LY12-5016-2

Data Language/I Disk Operating System/ Virtual Storage (DL/I DOS/VS)

Program Product

Logic Manual

Program Number 5746-XX1

This publication provides information on the internal operation of the DL/I system as an application program under DOS/VS. It is intended for use by persons involved in program maintenance and by system programmers who are altering the program design.

DL/I DOS/VS is a data management control system that assists the user in creating, accessing, and maintaining large common data bases. In conjunction with the Customer Information Control System (CICS/DOS/VS), DL/I DOS/VS can be used in an online teleprocessing environment.

Readers of this manual must be thoroughly familiar with the use of DOS/VS, and of CICS/DOS/VS, if DL/I DOS/VS is to be used in the online environment.

IBM

Third Edition (March 1977)

This edition applies to Version 1, Release 2 (Version 1.2) of IBM System/370 Data Language/I Disk Operating System/Virtual Storage (DL/I DOS/VS), Program Number 5746-XX1, and to all subsequent versions and modifications until otherwise indicated in new editions or Technical Newsletters. It supersedes LY12-5016-1 with Technical Newsletters LN12-5058, LN12-5060, and LN12-5076. Changes are continually made to the information herein; any such changes will be reported in subsequent revisions or Technical Newsletters.

Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Programming Publications, Dept. G60, P.O. Box 6, Endicott, New York 13760. Comments become the property of IBM.

This publication provides information on the internal operation of the DL/I system as an application program under DOS/VS. It is intended for use by persons involved in program maintenance and by system programmers who are altering the program design.

Because DL/I DOS/VS is a functional subset of the IBM Information Management System/Virtual Storage (IMS/VS), some specific IMS or OS terms are used in this manual. These terms are used to allow easy reference to the documentation of the related systems.

A list of abbreviations is provided at the back of this manual.

Related Publications

DL/I DOS/VS General Information Manual, GH20-1246.

DL/I DOS/VS Application Programming Reference Manual, SH12-5411.

DL/I DOS/VS Utilities and Guide for the System Programmer, SH12-5412.

DL/I DOS/VS System/Application Design Guide, SH12-5413.

DL/I DOS/VS Operator's Reference Manual and Messages and Codes, SH12-5414.

For DOS/VS messages and return codes:

DOS/VS Messages Reference, GC33-5379.

DOS/VS Access Method Services Manual, GC33-5382.

DOS/VS Supervisor and I/O Macros, GC33-5373.

Users employing DL/I DOS/VS in an online environment should have access to the following CICS/DOS/VS publications:

CICS/DOS/VS System Programmer's Reference Manual, SH20-9004.

CICS/DOS/VS Application Programmer's Reference Manual, SH20-9003.

CICS/DOS/VS System/Application Design Guide, SH20-9002.

CONTENTS

FIGURES

The DL/I batch system executes as an application program in a virtual storage environment under DOS/VS. The DOS/VS partition in which the DL/I batch system executes is composed of the elements shown in figure 1.1. These are:

- The system control facility
- The DL/I facility
- The DOS/VS VSAM and SAM data management modules
- The user application program

The major components of the DL/I system are the system control facility and the DL/I facility. The system control facility receives control from DOS/VS job control, initializes the DL/I batch system, and interfaces between DL/I and the user application program. The DL/I facility interfaces with the DOS/VS VSAM and SAM data management modules when performing the data base call function requested by the user application.

SYSTEM CONTROL FACILITY

The system control facility is divided into four functional areas (see Figure 1.2):

- Region control
- Application program control
- Language interface
- Program request handler.

Region control is responsible for a general group of housekeeping functions common to various optional processing modes of the DL/I DOS/VS partition (also called a region). These functions are:

- Initial interface with DOS/VS job management
- Analysis and validity checking of DL/I parameter information
- Loading the batch nucleus.

Application program control is entered from region control and performs the following functions:

- Loading the DL/I application control blocks (PSB and DMBs) and relocating the control block addresses.
- Creation of the PSB intent list and the DMB directory (DDIR).
- Acquiring and formatting storage for the buffer pool control blocks and their related I/O buffers.
- Loading the DL/I facility modules.
- Loading the application program and passing control to it.

The language interface provides communication between the application program and the program request handler. This module is link-edited with the application program and provides a common interface for DL/I calls written in PL/I, COBOL, or Assembler language.

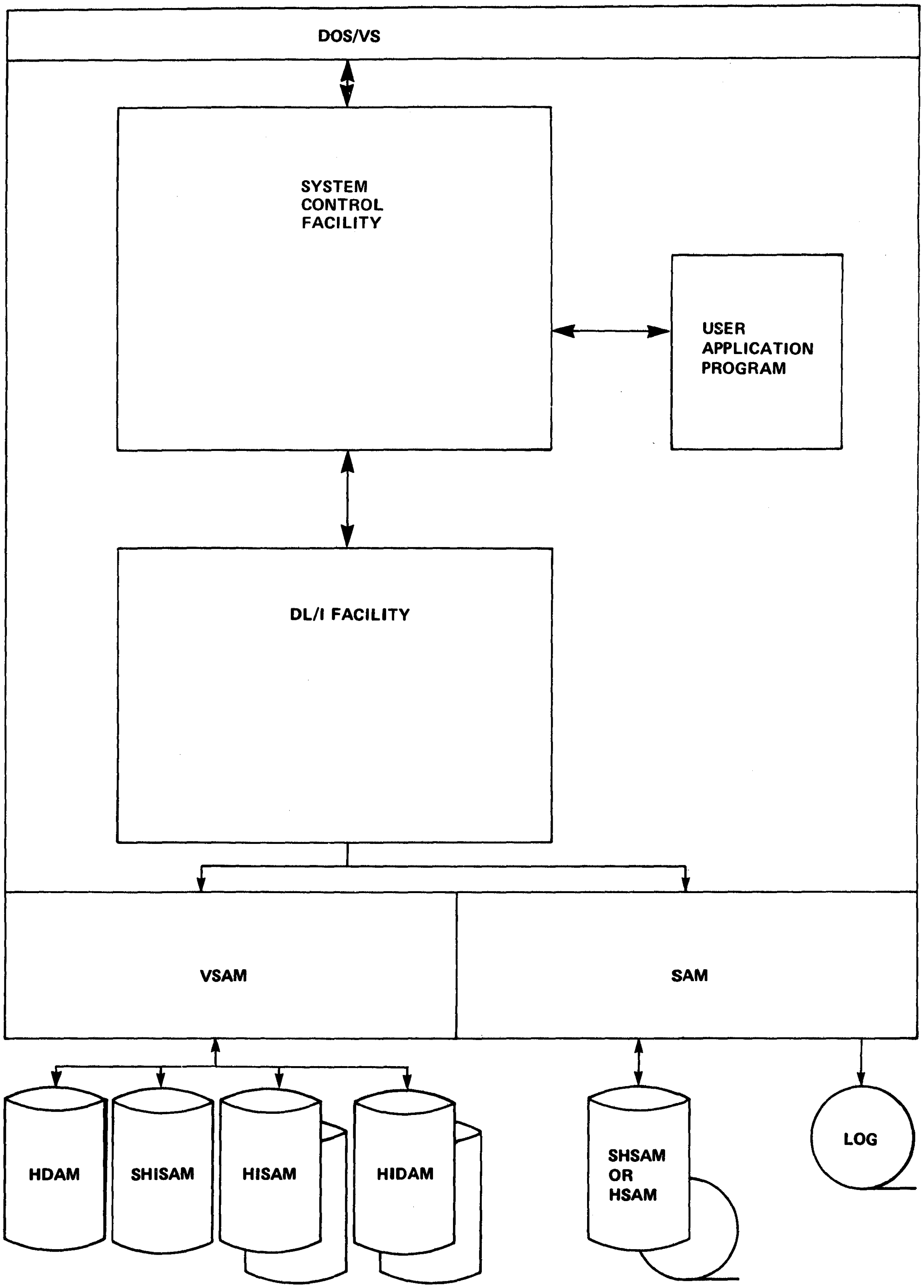


Figure 1.1. Elements of a DL/I DOS/VS Batch Partition

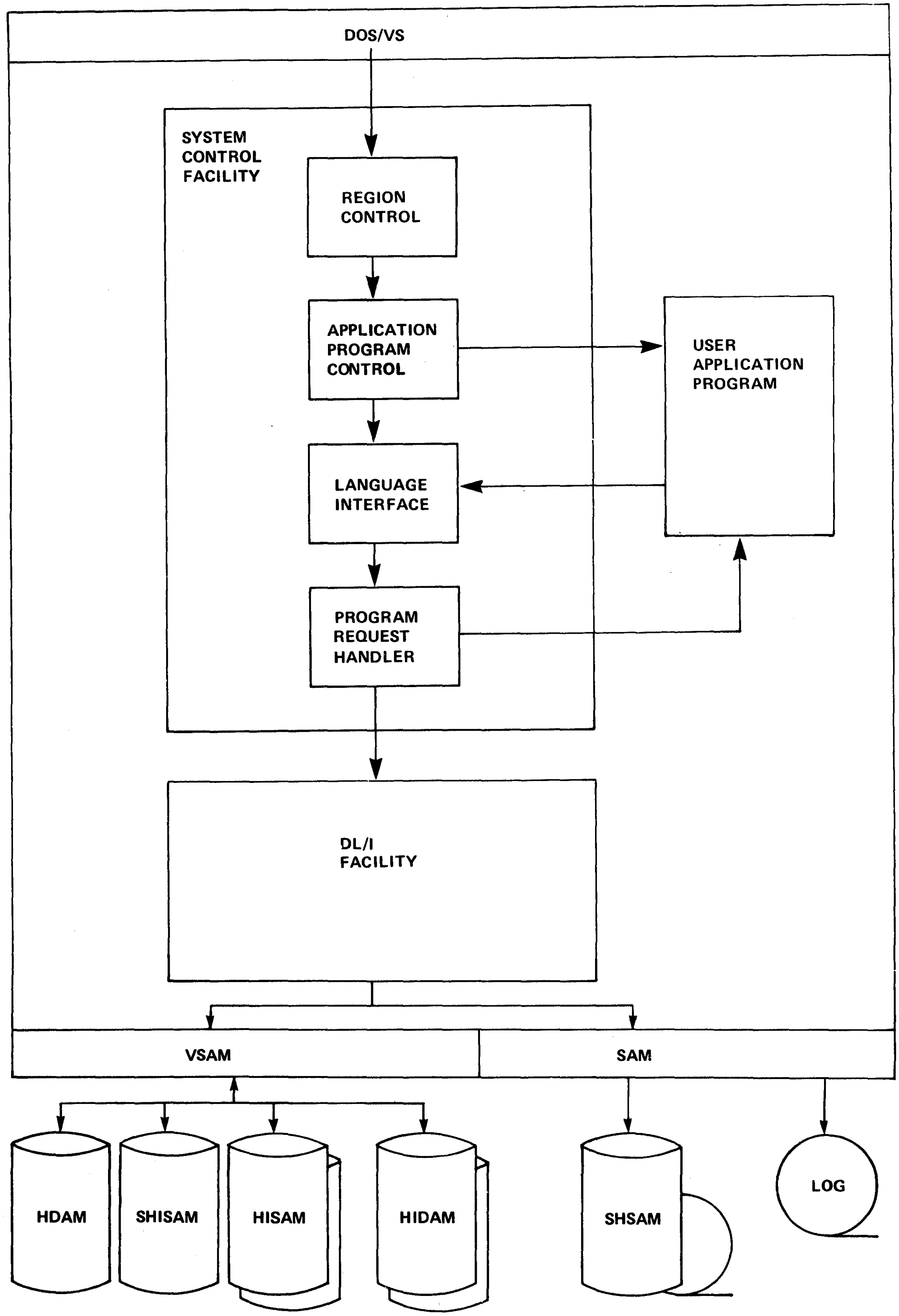


Figure 1.2. System Control Facility Relationships

The program request handler receives the DL/I call from the user application program via the language interface. It performs the following functions:

- Checks validity and, if necessary, reformats the caller's parameter lists and submits them to the DL/I facility.
- Accepts parameter lists from the DL/I facility and moves data to the user's work area, if required.
- Returns control directly to the user application program.

See Chapter 3 for a detailed description of each of these modules.

DL/I FACILITY

The functions of data base creation, access, maintenance, and reorganization are accomplished by the DL/I facility (see Figure 1.3). The DL/I call is passed from the system control facility to the DL/I call analyzer, which is the focal point of the DL/I facility. The type of call is analyzed (DL/I call, pseudo call, or internal call resulting from a DL/I call), and control is passed to the appropriate action module to process the call.
The action modules of the DL/I facility, together with their major functions, are listed below:

- Open/Close Module
    - Open DL/I data bases
    - Close DL/I data bases
    - Interface with data base logger to write data set open record to log tape
- Delete/Replace Module
    - Delete segment of DL/I data base in conjunction with buffer handler
    - Replace segment of a DL/I data base in conjunction with buffer handler
    - Interface with data base logger to record changes to log tape
    - Interface with space management for HDAM and HIDAM data bases
    - Interface with index maintenance for data bases with indexes
- Load/Insert Module
    - Load segments into a DL/I data base in conjunction with the buffer handler
    - Insert segments into a DL/I data base in conjunction with the buffer handler
    - Interface with data base logger to record changes to log tape
    - Interface with space management for HDAM and HIDAM data bases
    - Interface with index maintenance for data bases with indexes
    - Issue I/O for Simple HSAM and HSAM data bases
- Retrieve Module
    - Retrieve a segment of a DL/I data base in conjunction with the buffer handler
    - Perform data base positioning for load/insert
    - Issue I/O for Simple HSAM and HSAM data bases

- Index Maintenance

  - Maintain any indexes for HDAM or HIDAM data bases in conjunction with the buffer handler
  - Interface with data base logger to record changes to log tape

- Space Management

  - Allocate and maintain free space on DASD in conjunction with the buffer handler for storage of DL/I segments for HDAM and HIDAM data bases
  - Interface with data base logger to record changes to log tape

- Buffer Handler

  - For HDAM or HIDAM data base, satisfy requests for segments or records from data currently available in the buffer pool
  - Issue I/O to VSAM for HDAM or HIDAM data base requests that cannot be satisfied from the buffer pool
  - Issue I/O to VSAM for all Simple HISAM and HISAM data base requests

- Data Base Logger

  - Record all data base modifications on the log tape using DOS/VS SAM

See Chapter 4 for a detailed description of the modules.

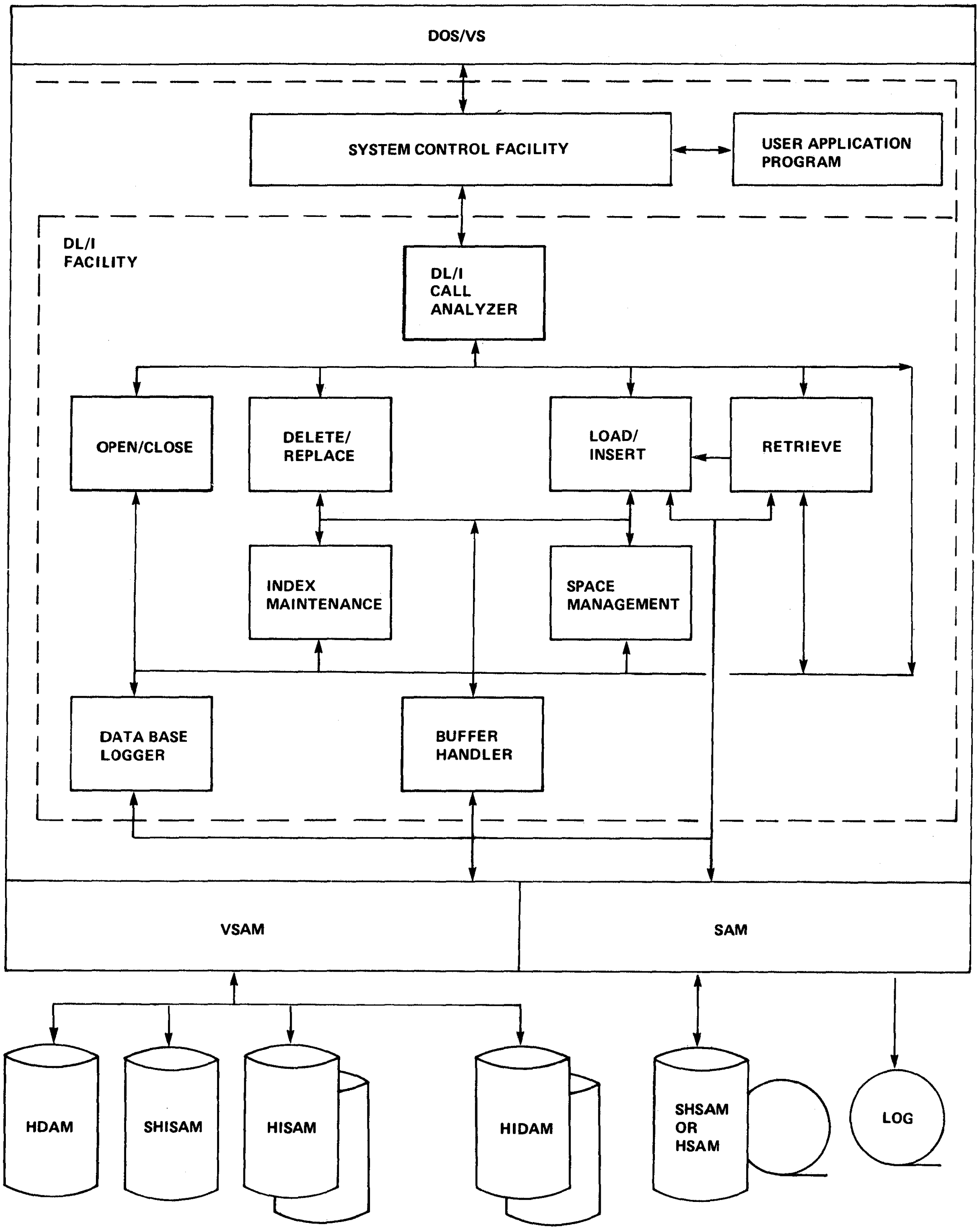
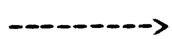


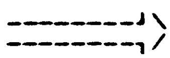
Figure 1.3. DL/I Facility Relationships

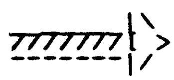
In this chapter, DL/I DOS/VS is divided into major functional areas. Within each area the functions of the various DL/I components are described in the form of HIPO (Hierarchy-Input-Processing-Output) charts.

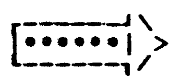
HIPO charts present a function in a more visual mode than flow charts, by showing for each component, what input is processed (left column), how it is processed (middle column), and what the resulting output is (right column). They are organized hierarchically, which means, for a gross overview a component's function, only the higher lever charts have to be read, while for more detailed information the reader can selectively go to more detailed lower level charts and eventually to the corresponding source module listings.

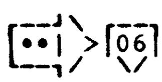
Graphic symbols used in the following HIPO charts are mostly self-explanatory. The meaning of the six kinds of arrows used is as follows:

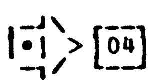
 Data reference

 Data movement

 Data modification

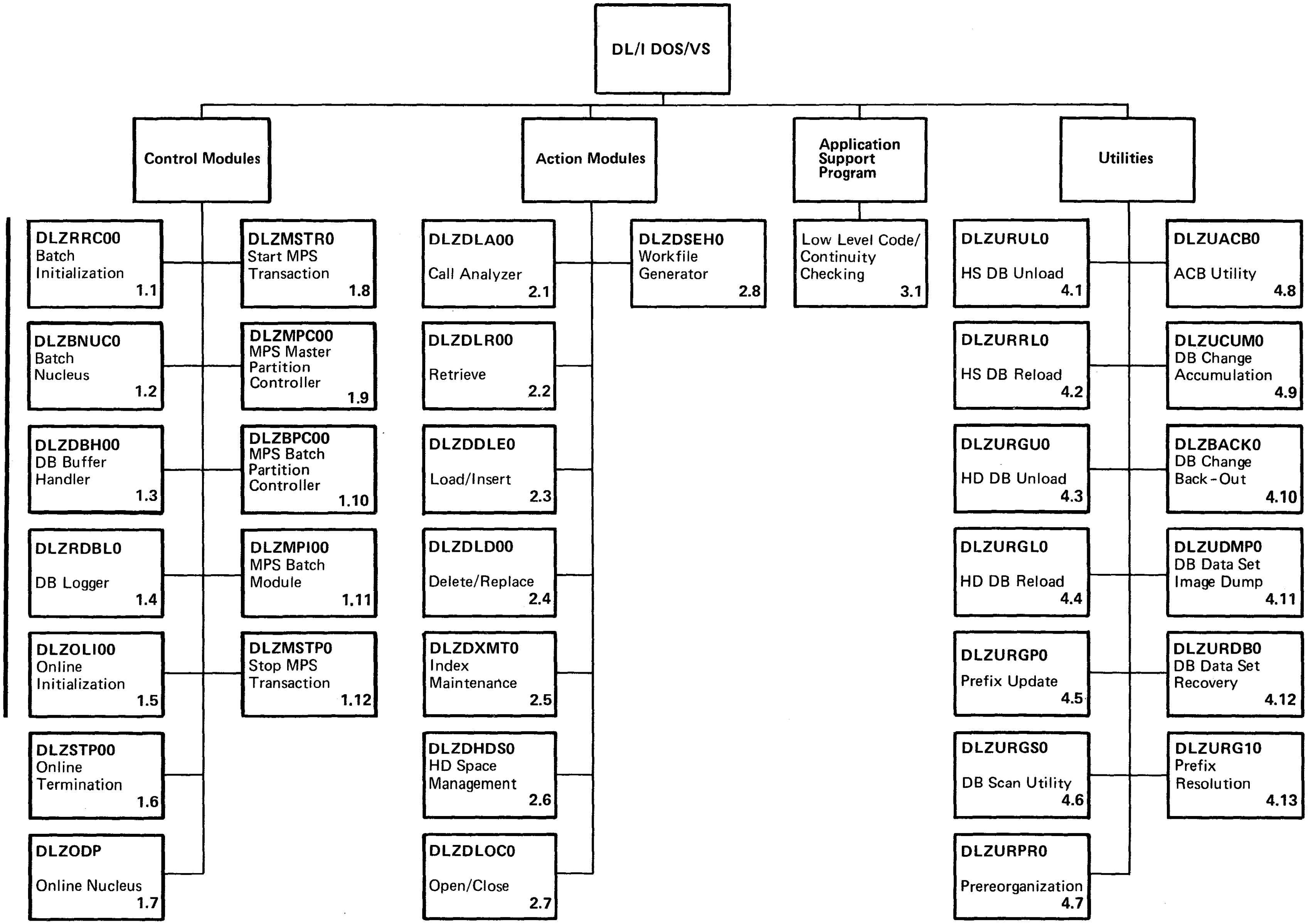
 Control flow

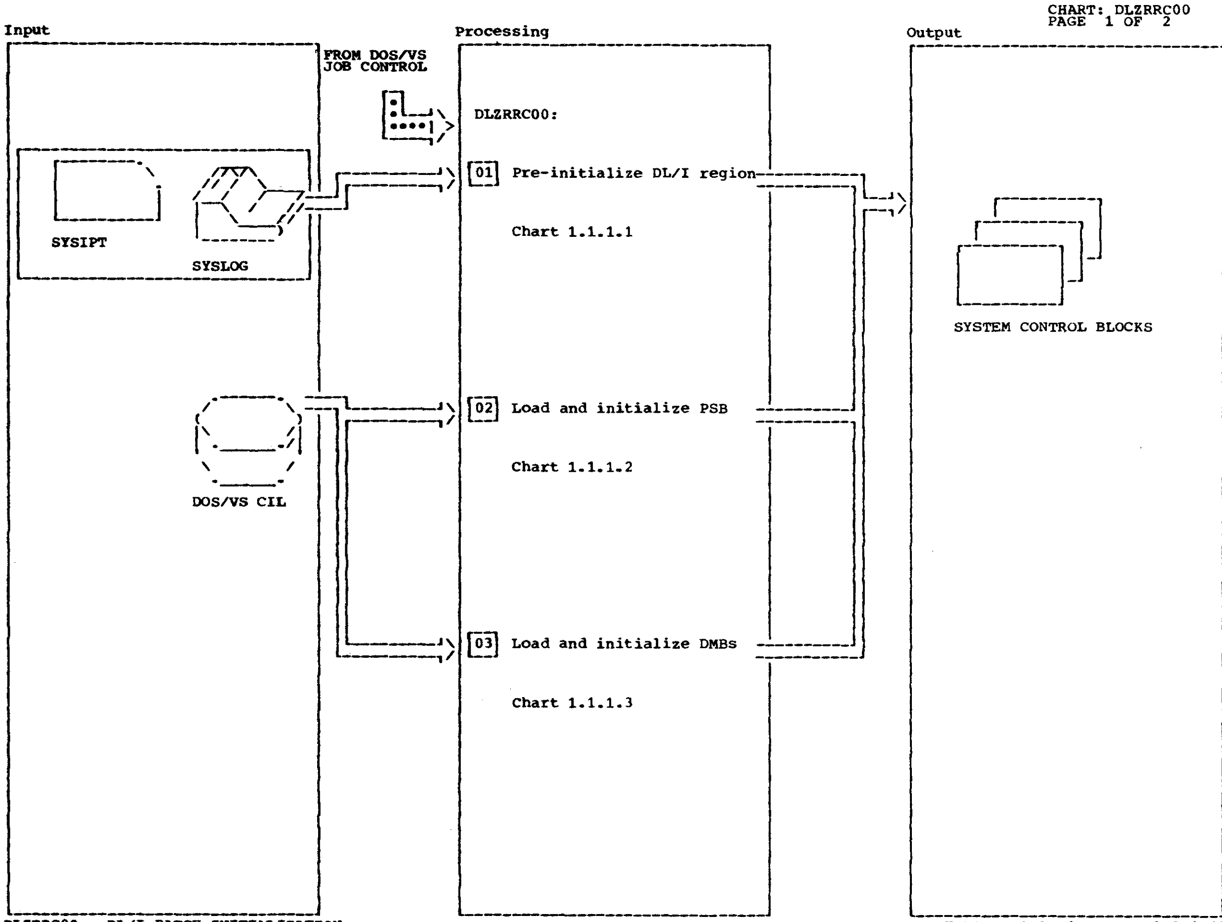
 Off-page connector

 On-page connector

The charts are numbered hierarchically; for instance, a chart x.y may refer to a lower level chart for more detailed explanation of one of it processing steps. The lower level chart would then be numbered x.y.z and so on.

Diagram 0.1 Visual Table of Contents for DL/I DOS/VS HIPO Charts





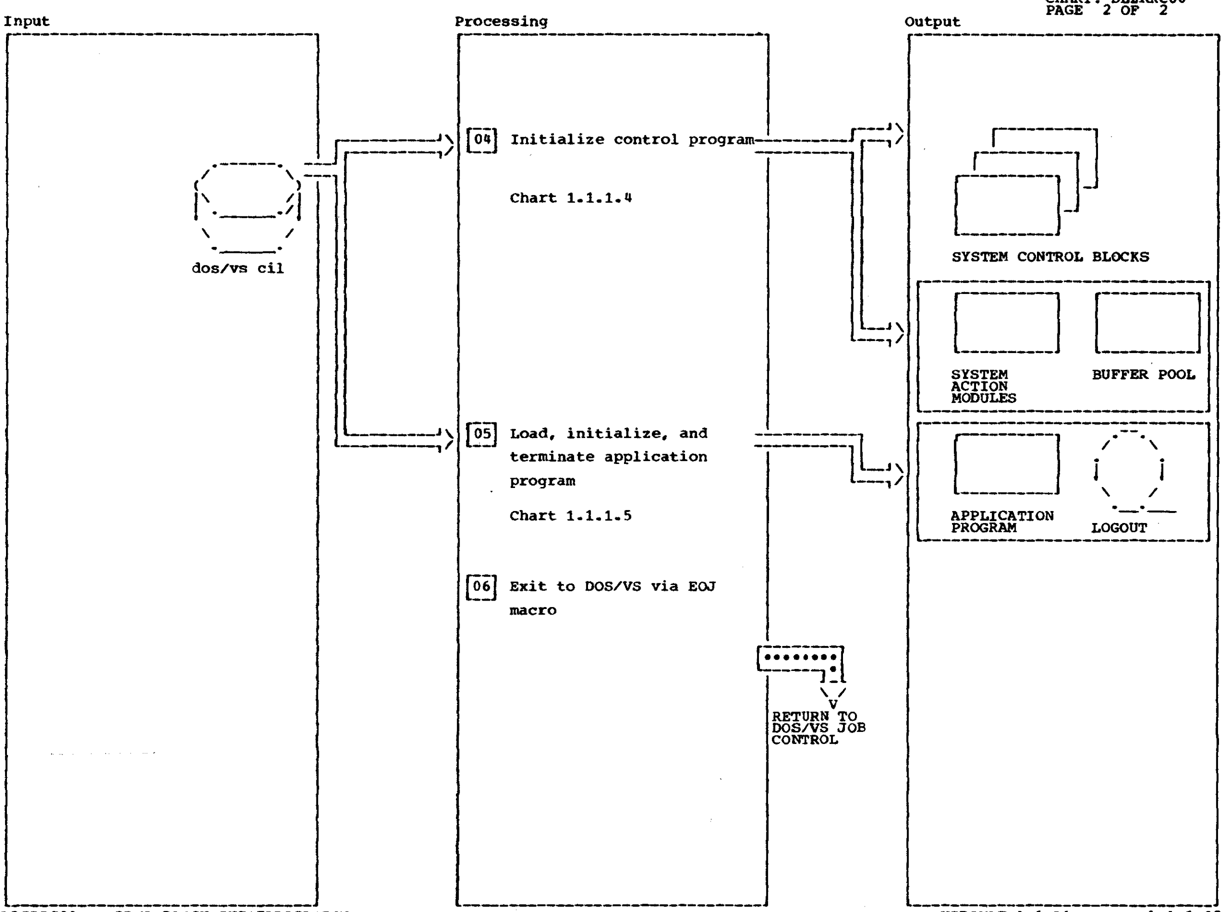
DLZRRC00 - DL/I BATCH INITIALIZATION

HIPOMAT 1.1 Diagram - 1.1.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

DLZRRC00 - DL/I BATCH INITIALIZATION

HIPOMAT 1.1 Diagram - 1.1.1-01

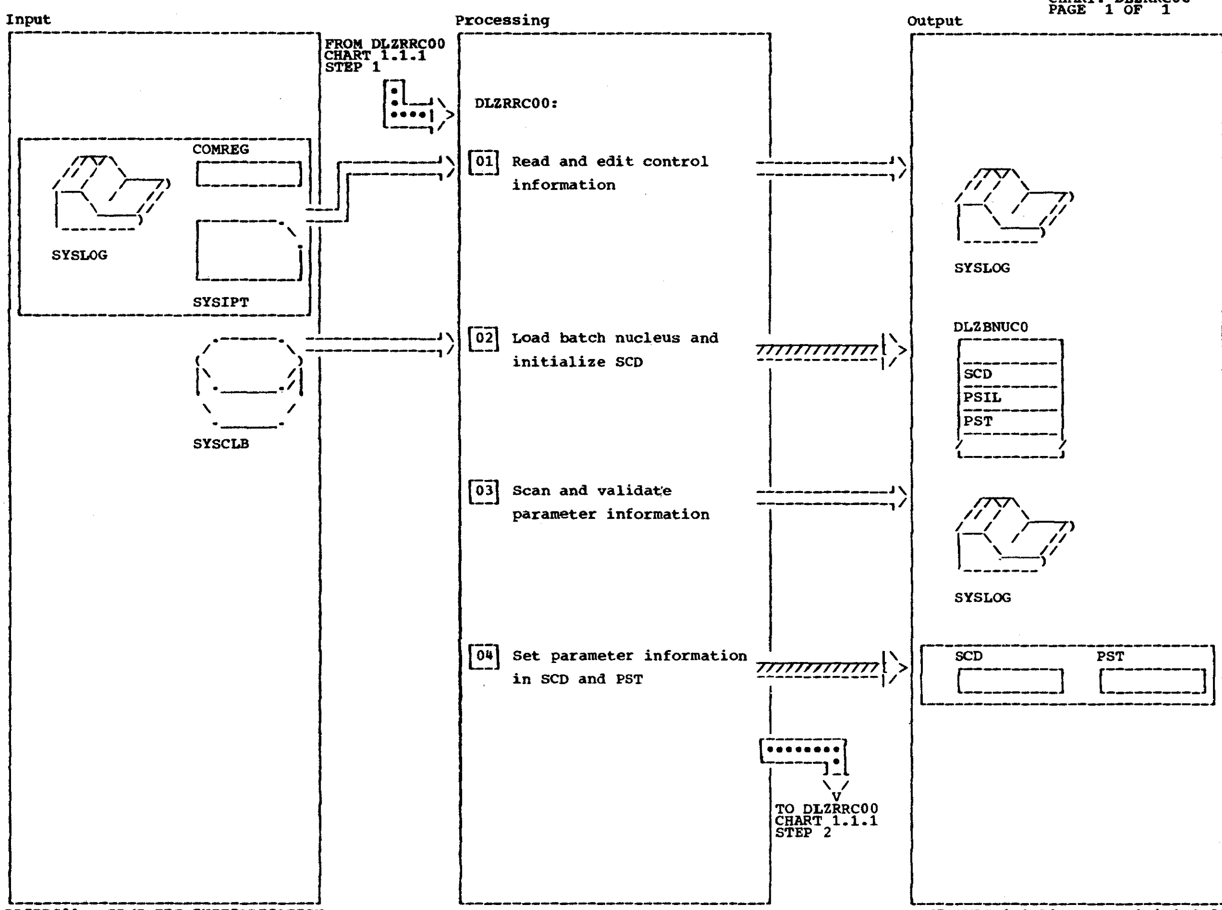


DLZRRC00 - DL/I BATCH INITIALIZATION

HIPOMAT 1.1 Diagram - 1.1.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

CHART: DLZRRC00
PAGE 1 OF 1



DLZRRC00 - DL/I PRE-INITIALIZATION

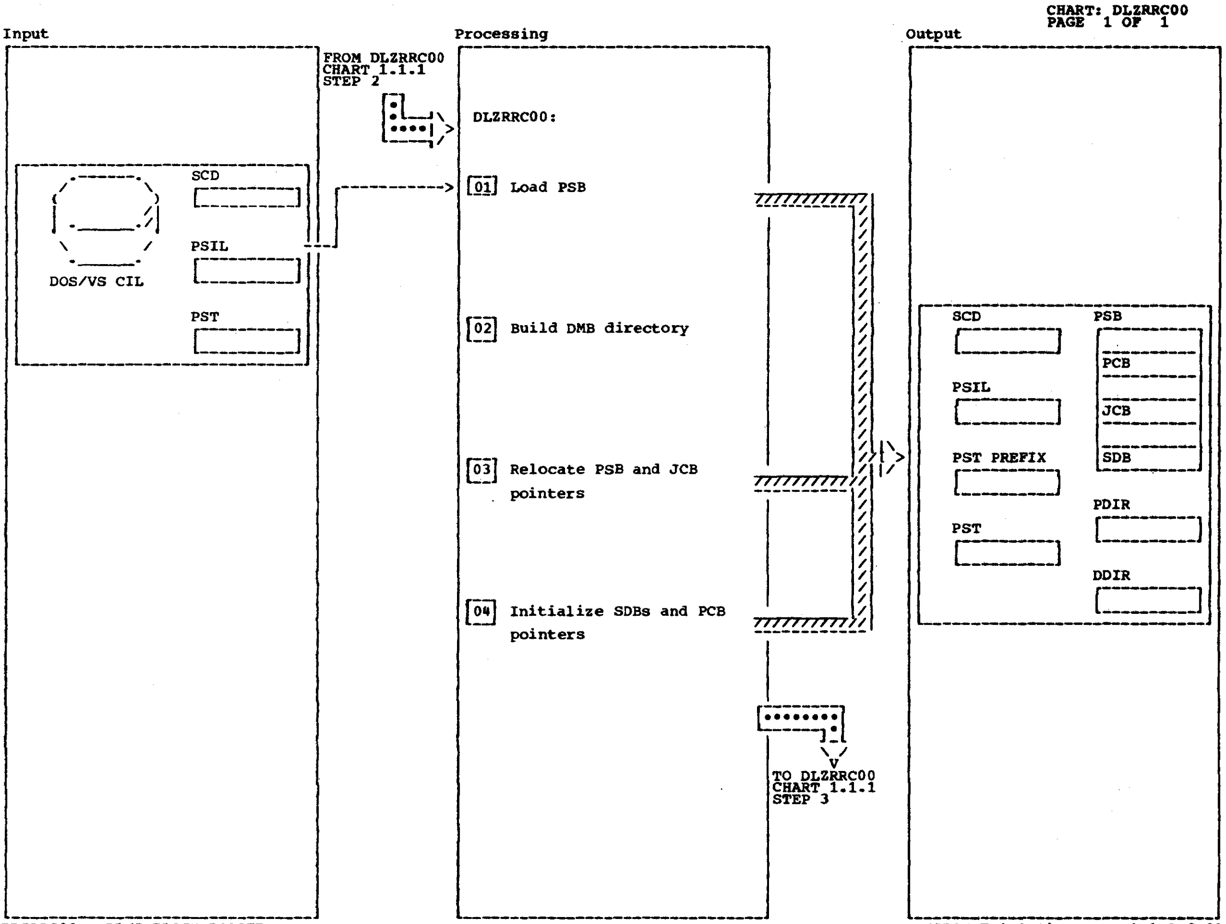
HIPOMAT 1.1 Diagram - 1.1.1.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

DLZRRC00 - DL/I PRE-INITIALIZATION

HIPOMAT 1.1 Diagram - 1.1.1.1-01

CHART: DLZRRC00
PAGE 1 OF 1



DLZRRC00 - DL/I BLOCK LOADER

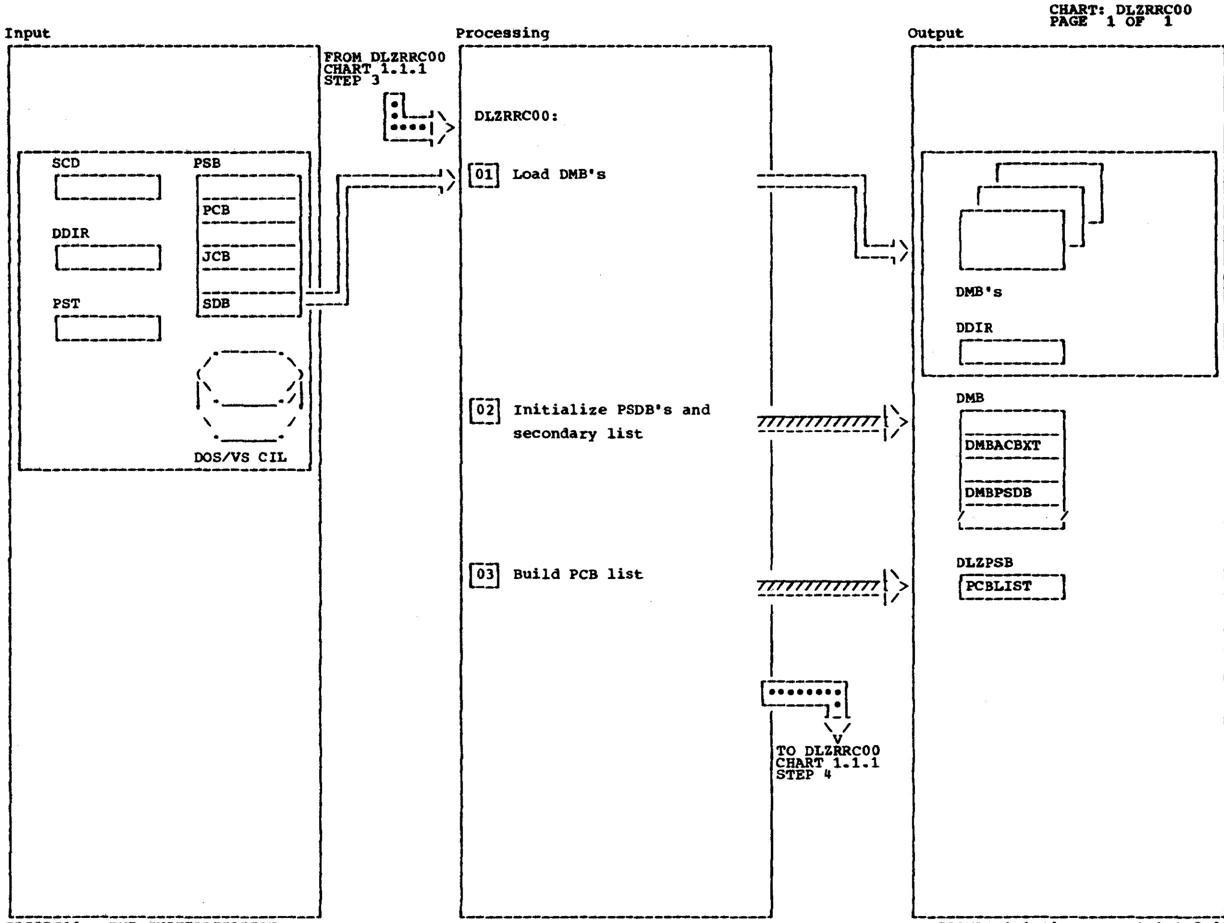
HIPOMAT 1.1 Diagram - 1.1.1.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

DLZRRC00 - DL/I BLOCK LOADER

HIPOMAT 1.1 Diagram - 1.1.1.2-01

CHART: DLZRRC00
PAGE 1 OF 1



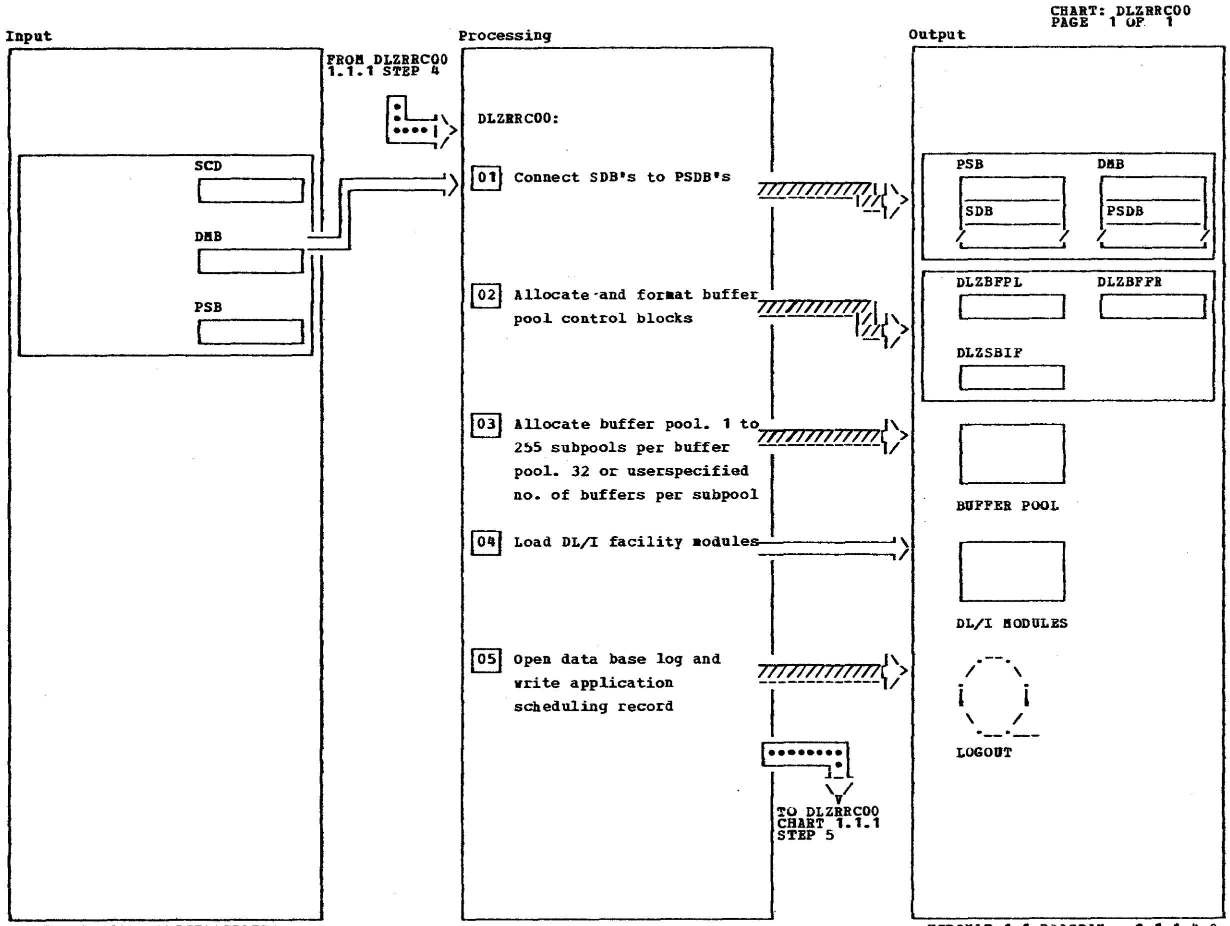
DLZRRC00 - DMB INITIALIZATION

HIPOMAT 1.1 Diagram - 1.1.1.3-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

DLZRRC00 - DMB INITIALIZATION

HIPOMAT 1.1 Diagram - 1.1.1.3-01



CONTROL PROGRAM INITIALIZATION

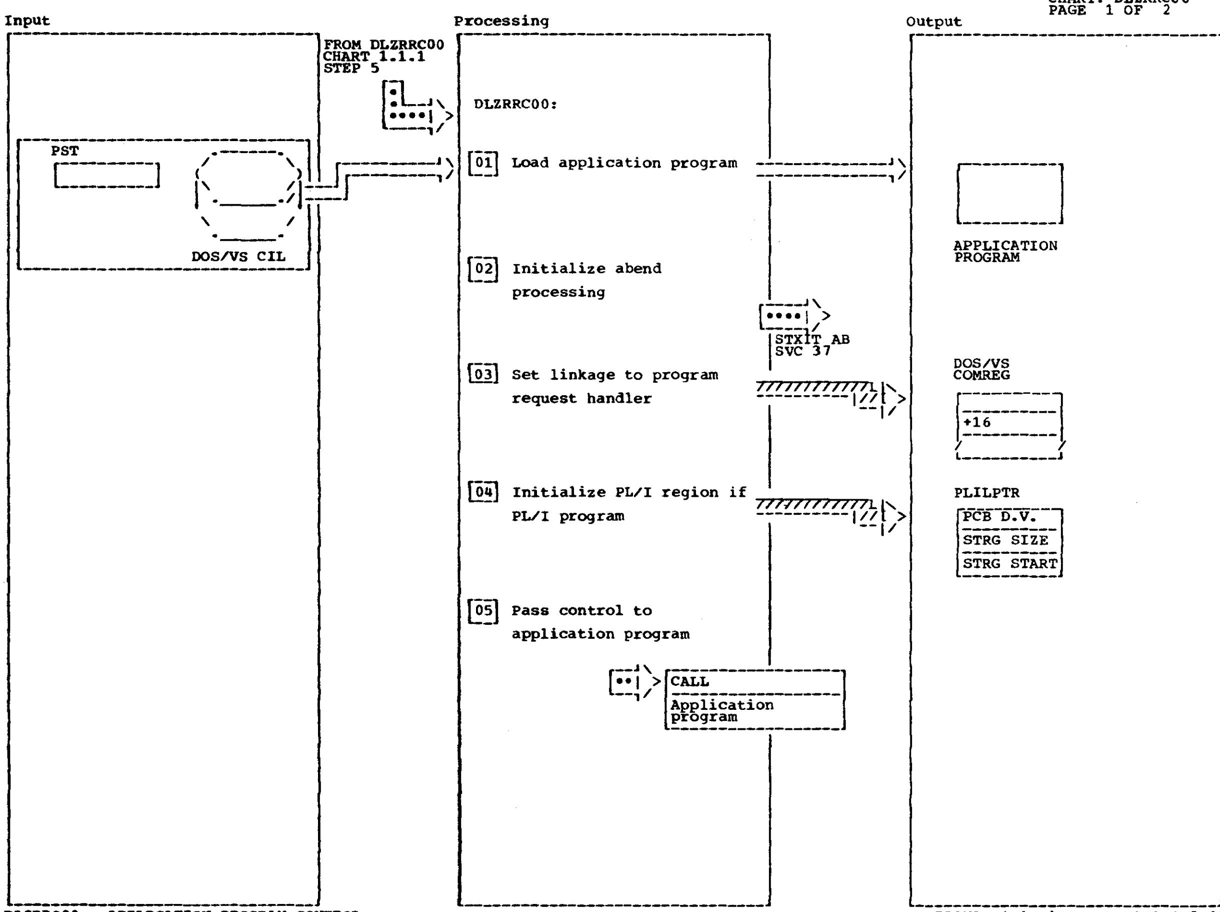
HIPOMAT 1.1 DIAGRAM - 1.1.1.4-0

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

CONTROL PROGRAM INITIALIZATION

HIPOMAT 1.1 DIAGRAM - 1.1.1.4-0

CHART: DLZRRC00
PAGE 1 OF 2



DLZRRC00 - APPLICATION PROGRAM CONTROL HIPOMAT 1.1 Diagram - 1.1.1.5-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
03 Linkage to program request handler is done via MOVECOM macro.							

DLZRRC00 - APPLICATION PROGRAM CONTROL HIPOMAT 1.1 Diagram - 1.1.1.5-01

Input	Processing	Output
	06 Log termination and close data base log	LOGOUT
	07 Issue UNLD call	
	DLZDLA00 UNLD call 2.1.1	
	TO CHART DLZRRC00 1.1.1 STEP 6	

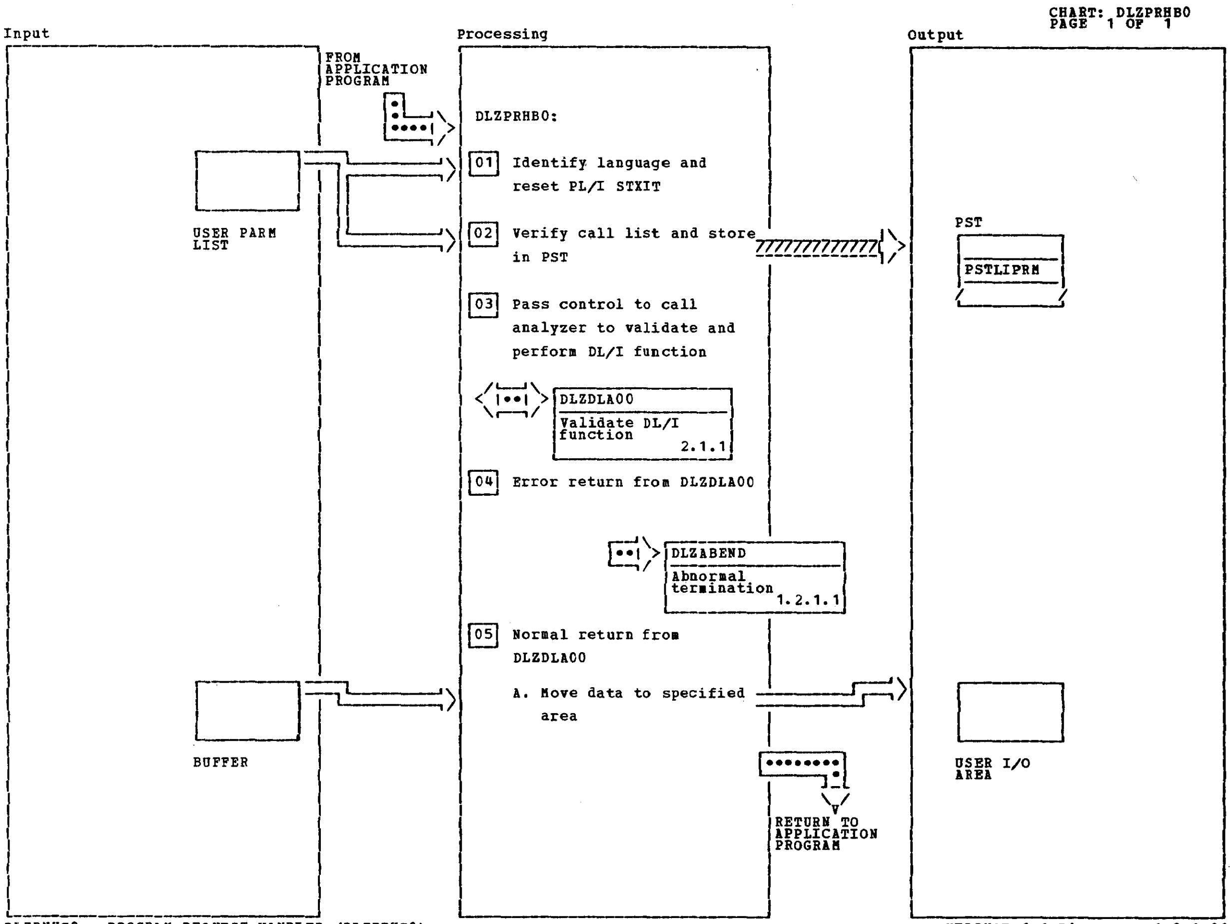
DLZRRC00 - APPLICATION PROGRAM CONTROL

HIPOMAT 1.1 Diagram - 1.1.1.5-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

DLZRRC00 - APPLICATION PROGRAM CONTROL

HIPOMAT 1.1 Diagram - 1.1.1.5-02



DLZBNUC0 - PROGRAM REQUEST HANDLER (DLZPRHB0) HIPOMAT 1.1 Diagram - 1.2.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 When control is passed to the program request handler, general purpose register 1 must point to the user call list and register 13 to a standard save area.							

Input | Processing | Output

FROM CALLER (NOTE 1)

DLZABEND:

01 Establish SCD address

02 Close data base log → LOGOUT

03 Close workfile if required.

04 Issue DLZ001 message → CONSOLE

DLZERRMS
Error message writer

05 Issue UNLD call

DLZDLA00
DL/I Analyzer Module 2.1.1

06 Issue DLZ002 message → CONSOLE

DLZERRMS
Error message writer

07 If DUMP option active, write formatted dump → SYSLST

DOS/VS SVC VIA DUMP OR EOJ MACRO

ABNORMAL TERMINATION

HIPOMAT 1.1 Diagram - 1.2.1.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 The abend routine is envoked by direct branch for DL/I pseudo abends and by DOS/VS LPSW in case of STXIT AB invocation.							
03 If the HD Reorganization Reload Module (DLZURGL0) is running for either a standard reload or reload restart, close the workfile if it is open.							
07 Dump module DLZFSDP0 is loaded into a GETVIS area and executed.							

ABNORMAL TERMINATION

HIPOMAT 1.1 Diagram - 1.2.1.1-01

Input	Processing	Output
FROM CALLER; PST: PSTFNCTN	DLZDBH00: What function is requested?	

DLZDBH00:

What function is requested?

[01] HD organization

A. Byte locate (chart 1.3.1.1)

B. Block locate (chart 1.3.1.2)

C. Byte alter (chart 1.3.1.3)

D. Mark buffer as altered (chart 1.3.1.4)

E. Get buffer space (chart 1.3.1.5)

F. Free buffer space (chart 1.3.1.6)

G. Mark buffer as empty (chart 1.3.1.6)

H. Purge all buffers modified by a given user (chart 1.3.1.7)

DLZDBH00 - BUFFER HANDLER

HIPOMAT 1.1 Diagram - 1.3.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[01]							
A. Return the address of a buffer which contains the requested data base segment identified by a RBA - or,if the requested CI is not yet existing,return the address of a buffer, into which the CI is to be inserted.							
B. Same as 1 A except that the request is for a data base CI identified by a CI number.							
C. Byte alter is the combination of byte locate and buffer alter.							

Input	Processing	Output
	[02] HISAM/HIDAM INDEX: A. Byte locate B. SETL equal C. SETL begin D. Add a record to a HISAM ESDS E. Insert a record into a HISAM KSDS F. Insert records in key order sequence into a HISAM KSDS G. Update a record in a HISAM KSDS or ESDS H. Get the next record from a HISAM KSDS I. Erase a record from a SHISAM data base [03] Exit RETURN TO CALLER	

DLZDBH00 - BUFFER HANDLER

HIPOMAT 1.1 Diagram - 1.3.1-02

Notes	Routine	Label	Ref
[02]			
A. Get a HISAM record(from KSDS or ESDS) by using an RBA.	DLZDBH02	HSREAD	
B. Get a HISAM record (KSDS) by root key.	DLZDBH02	STLEQ	
C. Get the HISAM record with the first root segment in the data base.	DLZDBH02	STLBG	
D.	DLZDBH02	LOWRITE	
E.	DLZDBH02	PUTKY	
F.	DLZDBH02	MSPUT	
G.	DLZDBH02	HSWRITE	
H.	DLZDBH02	GETNX	
I.	DLZDBH02	HSWRITE	

Notes	Routine	Label	Ref

DLZDBH00 - BUFFER HANDLER

HIPOMAT 1.1 Diagram - 1.3.1-02

Input	Processing	Output
PST: PSTBYTNM	FROM DLZDBH00 CHART 1.3.1 STEP 1A	
	BYLCT:	
	01 Convert the given RBA to a CI number and an offset	PST: PSTBLKNM, PSTOFFST
	02	
	LOCATE 1.3.1.8	
	03 Exit	
	TO DLZDBH00 CHART 1.3.1 STEP 3	

DLZDBH00 - BYTE LOCATE

HIPOMAT 1.1 Diagram - 1.3.1.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01	DLZDBH00	CONVER					

Input

Processing

Output

FROM DLZDBH00
CHART 1.3.1
STEP 1B

BKLCT:

01

LOCATE
1.3.1.8

PST
PSTBLKNM

02 Convert the CI number to a RBA

PST
PSTBYTNM

03 Exit

TO DLZDBH00
CHART 1.3.1
STEP 3

DLZDBH00 - BLOCK LOCATE

HIPOMAT 1.1 Diagram - 1.3.1.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

Input	Processing	Output
	FROM DLZDBH00 CHART 1.3.1 STEP 1 C	
	BYALT:	
PST: PSTBYTNM	01 Convert the RBA to a block number and an offset	PST: PSTBLKNM, PSTOFFST
	02	
	LOCATE 1.3.1.8	
PPST: PPSTID	03 Turn on a bit in the buffer prefix indicating that this user altered this buffer	BUFFER PREFIX
	04 Exit	
	TO DLZDBH00 CHART 1.3.1 STEP 3	

DLZDBH00 - BYTE ALTER

HIPOMAT 1.1 Diagram - 1.3.1.3-01

Notes	Routine	Label	Ref
03 See note for BFALT routine (chart 1.3.1.4).			

Notes	Routine	Label	Ref

DLZDBH00 - BYTE ALTER

HIPOMAT 1.1 Diagram - 1.3.1.3-01

Input	Processing	Output
PPST: PPSTID	FROM DLZDBH00 CHART 1.3.1 STEP 1 D	BUFFER PREFIX
	BFALT:	
	01 Turn on the bit in the buffer prefix indicating that this user modified this buffer	
	02 Exit	
	TO DLZDBH00 CHART 1.3.1 STEP 3	

DLZDBH00 - MARK BUFFER AS ALTERED

HIPOMAT 1.1 Diagram - 1.3.1.4-01

Notes	Routine	Label	Ref
01 The bit is turned on in the 2-byte field BFFRUSID.The 16 bits correspond from right to left to the user id as indicated in the PPST. If a higher user id than 16 is assigned two or more users share the same bit.	DLZDBH00	MARKALT	

Input | Processing | Output

FROM DLZDBH0C
CHART 1.3.1
STEP 1 E

PST
PSTBYTNM

SUBP INFORM TABLE

BUFFER PREFIXES

BFPL
BFPLRQCT

GBSPC:

01 Get the address of the appropriate buffer subpool

02 Search thru the buffer prefixes of the subpool to find a buffer that can be used

03 If a reusable buffer couldn't be found, wait until a buffer becomes available

DLZOWAIT 1.7.5

04 Move the CI id into the buffer prefix

05 If the buffer is busy, wait for it

DLZOWAIT 1.7.5

BUFFER PREFIX

DLZDBH00 - GET BUFFER SPACE

HIPOMAT 1.1 Diagram - 1.3.1.5-01

Notes	Routine	Label	Ref
01 The subpool information table is used to find the buffer subpool with the buffers containing the least excessive number of bytes for this space request.			
02 Only buffers are used that are not 'non reusable' and that are not permanent write error buffers.			

Input	Processing	Output
DL/I BUFFER	06 If the buffer needs to be written, write it	DATA BASE
A	07 Mark buffer as 'non reusable'	B
	08 Put the address of the buffer and the size of it into the PST	PST: PSTDATA, PSTBYTNM
	09 Exit	

TO DLZDBH00
CHART 1.3.1
STEP 3

DLZDBH00 - GET BUFFER SPACE

HIPOMAT 1.1 Diagram - 1.3.1.5-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

Input

PST
- PSTBLKNM
- PSTDMBNM
- PSTACBNM

DMB SUBPOOL DIR

BUFFER PREFIXES

BUFFER PREFIX

Processing

FROM DLZDBH00
CHART 1.3.1
STEP 1 F AND G

MRKEMPT:

[01] Get the first buffer prefix in the subpool that applies

[A] [02] If PSTDMBNM,PSTACBNM,PSTBLKNM don't match BFFRDMB,BFFRDCB,BFFRCIID , go to step 4

[A] [A] [03]

A. Mark the buffer as empty

B. Issue RELPAG macro

C. Put the buffer on the bottom of the use chain

[04] If this is not the last buffer prefix, get the next buffer prefix and go to step 2

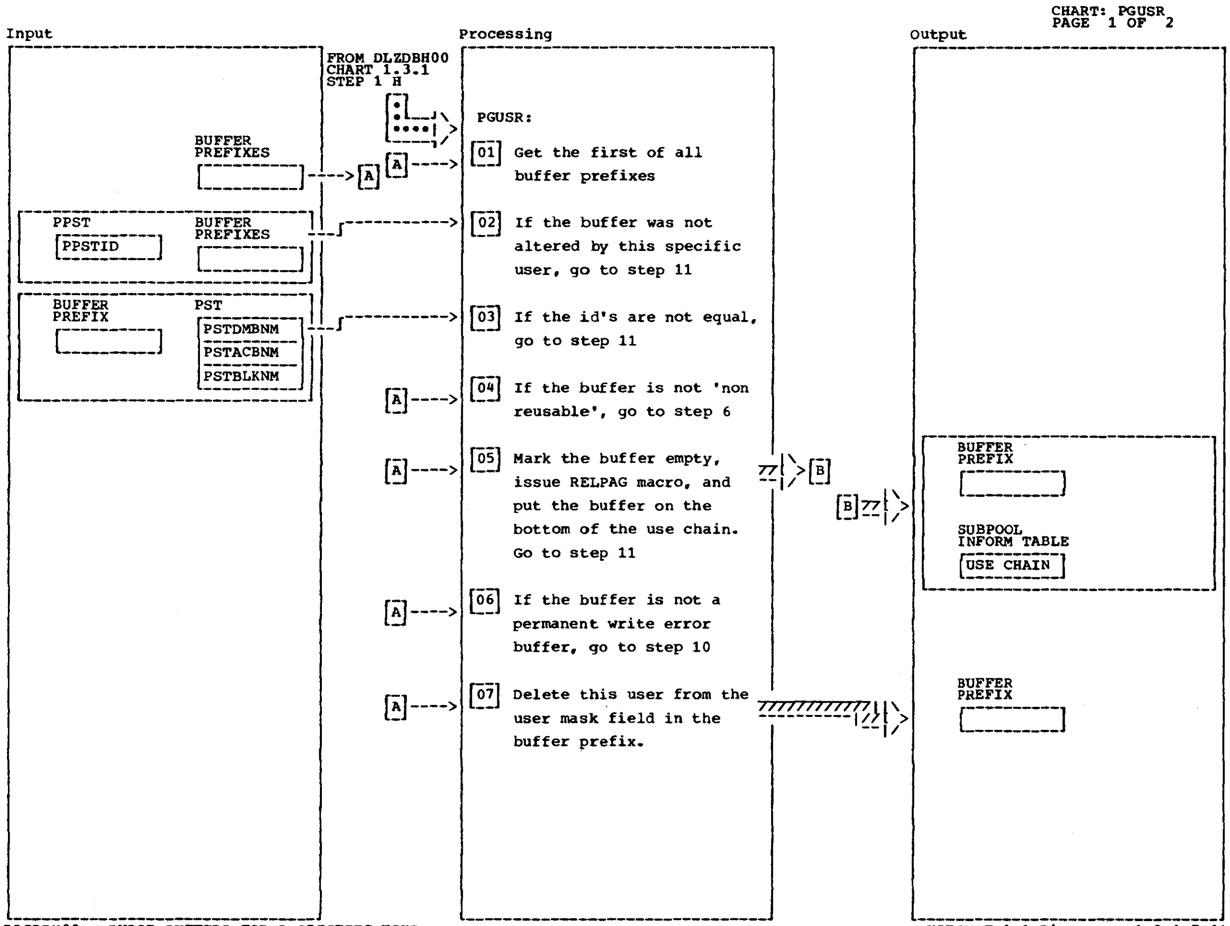
TO DLZDBH00
CHART 1.3.1
STEP 3

Output

BUFFER PREFIXES

SUBPOOL INFORM TABLE

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[01] To find the buffer subpool that applies to the call , the DMB SUBPOOL DIR and the DMB number in PSTDMBNM are used.							
[02] The caller can have the buffer handler free only one buffer(PSTDMBNM,PSTACBNM, PSTBLKNM not 0) or all buffers of a data set (PSTDMBNM,PSTACBNM not 0,PSTBLKNM=0) or all buffers of a data base(PSTDMBNM = DMB number of the data base,PSTACBNM,PSTBLKNM=0).							



DLZDBH00 - PURGE BUFFERS FOR A SPECIFIC USER HIPOMAT 1.1 Diagram - 1.3.1.7-01

Notes	Routine	Label	Ref
01 This routine scans thru all buffer prefixes.			
03 The caller may select a certain data base, a certain data set or a certain buffer to be purged. He indicates his choice by putting the number of the desired item into PSTDMBNM, PSTACBNM or PSTBLKNM. Zeroes in these fields indicate that purging of all components of the item on the next higher level is desired. This module checks the contents of the above mentioned PST fields against the contents of BFFRDMB, BFFRDCB, BFFRCIID in the buffer prefix.			
04 Buffers that are 'non reusable' are freed during a purge call.			
06 Permanent write error buffers are not freed until all tasks, which either altered the buffer or might be interested in it, because they use the data base, have terminated.			

Notes	Routine	Label	Ref
07 Before the bit in BFFRUSID, which corresponds to the user id (in the PPST) of the current task, is turned off, a check is made whether any tasks are active that would share the bit with the current task (refer to notes of the chart for routine BFALT).			

DLZDBH00 - PURGE BUFFERS FOR A SPECIFIC USER HIPOMAT 1.1 Diagram - 1.3.1.7-01

Input

PSB
PCB
JCB
DSG

BUFFER PREFIX

DL/I BUFFER .
DATA BASE CI

Processing

08 If BFFRUSID is zero now or if there are no more potential users for this buffer,

A. Mark the buffer empty

B. Issue RELPAG macro

C. Put the buffer on the bottom of use chain

B

09 Go to step 11

10 Write buffer to disk

11 If this is not the last buffer prefix, get the next buffer prefix, go to step 2

12 Exit

TO DLZDBH00
CHART 1.3.1
STEP 3

Output

DATA BASE

DLZDBH00 - PURGE BUFFERS FOR A SPECIFIC USER

HIPOMAT 1.1 Diagram - 1.3.1.7-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
08 A task is a potential user of a buffer if at least one of the DSG's in the PSB has the same DMB and ACB number as in BFFRDMB and BFFRACB of the buffer prefix.							

Input

BUFFER PREFIXES

PST

DMB

DATA BASE

Processing

FROM DLZDBH00 CHARTS 1.3.1.1 OR 1.3.1.2

LOCATE:

01 Search a buffer for the requested block

Chart 1.3.1.8.1

02 Check,if the predecessor in the write chain is in the buffer pool and if necessary write buffer

Chart 1.3.1.8.2

03 If the block is new,put the buffer on the write chain and mark it as altered

Chart 1.3.1.8.3

04 If the block is not new, read the block into the buffer

Chart 1.3.1.8.4

05 Exit

TO DLZDBH00 CHARTS 1.3.1.1 OR 1.3.1.2

Output

DATA BASE

BUFFER PREFIXES

DL/I BUFFER

DLZDBH00 - LOCATE

HIPOMAT 1.1 Diagram - 1.3.1.8-0

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

Input

BUFFER PREFIXES

PST
- PSTBLKNM
- PSTACBNM
- PSTDMBNM

Processing

FROM LOCATE CHART 1.3.1.8 STEP 1

LOCATE:

01 Search thru the buffer prefixes for the requested data base CI

A. If the CI was not found, go to step 4

B. If the CI was found as pending in the buffer pool,enq on this CI. After having gotten control back, go to step 1

C. If the CI was found as existing in the buffer pool and is not busy, go to step 2

D. If the CI was found as existing in the buffer pool and is busy,enq on this CI. After having gotten control back go to step 1

02 Pass the buffer to the requestor

Output

PST
- PSTBUFFA
- PSTDATA

DLZDBH00 - LOCATE

HIPOMAT 1.1 Diagram - 1.3.1.8.1-01

Notes	Routine	Label	Ref
02 Passing the buffer consists of putting the buffer prefix address into PSTBUFFA and the buffer address into PSTDATA.			

Input

BUFFER PREFIXES

SUBPOOL INFORM TABLE

USE CHAIN

Processing

[03] Exit

RETURN TO CALLER

[04] Go on the use chain from bottom to top and search for a buffer that can be used

[05] If no buffer is available, enq on the pending CI of the buffer being on the bottom of the use chain. After having gotten control back go to step 4

[06] Exit

TO LOCATE CHART 1.3.1.8 STEP 2

Output

DLZDBH00 - LOCATE

HIPOMAT 1.1 Diagram - 1.3.1.8.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[04] A buffer can be used, if it is not marked as 'non reusable' and if it is not a permanent write error buffer and if it is currently not enqueued upon for a pending CI.							

DLZDBH00 - LOCATE

HIPOMAT 1.1 Diagram - 1.3.1.8.1-02

Input | **Processing** | **Output**

FROM LOCATE
CHART 1.3.1.8
STEP 2

BUFFOUND:

Input:

PST
- PSTBLKNM
- PSTACBNM
- PSTDMBNM

DMB
- DMBRLBLK

BUFFER PREFIXES

Processing:

01 Move the CI id into the buffer prefix of the buffer that is to be used

02 If the CI, which is being processed, is not new, go to step 5

03 If the predecessor in the write chain can be found in the buffer pool, go to step 5

04 Sequence error

Exit

RETURN TO CALLER

05 If the buffer is busy,enq on this buffer

06 If the buffer needs not to be written, go to step 8

Output:

BUFFER PREFIX

PST
- PSTRTCDE

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 This moving in of the CI id means enqueuing on a pending CI.							
02 This check is made to make sure that the CI's of the data base get initialized in sequence.							
04 x'04' is being stored into PSTRTCDE.							
05 A buffer is busy, if it is being read into, or being written or if it is waiting for its predecessor in write chain being written.							

Input	Processing	Output
DL/I BUFFER	[07] Write the buffer	DATA BASE
	[08] Take buffer over	BUFFER PREFIX; SUBPOOL INFORM TABLE: USE CHAIN
	[09] Exit	

TO LOCATE CHART 1.3.1.8 STEP 3

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[08] 'Taking over' a buffer consists of moving the CI id from BFFRNPST to BFFRPST, turning off BFFRPNNQ/turning on BFFREXNQ in BFFRSW, putting the buffer on the top of the use chain and clearing the buffer with zeroes.	DLZDBH00	BFSWAP					

DLZDBH00 - LOCATE
HIPOMAT 1.1 Diagram - 1.3.1.8.2-02

Input | Processing | Output

FROM LOCATE
CHART 1.3.1.8
STEP 3

TESTNEW1:

BUFFER
PREFIXES
---> A

A ---> 01 Search in buffer pool for the predecessor in the write chain. If it is found, go to step 3

02 The predecessor cannot be found - system abend

DLZABEND
ABEND 845
1.2.1.1

A ---> 03 If the predecessor is not being written, go to step 5

04 Enq on predecessor

A ---> 05 If the predecessor is not a permanent write error buffer, go to step 7

06 Mark the current buffer as a permanent write error buffer. Go to step 9 ---> B

B ---> BUFFER
PREFIX

DLZDBH00 - LOCATE

HIPOMAT 1.1 Diagram - 1.3.1.8.3-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
04 The purpose for enqueuing on the predecessor is to wait for completion of the writing. This is necessary to find out if the buffer then is a permanent write error buffer.							

DLZDBH00 - LOCATE

HIPOMAT 1.1 Diagram - 1.3.1.8.3-01

Input

PPST
PPSTID

Processing

07 Put the buffer on the bottom of the write chain — B

08 Mark buffer as altered — B

09 Put buffer prefix address and buffer address into the PST

10 Exit

TO LOCATE CHART 1.3.1.8 STEP 5

Output

PST
PSTBUFFA
PSTDATA

DLZDBH00 - LOCATE

HIPOMAT 1.1 Diagram - 1.3.1.8.3-02

Notes	Routine	Label	Ref

Notes	Routine	Label	Ref

DLZDBH00 - LOCATE

HIPOMAT 1.1 Diagram - 1.3.1.8.3-02

Input

DATA BASE

RPL
RPLFDBK

Processing

FROM LOCATE
CHART 1.3.1.8
STEP 4

ISSREAD:

01 Read the requested CI from the data base

02 If no read error occurred, go to step 4

03 Indicate in the PST, that a I/O error occurred

Exit

RETURN TO CALLER

04 Put the buffer prefix address and the buffer address into the PST

05 Exit

TO LOCATE
CHART 1.3.1.8
STEP 5

Output

DL/I BUFFER

PST
PSTRTCDE

PST
PSTBUFFA
PSTDATA

DLZDBH00 - LOCATE

HIPOMAT 1.1 Diagram - 1.3.1.8.4-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
03 A return code of X'08' (PSTIOERR) is stored into PSTRTCDE.							

Input

Processing

FROM CALLER

DLZRDBL0:

What function is requested? (note 5)

[01] Build a log record and move it to the log I/O area (chart 1.4.1.1)

[02] Move a log record, which has been built by another module, to the log I/O area (chart 1.4.1.2)

[03] Write log information physically to tape (chart 1.4.1.3)

[04] Exit

RETURN TO CALLER

Output

DLZRDBL0 - LOG MODULE

HIPOMAT 1.1 Diagram - 1.4.1-01

Notes	Routine	Label	Ref
[01]	DLZRDBL0	DLZIDBL0	
[02] This applies to scheduling and termination log records built by the scheduling resp. termination routine.	DLZRDBL0	LOGWR	
[03] This function is used by DLZDBH00, when log information associated with a data base update has not yet been written to tape at the time the data base update is being done .	DLZRDBL0	WRIAHEAD	
[05] The three different functions of this module are associated with three different entry points into it.			

Notes	Routine	Label	Ref

DLZRDBL0 - LOG MODULE

HIPOMAT 1.1 Diagram - 1.4.1-01

Input

JCB
- JCBPRESF

PST
- PSTBYTNM
- PSTBLKNM
- PSTDATA
- PSTOFFST
- PSTWRK1 - PSTWRK4

DSG
- DSGDCBA
- DSGINDA
- DSGDMBNO
- DSGACBNO

DMB
- DCBLRECL
- DMBDL
- DMBPRSZ

DL/I OR VSAM BUFFER

SCD
- SCDCWRKL

LOG WORK AREA

FROM DLZRDBLO
CHART 1.4.1
STEP 1

Processing

DLZIDBLO:

01 Build the log record

A. Detect the kind of log call

B. Move header information into workarea

C. Move data into workarea

D. If data does not all fit into record ,move as much data as possible into the log record

02 Move date and time into log record

03 Move the log record to the I/O area

A. If log record will not fit into I/O AREA go to step 5

B. Move the log record into I/O AREA

04 Go to step 6

Output

LOG WORK AREA

IOAREA

DLZRDBLO - LOG RECORD MOVING

HIPOMAT 1.1 Diagram - 1.4.1.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Dependant on the kind of DL/I call which is being processed, the logger builds a log record of one of the following types : Physical insert record physical replace record physical delete record logical delete record pointer maintenance record The maximum logical recordsize for a log record is 512. The blocks are undefined with a maximum of 1024 bytes.							

DLZRDBLO - LOG RECORD MOVING

HIPOMAT 1.1 Diagram - 1.4.1.1-01

Input

IOAREA

SCD
SCDLOCOU

Processing

[05] Write the contents of the I/O AREA to tape

A. If logging is being done in a batch environment, issue the PUT immediately

B. If logging is being done in an online environment, give control to the asynchronous log subtask (chart 1.4.1.11)

[06] If the log request is associated with a buffer, move the number of the last written log block from SCDLOCOU into the buffer prefix

[07] If more data is to be logged, go to step 1 C

[08] Exit

TO DLZRDBL0
CHART 1.4.1
STEP 4

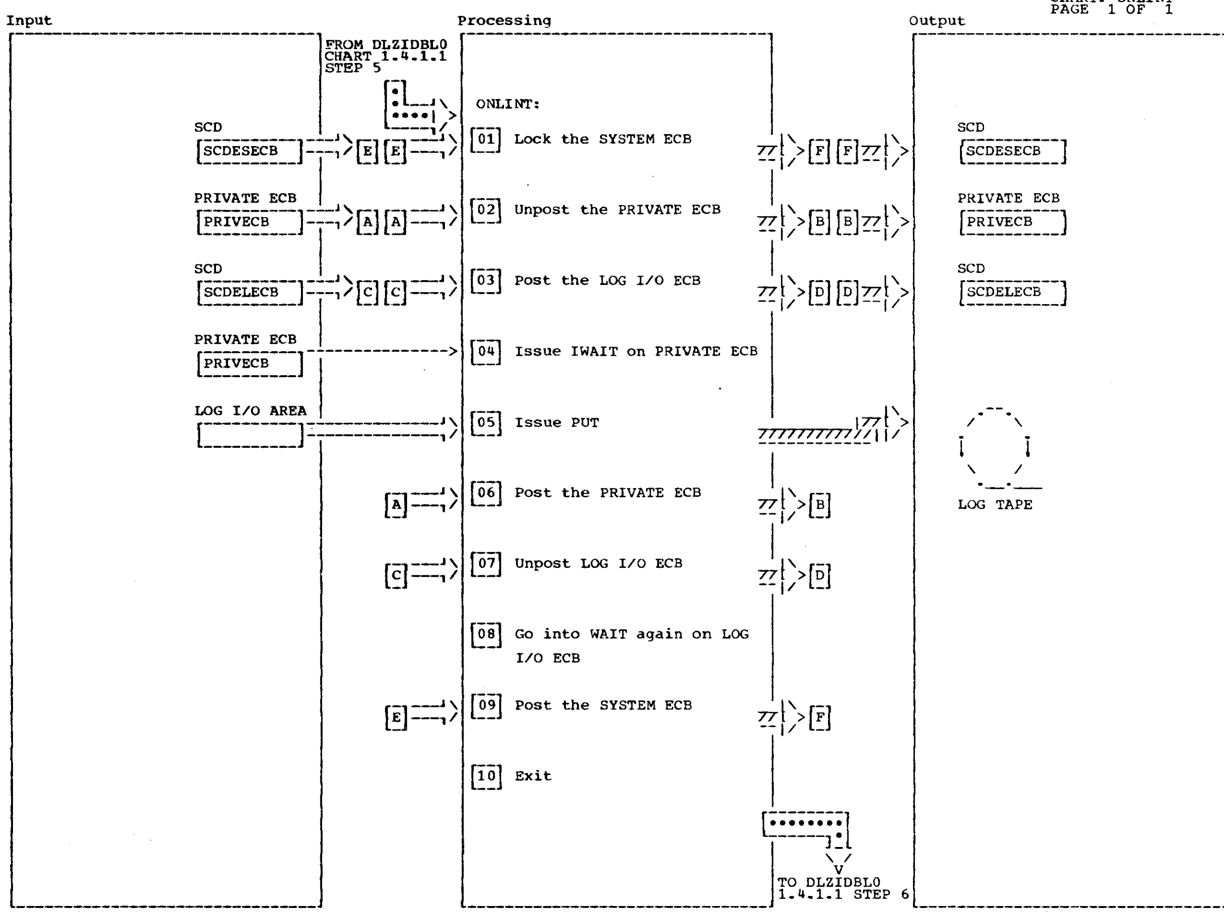
Output

LOG TAPE

BUFFER PREFIX
BFFRLOCO

DLZRDBL0 - LOG RECORD MOVING — HIPOMAT 1.1 Diagram - 1.4.1.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[05] B. The PUT is issued in an online environment from an asynchronously running subtask in order to avoid loosing tasks when EOV is encountered on the log tape. Refer to chart 1.4.1.11 for a description of this asynchronous log subtask.							
[06] The purpose for keeping the number of the last written log block in ths SCD and in the buffer prefix is to enable DLZDBH00 to determine, whether a log buffer has to be written out before an update is applied to a data base.							
[07] This happens, if the data to be logged doesn't fit into one log record. Refer to step 1 D.							



DLZRDBL0 - ISSUING PUT FROM THE ASYNCHRONOUS LOG SUBTASK HIPOMAT 1.1 Diagram - 1.4.1.11-01

Notes	Routine	Label	Ref
01 The SYSTEM ECB is used for communication between DLZRDBL0 and the DL/I ONLINE NUCLEUS. It is locked in order to prevent any other task from entering the logger while the I/O is going on.			
02 The PRIVATE ECB, which is defined in the asynchronous logwriter subtask (Csect ONLLOGWR), is used for communication between the asynchr. log subtask and DLZRDBL0 about the completion of the I/O.			
03 The LOG I/O ECB is used for communication between DLZRDBL0 and the asynchronous log subtask about the necessity to issue a PUT. The asynchronous log subtask is waiting on this ECB and when it gets posted, DOS will mark this subtask as dispatchable.			
04 This IWAIT will have the effect, that the DL/I 'maintask' will be put into wait. The asynchronous log write subtask can then be started by DOS.			
The steps 1,2,3,4,9 are performed within csect DLZRDBL0.	DLZRDBL0	ONLINT	
The steps 5,6,7,8 are performed within the asynchronous log writer subtask.	DLZRDBL0	ONLLOGWR	

DLZRDBL0 - ISSUING PUT FROM THE ASYNCHRONOUS LOG SUBTASK HIPOMAT 1.1 Diagram - 1.4.1.11-01

Input

LOG I/O AREA
CURSIZE

LOG RECORD
DLENGTH

LOG I/O AREA

LOG WORK AREA
LOG RECORD

Processing

FROM DLZRDBL0
CHART 1.4.1
STEP 2

LOGWR:

01 If the log record fits into the space currently available in the I/O AREA, go to step 3

02 Write the current contents of the LOG I/O AREA to the log tape

03 Move the log record to the LOG I/O AREA

04 Exit

TO DLZRDBL0
CHART 1.4.1
STEP 4

Output

LOG TAPE

LOG I/O AREA

DLZRDBL0 - LOG RECORD MOVING | HIPOMAT 1.1 Diagram - 1.4.1.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
This function is used for open log records ID X'2F', for scheduling records ID X'08' and for termination records ID X'07'.							

DLZRDBL0 - LOG RECORD MOVING | HIPOMAT 1.1 Diagram - 1.4.1.2-01

Input	Processing	Output
LOG I/O AREA	FROM DLZRDBLO CHART 1.4.1 STEP 3 WRIAHEAD: 01 Issue PUT to write current contents of the log I/O area to the log tape 02 Exit TO DLZRDBLO CHART 1.4.1 STEP 4	LOG TAPE

DLZRDBLO - FORCE WRITE ROUTINE

HIPOMAT 1.1 Diagram - 1.4.1.3-01

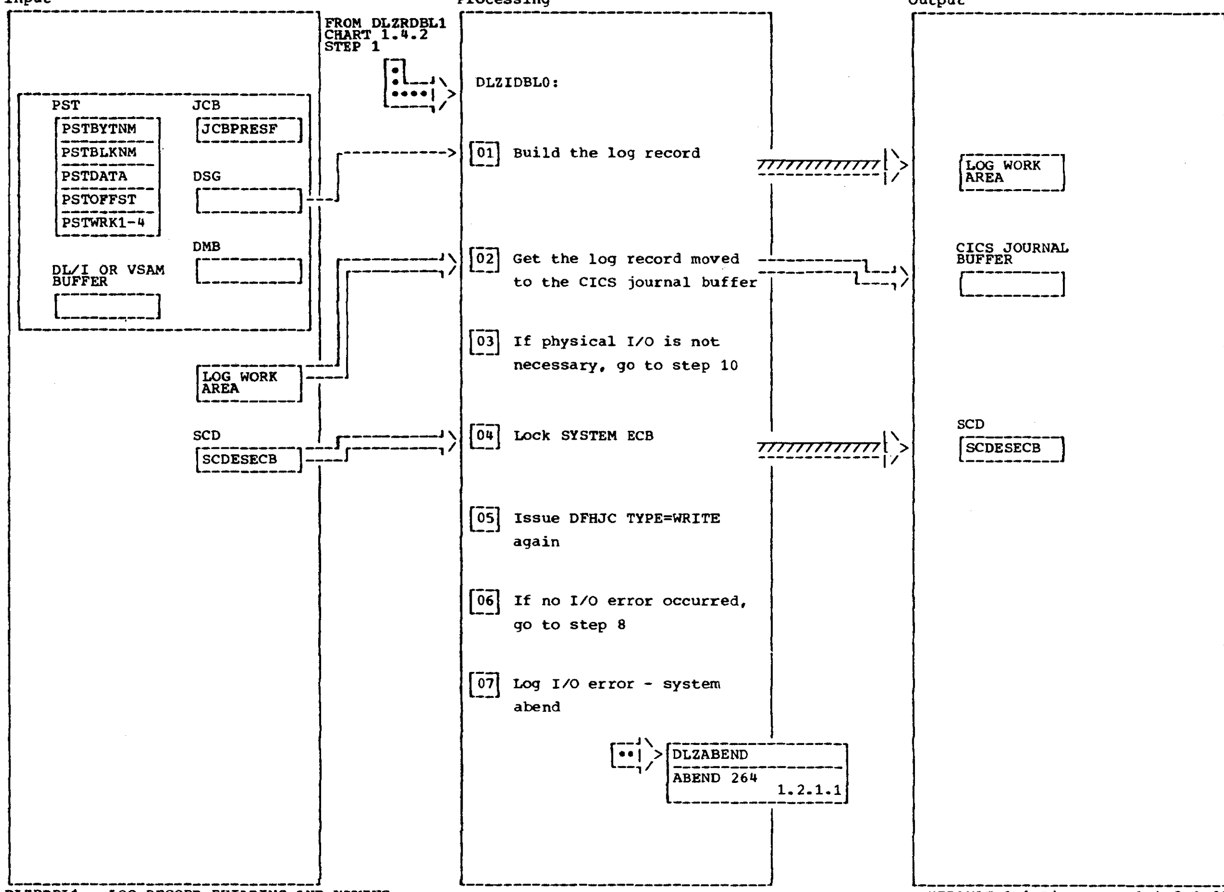
Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

Input	Processing	Output
	FROM CALLER DLZRDBL1: What function is requested? (note 6) 01 Build a log record and move it to the CICS journaling buffer (chart 1.4.2.1) 02 Build open log record(s) and move it (them) to the CICS journaling buffer (chart 1.4.2.2) 03 Move a log record, which has been built by another module, to the CICS journ. buffer (chart 1.4.2.3) 04 Write journal information physically to tape (chart 1.4.2.4) 05 Exit RETURN TO CALLER	

DLZRDBL1 - LOG MODULE USING CICS JOURNALING HIPOMAT 1.1 Diagram - 1.4.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01	DLZRDBL1	DLZIDBL0					
02 Since the CICS journal tape is not yet open at DL/I initialization time,the open log record(s) are built and moved before the first scheduling call is logged.	DLZRDBL1	OPLOG					
03 This applies to scheduling and termination log records built by the scheduling resp. termination routine.	DLZRDBL1	WRITEEXT					
04 This function is used by DLZDBH00, when log information associated with a data base update has not yet been written to tape at the time the data base update is being done .	DLZRDBL1	WRIAHEAD					
06 The four different functions of this module are associated with four different entry points into it.							

DLZRDBL1 - LOG MODULE USING CICS JOURNALING HIPOMAT 1.1 Diagram - 1.4.2-01



DLZRDBL1 - LOG RECORD BUILDING AND MOVING

HIPOMAT 1.1 Diagram - 1.4.2.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
02 A DFHJC TYPE=(WRITE,DL/I) is issued.							
04 The SYSTEM ECB is locked in order to prevent any other task from entering the logger while the I/O is going on.	DLZRDBL1	IONEC1					

DLZRDBL1 - LOG RECORD BUILDING AND MOVING

HIPOMAT 1.1 Diagram - 1.4.2.1-01

Input	Processing	Output
JCT JCTBLKNM	08 Get the new number of the last written journal block from the JCT	SCD SCDLOCOU
	09 Post the SYSTEM ECB	
	10 If the log request was not associated with a buffer, go to step 12	
SCD SCDLOCOU	11 Store the block number of the last journal buffer written to tape into the buffer prefix	BUFFER PREFIX
	12 If the data to be logged didn't fit into one log record, go to step 1	
	13 Exit	

TO DLZRDBL1
CHART 1.4.2
STEP 5

DLZRDBL1 - LOG RECORD BUILDING AND MOVING

HIPOMAT 1.1 Diagram - 1.4.2.1-02

Notes	Routine	Label	Ref
08 The purpose for keeping the CICS event control number is to enable DLZDBH00 to determine, whether a log buffer has to be written before an update is applied to a data base.	DLZRDBL1	GETECN	
11 The number is stored into BFFRLOCO.			

Input	Processing	Output
	FROM DLZRDBL1 CHART 1.4.2 STEP 2	
	OPLOG:	
DDIR	[01] Locate the first DDIR entry	
	[02] If the data base was not opened, go to step 5	
DMB: DMB, ACB EXT	[03] Build the open log record for the data base referenced by this DDIR entry	LOG WORK AREA
	[04] Move it to CICS journaling buffer	
	[05] If this is not the last DDIR entry, get the next DDIR entry and go to step 2	
	[06] Exit	
	TO DLZRDBL1 CHART 1.4.2 STEP 5	

DLZRDBL1 - BUILD AND MOVE OPEN LOG RECORDS

HIPOMAT 1.1 Diagram - 1.4.2.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[02] A data base might not have been opened because of the OPEN=DEFERRED option or because of an open error.							
[04] Refer to chart DLZIDBL0 1.4.2.1 step 2 to step 9							

DLZRDBL1 - BUILD AND MOVE OPEN LOG RECORDS

HIPOMAT 1.1 Diagram - 1.4.2.2-01

Input	Processing	Output
LOG WORK AREA	FROM DLZRDBL1 CHART 1.4.2 STEP 3 WRITEEXT: 01 Get the prebuilt log record moved to the CICS journal buffer 02 Exit TO DLZRDBL1 CHART 1.4.2 STEP 5	CICS JOURNAL BUFFER

DLZRDBL1 - MOVING PREBUILT LOG RECORDS

HIPOMAT 1.1 Diagram - 1.4.2.3-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Refer to DLZRDBL1 chart 1.4.2 step 2 - step 9.							

Input

FROM DLZRDBL1 CHART 1.4.2 STEP 4

SCD: SCDLOCOU

SCD: SCDESECB

CICS JOURNAL BUFFER

JCT: JCTBLKNM

SCD: SCDESECB

Processing

WRIAHEAD:

[01] Store the block number of the block that is going to be written in the JCA

[02] Lock the SYSTEM ECB

[03] Issue DFHJC TYPE=(WAIT,DL/I) to get current contents of CICS journal buffer written to tape

[04] Get the block number of the last written journal block from the JCT

[05] Post the SYSTEM ECB

[06] Exit

TO DLZRDBL1 CHART 1.4.2 STEP 5

Output

JCA: JCAECN

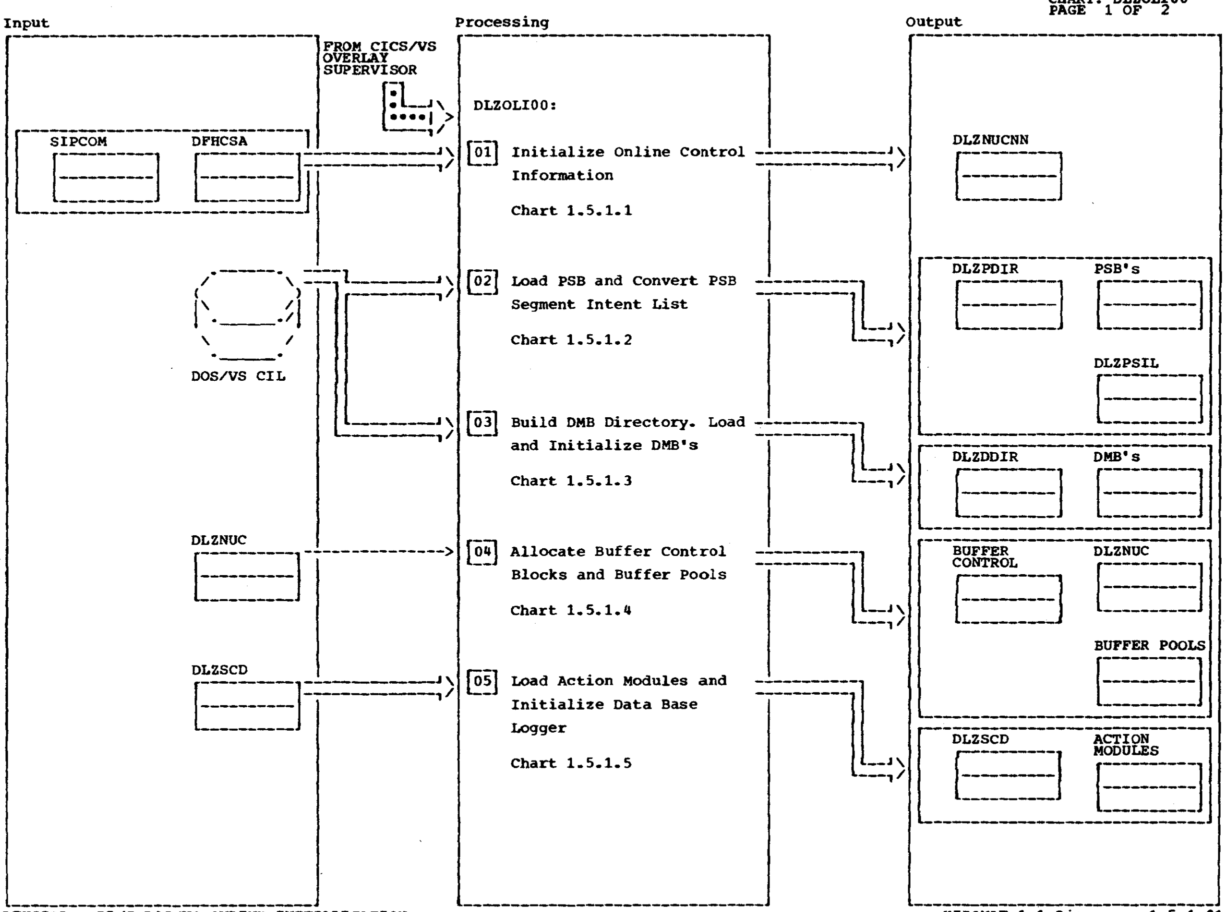
SCD: SCDESECB

LOG TAPE

SCD: SCDLOCOU

SCD: SCDESECB

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[01] Refer to note on step 8 of chart DLZIDBL0 1.4.2.1 .							
[02] Refer to the note on step 4 in chart DLZIDBL0 1.4.2.1 .							



DFHSIDL - DL/I DOS/VS ONLINE INITIALIZATION

HIPOMAT 1.1 Diagram - 1.5.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

DFHSIDL - DL/I DOS/VS ONLINE INITIALIZATION

HIPOMAT 1.1 Diagram - 1.5.1-01

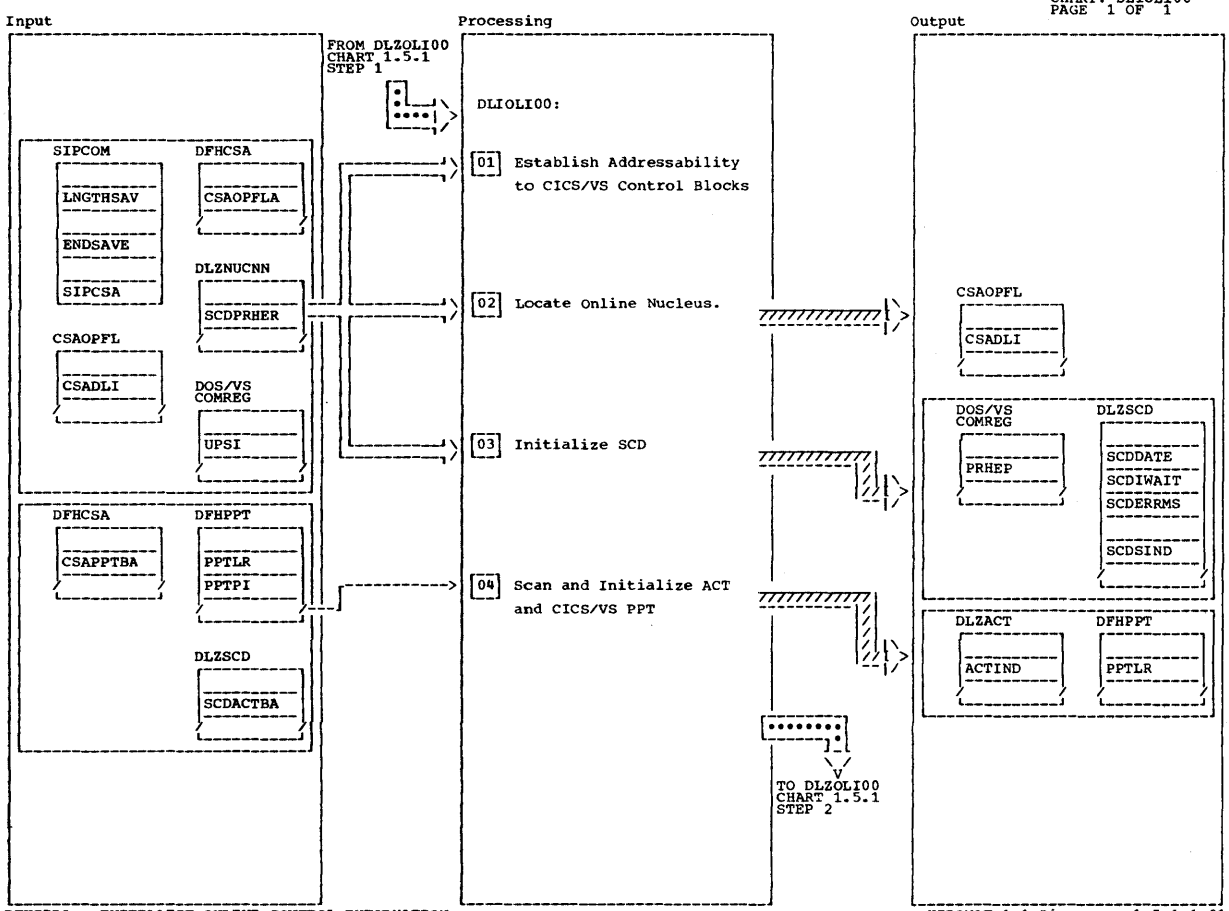
Input	Processing	Output
DLZPDIR, DLZSCD	06 Move and Relocate PSB'S Chart 1.5.1.6	DLZPDIR, PSB's
DLZDDIR	07 Open Data Bases Chart 1.5.1.7	DLZDDIR, DATA BASE LOG
	08 Return to CICS/VS SIP Overlay Supervisor	
	RETURN TO CICS/VS SIP	

DFHSIDL - DL/I DOS/VS ONLINE INITIALIZATION

HIPOMAT 1.1 Diagram - 1.5.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

CHART: DLIOLI00
PAGE 1 OF 1



DFHSIDL - INITIALIZE ONLINE CONTROL INFORMATION

HIPOMAT 1.1 Diagram - 1.5.1.1-01

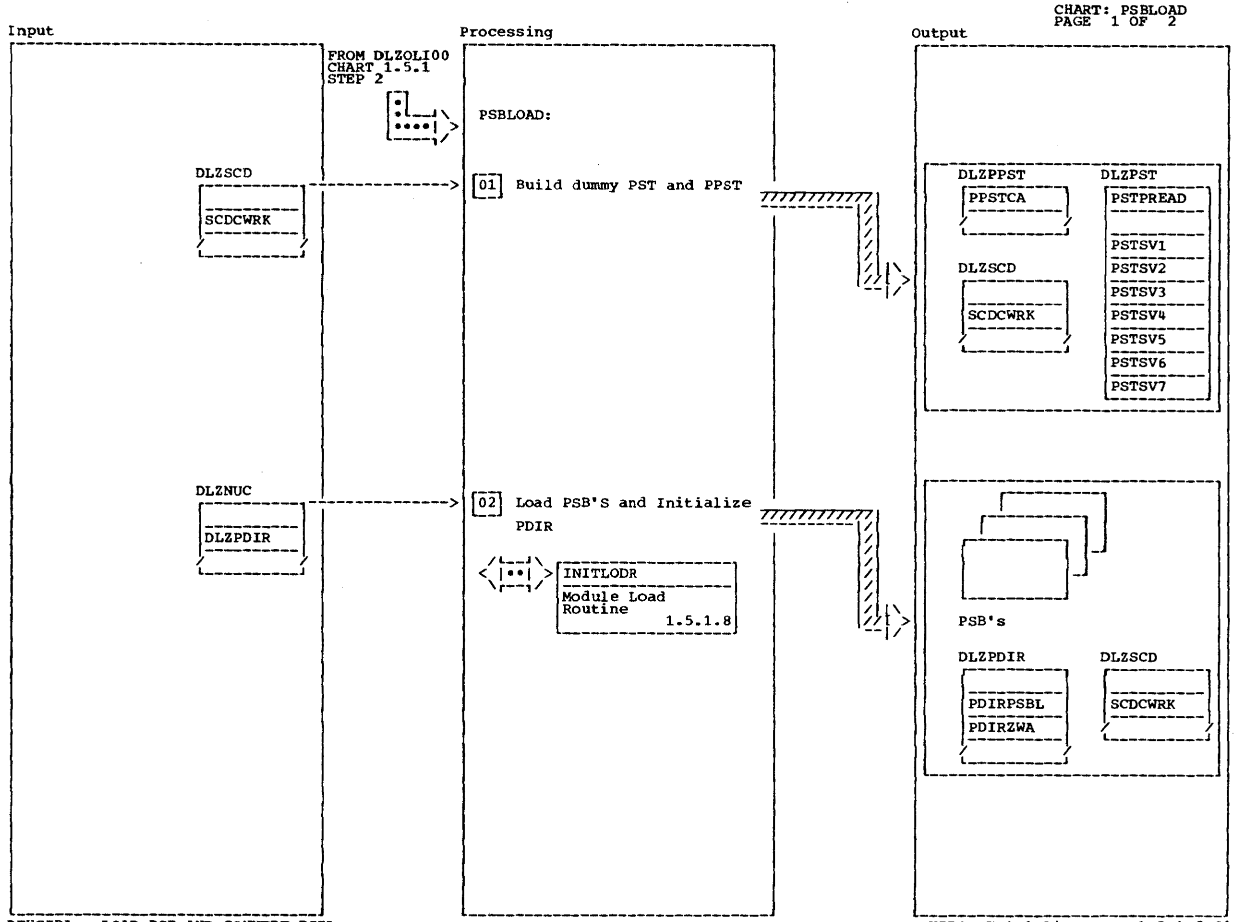
Notes	Routine	Label	Ref
01 Upon entry from the CICS/VS Overlay Supervisor, SIPBAR2 contains the overlay entry point, SIPBAR1 contains SIP Common Communications Area. The current storage allocation information is saved in order to release storage if DL/I initialization fails.		DLIOLI00	
02 The DL/I System Contents Directory is located from the CSA Optional Features List (CSAOPFL) field CSADLI. CSADLI is modified to point to the table of entry points for the Task & System Scheduling and Termination routines (DFHDLIAL).			
03 SCDSIND is initialized with bits 6&7 of the UPSI switch from the COMREG. The Program Request Handler entry point is moved to byte 16 of the Comreg and temporary entry points are established for the Error Message Routine and the DL/I Wait Routine.		NUCFOUND	

Notes	Routine	Label	Ref
04 Indicators are set in the CICS/VS PPT marking the program eligible for DL/I services. They are set in the DL/I ACT entry indicating the program was located in the PPT.		ACTCKLUP	

DFHSIDL - INITIALIZE ONLINE CONTROL INFORMATION

HIPOMAT 1.1 Diagram - 1.5.1.1-01

CHART: PSBLOAD
PAGE 1 OF 2

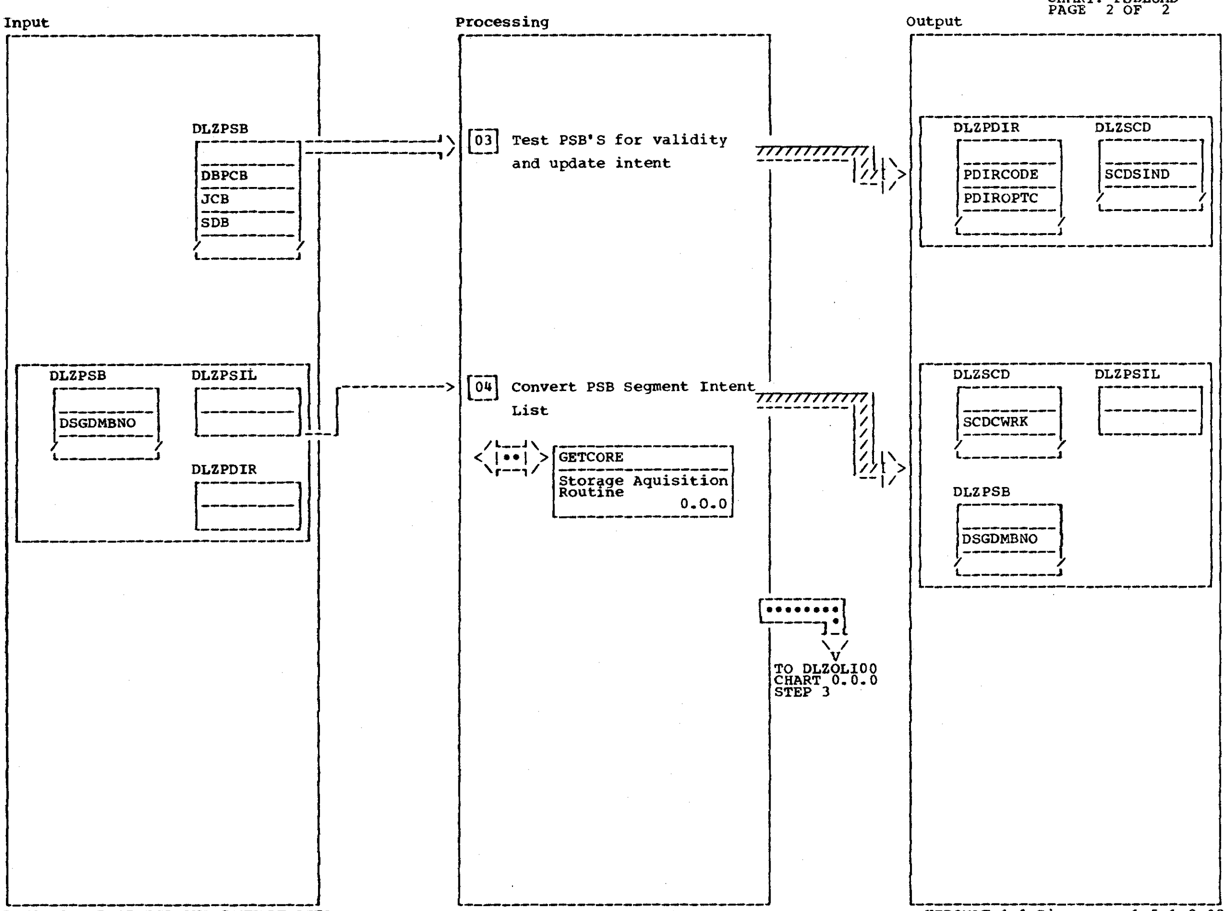


DFHSIDL - LOAD PSB AND CONVERT PSIL

HIPOMAT 1.1 Diagram - 1.5.1.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 The PST and PPST are built directly after the initialization overlay high storage address. The save areas are chained and SCDCWRK is updated to indicate the new upward core allocation starting address.		PSBLOAD					
02 The PDIR address is located in the SCD and each PSB is loaded temporarily, directly behind the dummy PST. If PSB initialization is successful, it will be moved up prior to completion of initialization.		PSBILUP					

CHART: PSBLOAD
PAGE 2 OF 2



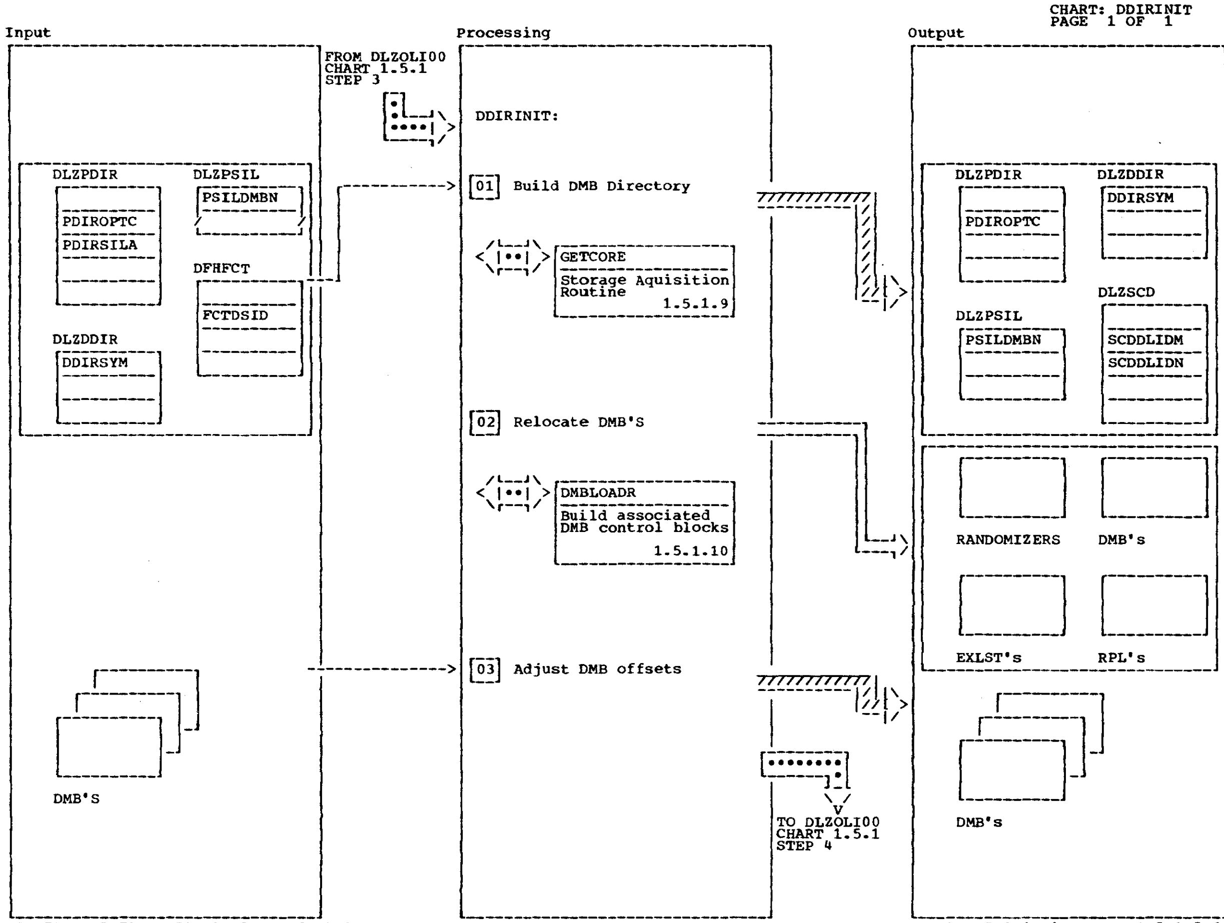
DFHSIDL - LOAD PSB AND CONVERT PSIL

HIPOMAT 1.1 Diagram - 1.5.1.2-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
03 Each PSB is tested for correct processing options. Indicators are switched in the coressponding PDIR entry to indicate validity.							
04 For each valid PSB the Segment Intent List is removed, translated to indicate read only conflict areas, and moved to the next available area via CICS GETMAIN.							

DFHSIDL - LOAD PSB AND CONVERT PSIL

HIPOMAT 1.1 Diagram - 1.5.1.2-02



DFHSIDL - BUILD DDIR, LOAD & INITIALIZE DMBS

HIPOMAT 1.1 Diagram - 1.5.1.3-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 The PSIL's are scanned for DMB names and an entry in the DDIR is created for each unique DMB encountered. The address of the DDIR replaces the respective DMBNAME in each PSIL.		DDIRINIT					

DFHSIDL - BUILD DDIR, LOAD & INITIALIZEDMBS

HIPOMAT 1.1 Diagram - 1.5.1.3-01

Input

DLZPST: PSTWK3
DLZSCD: SCDDBFPL

DLZSCD: SCDDBFPL
DLZPST: PSTWRK3
DLZBFPL: SUBINFTA
DLZDMB: DMBRBASN

Processing

FROM DLZOLI00
CHART 1.5.1
STEP 4

DLZCPI00:

01 If no buffers are required, go to step 6

02 Determine number of subpools required

03 Aquire buffer pool prefix and format

GETCORE
Storage aquisition routine
1.5.1.9

04 Allocate subpool sizes and subpool information table according to user specifications. For remaining subpools ,

A. Arrange subpools into ascending control interval sizes

B. Assign DMB's by corresponding control interval sizes

Output

DLZSCD: SCDBFPL

DLZBFPL: SUBINFTA

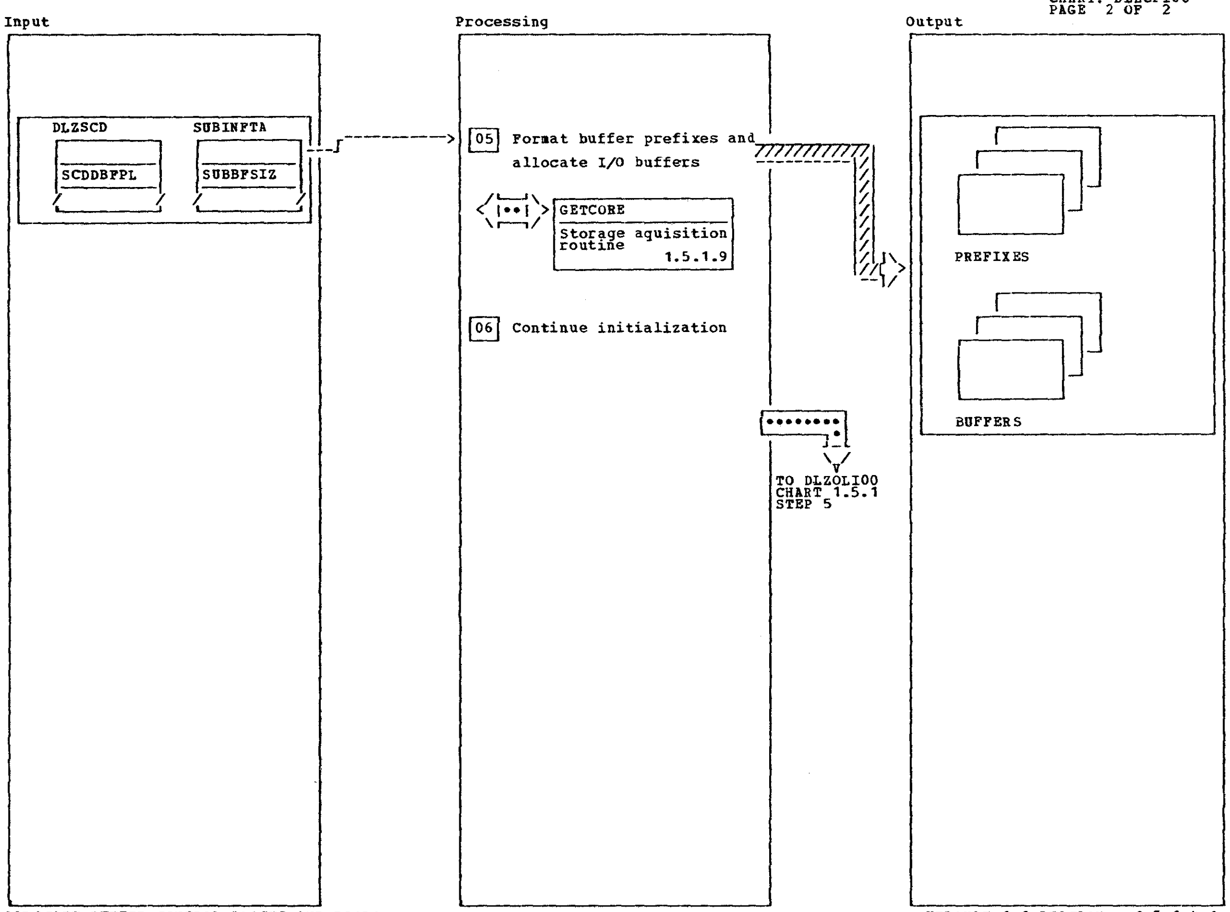
SUBINFTA: SBBFSIZ, SUBDMBCT

ALLOCATE BUFFER CONTROL BLOCKS AND POOLS — HIPOMAT 1.1 DIAGRAM - 1.5.1.4-0

Notes	Routine	Label	Ref
01 Zero buffer situation may occur if simple HISAM is the only access method specified.		SLCBP	
02 Buffer allocation is done by a subroutine. The required number is set to the user specified amount if the user number is smaller than required.		BUFALLOC	
03		BFPREADY	
04 At this point the size of the subpools are determined. They are allocated, largest first, until the specified number is exausted. Remaining DMB's requiring subpools are assigned evenly across all existing subpools. If the user specified more subpools than neccessary an additional pool, of 512 buffer size, is allocated for delete workspace.		SUBPALUP	
A. The subpool sizes are sorted so that the largest subpool appears first in the		SUBTSHFL	

Notes	Routine	Label	Ref
information table.			
B. Each DMB is assigned by placing its DDIR position pointer into the subpool table.		DMBSUBLP	

ALLOCATE BUFFER CONTROL BLOCKS AND POOLS — HIPOMAT 1.1 DIAGRAM - 1.5.1.4-0



ALLOCATE BUFFER CONTROL BLOCKS AND POOLS

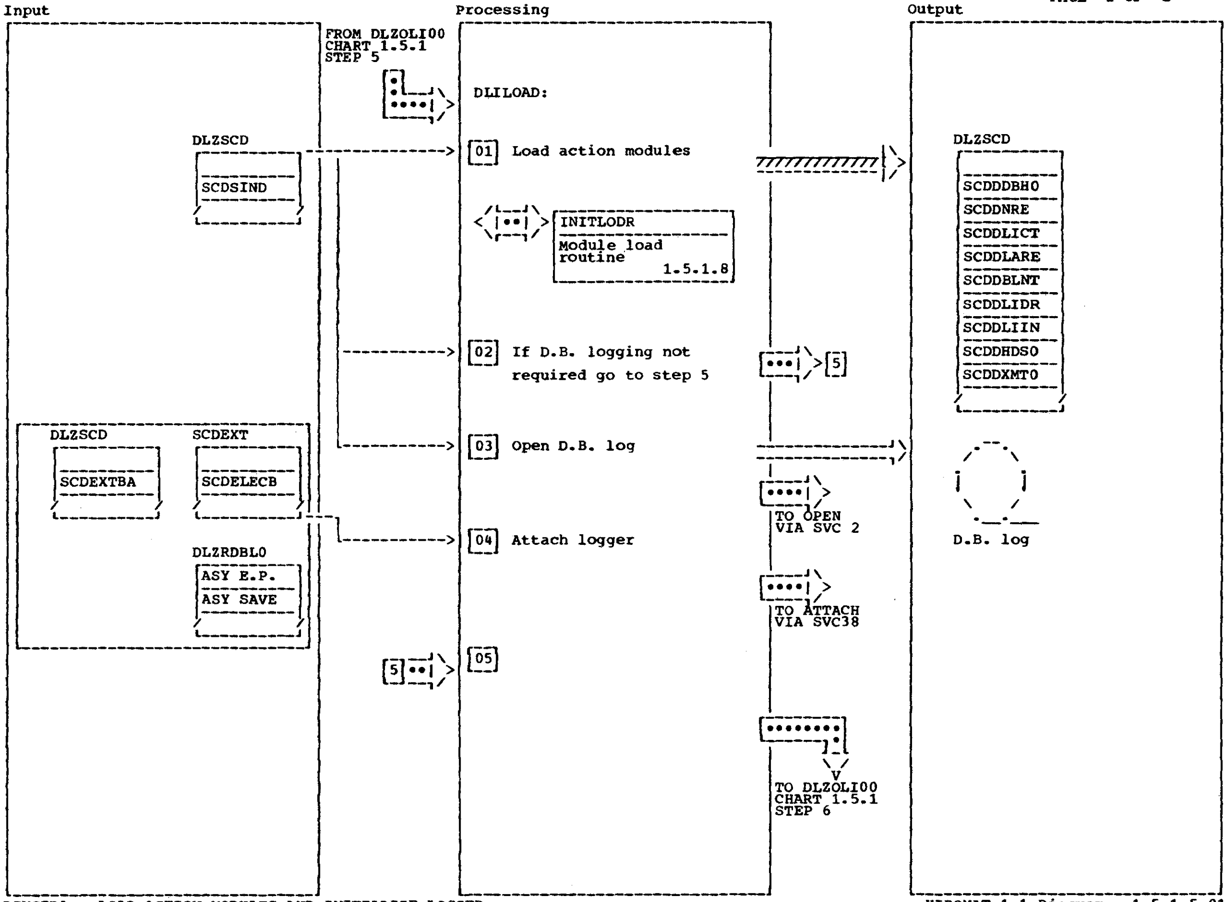
HIPOMAT 1.1 DIAGRAM - 1.5.1.4-0

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
05 The user-specified number of buffers is allocated per pool. default is 32.		BFRINIT					

ALLOCATE BUFFER CONTROL BLOCKS AND POOLS

HIPOMAT 1.1 DIAGRAM - 1.5.1.4-0

CHART: DLILOAD
PAGE 1 OF 1



DFHSIDL - LOAD ACTION MODULES AND INITIALIZE LOGGER

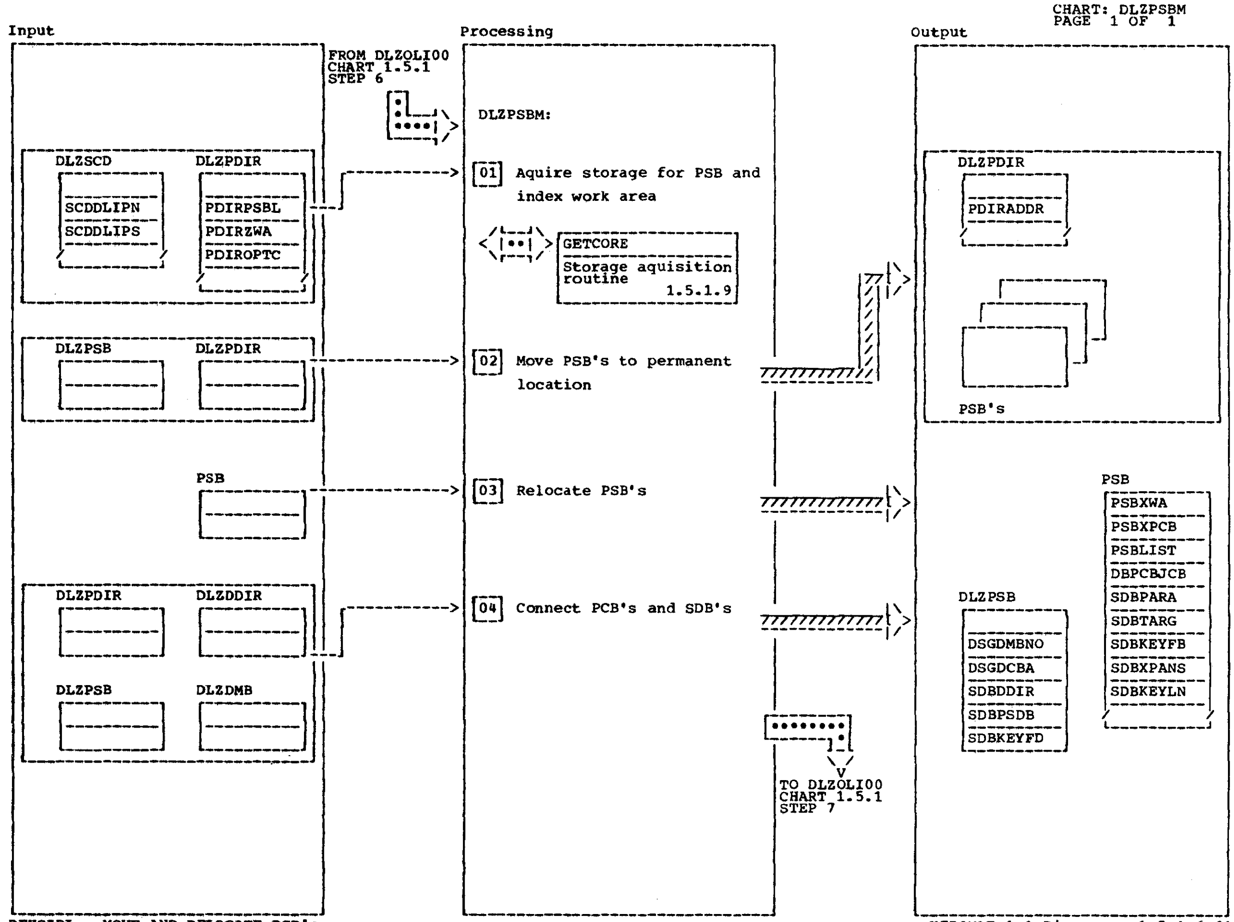
HIPOMAT 1.1 Diagram - 1.5.1.5-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 The nine action modules are loaded and their entry points moved to the SCD. If d.b. logging is not required, SCDDBLNT contains a pointer to a branch register 14.		DLILOAD					
03 The data base log is supported on magnetic tape assigned to DOS/VS logical unit SYS011.							
04 The address list for the asynchronous portion of the database logger and its save area address are located in the database log load module just prior to the entry point. If the attach fails the d.b. log is closed and the system continues without log support.							

DFHSIDL - LOAD ACTION MODULES AND INITIALIZE LOGGER

HIPOMAT 1.1 Diagram - 1.5.1.5-01

CHART: DLZPSBM
PAGE 1 OF 1



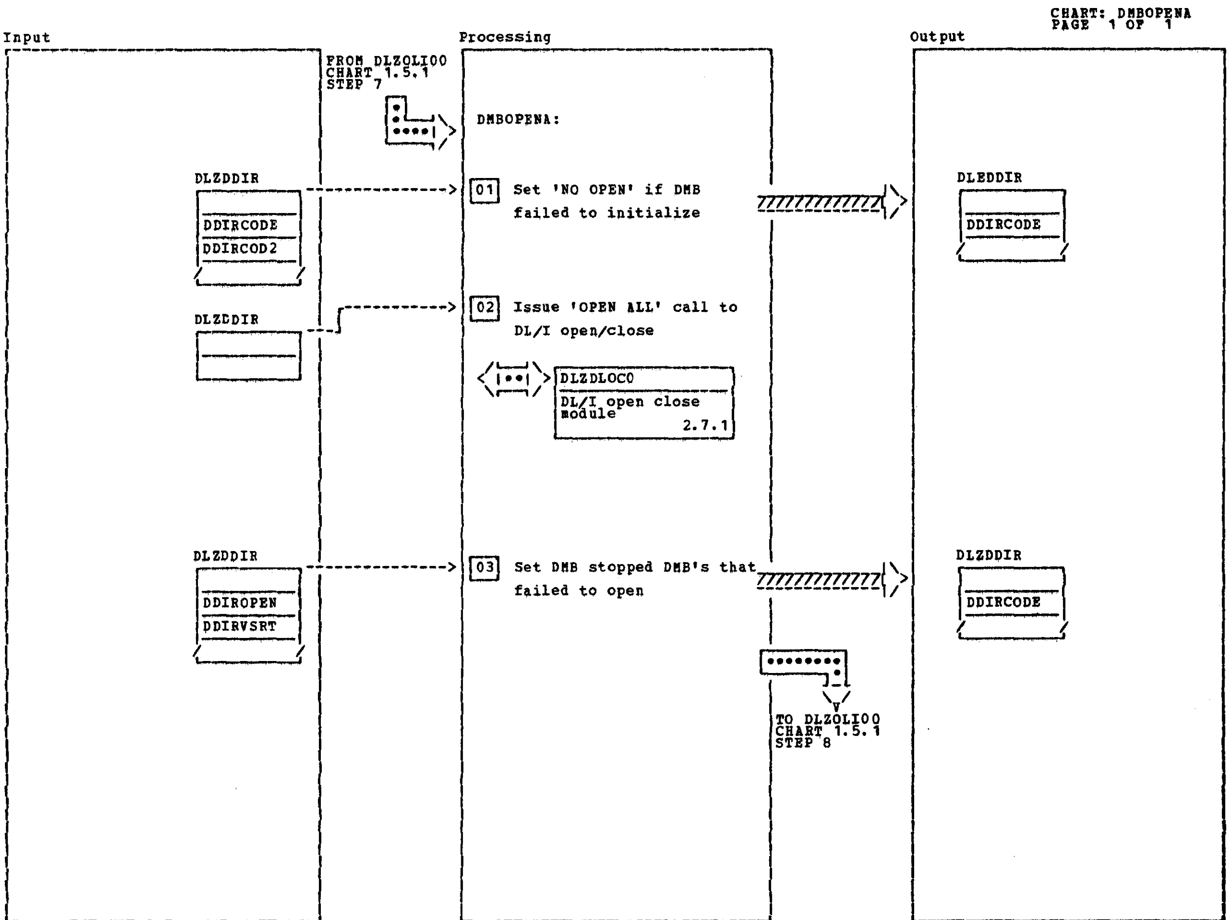
DFHSIDL - MOVE AND RELOCATE PSB's

HIPOMAT 1.1 Diagram - 1.5.1.6-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

DFHSIDL - MOVE AND RELOCATE PSB's

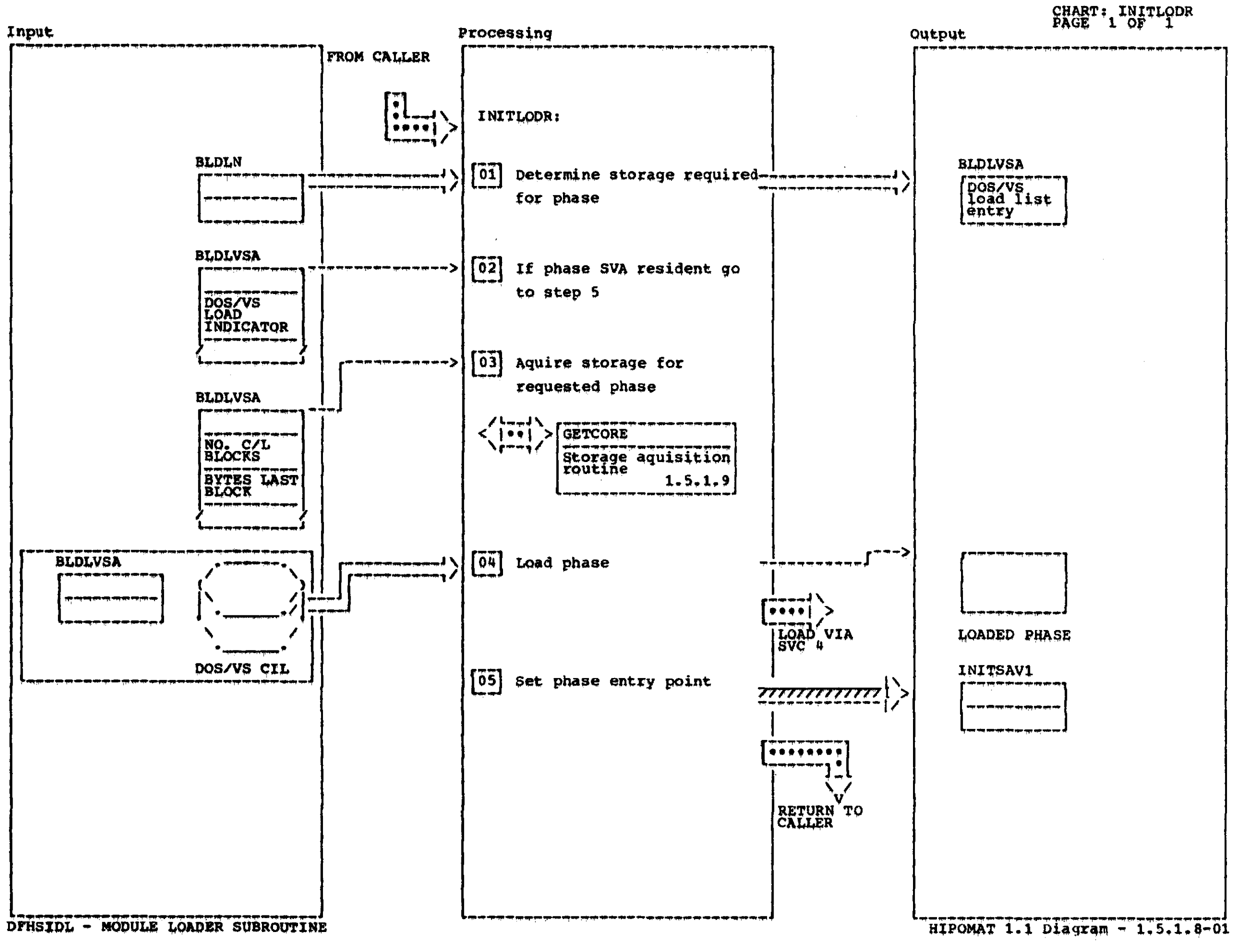
HIPOMAT 1.1 Diagram - 1.5.1.6-01



DFHSIDL - OPEN DATA BASES

HIPOMAT 1.1 Diagram - 1.5.1.7-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

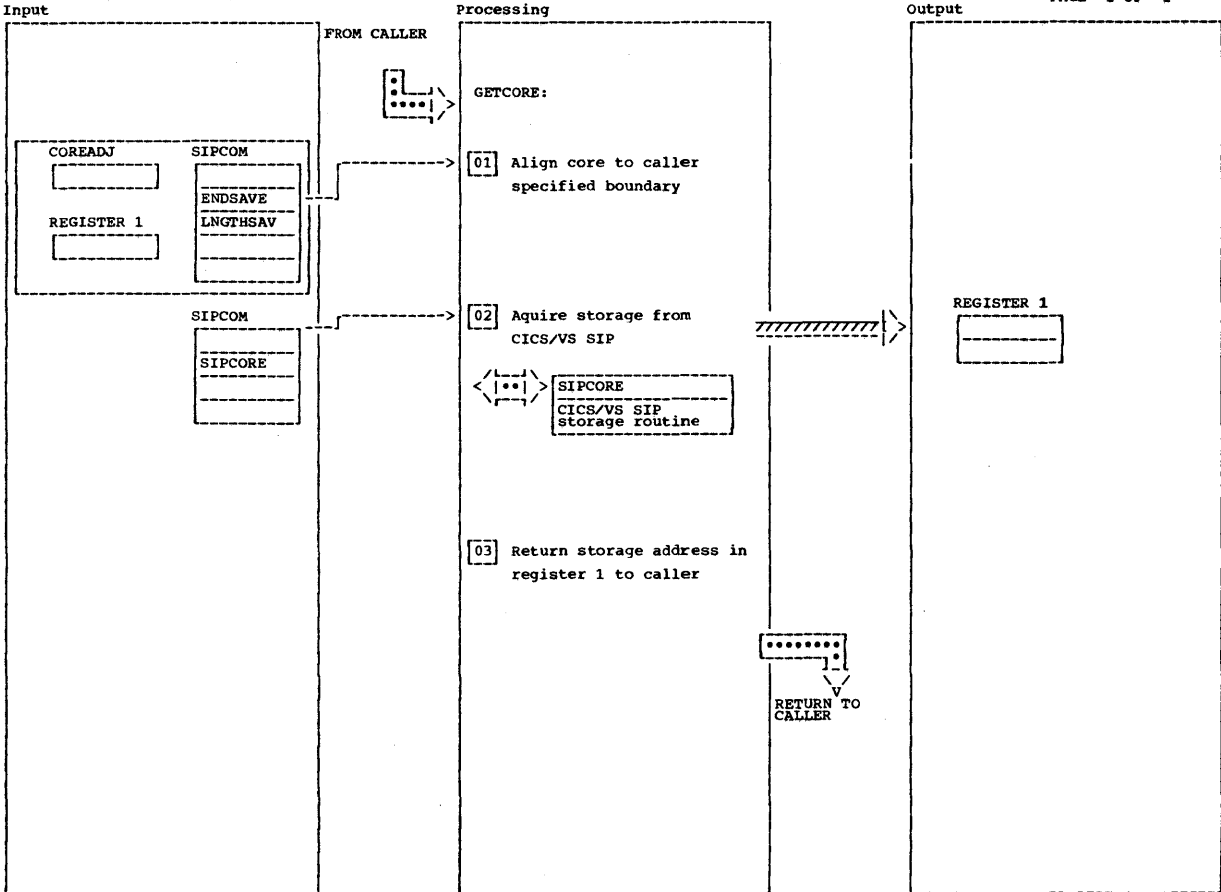


Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Caller passes requested phase name in a work field BLDLN. The output of the load call is a DOS/VS directory entry at BLDLVSA.		INITLODR					
03 Amount of storage is determined by number of library block * 1024 plus number of bytes in last block.							

DFHSIDL - MODULE LOADER SUBROUTINE

HIPOMAT 1.1 Diagram - 1.5.1.8-01

CHART: GETCORE
PAGE 1 OF 1



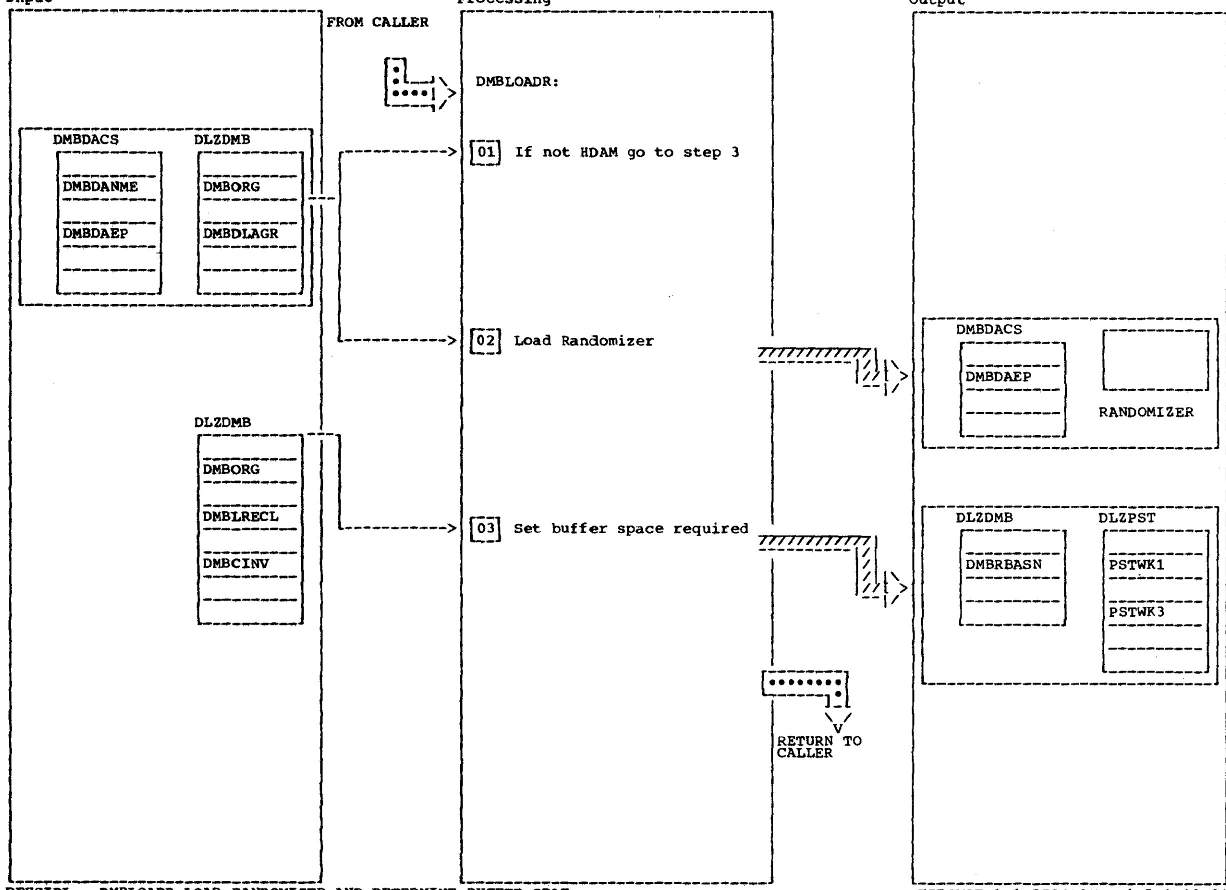
DFHSIDL - STORAGE AQUISITION ROUTINE

HIPOMAT 1.1 Diagram - 1.5.1.9-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 This routine aquires storage from CICS/VS SIP 'SIPCORE'.							

DFHSIDL - STORAGE AQUISITION ROUTINE

HIPOMAT 1.1 Diagram - 1.5.1.9-01



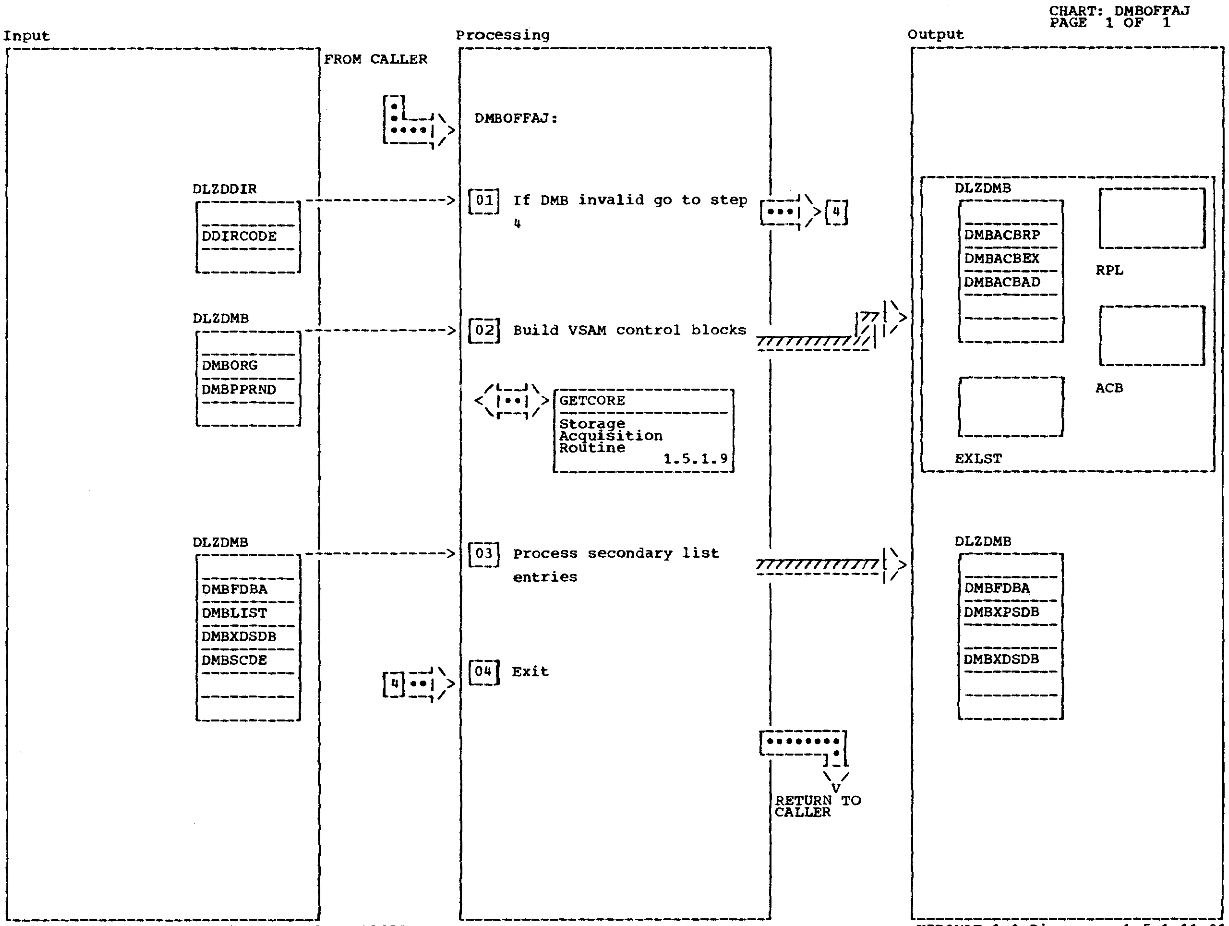
DFHSIDL - DMBLOADR LOAD RANDOMIZER AND DETERMINE BUFFER SPAE

HIPOMAT 1.1 DIAGRAM - 1.5.1.10.01

Notes	Routine	Label	Ref
02 Before loading the randomizer a check is made with all currently loaded randomizers. If one of the same name as that we are loading, the entry point is resolved and the actual load is bypassed.		RANCKLUP	
03 If buffer pool space is required, the size of each Control Interval, rounded to the next multiple of 512, is stored in PSTWK1 for later allocation of the buffer pool.		GETBUFRS	

DFHSIDL - DMBLOADR LOAD RANDOMIZER AND DETERMINE BUFFER SPAE

HIPOMAT 1.1 DIAGRAM - 1.5.1.10-0



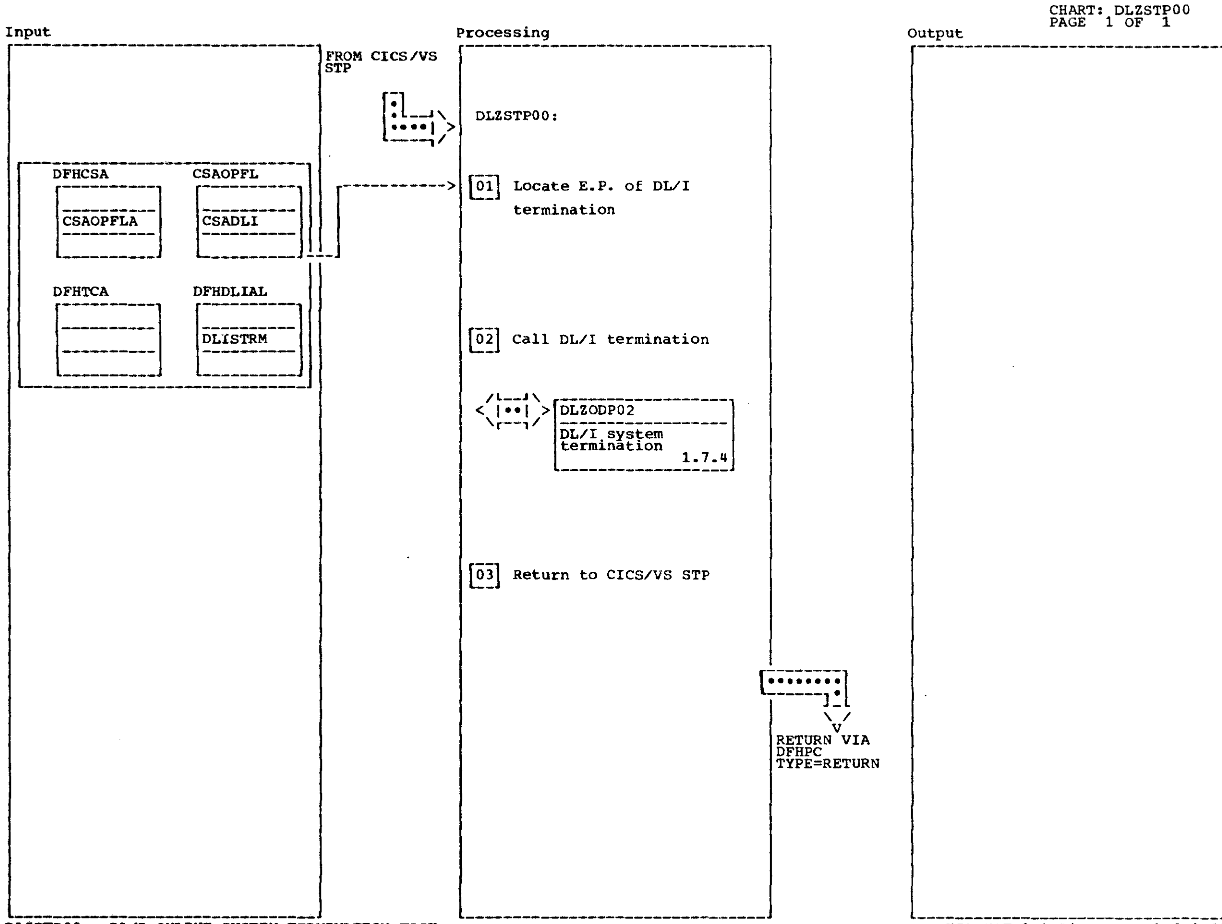
DFHSIDL - DMB RELOCATE AND VSAM BLOCK BUILD

HIPOMAT 1.1 Diagram - 1.5.1.11-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
02 If HISAM two sets of control blocks wil be built.		ACBADLUP					
03 The pointer to the FDB is relocated. If a secondary list is present, it's code is tested, and referenced DMBs are resolved to DDIR pointers and placed in the list.		PSDBROUT					

)FHSIDL - DMB RELOCATE AND VSAM BLOCK BUILD

HIPOMAT 1.1 Diagram - 1.5.1.11-01



DLZSTP00 - DL/I ONLINE SYSTEM TERMINATION TASK

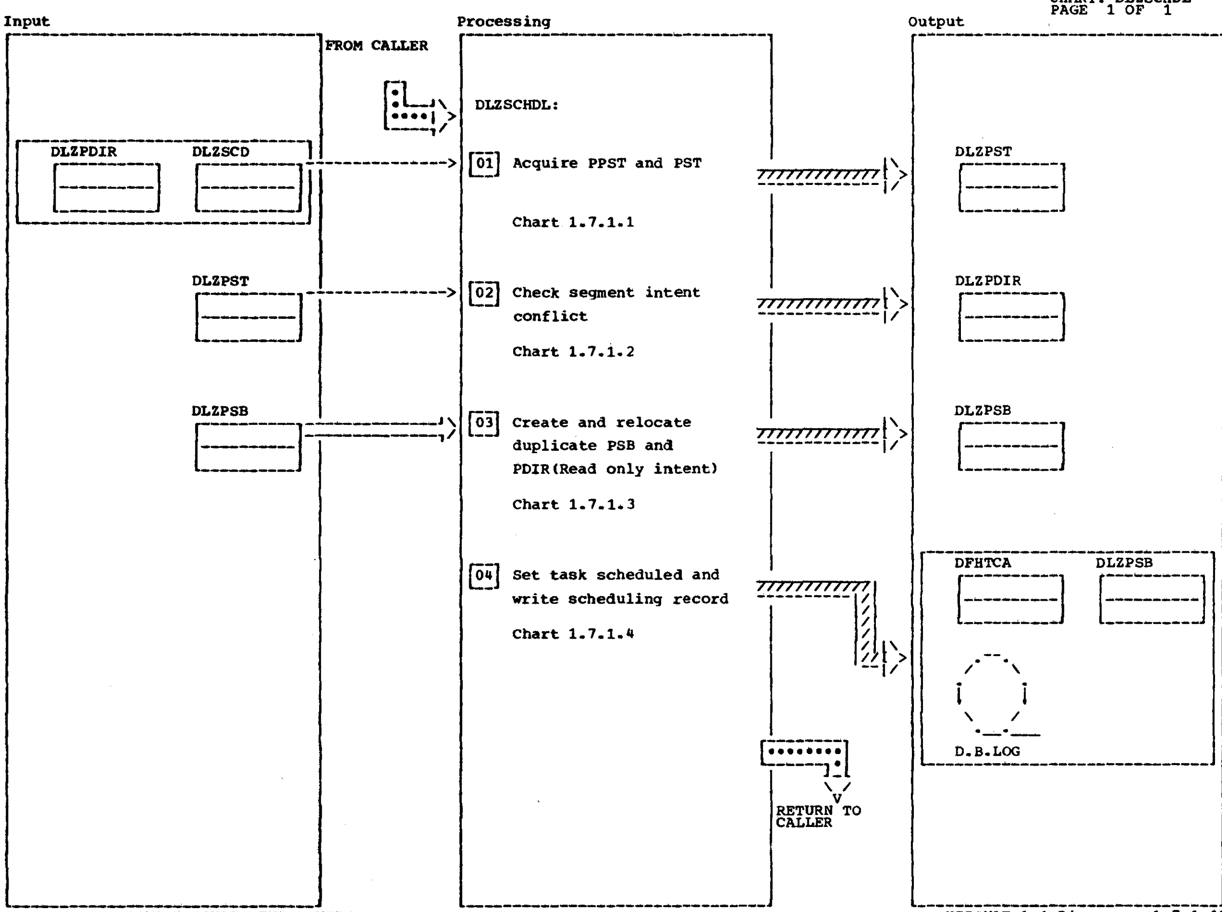
HIPOMAT 1.1 Diagram - 1.6.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Control is gained from CICS/VS stp via programs presence in DFHPLT.							

DLZSTP00 - DL/I ONLINE SYSTEM TERMINATION TASK

HIPOMAT 1.1 Diagram - 1.6.1-01

CHART: DLZSCHDL
PAGE 1 OF 1

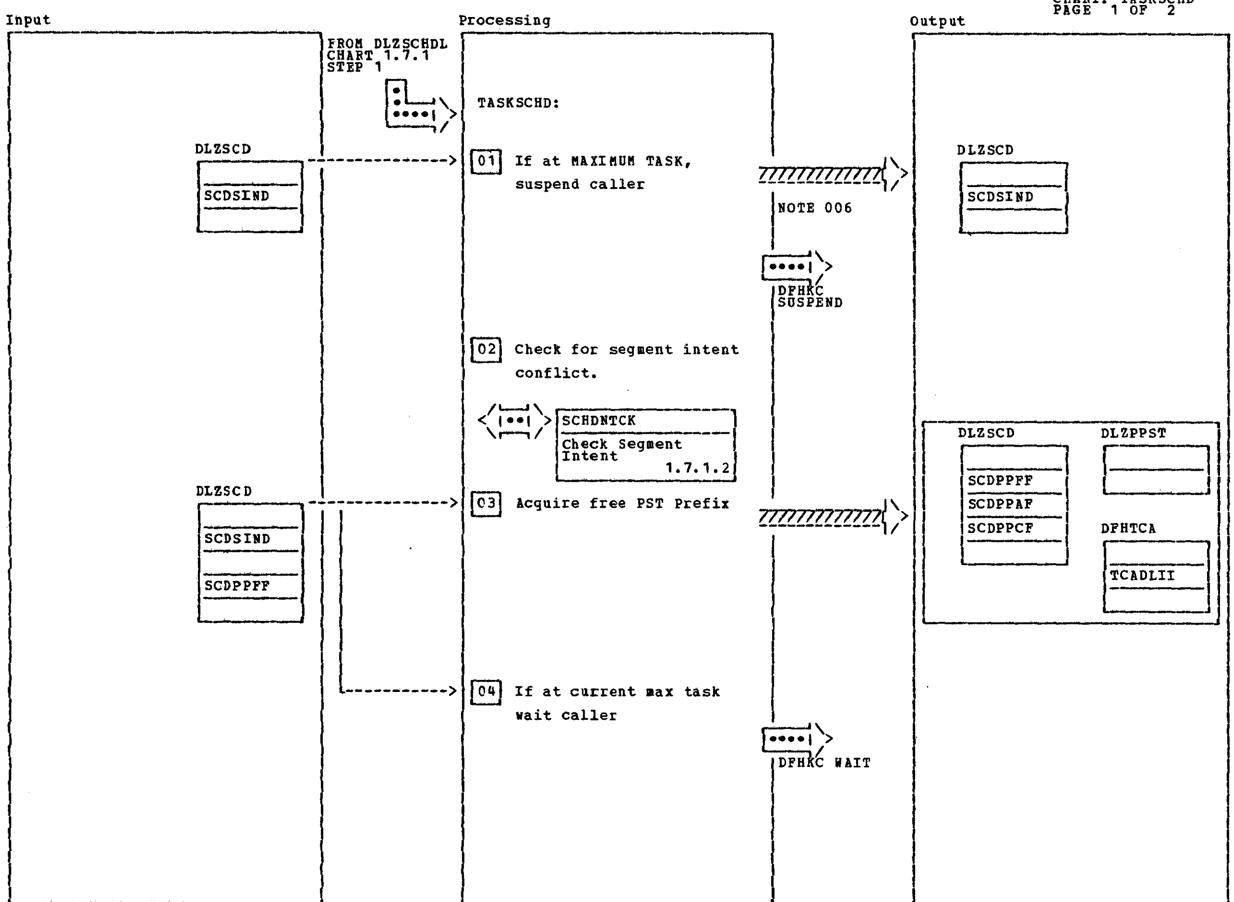


DLZNUCxx - RESOURCE SCHEDULING ROUTINE

HIPOMAT 1.1 Diagram - 1.7.1-01

Notes	Routine	Label	Ref
01 Task wait or suspend may be issued.		DLZSCHDL	
02 Task wait may be issued.		SCHDTCK	
03 Suspend due to storage request may occur.		TASKDUPP	

Notes	Routine	Label	Ref



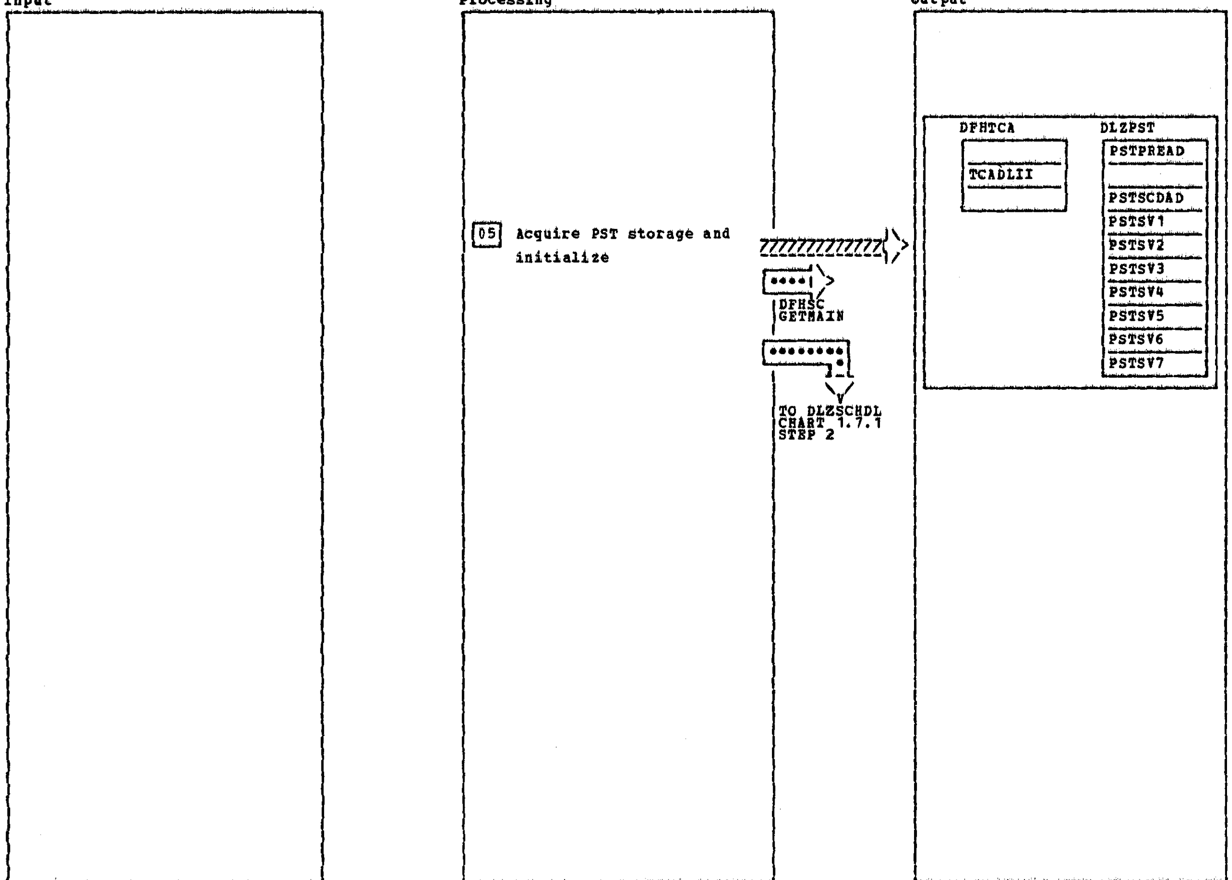
DLZNUCxx - TASKSCHD ACQUIRE PPST AND PST

HIPOMAT 1.1 Diagram - 1.7.1.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Maximum task is a condition where all PST Prefixes are in use.		TASKSCHD					
03 A prefix is acquired from the free chain and placed on the active chain.							

DLZNUCxx - TASKSCHD ACQUIRE PPST AND PST

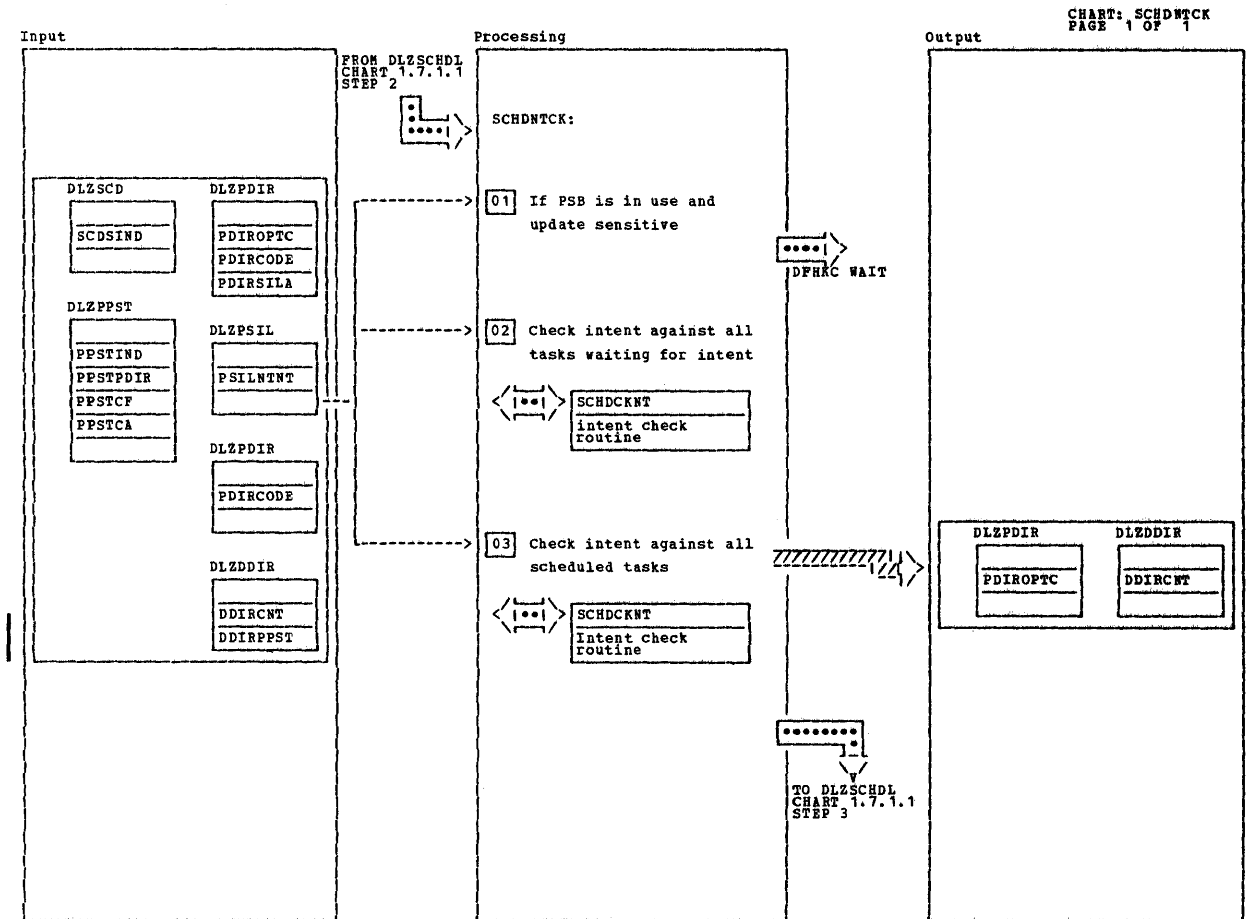
HIPOMAT 1.1 Diagram - 1.7.1.1-0



DLZNUCxx - TASKSCHD ACQUIRE PPST AND PST

HIPOMAT 1.1 Diagram - 1.7.1.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
06 If the last PPST is used, the maxtask indicator will be turned on.							



DLZNUCxx - CHECK SEGMENT INTENT

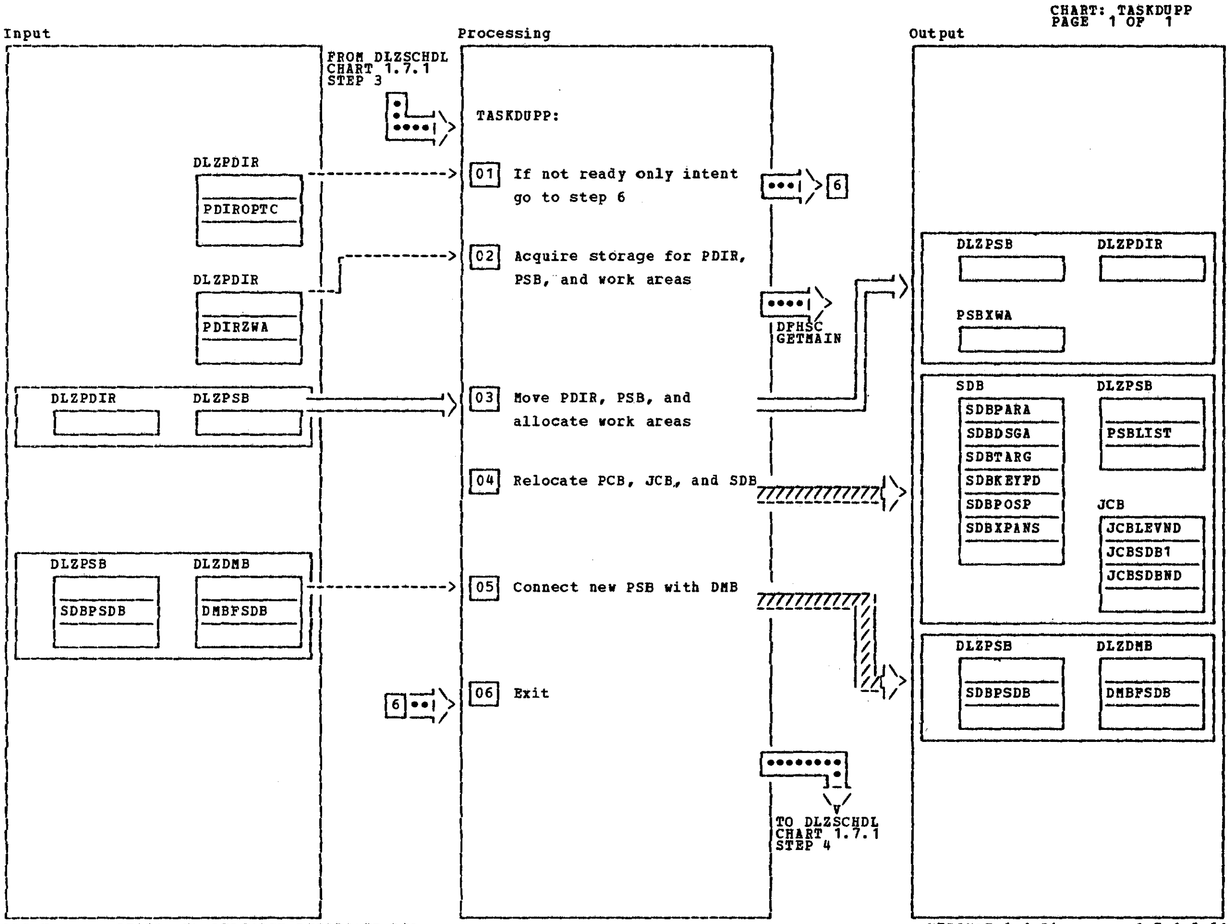
HIPOMAT 1.1 Diagram - 1.7.1.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 If this condition exists and the holder is not an MPS scheduled task, a wait is issued without going through intent checking.		SCHDNTCK					
02		SCHDFCLP					
03 If a conflict exists and the holder is not an MPS scheduled task, a wait is issued. If the holder is an MPS scheduled task, a data base-not-open return code is returned to the user.		SCHDCKNT					

DLZNUCxx - CHECK SEGMENT INTENT

HIPOMAT 1.1 Diagram - 1.7.1.2-0'

CHART: TASKDUPP
PAGE 1 OF 1



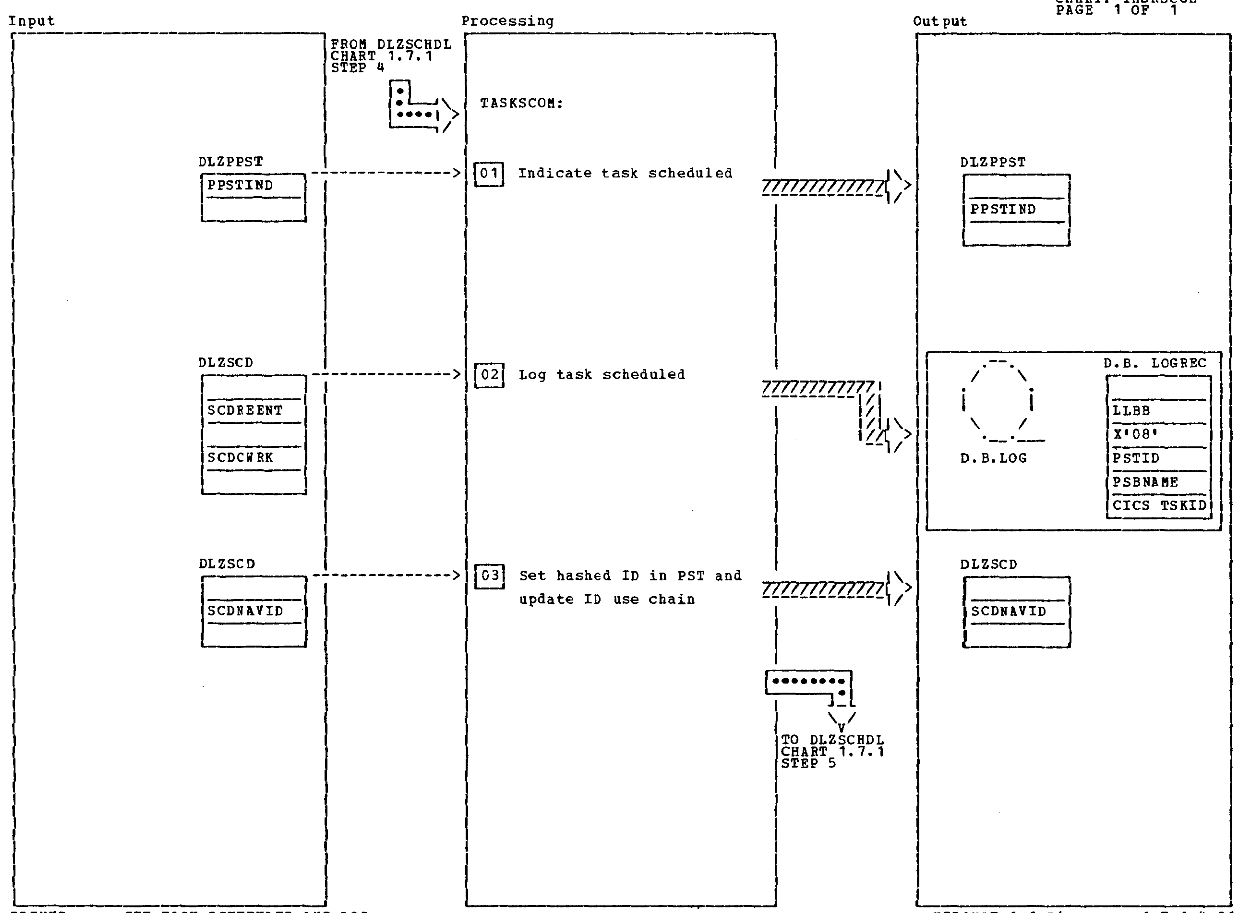
DLZNUCxx - CREATE AND RELOCATE DUPLICATE PSBs HIPOMAT 1.1 Diagram - 1.7.1.3-01

Notes	Routine	Label	Ref
01		TASKDUPP	
02 Task suspend possible due to unconditional CICS/VS GETMAIN.			
03		SCHDDPL	
05 Each PSB is scanned and the SDBs are placed in the corresponding SDB chain which starts in the DMB at DMBFSDB.			

Notes	Routine	Label	Ref

DLZNUCxx - CREATE AND RELOCATE DUPLICATE PSBs HIPOMAT 1.1 Diagram - 1.7.1.3-01

CHART: TASKSCOM
PAGE 1 OF 1



DLZNUCxx -- SET TASK SCHEDULED AND LOG

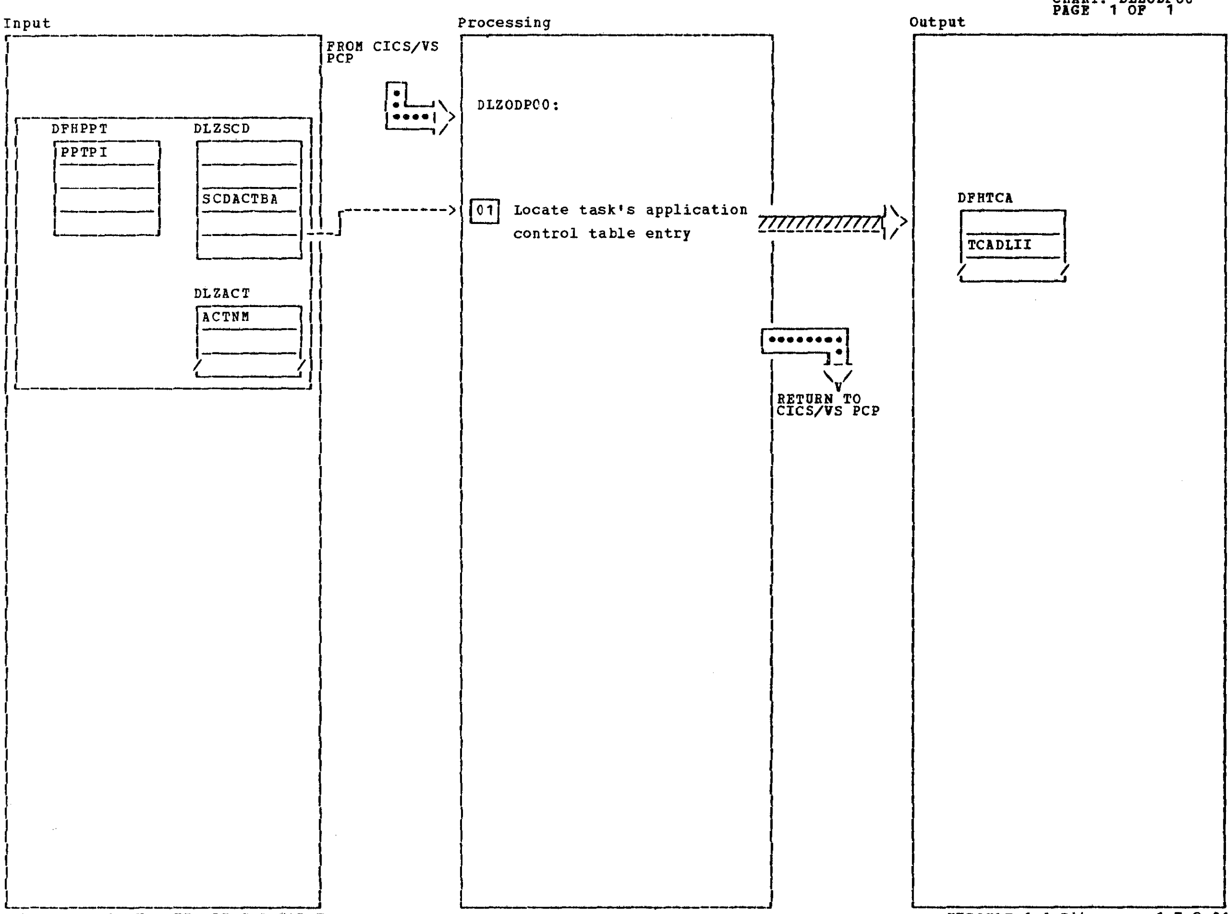
HIPOMAT 1.1 Diagram - 1.7.1.4-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01		TASKCOM					
02 The scheduling record is only created for tasks with update intent.							
03 The 'HASHED ID' is used by SPACE MANAGEMENT to prevent freed space from being re-used before the task terminates.							

DLZNUCxx -- SET TASK SCHEDULED AND LOG

HIPOMAT 1.1 Diagram - 1.7.1.4-01

CHART: DLZODP00
PAGE 1 OF 1

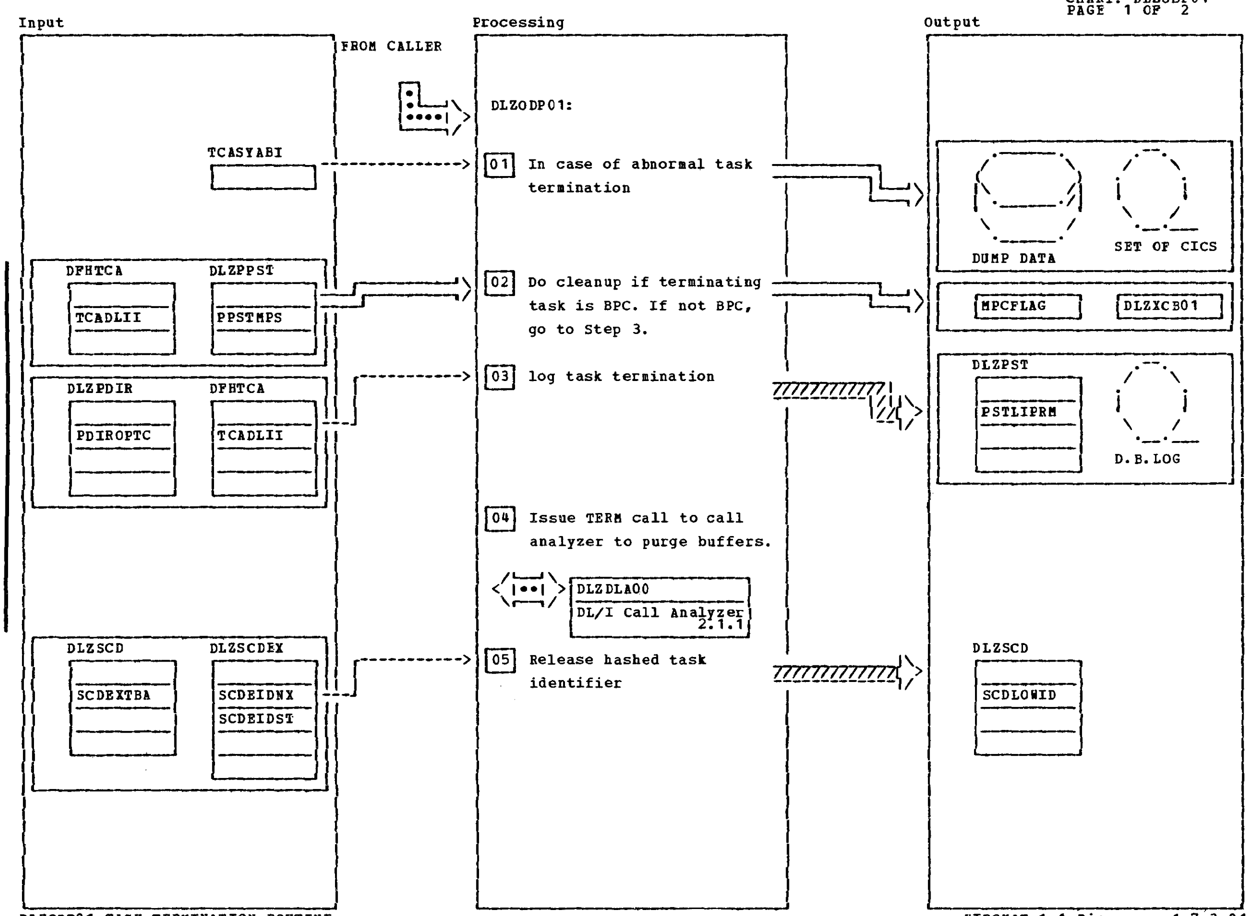


DLZNUCxx - TASK SCHEDULING ROUTINE — HIPOMAT 1.1 Diagram - 1.7.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 This step checks the authorization of the CICS application program to use DL/I.							
If the program name is not located in the act an error indicator is turned on in the TCA. If the act search is successful the act entry address is placed in the TCA.		DLZODP00					

DLZNUCxx - TASK SCHEDULING ROUTINE — HIPOMAT 1.1 Diagram - 1.7.2-01

CHART: DLZODP01
PAGE 1 OF 2



DLZODP01 TASK TERMINATION ROUTINE

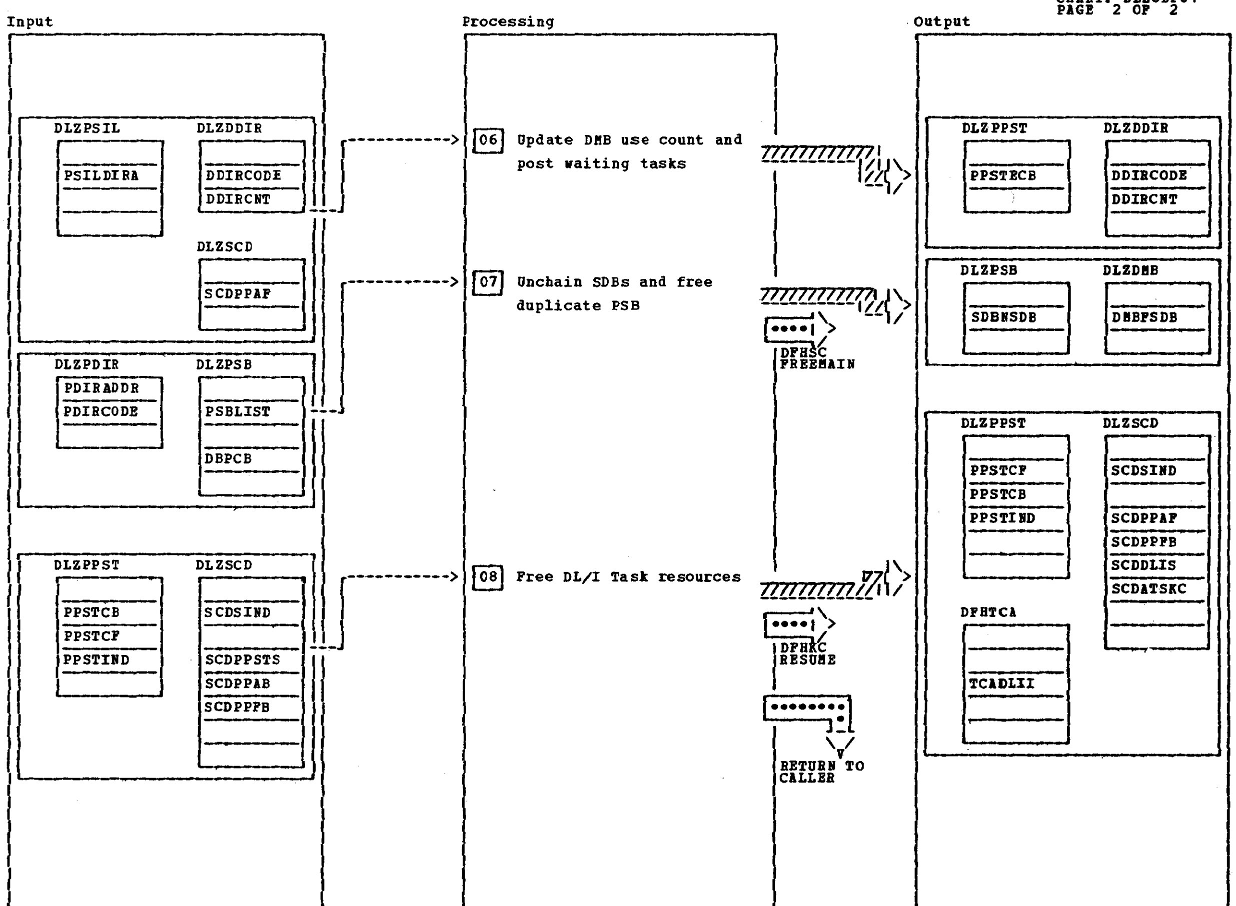
HIPOMAT 1.1 Diagram - 1.7.3-01

Notes	Routine	Label	Ref
01 No formatted dump will be produced in case of missing PST or insufficient core available.			
DL/I system abend will be reduced to task abend's : In case of DL/I system abend all DL/I tasks will be abended by DL/I. For each task then DLZFTDP0 will be called.			
DLZFTDP0 uses the CICS DUMP macro DFHDC, that dumps DL/I blocks on the CICS DUMP data set. To get the dump on printer use offline CICS program DFHDUP.			
02 If BPC (DLZBPC00) is the terminating task, the POST bit in DLZXCB01 is set on to signal MPC (DLZMPC00) that BPC abended.			
03 The termination record is logged for normal termination of update users only.		DLZTKTRM	
05 The lowest active identifier is maintained in the SCD. DL/I		TRMLGBY	

Notes	Routine	Label	Ref
Space Management uses the low and high identifiers to exclude free space belonging to active tasks from re-use.			

DLZODP01 TASK TERMINATION ROUTINE

HIPOMAT 1.1 Diagram - 1.7.3-01



DLZODP01 TASK TERMINATION ROUTINE

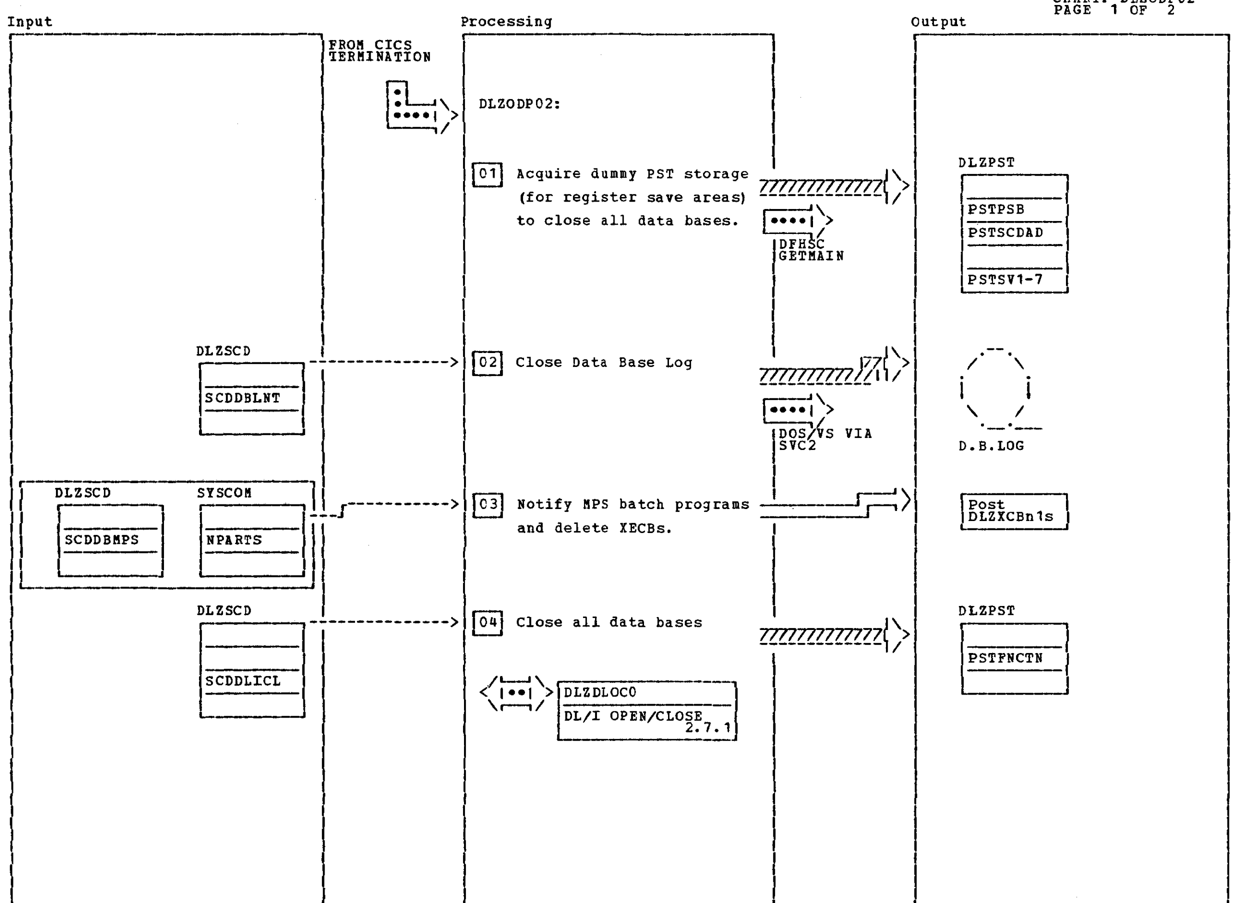
HIPOMAT 1.1 Diagram - 1.7.3-02

Notes	Routine	Label	Ref
06 All tasks waiting for intent are posted. At the next dispatch cycle they will attempt to schedule.		TRMPOLUP	
07 Read Only , duplicate PSBs are identified by the PDIR indicator PDIRDUPL.		TRMFRESB	
08 This routine cleans up the terminating PPST, removes it from the active chain, and places it on the free chain. It checks MAXTASK and resumes a task suspended due to MAXTASK.		TRMFREPP	

Notes	Routine	Label	Ref

DLZODP01 TASK TERMINATION ROUTINE

HIPOMAT 1.1 Diagram - 1.7.3-02



DLZODP02 SYSTEM TERMINATION

HIPOMAT 1.1 Diagram - 1.7.4-01

Notes	Routine	Label	Ref
01		STPBFFL	
02 This module is entered twice. First by DLZSTP00, then by DFHSTP.		DLZODP02	
03 MPS batch programs may be active or waiting for online MPS processing and are therefore notified if CICS terminates. Online XECBs defined for MPS are also deleted.			

DLZODP02 SYSTEM TERMINATION

HIPOMAT 1.1 Diagram - 1.7.4-01

Input

Processing

05 If error ocurred during DL/I OPEN/CLOSE or if CICS system terminated abnormally write formatted dump of DL/I blocks

RETURN TO CICS

Output

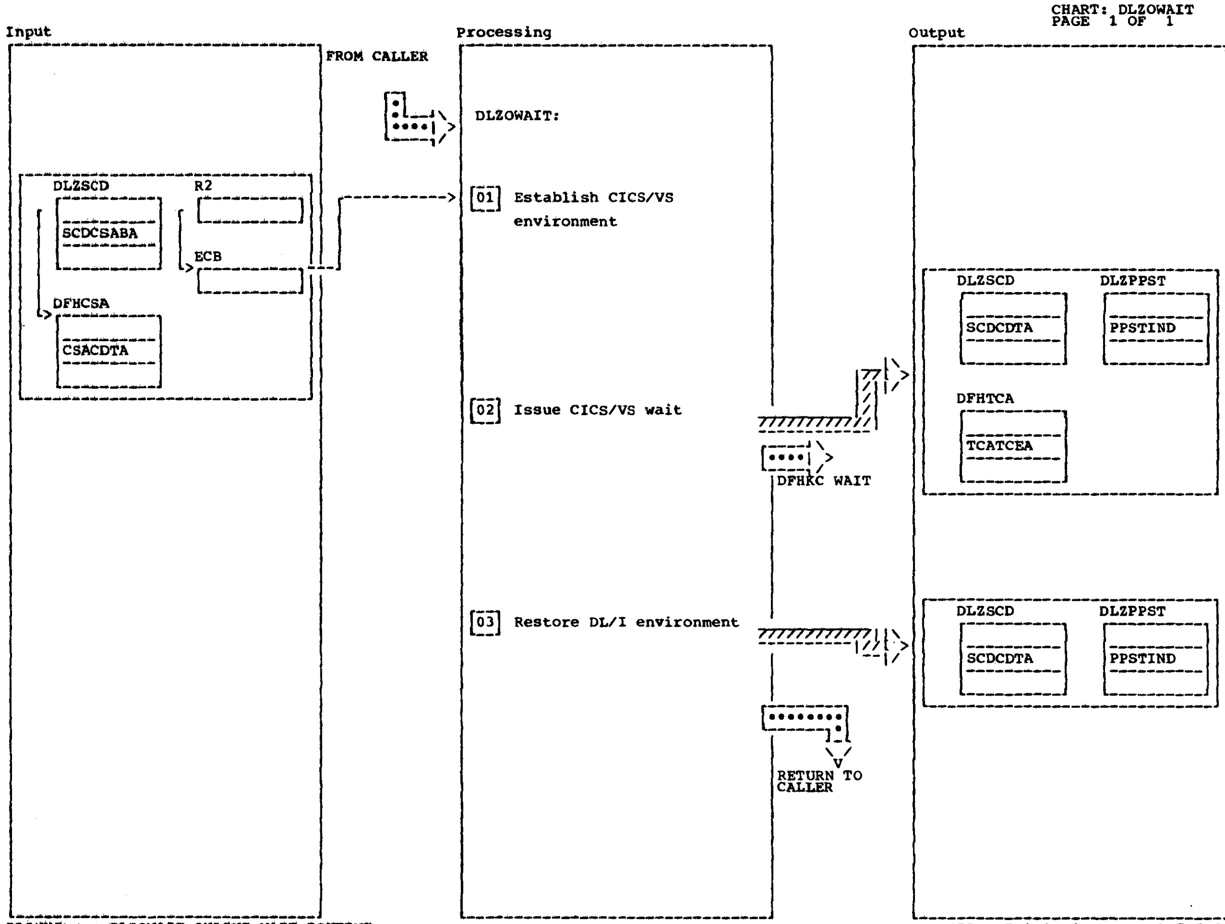
SYSLIST

DLZODP02 SYSTEM TERMINATION

HIPOMAT 1.1 Diagram - 1.7.4-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
05 Dump module DLZFSDP0 is loaded via DFHPC TYPE=LOAD and executed.							

CHART: DLZOWAIT
PAGE 1 OF 1



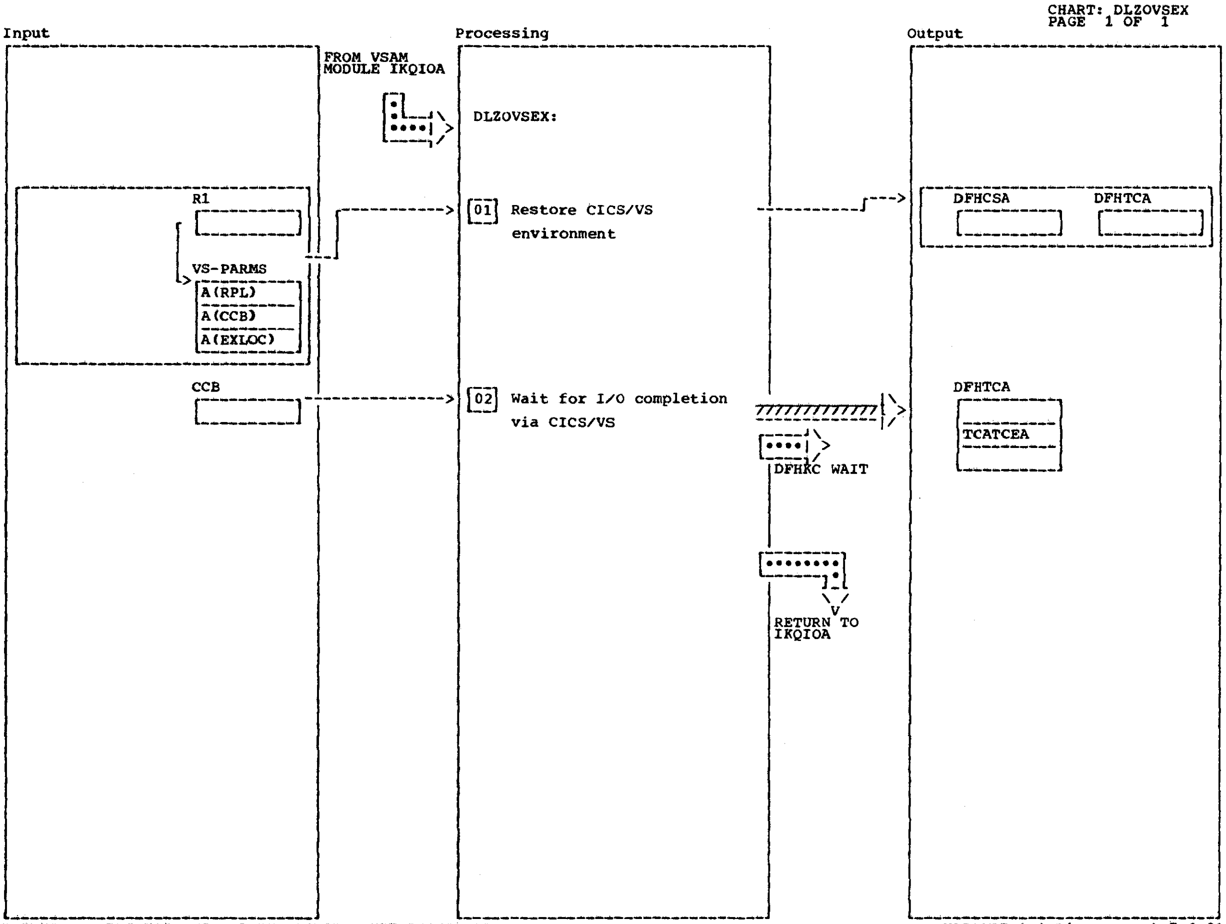
DLZNUCxx - DLZOWAIT ONLINE WAIT ROUTINE

HIPOMAT 1.1 Diagram - 1.7.5-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

DLZNUCxx - DLZOWAIT ONLINE WAIT ROUTINE

HIPOMAT 1.1 Diagram - 1.7.5-01

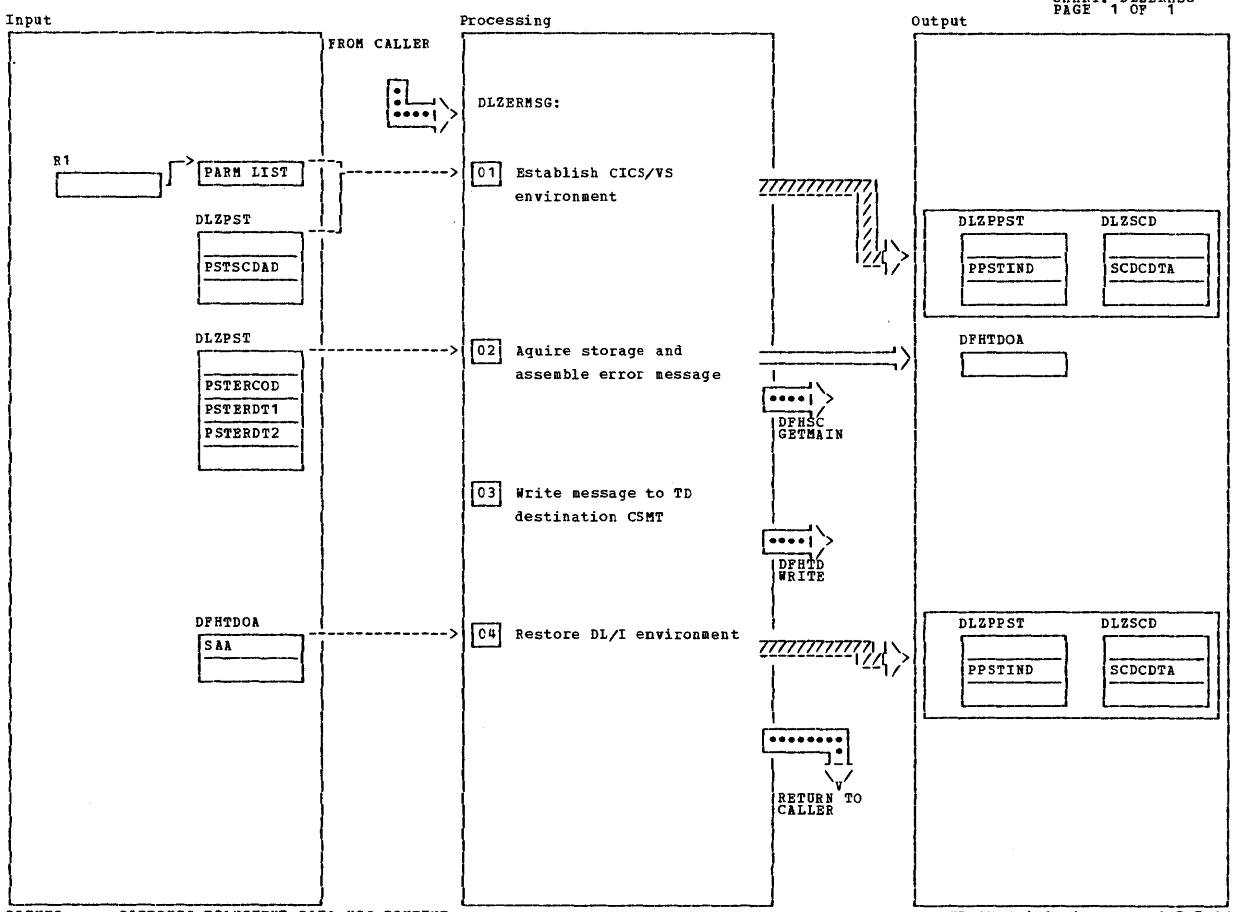


DLZNUCxx - DLZOVSEX VSAM ASYNCHRONOUS EXIT PROCESSOR HIPOMAT 1.1 Diagram - 1.7.6-01

Notes	Routine	Label	Ref

Notes	Routine	Label	Ref

DLZNUCxx - DLZOVSEX VSAM ASYNCHRONOUS EXIT PROCESSOR HIPOMAT 1.1 Diagram - 1.7.6-01



DLZNUCxx -- DLZERMSG TRANSIENT DATA MSG ROUTINE

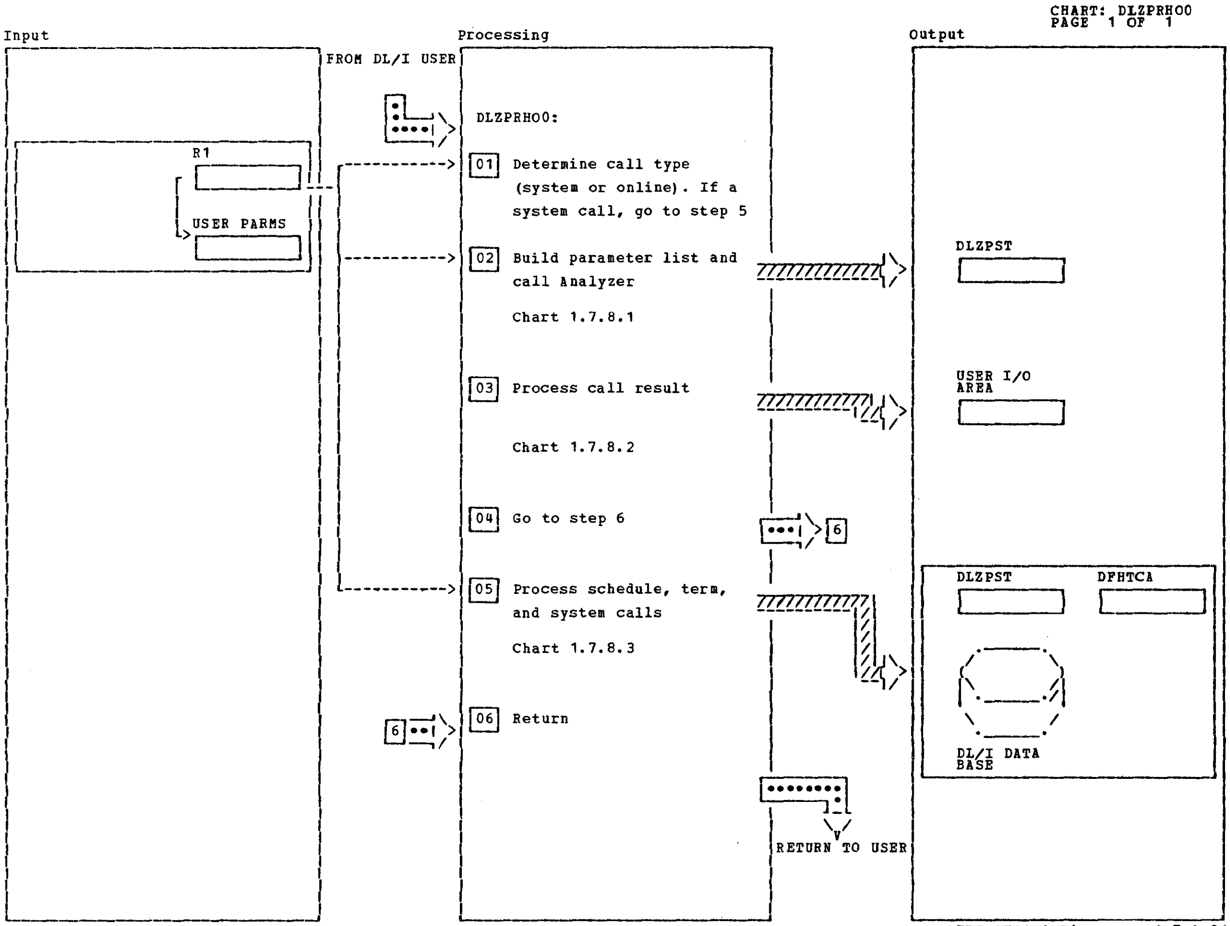
HIPOMAT 1.1 Diagram - 1.7.7-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 The parameter list is used only if PST is not available.							

DLZNUCxx -- DLZERMSG TRANSIENT DATA MSG ROUTINE

HIPOMAT 1.1 Diagram - 1.7.7-01

CHART: DLZPRH00
PAGE 1 OF 1



DLZPRH00 - ONLINE PROGRAM REQUEST HANDLER

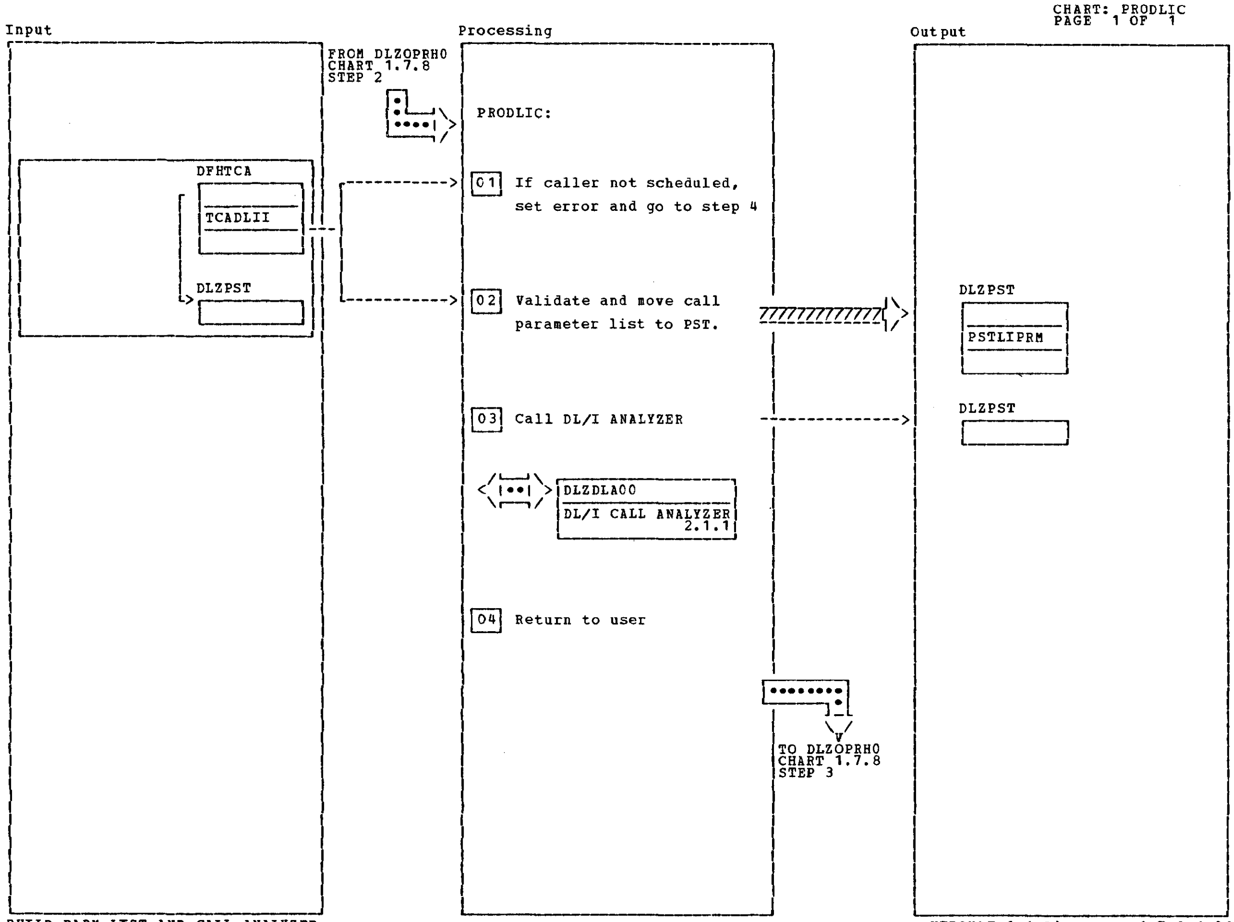
HIPOMAT 1.1 Diagram - 1.7.8-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Register 1 points to the call parameter list. The function is compared for scheduling, termination, or system calls.		DLZPRH00					
02		PRODLIC					
03 A CICS/VS abend may be issued here.							

DLZPRH00 - ONLINE PROGRAM REQUEST HANDLER

HIPOMAT 1.1 Diagram - 1.7.8-01

CHART: PRODLIC
PAGE 1 OF 1



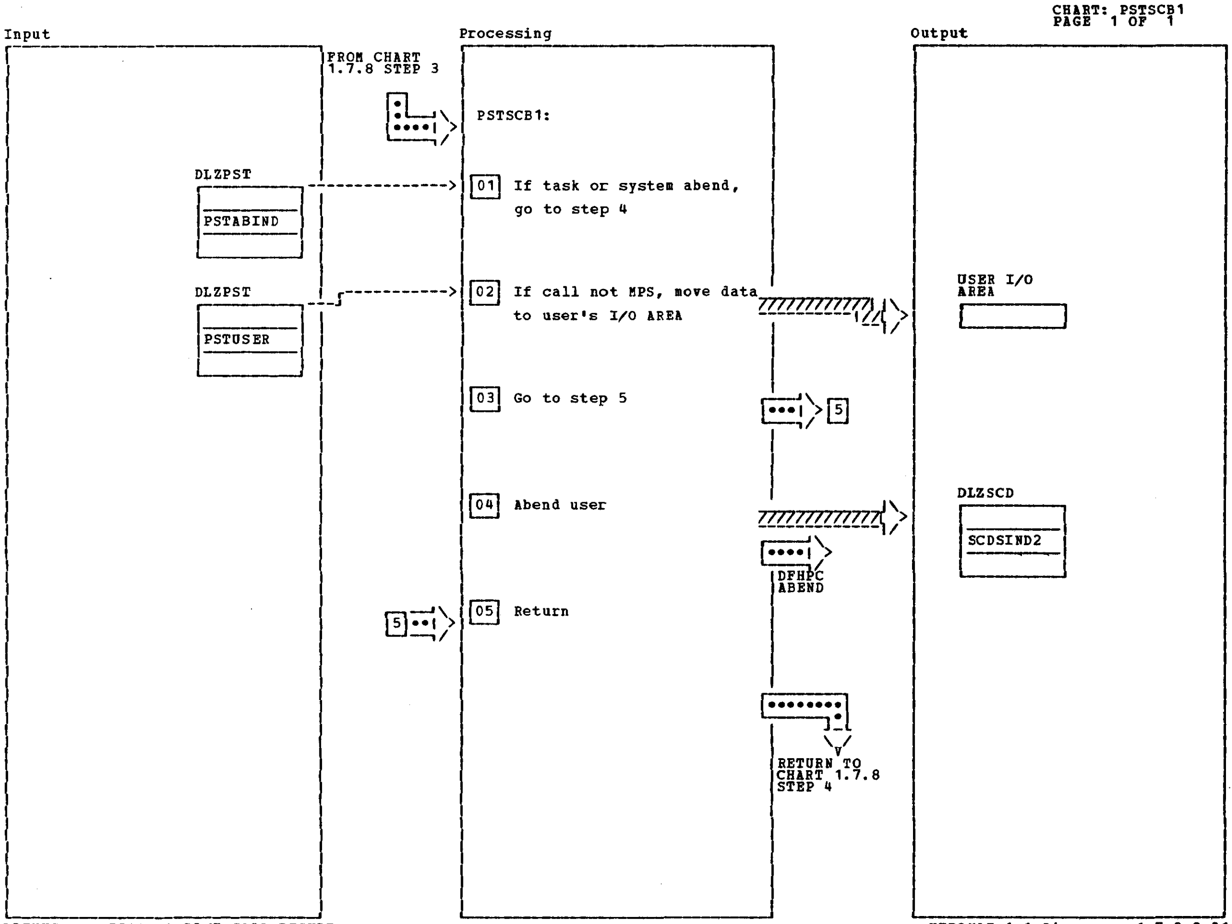
BUILD PARM LIST AND CALL ANALYZER

HIPOMAT 1.1 Diagram - 1.7.8.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
02 If call was not MPS, the addresses in the parameter list are validated against the limits of the partition, and the parameters are counted to insure they don't exceed 18.							
03 At this point the system is switched from the CICS state to the DL/I state. That is, standard register assignments.		EXITANAL					

BUILD PARM LIST AND CALL ANALYZER

HIPOMAT 1.1 Diagram - 1.7.8.1-01



DLZNUCxx - PROCESS DL/I CALL RESULT

HIPOMAT 1.1 Diagram - 1.7.8.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 If either the task or system abend indicator is on and not an MPS call.							
04 If this is a task abend, a CICS abend is issued. In addition if this is a system abend the System abend indicator is set in the SCD and all active DL/I tasks will be abended at the earliest opportunity.		PRHABEND					

Input | Processing | Output

FROM CHART
1.7.8 STEP 5

TESTFUNC:

USER CALL FUNCTION

[01] If function eq PCB then schedule

Chart 1.7.8.3.1

[02] If not TERM call, go to step 6

DFHTCA
TCADLII

[03] Call Task Termination

DLZODP01
Task Termination
1.7.3

[04] Free PST storage

DFHSC
FREEMAIN

[05] Go to step 8

USER CALL FUNCTION

[06] If function is system call,

Chart 1.7.8.3.2

[07] If function invalid, set INVREQ

DFHTCA
TCAFCTR

[08] Return

RETURN TO
CHART 1.7.8

DLZNUCxx - TESTFUNC PROCESS SCHEDULING AND SYSTEM CALLS

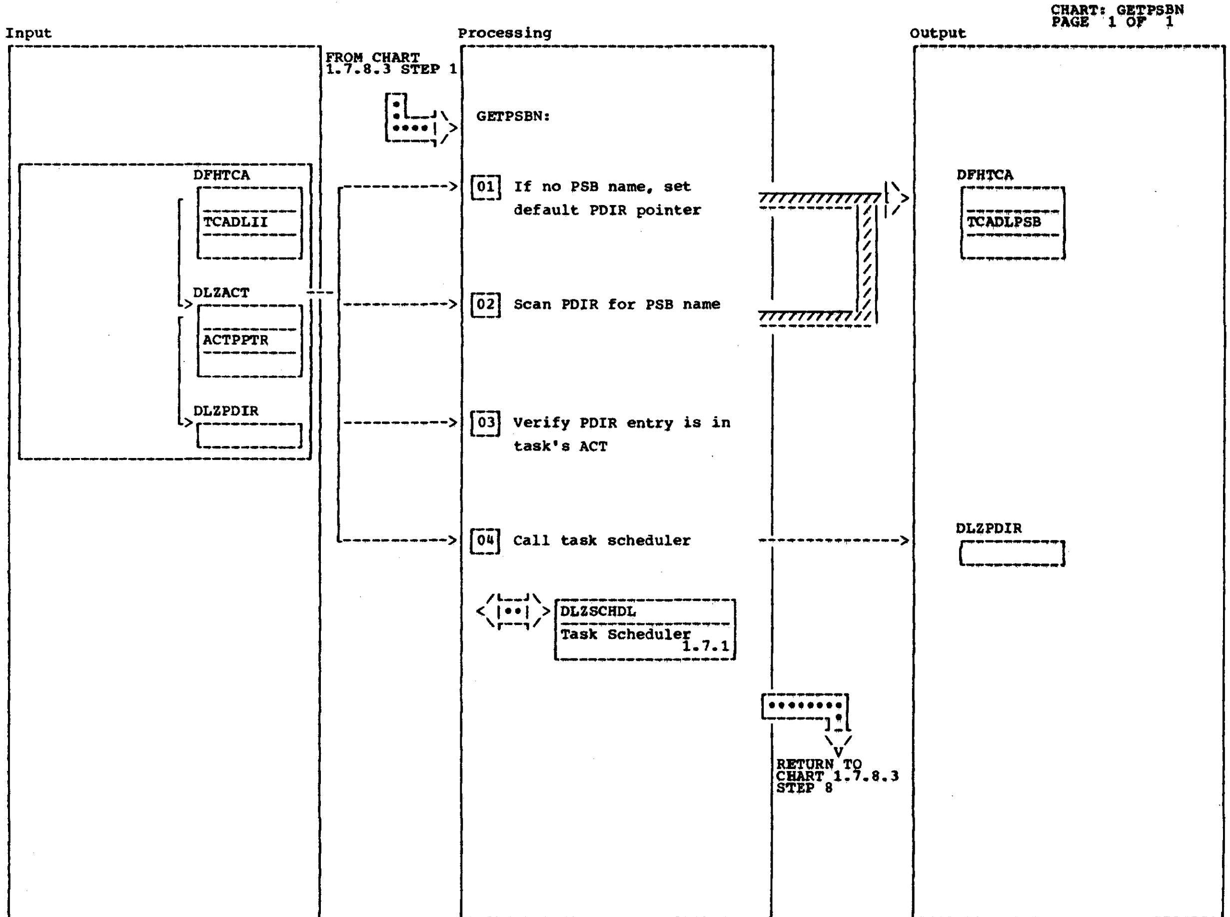
HIPOMAT 1.1 Diagram - 1.7.8.3-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[02]		ISUTERM					

DLZNUCxx - TESTFUNC PROCESS SCHEDULING AND SYSTEM CALLS

HIPOMAT 1.1 Diagram - 1.7.8.3-01

CHART: GETPSBN
PAGE 1 OF 1

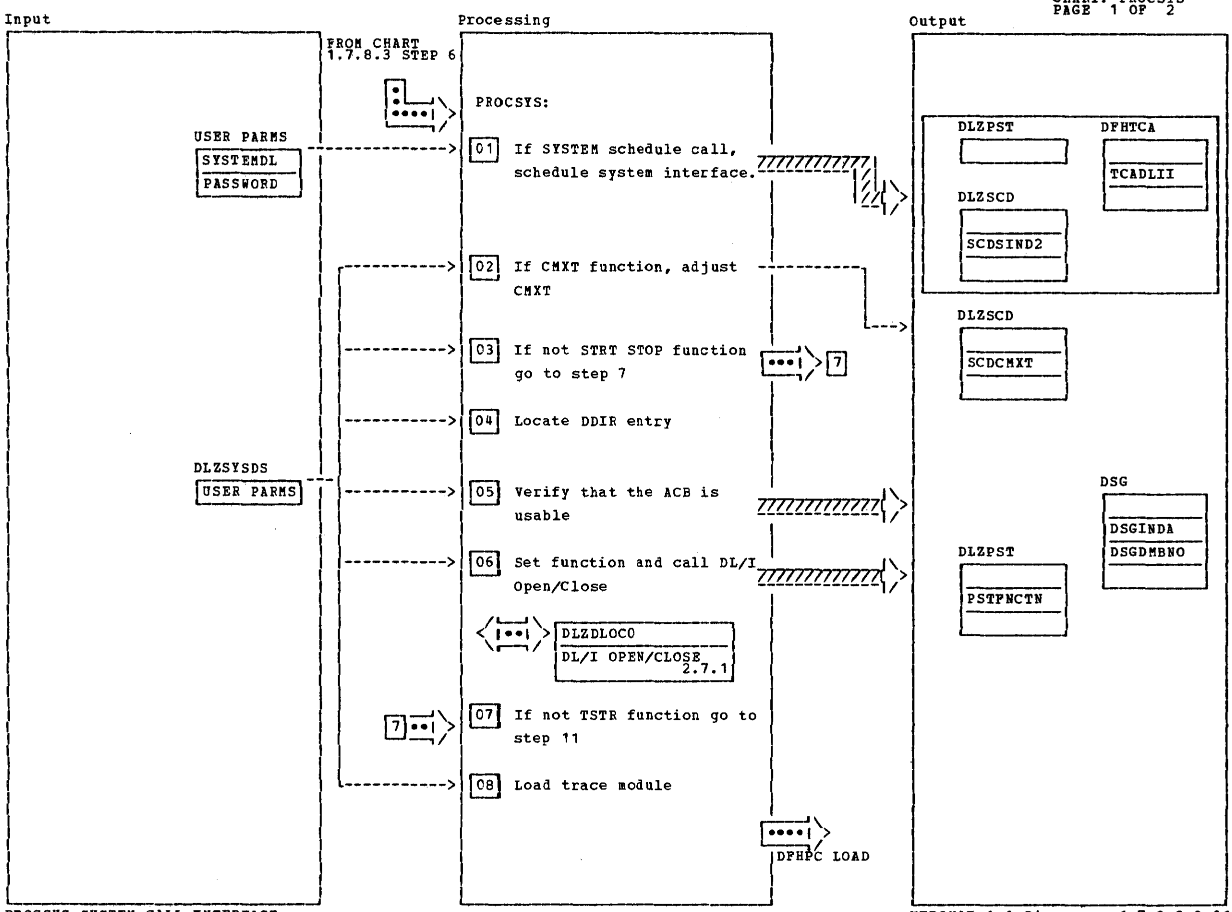


DLCNUCxx - GETPSBN SCHEDULE PSB

HIPOMAT 1.1 Diagram - 1.7.8.3.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 The first PDIR pointer is taken from the tasks's ACT entry and used to generate the PSB name.		GETPSBN					
02 If the PSB name is not in the PDIR, INVREQ is set in the TCA at TCAFCTR.							
03 If the PDIR entry is not found, INVREQ is set in the TCA at TCAFCTR.							

CHART: PROCSYS
PAGE 1 OF 2



PROCSYS SYSTEM CALL INTERFACE

HIPOMAT 1.1 Diagram - 1.7.8.3.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 A PST is acquired and initialized. If the password does not match, the caller is abended DLPV.		PROCSYS					
02 The value passed by the user is validated and moved to the SCD.		PROCMXT					
04 The DMB name passed by the caller is used to scan the DDIR.		PROICON					
05 The ACBs are checked for open/close status.							
06 The OPEN/CLOSE module issues an SVC 2 and CICS/VS loses control for the duration of the call.		PROCOCR					

PROCSYS SYSTEM CALL INTERFACE

HIPOMAT 1.1 Diagram - 1.7.8.3.2-01

Input

DLZSCD

SCDTRACE

SCDTRCNM

Processing

09 Issue initialization call to trace module

10 Store E.P. and name of trace module

Go to step 14

11 If not TSTP call go to step 14

12 Issue stop call to trace module

13 Free storage

DFHPC DELETE

14 Return

RETURN TO CHART 1.7.8.3 STEP 8

Output

DLZSCD

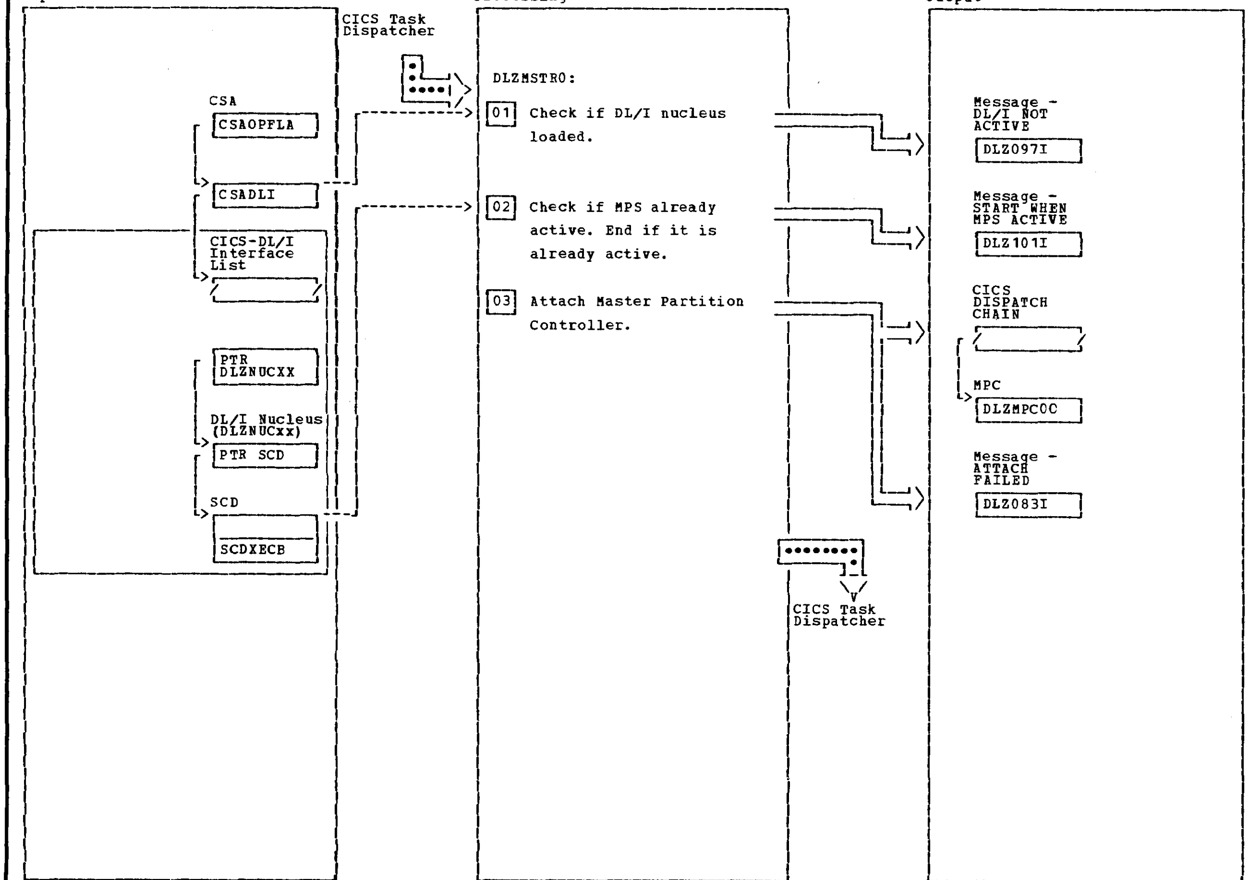
SCDTRACE

SCDTRCNM

PROCSYS SYSTEM CALL INTERFACE

HIPOMAT 1.1 Diagram - 1.7.8.3.2-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref



DLZMSTRO - Start MPS Transaction

HIPOMAT 1.1 Diagram - 1.8.0-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Message issued if nucleus not loaded or not active.							
02 Message issued if SCDXECB indicates MPS XECBs already defined.							

DLZMSTRO - Start MPS Transaction

HIPOMAT 1.1 Diagram - 1.8.0-01

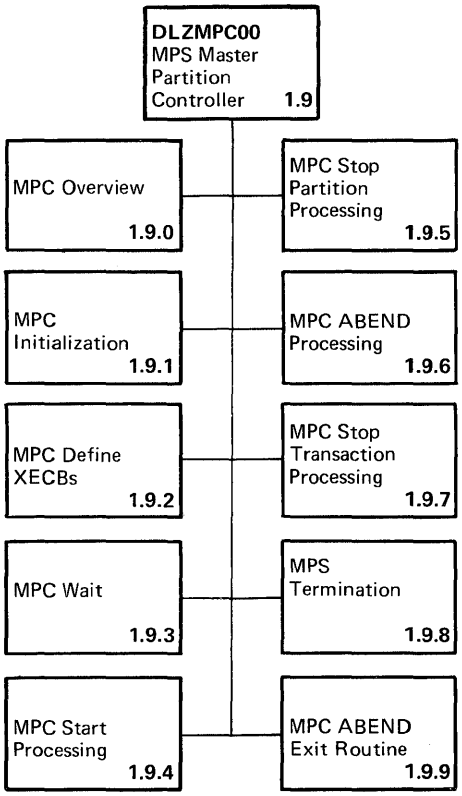
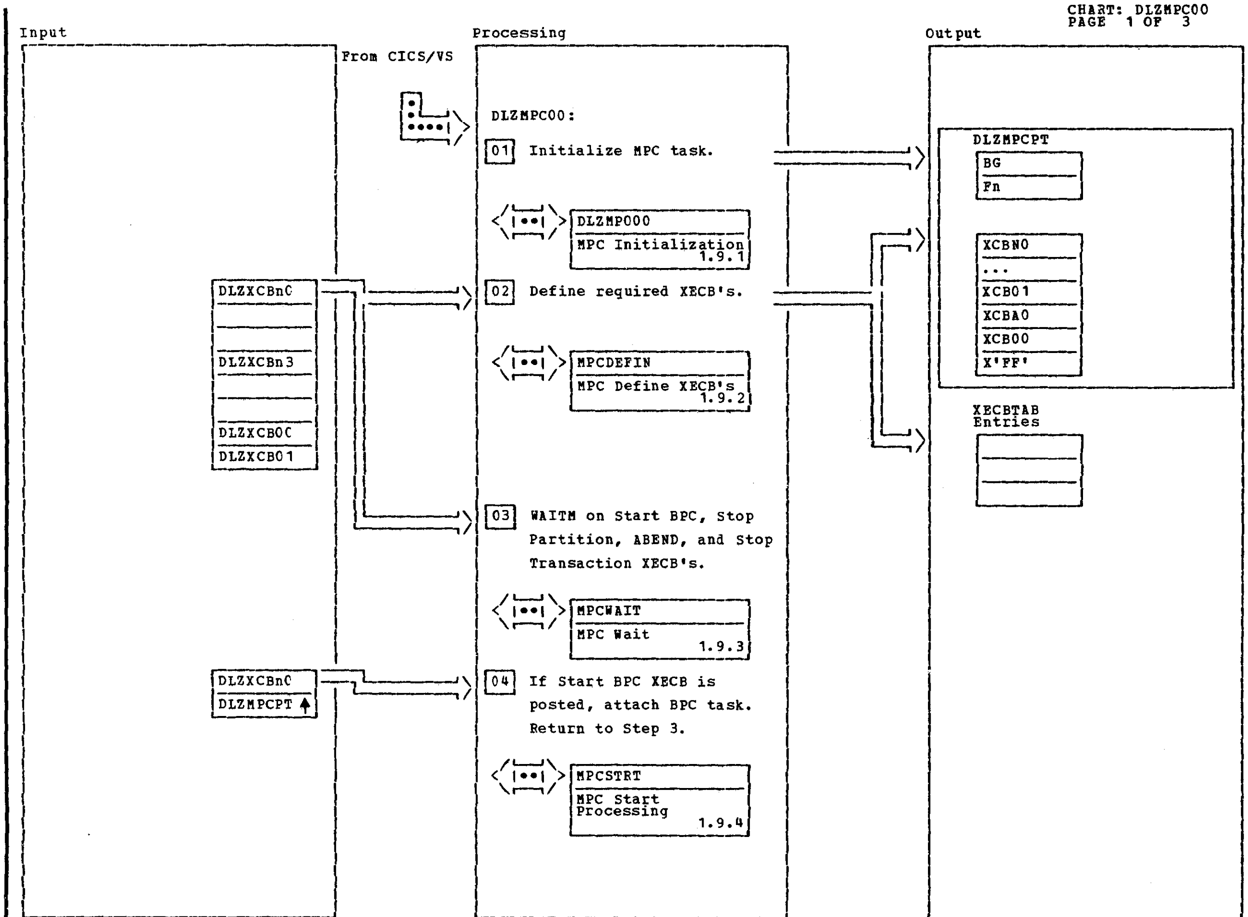


Diagram 1.9 Visual Table of Contents for MPS Master Partition Controller

CHART: DLZMPC00
PAGE 1 OF 3



DLZMPC00 - MPC Overview HIPOMAT 1.1 Diagram - 1.9.0-01

Notes	Routine	Label	Ref
01 DLZMPCPT is the name of the MPC partition table. Note - XECBname partition identifiers vary depending on the number of partitions defined during system generation (SYSGEN). For example: If two partitions defined: BG ID = F1 F1 ID = F2 If four partitions defined: BG ID = F1 F1 ID = F4 F2 ID = F3 F3 ID = F2			
02 DLZXCBn0 is the XECB name to start a batch partition controller for a specific partition. n is the partition identifier F1 thru F5, based on the key of the partition. DLZXCBn3 is the XECB name for handling an ABEND situation for a specific partition. n is the partition identifier F1 thru F5, based on the key of the partition. DLZXCB00 is the XECB name to stop the MPS transaction. DLZXCB01 is the XECB name to stop a partition. XECBTAB is a table in the supervisor containing information on all XECB's defined.			

DLZMPC00 - MPC Overview HIPOMAT 1.1 Diagram - 1.9.0-01

Input

DLZXCB01

DLZMPCPT

BG

Fn

DLZXCBn3

DLZXCB0C

DLZMPCPT

BG

Fn

Processing

05 If Stop Partition XECB is posted, do stop processing. Return to Step 3.

MPCSTOP
MPC Stop Partition Processing
1.9.5

06 If ABEND XECB is posted, do ABEND processing. Return to Step 3.

MPCABND
MPC Abend Processing
1.9.6

07 If Stop Transaction XECB is posted, do stop processing.

MPCSTRN
MPC Stop Transaction Processing
1.9.7

08 If any batch partition is active, go to Step 3.

Output

DLZMPC00 - MPC Overview

HIPOMAT 1.1 Diagram - 1.9.0-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

DLZMPC00 - MPC Overview

HIPOMAT 1.1 Diagram - 1.9.0-02

Input	Processing	Output
ECB List DLZMPCPT	09 Terminate MPC task. MPCEXIT MPS Termination 1.9.8 10 Return. To CICS/VS	

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

CHART: DLZMP000
PAGE 1 OF 2

Input	Processing	Output
From CICS/VS	DLZMP000:	
	01 Initialize base register.	
DL/I Nucleus: SCD	02 Get address of DL/I SCD.	
SCD: On-line Message Module	03 Get address of on-line message module.	
SCD	04 If MPC is already active, issue message DLZ101I and return to CICS/VS.	CICS/VS
Core Location X'80'	05 Get address of SYSCOM.	
SYSCOM + X'26': NPARTS	06 Get number of partitions defined during system generation (SYSGEN).	
	07 If only one partition was defined during system generation, issue messages DLZ088I and DLZ094I. Return to CICS/VS.	CICS/VS

DLZMPC00 - MPC Initialization

HIPOMAT 1.1 Diagram - 1.9.1-01

Notes	Routine	Label	Ref
01 MPC is attached by the MPS start transaction program via CICS/VS. On entry, R12 contains address of TCA and R13 contains address of CSA. R11 is initialized as base register.			
02 The address of DL/I SCD is obtained as follows: CSA entry CSAFLA points to OFL. OFL entry CSADLI points to DL/I interface list. DL/I interface entry at X'20' points to DL/I nucleus. First entry in DL/I nucleus points to SCD.			
03 SCDERRMS entry in SCD points to the on-line message module.			
04 MPC is active if SCDEXCB in SCDDBMPS field is set on. See Note 7 for message interface information.			
07 The following interface applies in all steps where a message is issued: R1 contains address of parameter list. R15 contains address of DL/I message module obtained from SCD (SCDERRMS) BALR R14,R15			

DLZMPC00 - MPC Initialization

HIPOMAT 1.1 Diagram - 1.9.1-01

Input

Transaction Work Area

Processing

[08] Get address of MPC partition table.

[09] Get address of CICS WAITM ECB list.

To Chart 1.9.2

Output

Notes	Routine	Label	Ref
[08] Transaction Work Area is a logical extension of the TCA.			

Notes	Routine	Label	Ref

Input

From Chart 1.9.1

Core Location C'14'

COMREG
BG COMREG

BG COMREG
PIK

DLZXCBn0
DLZXCBn3

Processing

MPCDEFIN:

10 Get address of partition COMREG.

11 Get address of BG COMREG.

12 Get partition identification key (PIK) of MPC partition.

13 Define (XWAIT) Start/ABEND XECB's for all partitions defined during system generation (SYSGEN) except MPC partitions.

14 If error return on define:

A. Issue message DLZ082I.

B. Delete any XECB's defined.

C. Go to Chart 1.9.8, step 26.

Output

XECBTAB Entries

DLZMPC00 - MPC Define XECB's HIPOMAT 1.1 Diagram - 1.9.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
12 The partition identification key is used to derive a unique XECB name (n) for the Start/ABEND XECB's. For example: X'10'=F1, X'20'=F2, etc.							
13 The XECBTAB/DEFINE macro is issued to initialize the XECBTAB table with XECB names. During the define processing, the MPC partition table is initialized with the partition ID (F1, F2, etc.) and address of ABEND XECB. Also, the CICS WAITM ECBlist is initialized with Start XECB pointers.							
14 Processing Step 14: A. See Chart 1.9.1, note 7, for message interface information. B. XECBTAB/DELETE macro issued.							

Input

DLZXCB00
DLZXCB01

DLZXCBn0
DLZXCBn3

Processing

15 Define (XPOST) Stop XECB's.

16 If error return on define:

A. Issue Message DLZ082I.

B. Delete any XECB's defined.

C. Go to Chart 1.9.8, step 26.

17 Set MPS XECB indicator in SCD.

A. Issue MPS started message DLZ093I.

B. Set ABEND routine exit for the master partition controller.

To Chart 1.9.3

Output

XECBTAB Entries

DLZMPC00 - MPC Define XECB's — HIPOMAT 1.1 Diagram - 1.9.2-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
15 The XECBTAB/DEFINE macro is issued to initialize the XECBTAB table with XECB names. The Stop XECB's are defined as XPOST so that they can be posted during abnormal system termination. The CICS WAITM ECB list is initialized with the Stop ECB pointers.							
16 Processing Step 16:							
A. See Chart 1.9.1, Step 7, for message interface information.							
B. XECBTAB/DELETE macro issued.							
17 Set flag bit SCDXECB at location SCDDBMPS.							
B. ABEND exit is established to routine MCPABEXT via the DFHPC TYPE=SETXIT.							

Input

- DLZXCBnC
  - Start Routine
  - DLZMPCTP Entry
- DLZXCBnC
  - Start Routine
  - DLZMPCPT Entry
- •
- •
- •
- DLZXCB0C
  - Stop Transact Routine
- DLZXCB01
  - Stop Partition Routine
- DLZXCBn3
  - ABEND Routine
- •
- •
- •

Processing

MPCWAIT:

18 Issue WAITM on XECB's defined.

Output

ECB List

DLZMPC00 - MPC WAIT

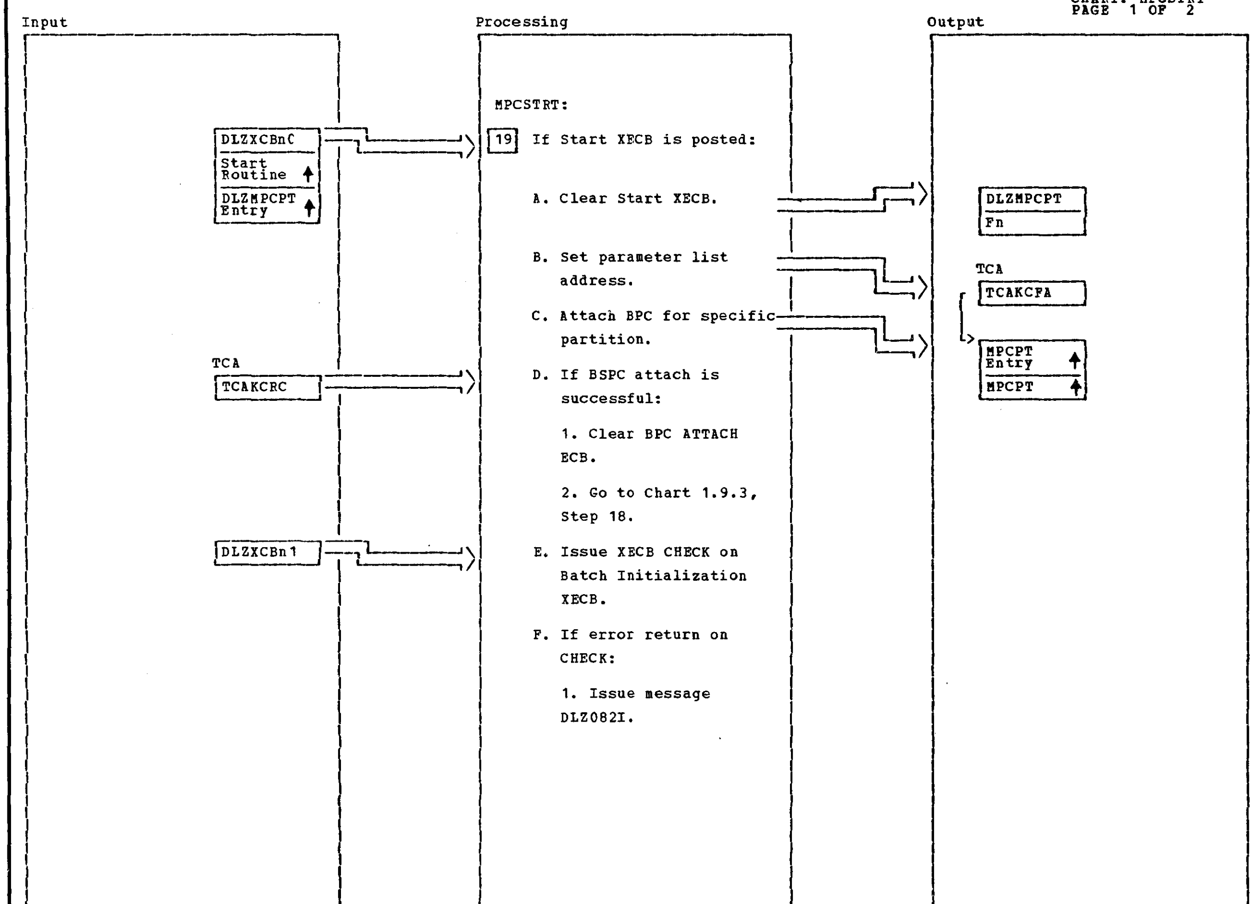
HIPOMAT 1.1 Diagram - 1.9.3-01

Notes	Routine	Label	Ref
18 CICS WAITM			
The ABEND XECB pointer is only placed in the ECB list when the BPC attach is unsuccessful. See Chart 1.9.4. Step 19.			
CICS macro:			
DFHKC TYPE=WAIT			
DCI=LIST			
When control is returned, MPC scans the ECB's to determine what action is to be taken.			

Notes	Routine	Label	Ref
The XECB's are posted on the following conditions:			
DLZXCBn0			
DLZMPI00 - Activate BPC for a specific partition.			
DLZXCB00			
DLZMSTP0 - Terminate MPS.			
DLZXCB01			
DLZBPC00 - Normal batch EOJ; error conditions in BPC or batch partition.			
DLZODP01 - ABEND.			
DLZXCBn3			
DLZMPI00 - BPC attach failure.			

DLZMPC00 - MPC WAIT

HIPOMAT 1.1 Diagram - 1.9.3-01



DLZMPC00 - MPC Start Processing HIPOMAT 1.1 Diagram - 1.9.4-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
19 XECB is posted by Batch Initialization - DLZMPI00. B. TCAKCFA field is initialized with pointer to parameter list. C. CICS macro: DFHKC TYPE=ATTACH, FCADDR=pointer to address of MPC partition table entry. TRANSID=CSDC. D. A check is made for X'31' in TCAKCRC which indicated ATTACH failure. E. Beginning at this point as much recovery is attempted on a BPC ATTACH failure to clean-up the Batch and MPC task without operator intervention. The XECBTAB/CHECK macro is issued to obtain address of Batch Partition's XECB.							

DLZMPC00 - MPC Start Processing HIPOMAT 1.1 Diagram - 1.9.4-01

Input

ECB List

ECB List

Processing

2. Indicate partition inactive.

3. Go to Chart 1.9.3, step 18.

G. Set error indicator in partition table.

H. Update CICS WAITM list with ABEND XECB.

I. XPOST Batch Initialization XECB.

J. If error return on XPOST:

1. Issue message DLZ084I.

2. Indicate partition inactive.

3. Delete error indicator.

4. Remove ABEND XECB from CICS WAITM list.

K. Go to Chart 1.9.3, Step 18.

Output

DLZMPCPT
Fn

DLZMPCPT
Fn

XCB00
...
XCB01
X'FF'

XCBNC
XCB00
XCBN3

DLZXCBn1

DLZMPCPT
Fn

XCBN0
...
XCB01

XCBN0
XCB00
X'FF'

DLZMPC00 - MPC Start Processing   HIPOMAT 1.1 Diagram - 1.9.4-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
F. See Chart 1.9.1, Note 7, for message interface information.							
H. The CICS ECB List is updated with the ABEND XECB pointer to provide recovery.							
I. The XPOST macro is issued to notify Batch Initialization of BPC ATTACH failure.							
J. See Chart 1.9.1, Note 7, for message interface information.							

Input	Processing	Output
DLZXCB01 Stop Routine	MPCSTOP: 20 If Stop Partition XECB posted: A. Clear Stop XECB. B. Scan table for specific entry. C. Set partition inactive indicator. D. Clear partition table entries. E. If Stop Transaction indicator set: 1. Delete inactive Start/ABEND XECB's. 2. If error, issue message DLZ082I. 3. If all batch partitions are inactive, go to Chart 1.9.8, Step 23. F. Go to Chart 1.9.3, step 18.	DLZMPCPT BG Fn
DLZMPCPT BG Fn		XECBTAB Entries

DLZMPC00 -- MPC Stop Partition Processing   HIPOMAT 1.1 Diagram - 1.9.5-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
20 XECB posted by BPC (DLZBPC00) or Task Termination (DLZODP01). B. A scan is done on every entry in the partition table to avoid losing a stop partition indication on a double post. E. XECBTAB/DELETE macro is issued. Set indicator in MPC partition table that XECB's have been deleted. See Chart 1.9.1, Note 7, for message interface information.							

DLZMPC00 -- MPC Stop Partition Processing   HIPOMAT 1.1 Diagram - 1.9.5-0

Input

DLZXCBn3

ABEND Routine ↑

Processing

MPCABMP:

[21] If ABEND XECB posted:

A. Clear ABEND XECB.

B. Reset ABEND indicator.

C. Reset error codes.

D. Indicate partition inactive.

E. Remove ABEND XECB from CICS WAITM list.

F. Go to Chart 1.9.3, Step 18.

Output

DLZMPCPT

Pn

XCBNO ↑

XCB00 ↑

X'FF'

...

XCB01 ↑

DLZMPC00 - MPC ABEND Processing   HIPOMAT 1.1 Diagram - 1.9.6-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[21] XECB is posted by Batch Initialization (DLZMPI00)on BPC ATTACH failure.							

DLZMPC00 - MPC ABEND Processing   HIPOMAT 1.1 Diagram - 1.9.6-01

Input

DLZXCB00
Stop Transact Routine

DLZMPCPT
BG
Fn

Processing

MPCSTRN:

$\boxed{22}$ If Stop Transaction XECB posted:

A. Clear Stop Transaction XECB.

B. Set Stop Transaction indicator.

C. Delete Stop Transaction XECB.

D. If error return on DELETE, issue message DLZ082I.

E. Delete Start/ABEND XECB's for inactive partitions.

F. If error return on DELETE, issue message DLZ082I.

G. If any batch partitions are active:

1. Issue message DLZ086I.

2. Go to Chart 1.9.3, Step 18.

Output

DLZMPCPT
BG
Fn

XECBTAB Entries

DLZMPC00 - MPC Stop Transaction Processing | HIPOMAT 1.1 Diagram - 1.9.7-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
$\boxed{22}$ XECB posted by Stop Transaction (DLZMSTP0).							
C. XECBTAB/DELETE macro issued.							
D. See Chart 1.9.1, Note 7, for message interface information.							
E. XECBTAB/DELETE macro issued. Set indicator in MPC partition table that XECB's have been deleted.							
F. See Chart 1.9.1, Note 7, for message interface information.							
G. See Chart 1.9.1, Note 7, for message interface information.							

DLZMPC00 - MPC Stop Transaction Processing | HIPOMAT 1.1 Diagram - 1.9.7-01

CHART: MPCEXIT
PAGE 1 OF 1

Input	Processing	Output
DLZXCB01	MPCEXIT:	
	23 Delete Stop Partition XECB.	XECBTAB Entries
	24 If error return on delete, issue message DLZ082I.	
	25 Reset MPS XECB indicator in SCD.	SCD
	26 Issue MPS stopped message DLZ094I.	
	27 Return to CICS.	
	To CICS/VS	

DLZMPC00 - MPS Termination

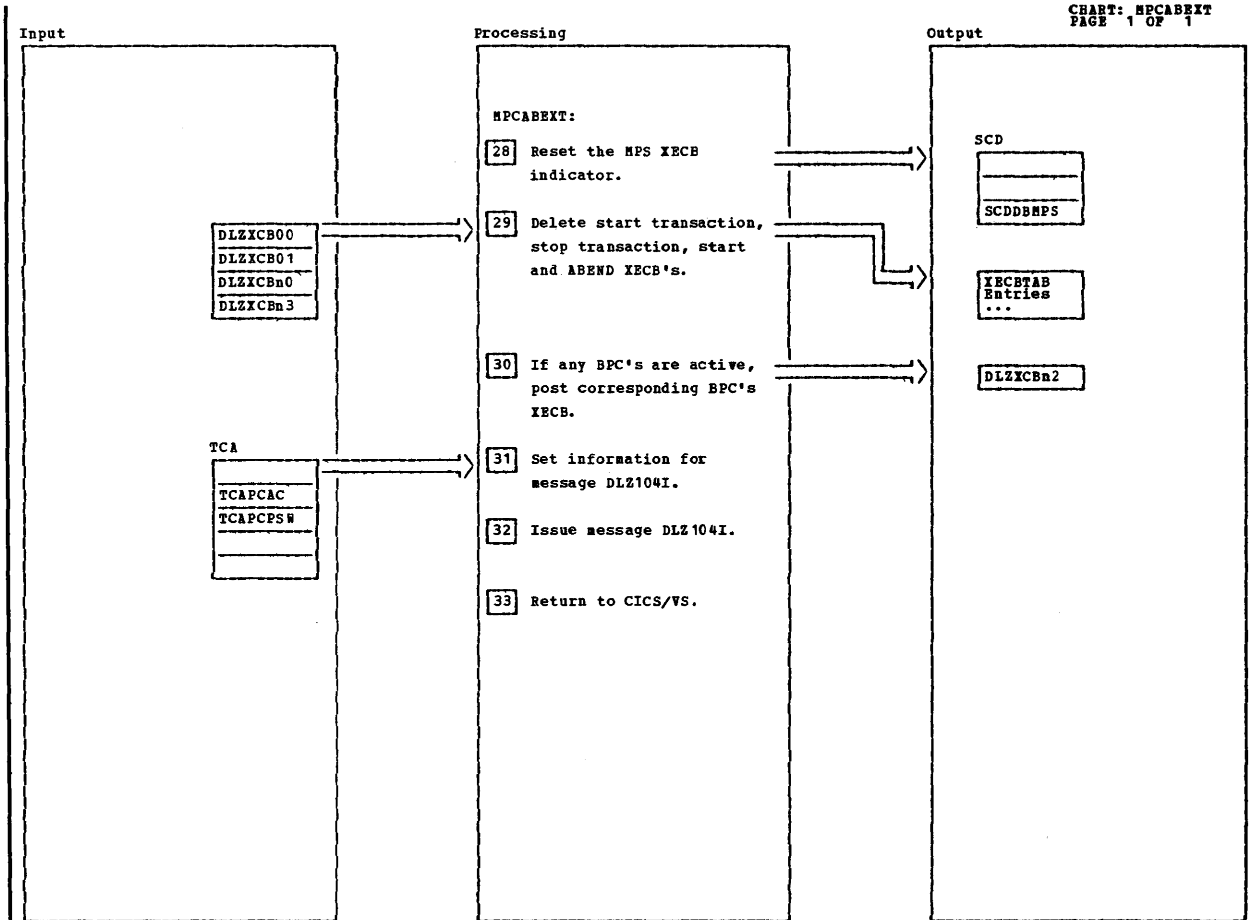
HIPOMAT 1.1 Diagram - 1.9.8-01

Notes	Routine	Label	Ref
23 XECBTAB macro TYPE=DELETE issued.			
24 See Chart 1.9.1, Note 7, for message interface information.			
25 Bit SCDXECB at location SCDDBMPS is turned off.			
26 See Chart 1.9.1, Note 7, for message interface information.			
27 CICS macro: OPHPC TYPE=RETURN			

DLZMPC00 - MPS Termination

HIPOMAT 1.1 Diagram - 1.9.8-01

CHART: MPCABEXT
PAGE 1 OF 1



DLZMPC00 - MPC ABEND Exit Routine

HIPOMAT 1.1 Diagram - 1.9.9-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
Note: Entry to this routine is from CICS/VS when linkage was established through the DFHPC TYPE=SETXIT in the MPC define XECB routine.							
28 Bit SCDXECB in SCD at location SCDDBMPS is turned off.							
29 XECBTAB macro TYPE=DELETE is issued.							
30 XECBTAB macro TYPE=CHECK is issued. Posting of the XECB is not XPOST.							
31 The TCAPCAC field contains the CICS/VS ABEND code. If this code is 'ASRA', the TCAPCPSW field contains the program interrupt PSW.							
33 Return is made to CICS/VS via DFHPC TYPE=ABEND, ABCODE=DMPC.							

DLZMPC00 - MPC ABEND Exit Routine

HIPOMAT 1.1 Diagram - 1.9.9-01

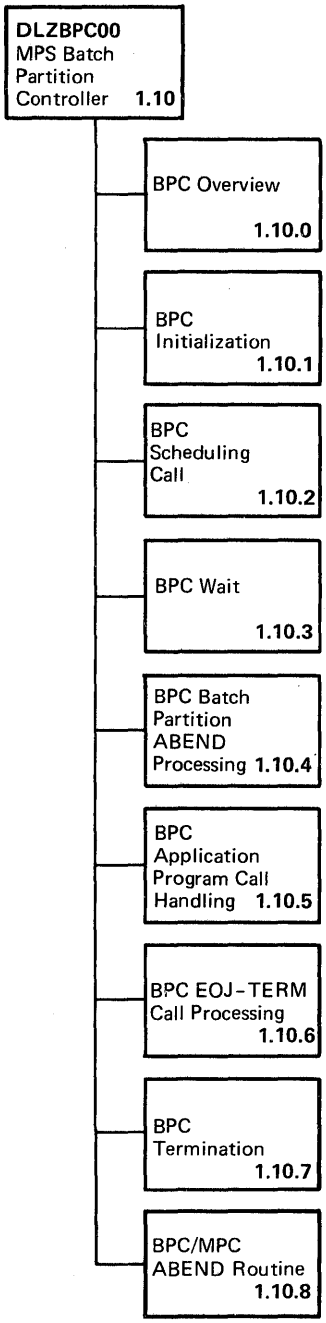
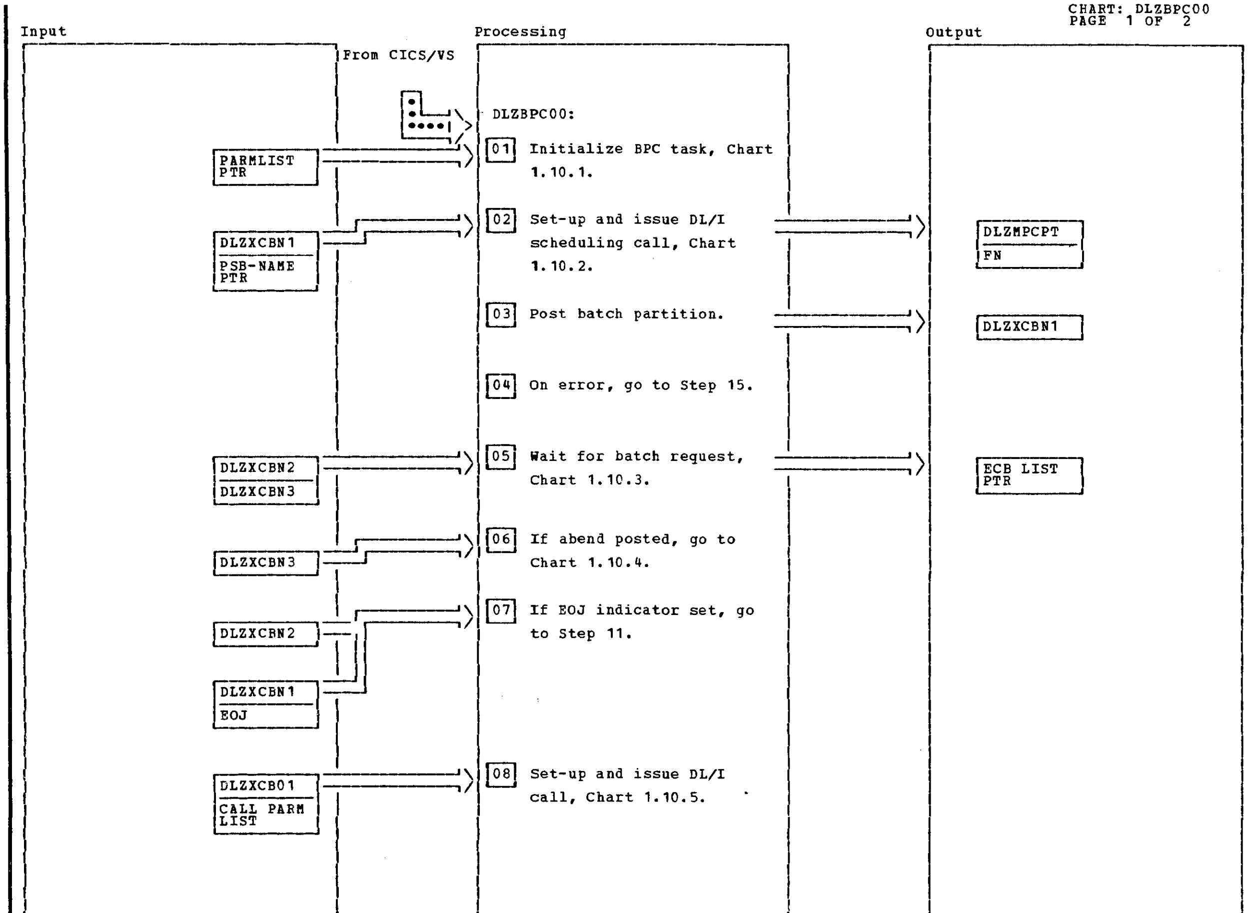


Diagram 1.10 Visual Table of Contents for MPS Batch Partition Controller



DLZBPC00-BATCH PARTITION CONTROLLER OVERVIEW

HIPOMAT 1.1 Diagram - 1.10.0-01

Notes	Routine	Label	Ref
DLZMPCPT = Name of MPC partition table.			
DLZXCBN1 = XECB name for batch partition, where N is the partition ID F1 thru F5 based on the key of the partition.			
DLZXCBN2 = XECB name for a BPC for a specific partition, where N is the partition ID F1 thru F5 based on key of the partition.			
DLZXCBN3 = XECB name for handling an abend condition for a specific partition, where N is the partition ID F1 thru F5 based on key of the partition.			
02 Assembler version of DL/I scheduling call issued on behalf of the batch partition.			
03 XPOST macro.			
05 CICS WAIT macro - DFHKC TYPE=WAIT			

Notes	Routine	Label	Ref
08 Version of the DL/I call is based on language of the application program.			

DLZBPC00-BATCH PARTITION CONTROLLER OVERVIEW

HIPOMAT 1.1 Diagram - 1.10.0-01

Input

Processing

09 Post batch program request handler, Chart 1.10.6. → DLZXCBN1

10 Go to Step 5.

11 Post batch program request handler, Chart 1.10.6. → DLZXCBN1

12 Clean up BPC task, Chart 1.10.7.

13 Return to CICS. → To CICS/VS

14 Clean up BPC task, Chart 1.10.7.

15 Return to CICS - ABEND. → To CICS/VS

Output

DLZXCBN1

DLZXCBN1

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
09 XPOST macro.							
11 XPOST macro.							
13 CICS macro DFHPC TYPE=RETURN.							
15 CICS macro DFHPC TYPE=ABEND.							

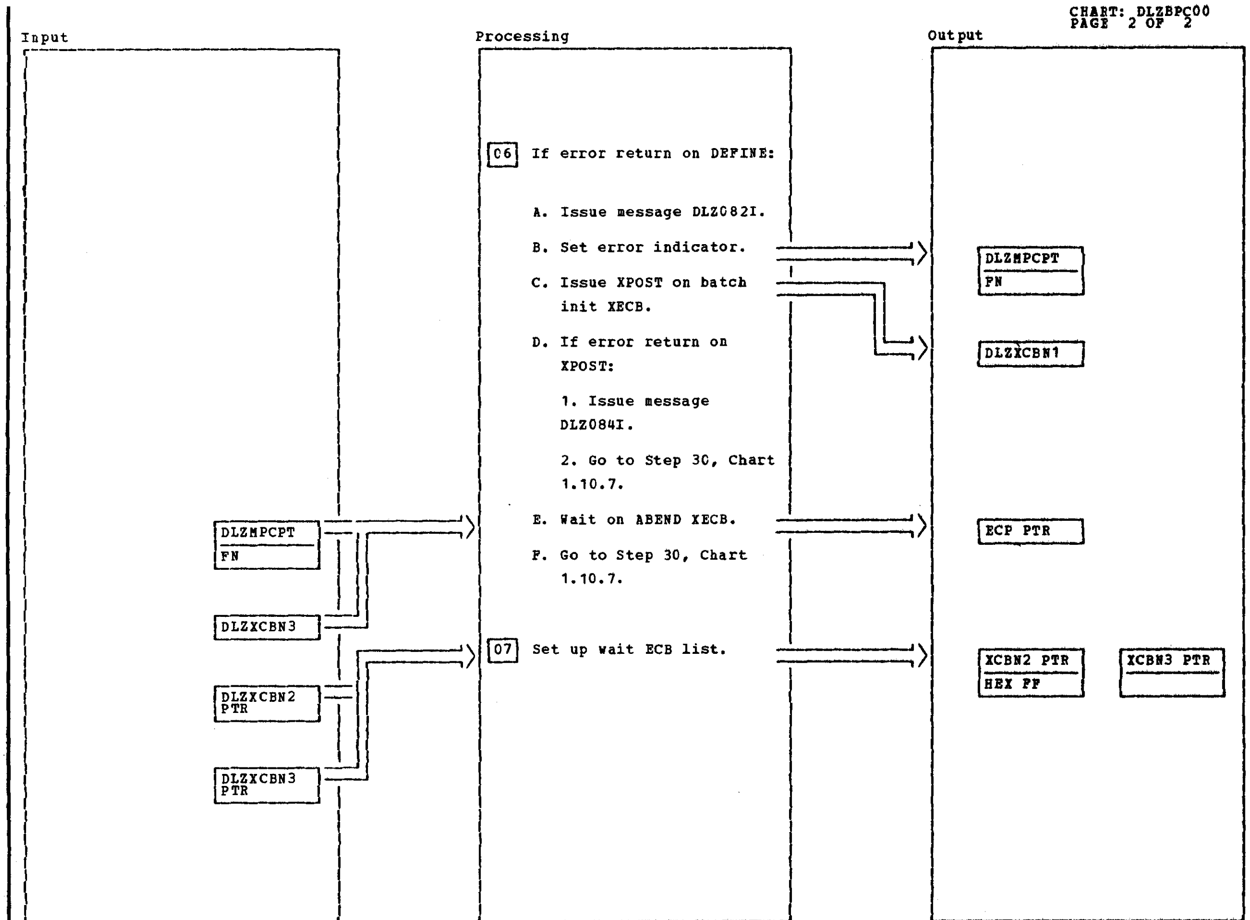
CHART: DLZBPC00
PAGE 1 OF 2

Input	Processing	Output
	DLZBPC00:	
PARM LIST PTR	01 Initialize base register, etc.	
	02 Set MPCPT address in transaction work area	Transact Work Area
	Set partition active indicator.	DLZMPCP1 / FN
DLZXCBN1	03 Issue CHECK on batch init XECB.	
	04 If error return on CHECK:	
	A. Issue message DLZ082I.	
	B. Go to step 30, Chart 1.10.7.	
DLZXCBN2	05 DEFINE (XWAIT) BPC XECB.	XECBTAB ENTRIES...

DLZBPC00 - BPC Initialization HIPOMAT 1.1 Diagram - 1.10.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 BPC is attached by the MPC via CICS/VS. On entry, register 12 contains address of the TCA and register 13 contains address of CSA. Register 11 initialized as base register.							
The MPC partition table pointer (DLZMPCPT) is obtained from the TCA whose pointer is in TCAFCAAA.							
The partition ID, and the MPCPT beginning address are stored in the transaction work area at locations: TWABPCID TWAMPCPT							
03 XECBTAB/CHECK macro is issued to obtain the address of the batch init XECB.							
05 XECBTAB/DEFINE macro is issued to define the BPC XECB for a specific partition.							

DLZBPC00 - BPC Initialization HIPOMAT 1.1 Diagram - 1.10.1-01

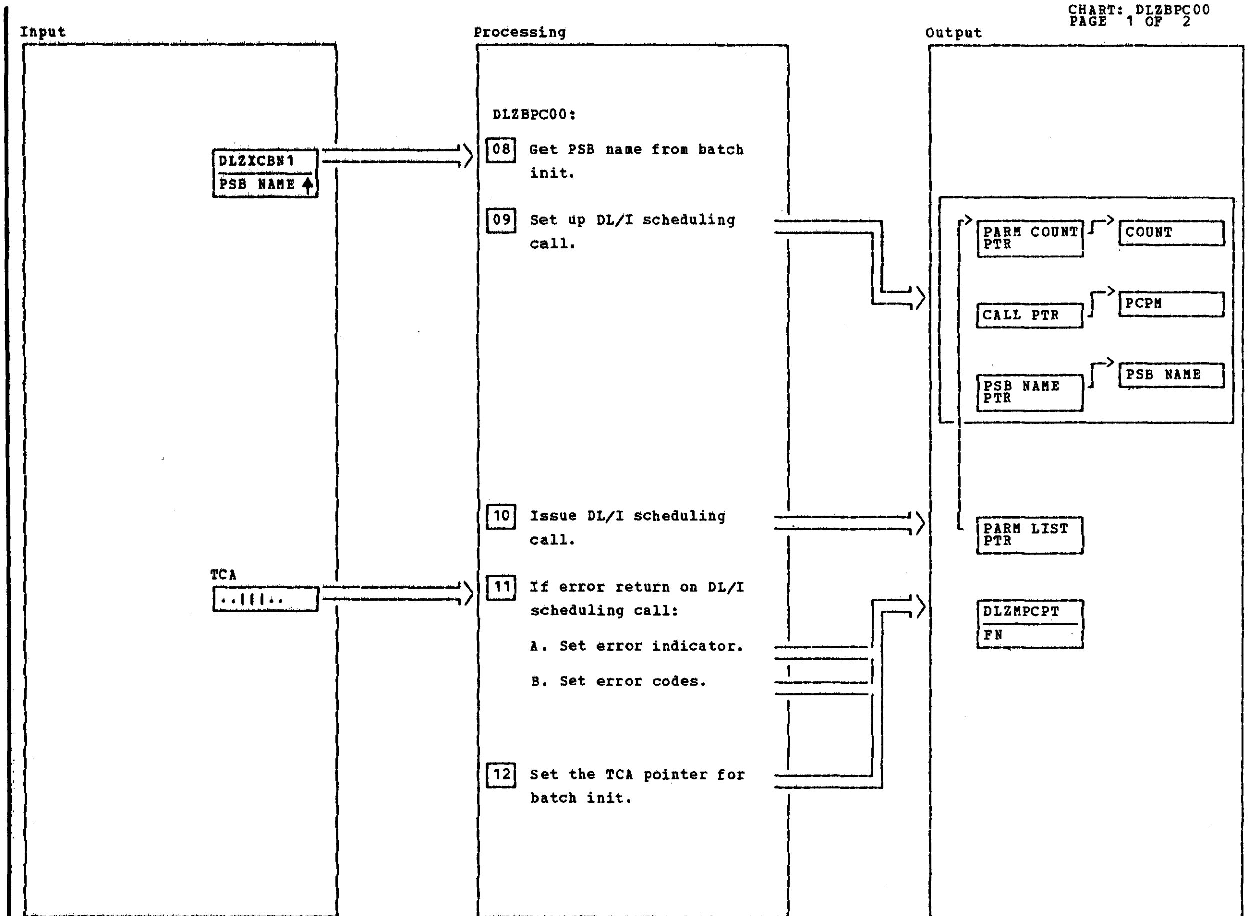


DLZBPC00 - BPC Initialization

HIPOMAT 1.1 Diagram - 1.10.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
06 C. XPOST on the batch init XECB is issued so that batch init can clean up the batch partition on error condition. E. CICS WAIT. Address of ABEND XECB is obtained from the MPC partition table. DPHK TYPE=WAIT, DCI=SINGLE 07 Address of the ABEND XECB is obtained from the MPC partition table (DLZMPCPT).							

CHART: DLZBPC00
PAGE 1 OF 2



DLZBPC00 - BPC Scheduling Call

HIPOMAT 1.1 Diagram - 1.10.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
10 Assembler version of the DL/I scheduling call is used.							
11 B. Error codes are obtained from TCA (TCAFCTR, TCADLTR).							

DLZBPC00 - BPC Scheduling Call

HIPOMAT 1.1 Diagram - 1.10.2-01

Input	Processing	Output
	[13] Issue XPOST on batch init XECB.	DLZXCBN1
	[14] If error return on XPOST:	
	A. Issue message DLZ084I.	
	B. Go to step 28, Chart 1.10.7.	

DLZBPC00 - BPC Scheduling Call — HIPOMAT 1.1 Diagram - 1.10.2-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[13] XPOST is issued on batch init XECB to notify batch init that DL/I scheduling call has been completed.							

DLZBPC00 - BPC Scheduling Call — HIPOMAT 1.1 Diagram - 1.10.2-02

Input	Processing	Output
DLZXCBN2 DLZXCBN3	DLZBPC00: 15 Issue WAITM on BPC/ABEND XECB A. If MPC abended, go to Chart 1.10.8, step 37.	ECB LIST POINTER

Notes	Routine	Label	Ref
15 CICS WAITM - XECBs are posted for the following conditions: DLZXCBN2 Process call on behalf of batch partition. EOJ has been encountered in batch partition. DLZXCBN3 An abend condition has been encountered in the batch partition. CICS MACRO: DFHKC TYPE=WAIT, DCI=LIST ECB list address pointer is placed in TCATCEA prior to issuing WAIT When CICS returns control after a WAIT, BPC must scan the ECB list to determine			

Notes	Routine	Label	Ref
which ECB is posted. A. Bit SCDXECB is tested in field SCDDBMPS of the SCD.			

Input	Processing	Output
DLZXCBN3 →	DLZBPC00: 16 If ABEND XECB posted: A. Clear ABEND XECB. B. Go to Step 28, Chart 1.10.7.	

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
16 Entered when abend condition has been encountered in the batch partition.							

Input

- DLZXCBN2 → 17
- DLZXCBN1 / EOJ → 18
- PST (..|.|..) → 20
- DLZXCBN1 / PARM LIST → 21A, 22

Processing

DLZBPC00:

17 Clear BPC XECB.

18 If EOJ posted:

A. Go to Step 26, Chart 1.10.6.

19 Set MPS indicator.

20 Determine application program language.

21 If PL/I:

A. Issue PL/I Version of DL/I call.

B. Go to Step 23, Chart 1.10.5.

22 Issue Assembler/COBOL version of DL/I call.

Output

- 19 → PPST (.|.|..)
- 21A → PARM LIST PTR
- 22 → PARM LIST PTR

DLZBPC00 - BPC Application Program Call Handling   HIPOMAT 1.1 Diagram - 1.10.5-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
19 Address of the PPST is obtained from the PST (PSTPREAD)							
20 Address of PST is obtained from the TCA (TCADLIPA).							
21 Entry point in the language interface program is PLITDLI.							
22 Entry point in the language interface program is ASMTDLI or CBLTDLI.							

DLZBPC00 - BPC Application Program Call Handling   HIPOMAT 1.1 Diagram - 1.10.5-01

Input	Processing	Output
	23 Issue XPOST on batch partition XECB.	DLZXCBN1
	24 If error return on XPOST: A. Issue message DLZ084I. B. Go to Step 28, Chart 1.10.7.	
	25 Go to Step 15, Chart 1.10.3.	

DLZBPC00 - BPC Application Program Call Handling HIPOMAT 1.1 Diagram - 1.10.5-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
23 XPOST issued on the batch partition XECB to notify that the call on behalf of the batch partition has completed.							
25 Return to WAITM on BPC/ABEND XECBs.							

Input	Processing	Output
	DLZBPC00: 26 Issue XPOST on batch partition XECB. 27 If error return on XPOST: A. Issue message DLZ084I. B. Go to Step 28, Chart 1.10.7.	DLZXCBN1

DLZBPC00 - BPC - EOJ - TERM Call Processing

HIPOMAT 1.1 Diagram - 1.10.6-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
26 XPOST the batch partition XECB in to notify the batch partition that end-of-job processing has completed.							

DLZBPC00 - BPC - EOJ - TERM Call Processing

HIPOMAT 1.1 Diagram - 1.10.6-01

Input	Processing	Output
DLZXCBN2	DLZBPC00:	
	28 Delete BPC XECB.	XECBTAB ENTRIES...
	29 If error return on DELETE, issue message DLZ082I.	
	30 Set stop partition indicator.	DLZMPCPT / FN
	31 Post stop partition XECB.	DLZXCB01
	32 Issue message DLZ104I.	
	33 If abnormal termination, go to Step 35.	
	34 Return to CICS/VS.	To CICS/VS
	35 Set ABEND indicator in transaction work area.	
	36 Return to CICS/VS - ABEND.	To CICS/VS

DLZBPC00 - BPC Termination

HIPOMAT 1.1 Diagram - 1.10.7-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
28 XECBTAB/DELETE macro issued to delete the XECBNAME from the XECBTAB table.							
31 This is not an XPOST since XECB was DEFINED in this partition.							
34 CICS macro: DFHPC TYPE=RETURN							
35 ABEND indicator TWABPCOK is set on in field TWAMPSFG in the transaction work area to indicate BPC ABEND processing was successful.							
36 CICS macro: DFHPC TYPE=ABEND ABCODE=DBPC DBPC ABEND code defines BPC failure for CICS/VS dump ID.							

DLZBPC00 - BPC Termination

HIPOMAT 1.1 Diagram - 1.10.7-01

Input	Processing	Output
DLZXCBN2	MPCABEND:	
	37 Delete the BPC's XECB.	XECBTAB ENTRIES ...
	38 Post the batch partition XECB.	DLZXCBN1
	39 Set abnormal condition for message DLZ103I.	
	40 Go to Chart 1.10.7, Step 32.	

DLZBPC00 - BPC/MPC ABEND Routine

HIPOMAT 1.1 Diagram - 1.10.8-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
37 The XECBTAB macro TYPE=DELETE is issued.							
38 XPOST is issued on batch partition XECB (DLZXCBN1).							

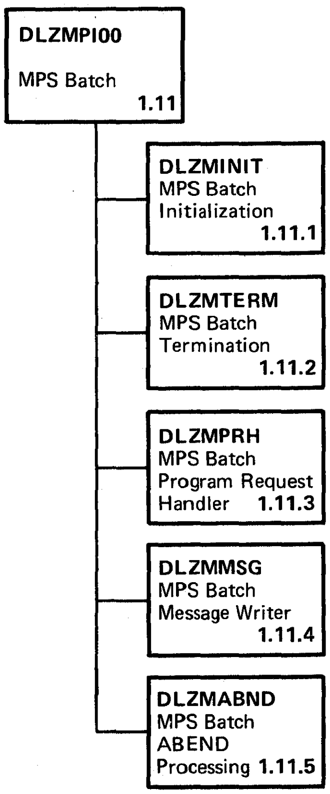
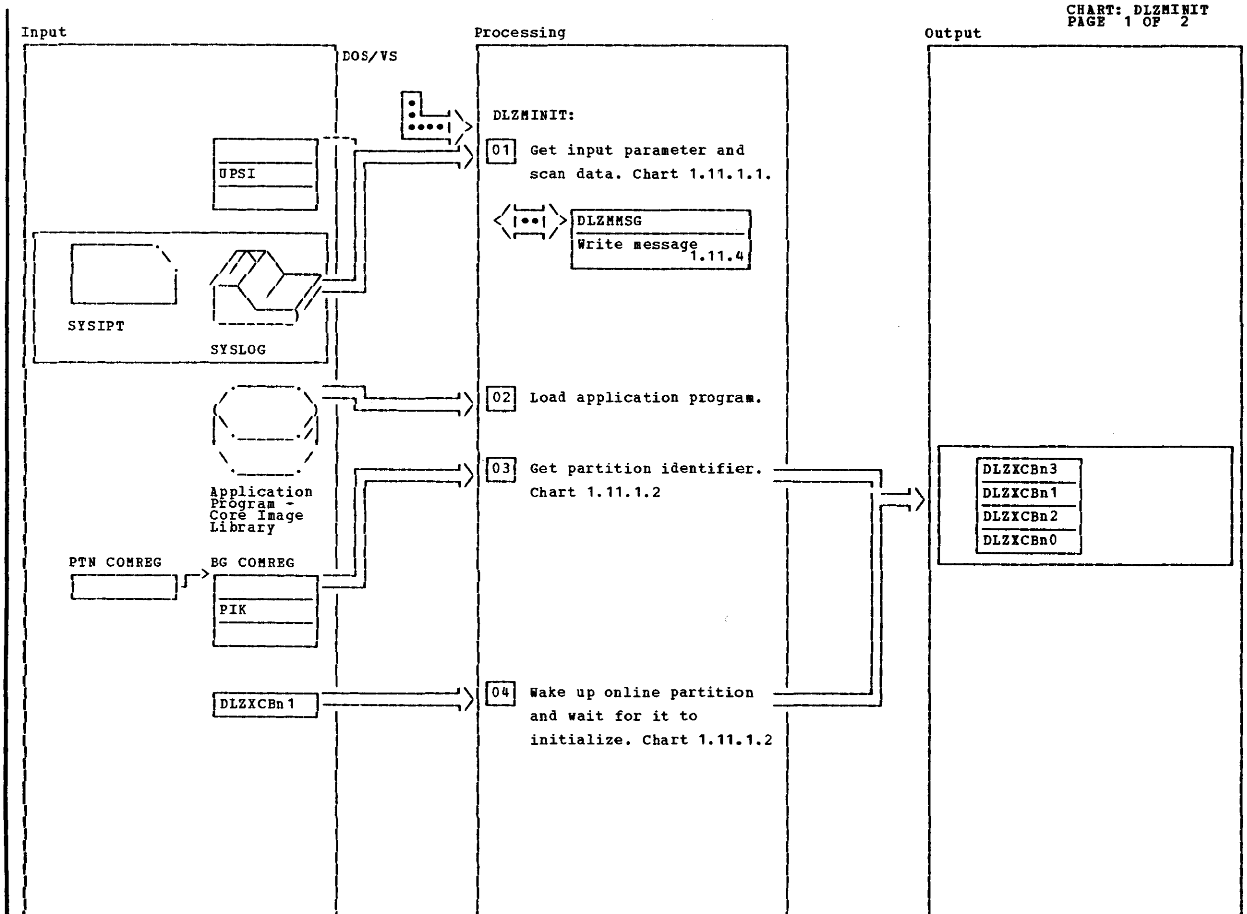


Diagram 1.11 Visual Table of Contents for MPS Batch



DLZMPI00 - MPS Batch Initialization

HIPOMAT 1.1 Diagram - 1.11.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Read from SYSLOG or SYSIPT depending on UPSI bit 0.							
03 Modify 'n' in XECB names based on partition running in.							
04 XPOST XECB DLZXCBn0 and XWAIT on DLZXCBn1.							

DLZMPI00 - MPS Batch Initialization

HIPOMAT 1.1 Diagram - 1.11.1-01

Input

Processing

05 Complete initialization. Chart 1.11.1.3

06 Exit to user application program.

User Program
Process using DL/I

DLZMTERM Chart 1.11.2

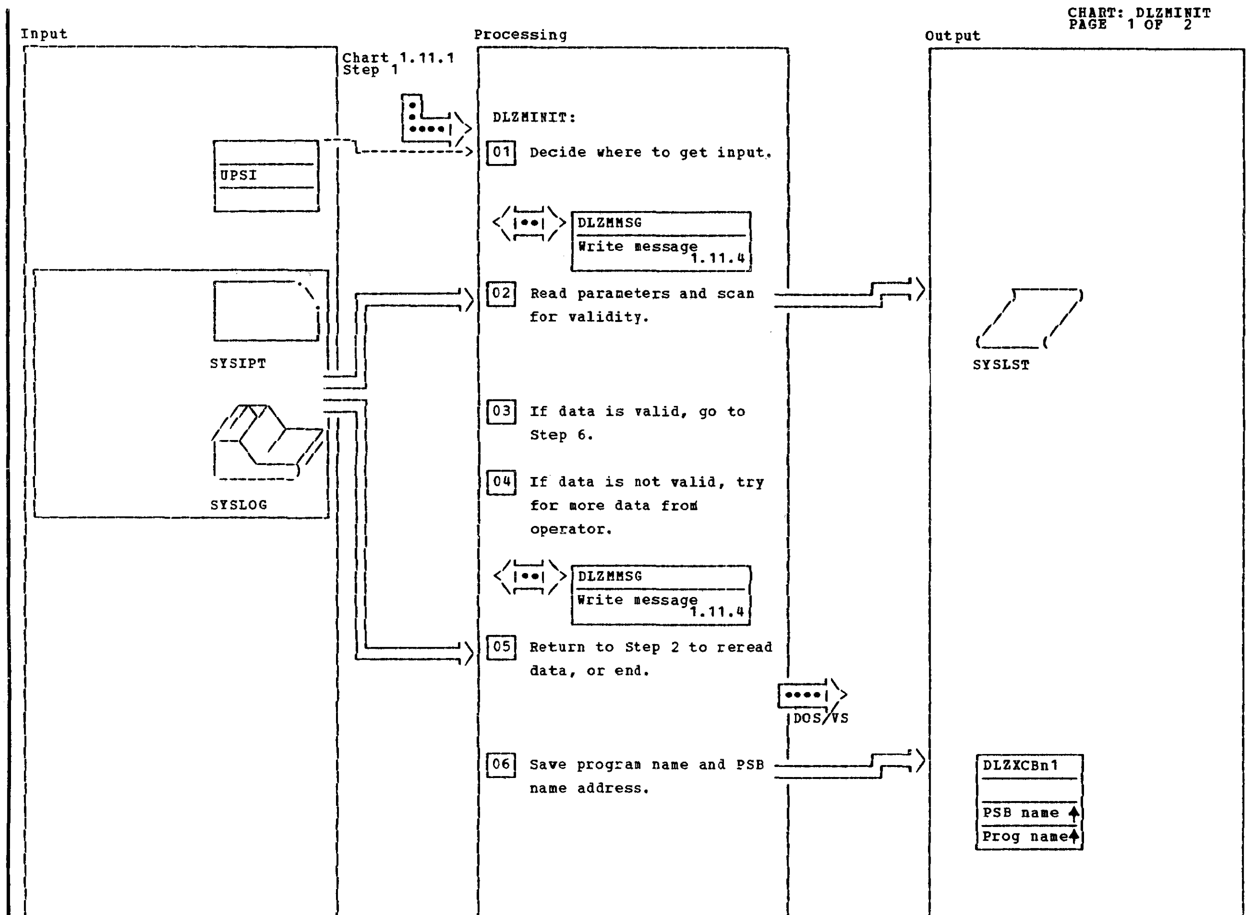
Output

Partition COMREG

PRH

DLZMPI00 - MPS Batch Initialization HIPOMAT 1.1 Diagram - 1.11.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
06 An application program runs as a subroutine of DL/I initialization. At normal termination XECB DLZICBn2 is XPOSTed and DLZXCBn1 is deleted.							



DLZMPIOC - MPS Batch Initialization

HIPOMAT 1.1 Diagram - 1.11.1.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 If UPSI byte bit 0 is on, input is from SYSLOG. Send message DLZ010A to have operator enter information.							
If UPSI byte bit 0 is off, input is from SYSIPT.							
02 Read parameters from SYSIPT or SYSLOG. Print parameters on SYSLST.							
04 If end of file on SYSIPT, send message DLZ014A.							
If data invalid, send message DLZ087A.							
05 Read operator reply to decide if more parameters provided or if job is cancelled.							
06 The pointers to program name and PSB name are kept in area behind XECB used for communicating with the online partition.							

DLZMPIC0 - MPS Batch Initialization

HIPOMAT 1.1 Diagram - 1.11.1.1-01

Input

Application program - Core Image Library

Processing

07 Load application program.

08 End processing if program not found.

DLZMMSG
Write message 1.11.4

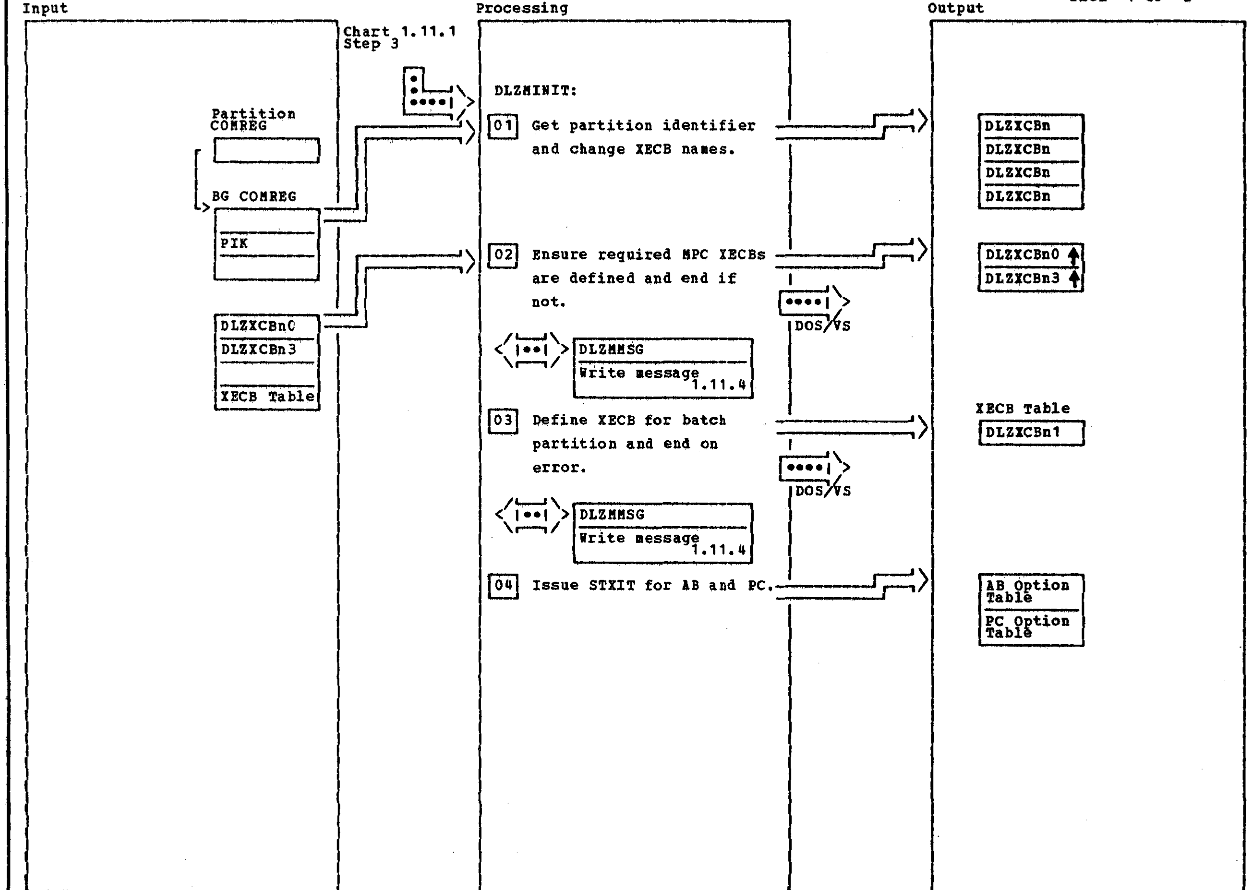
Chart 1.11.1 Step 3

Output

DLZMPI00 - MPS Batch Initialization

HIPOMAT 1.1 Diagram - 1.11.1.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
08 If program was not found, send message DLZ012I and cancel job.							



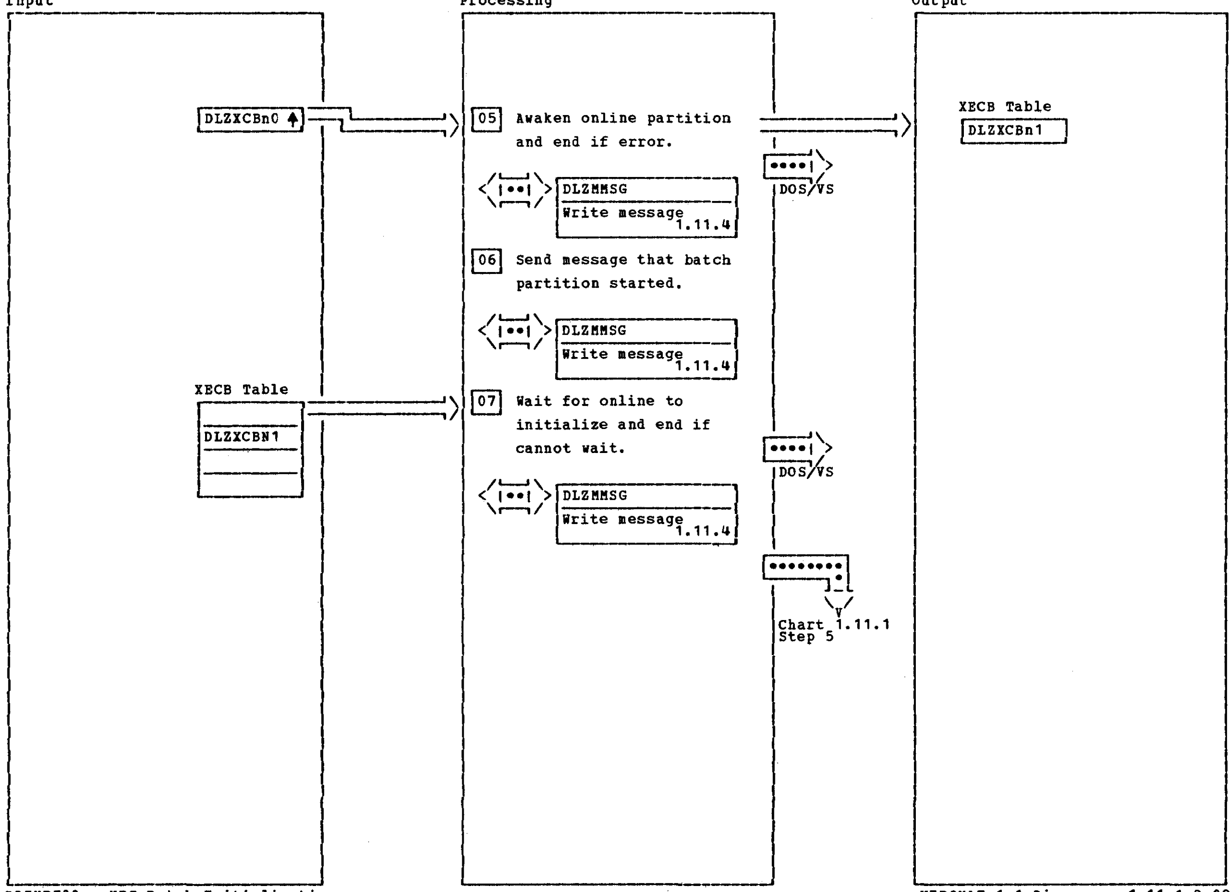
DLZMPI00 - MPS Batch Initialization

HIPOMAT 1.1 Diagram - 1.11.1.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Names of XECBs must be modified based on partition the job is running in.							
02 Use XECBTAB TYPE=CHECK macro for XECBs DLZXCBn0 and DLZXCBn3. If either is not found, message DLZ089I is issued.							
03 Use XECBTAB TYPE=DEFINE macro. If not successful, issue message DLZ082I and end.							

DLZMPI00 - MPS Batch Initialization

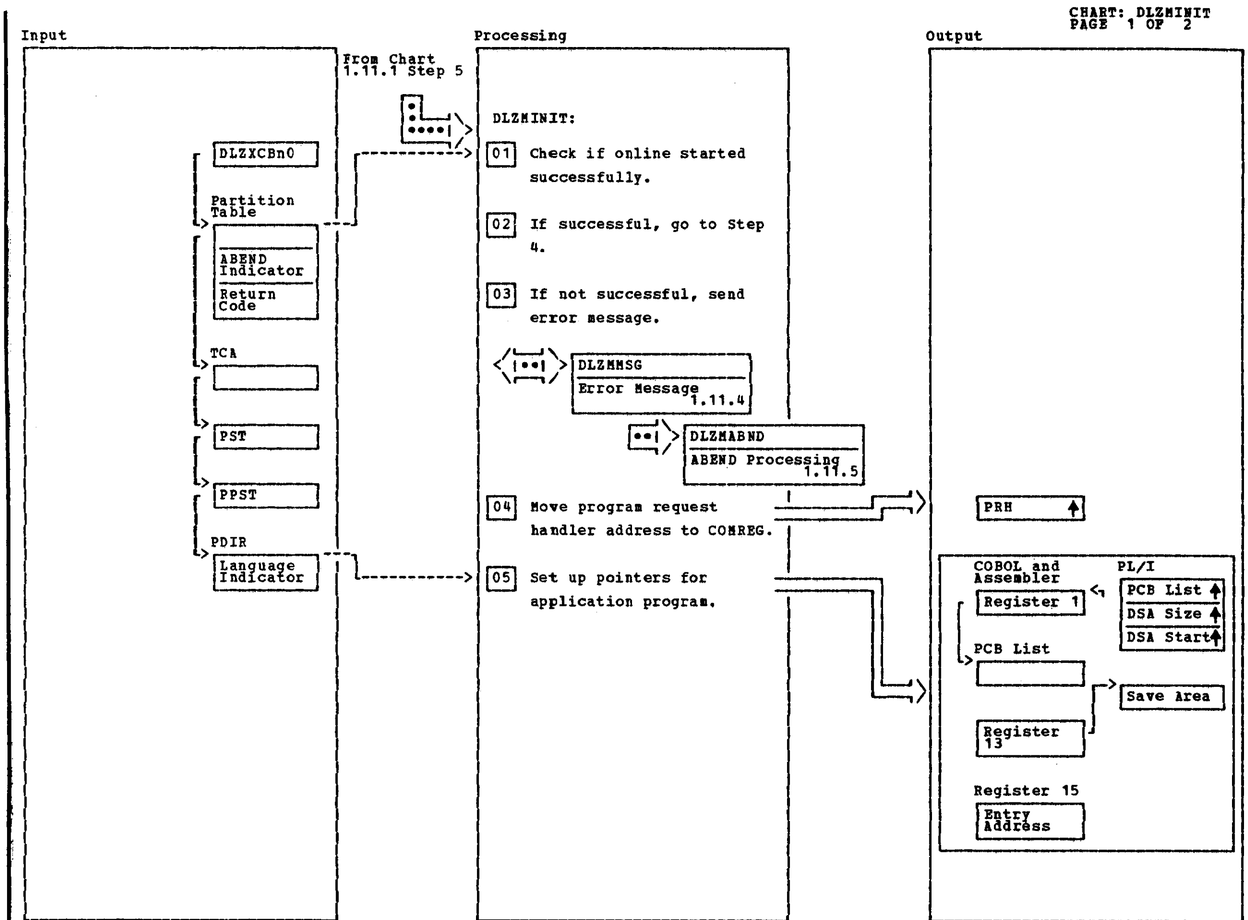
HIPOMAT 1.1 Diagram - 1.11.1.2-01



DLZMPI00 - MPS Batch Initialization

HIPOMAT 1.1 Diagram - 1.11.1.2-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
04 XPOST DLZXCBn0 XECB. If not successful, issue message DLZ084I and end.							
05 Issue message DLZ081I.							
06 When on line initialization completes, XPOST XECB DLZXCBn1.							



DLZMPI00 - MPS Batch Initialization

HIPOMAT 1.1 Diagram - 1.11.1.3-01

Notes	Routine	Label	Ref
03 If XECB DLZXCBn0 is not at the same place as before or no longer exists, issue messages DLZ082I and DLZ090I. If the BPC could not be started, issue message DLZ085I. If there was an error on the scheduling call, issue message DLZ095I.			
05 If PL/I --- a three-word list is set up with pointers to PCB list, amount of dynamic storage, and start of dynamic storage area for PL/I. if ANS COBOL or Assembler --- register 1 points to first PCBADDR.			

Notes	Routine	Label	Ref

DLZMPI00 - MPS Batch Initialization

HIPOMAT 1.1 Diagram - 1.11.1.3-01

Input

Processing

Output

06 Send error message and terminate if online no longer active.

DLZMMSG
Error Message 1.11.4

DLZMABND
ABEND Processing 1.11.5

To User Application Program

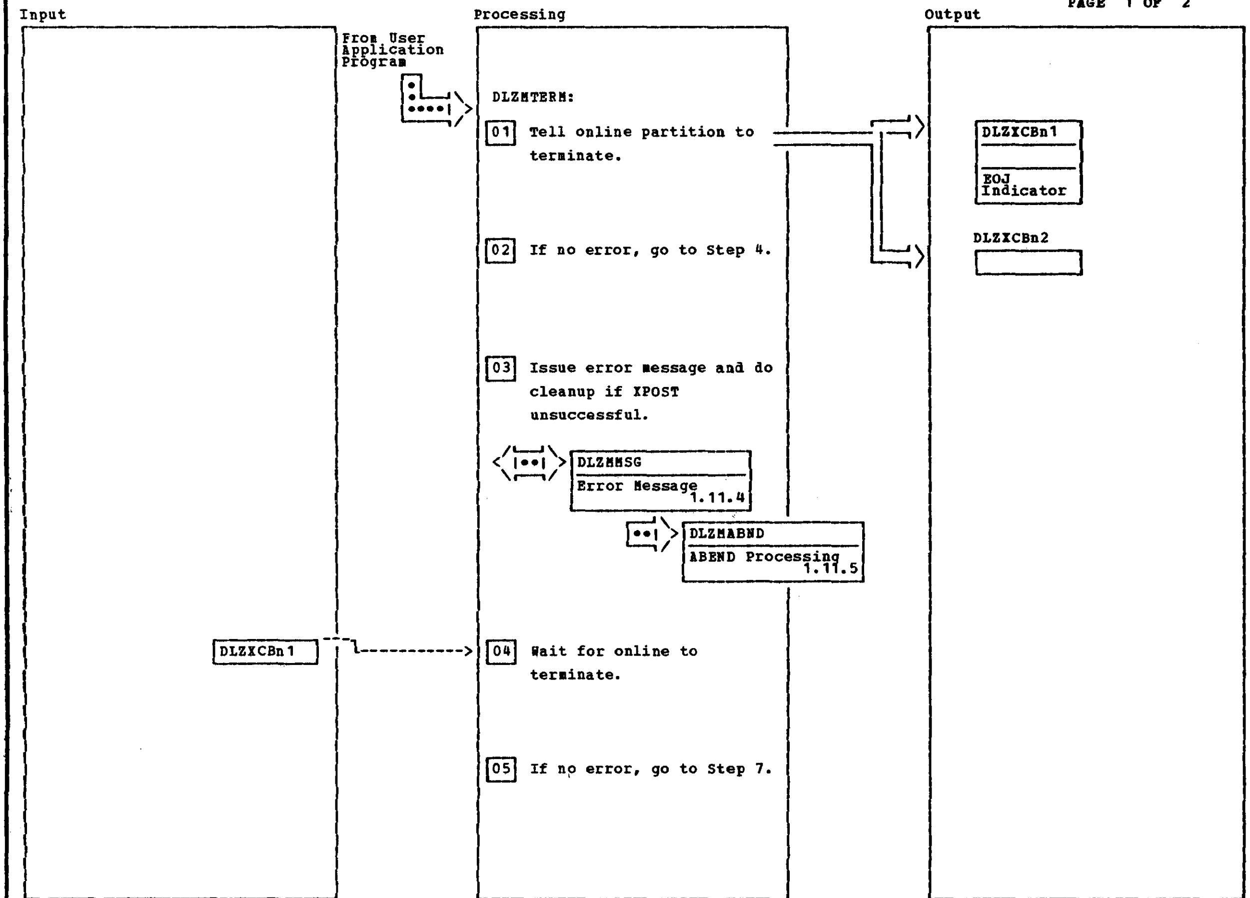
DLZMPI00 - MPS Batch Initialization

HIPOMAT 1.1 Diagram - 1.11.1.3-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
06 Issue message DLZ082I.							

DLZMPI00 - MPS Batch Initialization

HIPOMAT 1.1 Diagram - 1.11.1.3-02



DLZMPI00 - MPS Batch Termination

HIPOMAT 1.1 Diagram - 1.11.2-01

Notes	Routine	Label	Ref
01 The EOJ indicator is in flag byte following XECB DLZXCBn1. If on, XPOST DLZXCBn2.			
03 Issue Message DLZ090I. Dump if UPSI indicates to.			
04 XWAIT DLZXCBn1 until online completes.			

DLZMPI00 - MPS Batch Termination

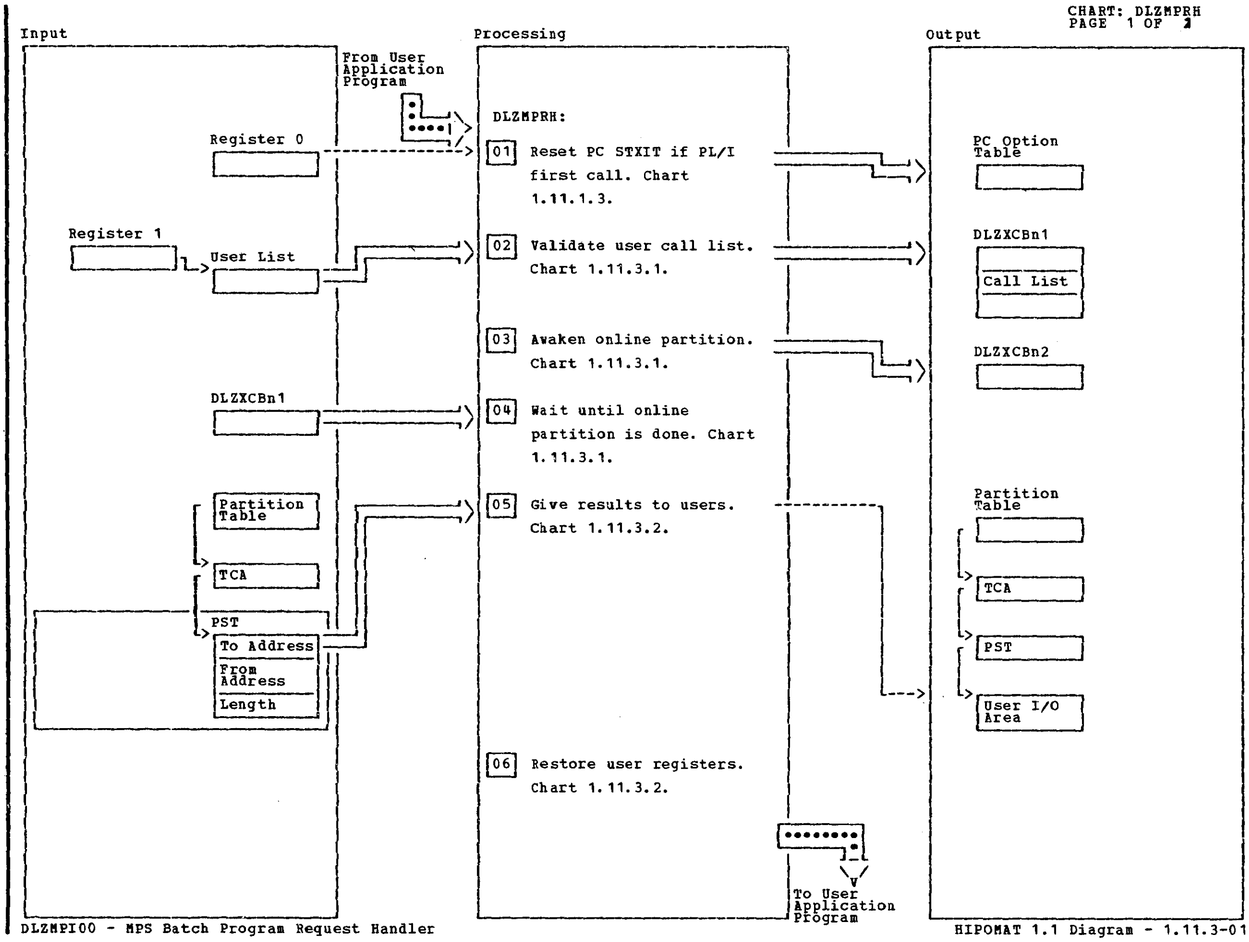
HIPOMAT 1.1 Diagram - 1.11.2-01

Input	Processing	Output
	06 Issue error message and do cleanup if XWAIT unsuccessful.	
	DLZMMSG — Error Message 1.11.4	
	DLZMABND — ABEND Processing 1.11.5	
	07 Delete batch XECB.	DLZXCBn1
	08 Notify online that batch is done.	DLZXCBn2
	To DOS/VS	

DLZMPI00 - MPS Batch Termination

HIPOMAT 1.1 Diagram - 1.11.2-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
06 Issue message DLZ090I. Dump if UPSI indicates to.							
08 XPOST DLZXCBn2.							



Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
02 Ensure that there are no more than 18 parameters and that all which should be are within the batch partition. Move list to follow DLZXCBn1 so that online can find it.							
03 XPOST DLZXCBn2.							
04 XWAIT on DLZXCBn1. Online will post it.							

Input

Register 0

Register 1

User List

From Chart 1.11.3 Step 1

A

Processing

DLZMPRH:

01 Reset PC STXIT if first call from PL/I.

02 Check if all user call list addresses which should be, are within batch partition.

03 If no error, go to step 5.

04 Issue error message DLZ092I and end.

DLZMMSG
Error Message 1.11.4

DLZMABND
ABEND Processing 1.11.5

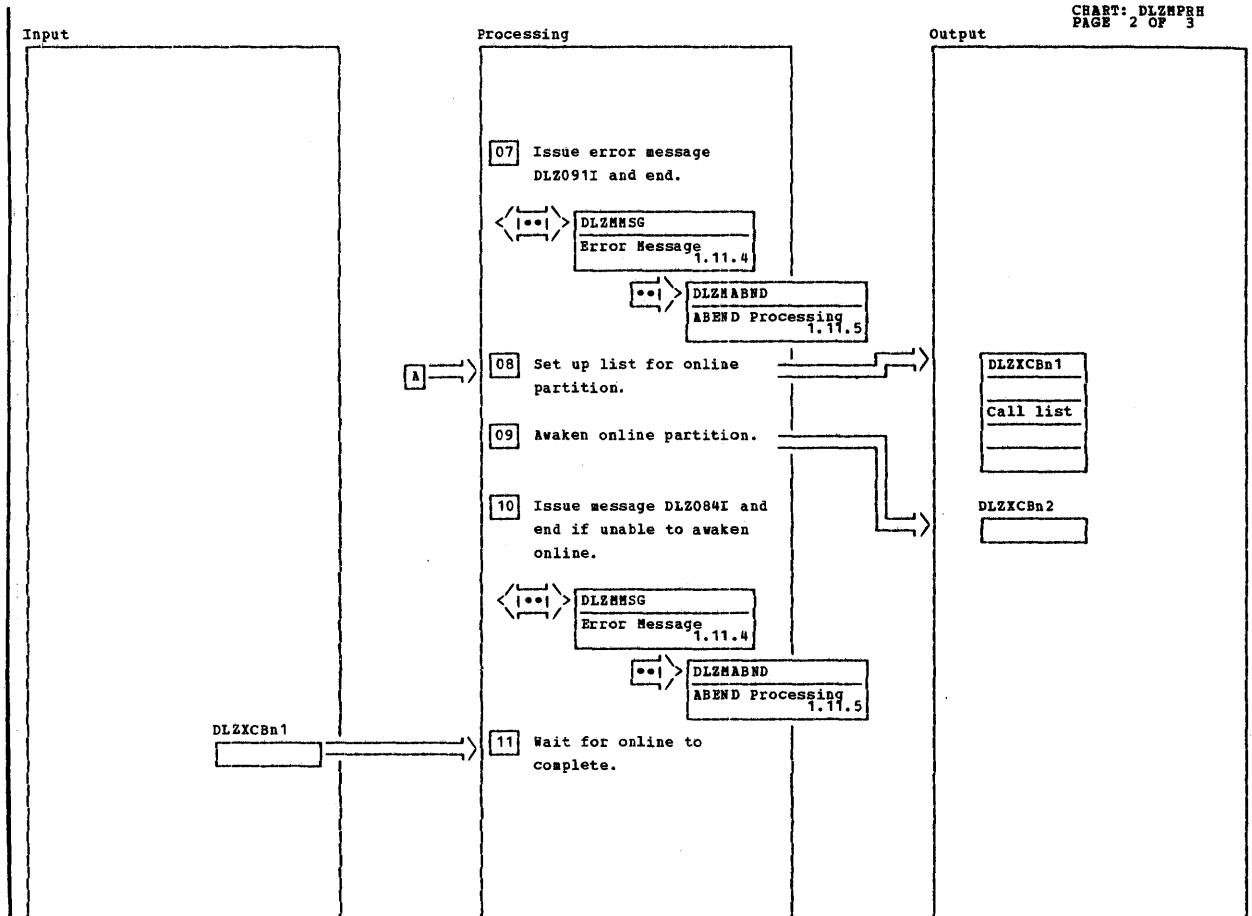
05 Check list to ensure it does not contain more than 18 parameters.

06 If no error, go to step 8.

Output

PC Option Table

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Language is PL/I if register 0 has 1. Language is COBOL or Assembler if register 0 has 0. PL/I reissues STXIT PC when application program starts, therefore, DL/I must reissue STXIT to get control after PL/I issues its STXIT PC.							
02 Ensure that call list and addresses it points to are within batch partition. If PL/I, ensure that pointers pointed to by pointers, are within the batch partition.							



DLZMPI00 - MPS Batch Program Request Handler   HIPOMAT 1.1 Diagram - 1.11.3.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
08 List contains count field and up to 18 parameters in area behind XECB DLZXCBn1.							
09 XPOST XECB DLZXCBn2.							
11 Online partition will XPOST DLZXCBn1 when online processing complete.							

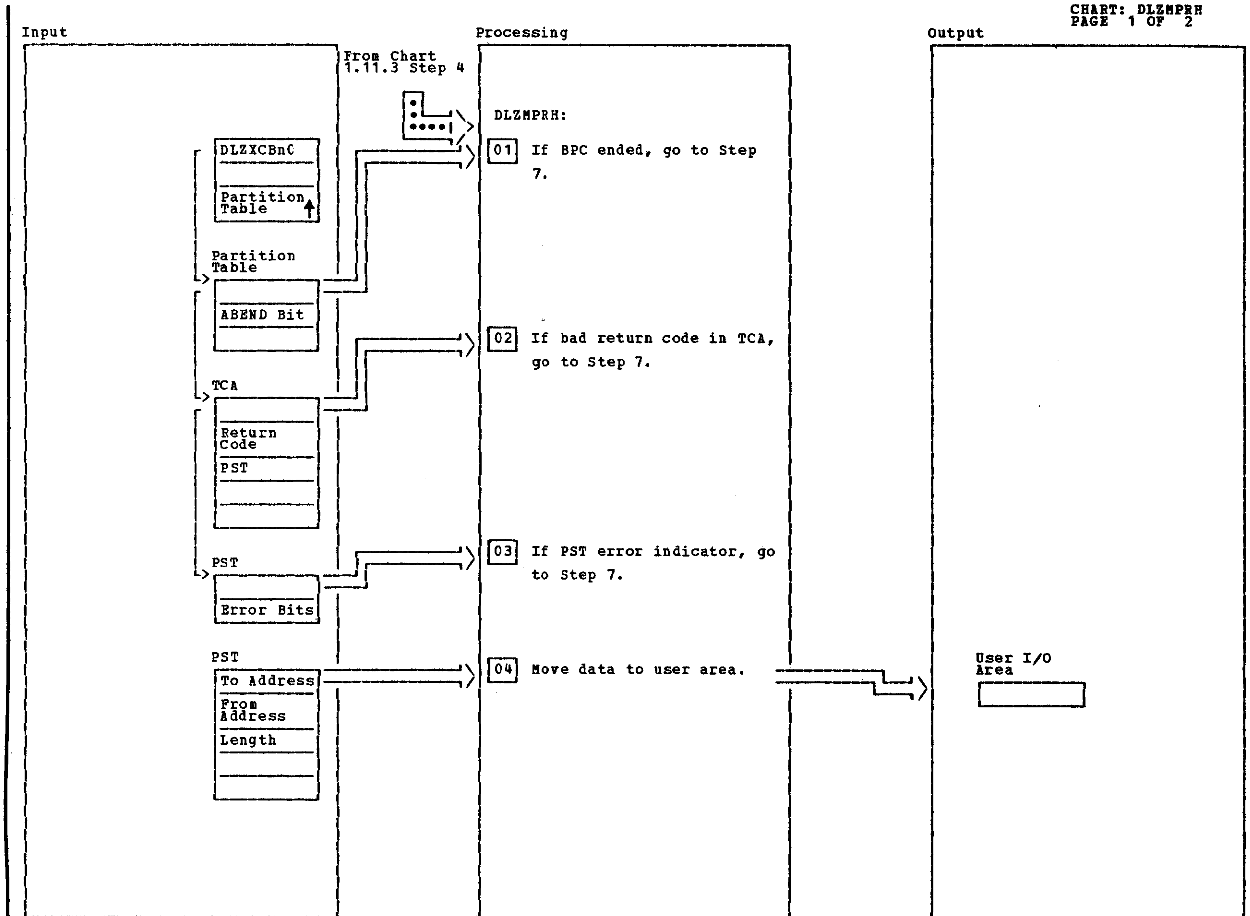
DLZMPI00 - MPS Batch Program Request Handler   HIPOMAT 1.1 Diagram - 1.11.3.1-02

Input	Processing	Output
	12 Issue message DLZ084I and end if wait is unsuccessful. DLZMMSG Error Message 1.11.4 To Chart 1.11.3 Step 5	

DLZMPI00 - MPS Batch Program Request Handler

HIPOMAT 1.1 Diagram - 1.11.3.1-03

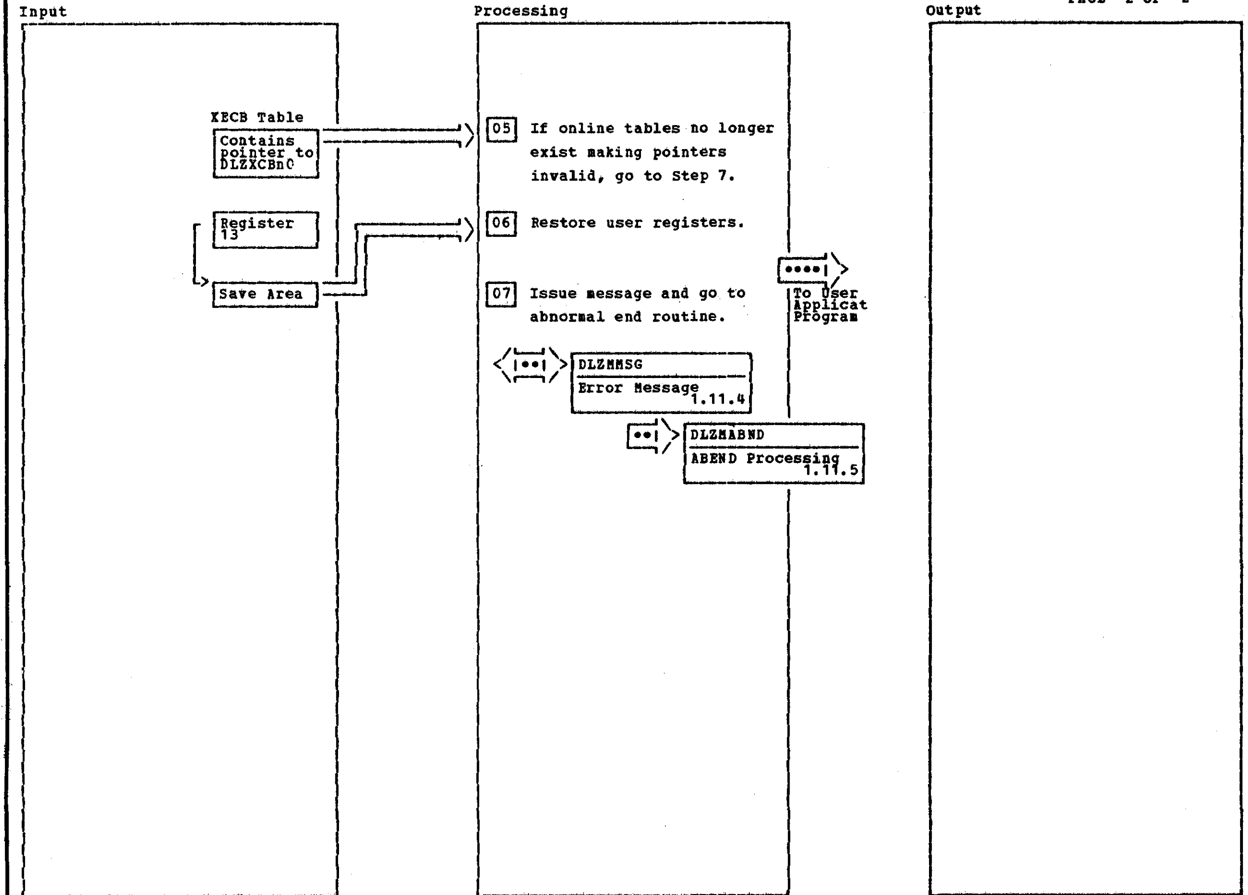
Notes	Routine	Label	Ref	Notes	Routine	Label	Ref



DLZMPI00 - MPS Batch Program Request Handler
HIPOMAT 1.1 Diagram - 1.11.3.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Message DLZ100I is issued if BPC ended.							
02 Message DLZ102I is issued if return code in TCA is invalid.							
03 Message DLZ098I is issued if PST contains an error.							

DLZMPI00 - MPS Batch Program Request Handler
HIPOMAT 1.1 Diagram - 1.11.3.2-01



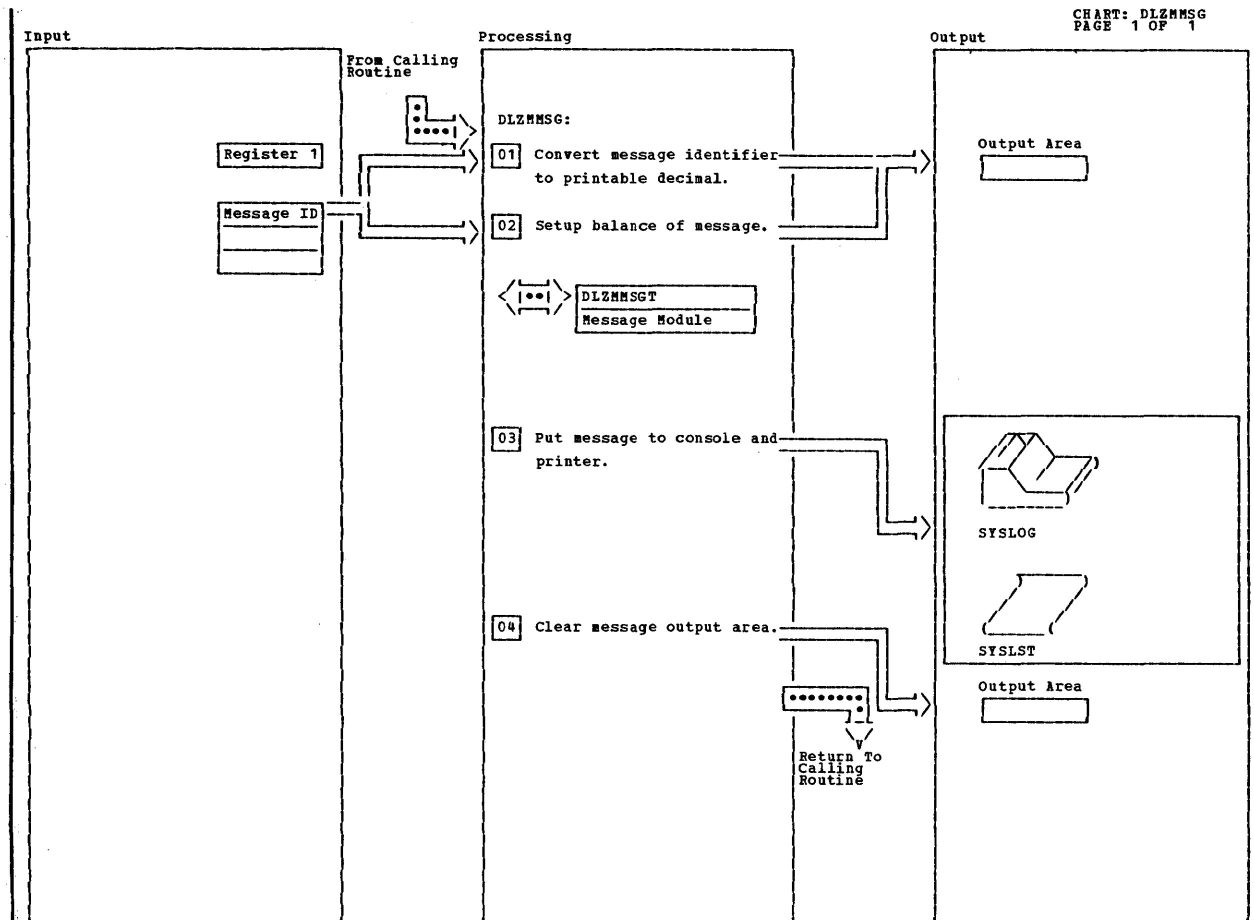
DLZMPI00 - MPS Batch Program Request Handler

HIPOMAT 1.1 Diagram - 1.11.3.2-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
05 Message DLZ082I is issued if DLZXCBn0 no longer exists or not where expected. XECBTAB TYPE=CHECK							

DLZMPI00 - MPS Batch Program Request Handler

HIPOMAT 1.1 Diagram - 1.11.3.2-02



DLZMPI00 - MPS Batch Message Writer

HIPOMAT 1.1 Diagram - 1.11.4-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
02 The online message module, DLZMMSGT, now includes all messages that can be issued by MPS and is used in both the batch and online partitions.							
03 All messages are written to the system operator and to the printer.							

DLZMPI00 - MPS Batch Message Writer

HIPOMAT 1.1 Diagram - 1.11.4-01

Input

From STXIT AB Routine

Register 0

From STXIT PC Routine

From XPOST Entry

Processing

DLZMABND:

01 Save ABEND code.

02 Set AB entry indicator.

03 Go to step 5.

04 Set PC entry indicator.

05 Send error message DLZ096I.

DLZMMSG
Error Message 1.11.4

06 Notify the online partition not to wait for batch because batch is ending.

Output

AB Output Area

Flag Byte

Flag Byte

DLZXCBn3

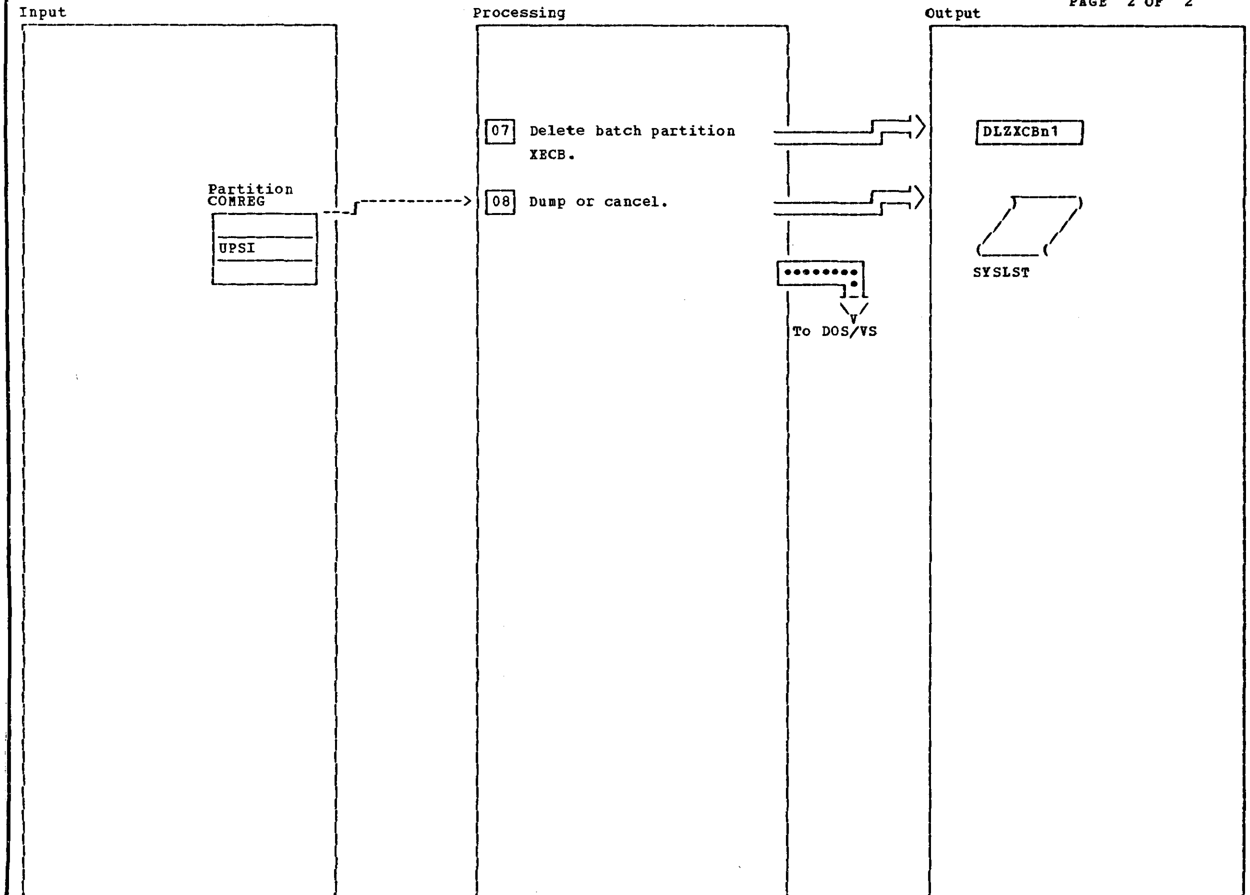
DLZMPI00 - MPS Batch ABEND Processing

HIPOMAT 1.1 Diagram - 1.11.5-01

Notes	Routine	Label	Ref
There are three entries to this routine:			
1. AB STXIT			
2. PC STXIT			
3. The MPS Batch Initialization, MPS Batch Termination, and MPS Program Request Handler routines - whenever XPOST is needed to tell online that batch completed unsuccessfully.			
01 The AB output area is located in storage following the DC C 'AB SAVE' indicator.			
The ABEND code is located in storage following the DC C 'AB REASON CODE ' indicator.			
04 The PC output area is located in storage following the DC C 'PC SAVE ' indicator.			
06 XPOST online XECB DLZXCBn3.			

DLZMPI00 - MPS Batch ABEND Processing

HIPOMAT 1.1 Diagram - 1.11.5-01

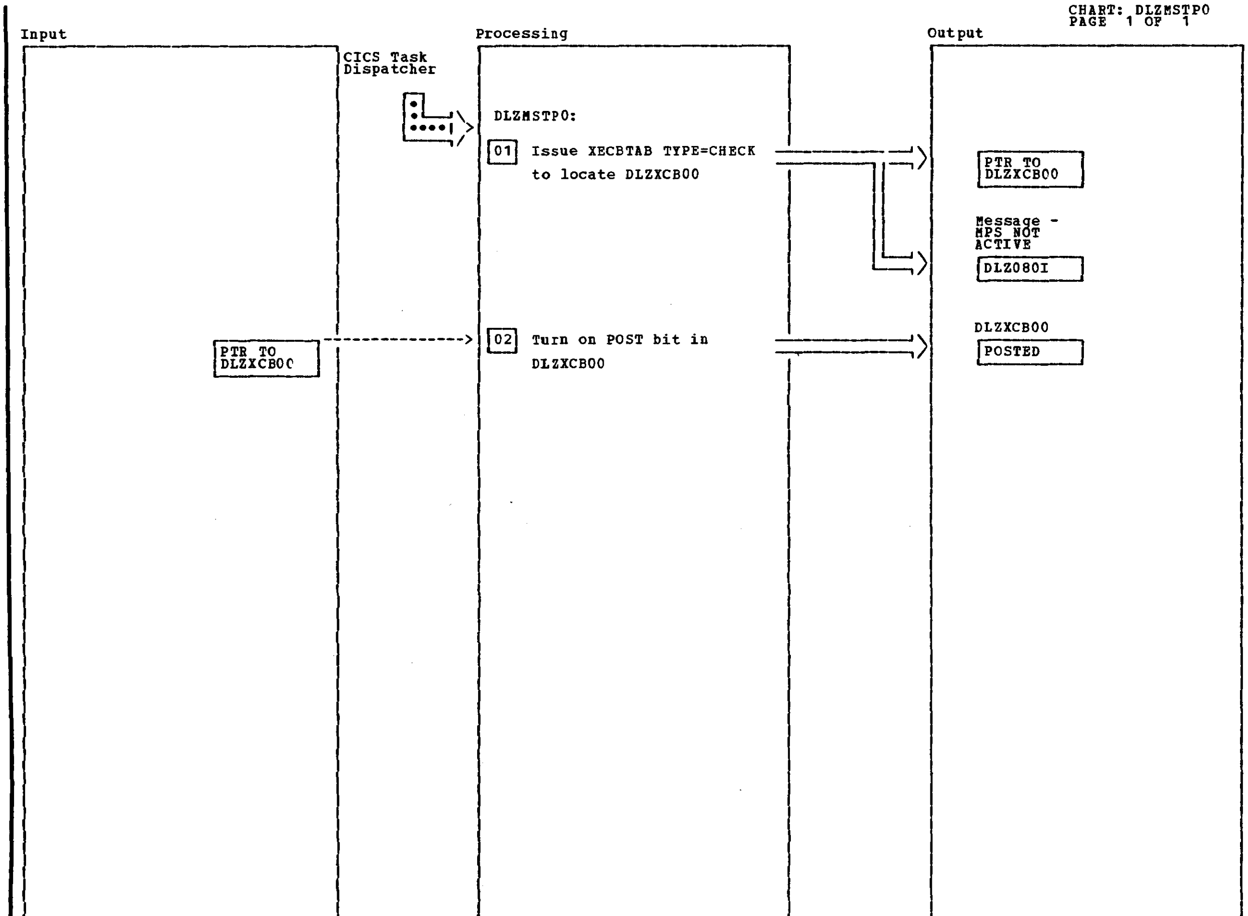


DLZMPI00 - MPS Batch ABEND Processing

HIPOMAT 1.1 Diagram - 1.11.5-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
07 XECBTAB TYPE=DELETE for XECB DLZXCBn1.							
08 JDUMP if UPSI bit 5=0, otherwise CANCEL.							

CHART: DLZMSTPO
PAGE 1 OF 1



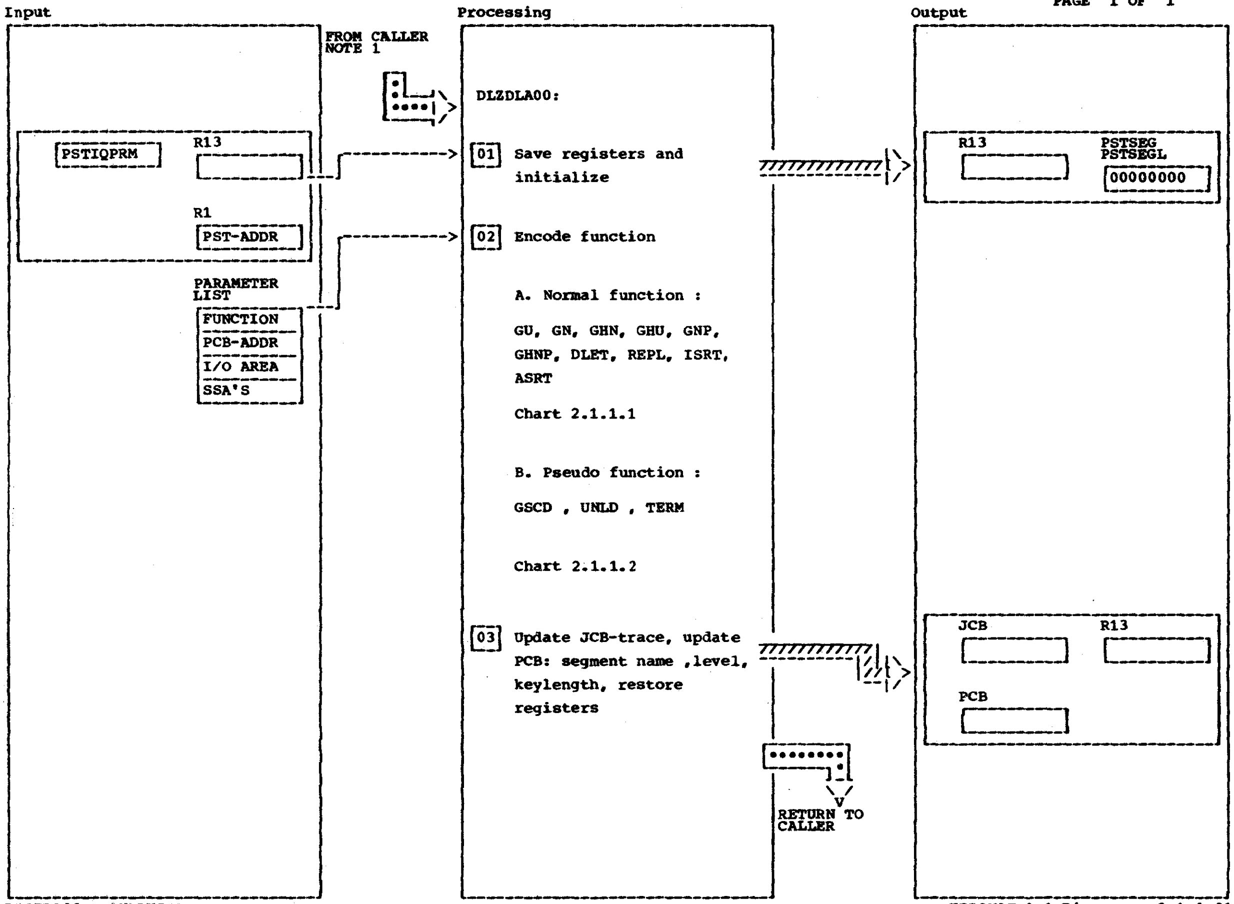
DLZMSTPO - Stop MPS Transaction

HIPOMAT 1.1 Diagram - 1.12.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Message issued if DLZXCB00 does not exist							
02 Performed only if message wasn't written in Step 1.							
NOTE: User should include DLZMSTPO in CICS Transaction List Table (XLT)							

DLZMSTPO - Stop MPS Transaction

HIPOMAT 1.1 Diagram - 1.12.1-01



DLZDLA00 - ANALYZER   HIPOMAT 1.1 Diagram - 2.1.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 DLZDLA00 is called from program request handler - DLZBPR00 - in a batch system. From - DLZODP00 - in an online system. At termination it is called from either the application program control - DLZPCC00 - or from online task termination - DLZODP01. It is also called from DLZDXMT0.							
02 The function, - first parameter in list - is encoded. If no valid function function is found 'AD' status code is returned.							

DLZDLA00 - ANALYZER   HIPOMAT 1.1 Diagram - 2.1.1-01

Input

PST

SDB

FDB

PSB

FROM DLZDLA00
CHART 2.1.1
STEP 2A

Processing

01 Validate PCB address, update JCB and level table

02 Find user's I/O area

03 Validate SSA's

Chart 2.1.1.11

04 Perform key checking for loading

05 Validate sensitivity

06 Check length for variable length segments

07 Call DLZDLOC0 to open data sets

08 Call proper action module - DLZDLR00 - DLZDDLE0 - DLZDLD00

TO DLZDLA00
CHART 2.1.1
STEP3

Output

PCB

PST

JCB

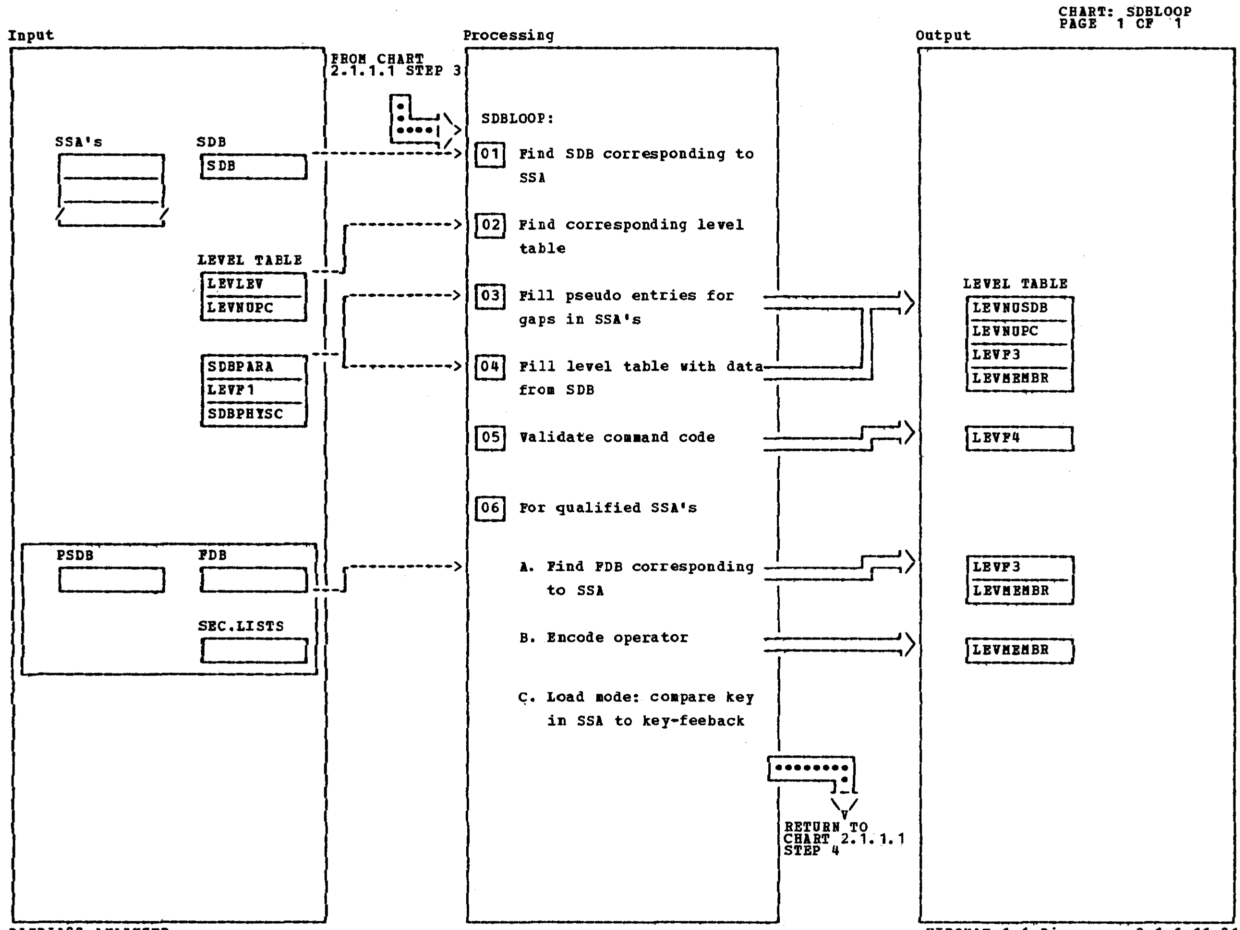
LEVTAB

DLZDLA00 - ANALYZER

HIPOMAT 1.1 Diagram - 2.1.1.1-01

Notes	Routine	Label	Ref
01 It no valid PCB address is provided, abend code '476' is returned. The JCB and PCB is updated and the second part of the level tables cleared.		TESTPCB VALIDCK2 DBPCBFND GETJCB	
02 If no I/O area is provided 'AB' status code is returned.			
03 All SSA's in the call are checked.		SDBLOOP SDBLOOP1	
04 Key checking is done for load mode, for the last SSA of an ISRT call. For PROCOPT=LS and for HISAM the root key is compared to the previously loaded root. Status code 'LB' indicates invalid sequence.		LDCHCK	
05 Sensitivity checking is done for ISRT - DLET - REPL calls. Violations return 'AM'. Extra checking is done for DLET - REPL calls, if successfull GH call was executed before -'DJ' status.		NOTLOAD7 FSTDATAL ISREPL TSTISRTS	

Notes	Routine	Label	Ref
06 For variable length segments the 2-byte length field in the user I/O area is compared to the maximum length and to the key+keyoffset. If it is greater or smaller 'V1' status is returned.		DOVLTST	
07 When the data base the PCB references is not open, DLZDLOC0 is called to open all data bases related to this PCB.		ANYSEN	
08 For get calls DLZDLR00 is called. For DLET/REPL calls DLZDLD00 is called. For ISRT/ASRT calls in load mode: DLZDDLE0 is called for all segments, except for HDAM root - where DLZDLR00 is called. for ISRT not load mode: DLZDLR00 is called for all segments, exept HISAM root - where DLZDDLE0 is called.		ACTION	



DLZDLA00 ANALYZER

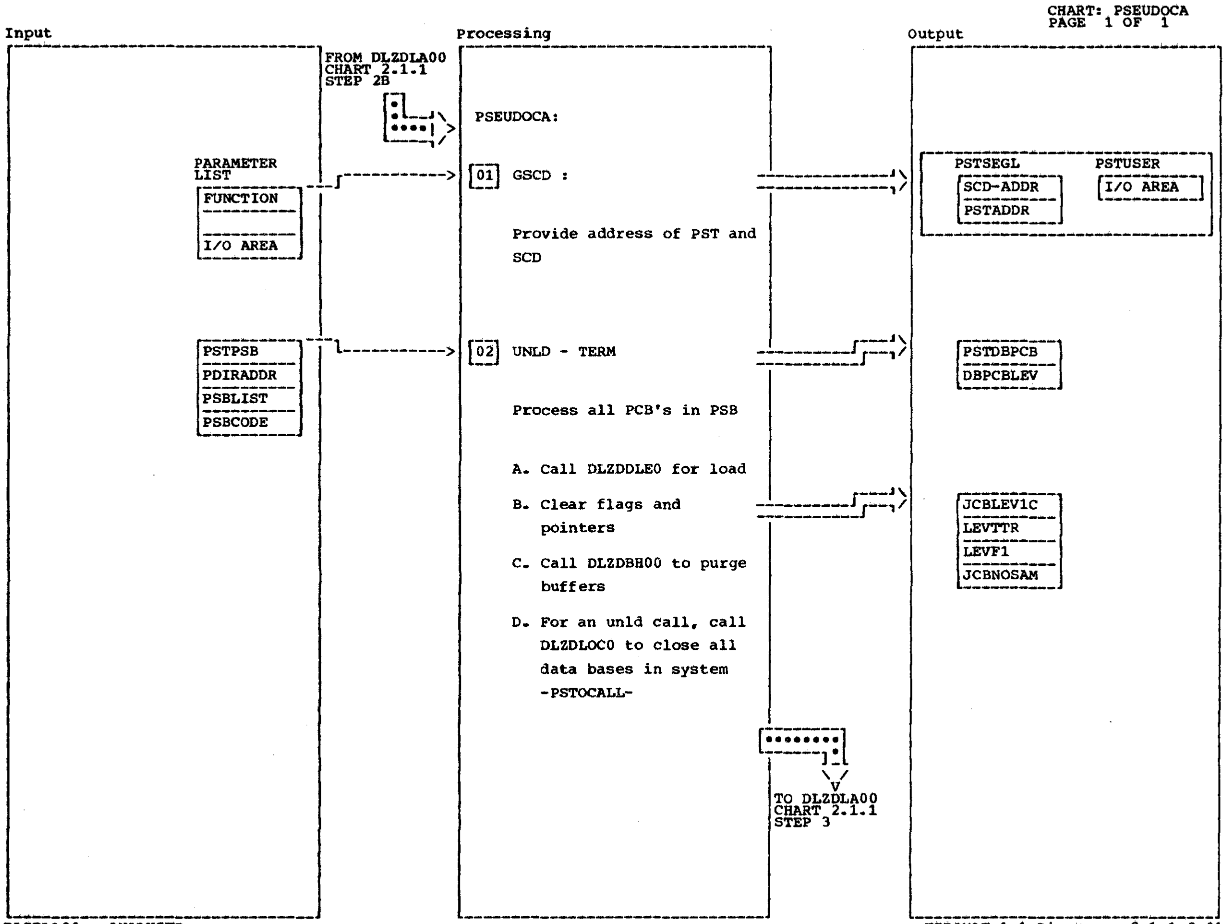
HIPOMAT 1.1 Diagram - 2.1.1.11-01

Notes	Routine	Label	Ref
01 When the segment name, specified in the SSA cannot be found in the SDB's 'AC' status is returned.		SDBLOOP	
02 When an hierarchy error is detected an 'AC' status code is returned.		GETLEV RIGHTLEV	
03 The levels corresponding to gaps in the SSA's are filled with data from the previous call. For loading no parent level may be empty, 'LD' status is returned.		PSEUDLOP	
04 Extra checks are made for DLET and REPL calls. When no GH call was made for this SDB previously a 'DJ' status is returned.			
05 Valid command codes are: C - D - F - L - N - T - X . Status code for invalid command code is 'AJ'. For D - call and no path sensitivity 'AM'.		NOTDORR	

Notes	Routine	Label	Ref
06 For errors in qualification format 'AJ' status is returned.			
A. Valid field names are: any normal field of the segment, the XDFLD-name, if the secondary processing sequence is used. For a concatenated segment field names of the logical child and of the destination parent are valid. 'AK' status for invalid field name, 'AC' if /CK or /SX is used.		NXTBOOL FDBEQUAL	
B. Invalid operator returns status code 'AJ'.		CODES ROHIT	
C. If qualified SSA's are specified for loading, the key has to correspond to the key-feed-back area, otherwise 'LD' status code.		NXTSEC11 R11ISOK	

DLZDLA00 ANALYZER

HIPOMAT 1.1 Diagram - 2.1.1.11-01

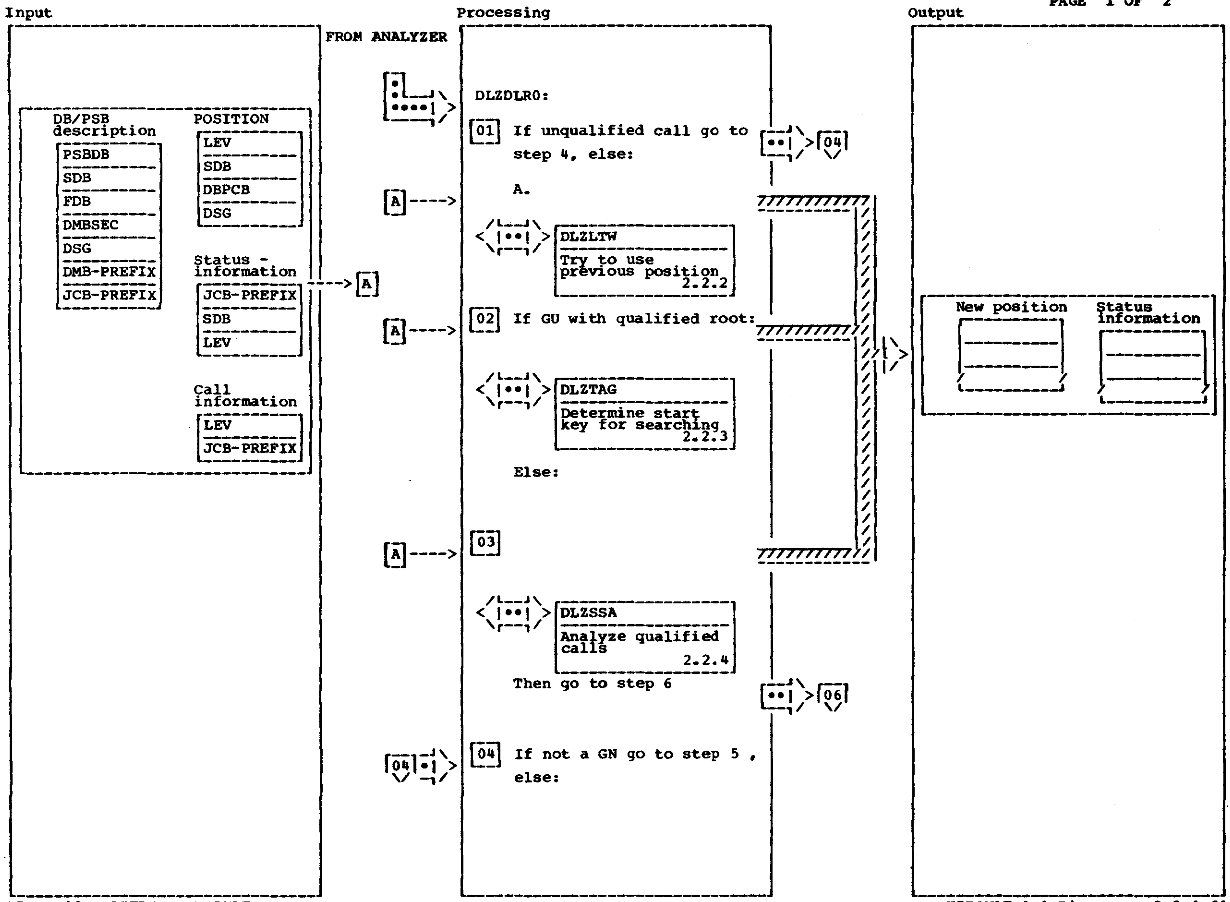


DLZDLA00 - ANALYZER HIPOMAT 1.1 Diagram - 2.1.1.2-01

Notes	Routine	Label	Ref
01 Input to the GSCD call is function and I/O area address. DLZDLA00 puts the SCD and PST address in PSTSEGL. Program request handler moves it to I/O area.		DLBGSCD	
02 The TERM call is issued in online to end a task, the UNLD call is used in batch to end the batch program.		DLBUNLD UNLDLOOP WASHLOAD NXTPCB	
A. If the UNLD call is made for load mode, DLZDDLE0 is called to write the last records for HSAM and HISAM. For HISAM and index data bases a record with FF-key's is written.			
B. Flags and pointers are cleared, so that the PSB can be used by another task.			
C. All buffer's of this user are written to the data base now. PSTFNCTN is - PSTPGUSR -, PSTBLKNM, DMBNM, ACBNM are cleared.			

Notes	Routine	Label	Ref

CHART: DLZDLR0
PAGE 1 OF 2



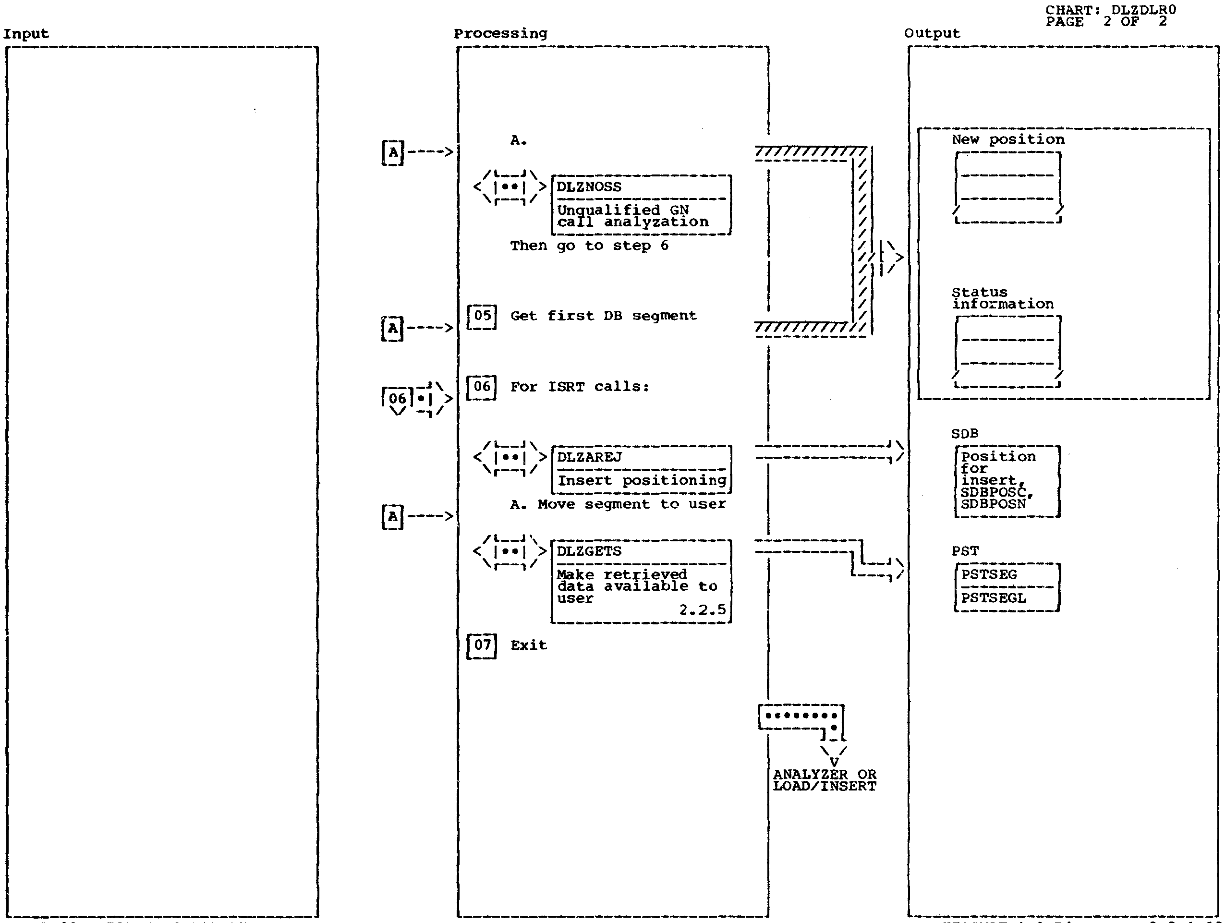
DLZDLR00 - RETRIEVE MODULE — HIPOMAT 1.1 Diagram - 2.2.1-01

Notes	Routine	Label	Ref
01 I/O - information :			
A. The position - block includes RBA of segment(HD) or lrec(HS), RBA of previous and next positions (HD), offset to segment from begin lrec (HS), concatenated key, level, block-no.(HSAM),block-no. and RAP-no. of current RAP (HDAM). RAP = root anchor point.			
B. The DB/PSB-description - block includes segment and data set descriptions, data base specifications, sensitivity and HDAM randomizing facility.			
C. The status-information - block includes prior status codes, segment status and -for output - pseudo abends (801 & 800)			
D. The call-information - block includes SSA and call-type.			

Notes	Routine	Label	Ref
E. Processing starts with initialization. Level of previous call stored in LASTLEV.			
Special conditions:			
No valid level from previous call. Qualified calls: DLZTAG, step (02) A.		XLTFINDR	
Unqual. calls: either next root, then DLZNOSS, step 4 A		MTNOSSA	
or step (05).		NOTEOD	
GNP calls: special processing, then enter main line:		HAVGNP	
Qualified calls step (01) A		LTWSSA	
Unqualified calls step (04)		NOSSA	
Start of general processing		GUGNGNP	

DLZDLR00 - RETRIEVE MODULE — HIPOMAT 1.1 Diagram - 2.2.1-01

CHART: DLZDLR0
PAGE 2 OF 2

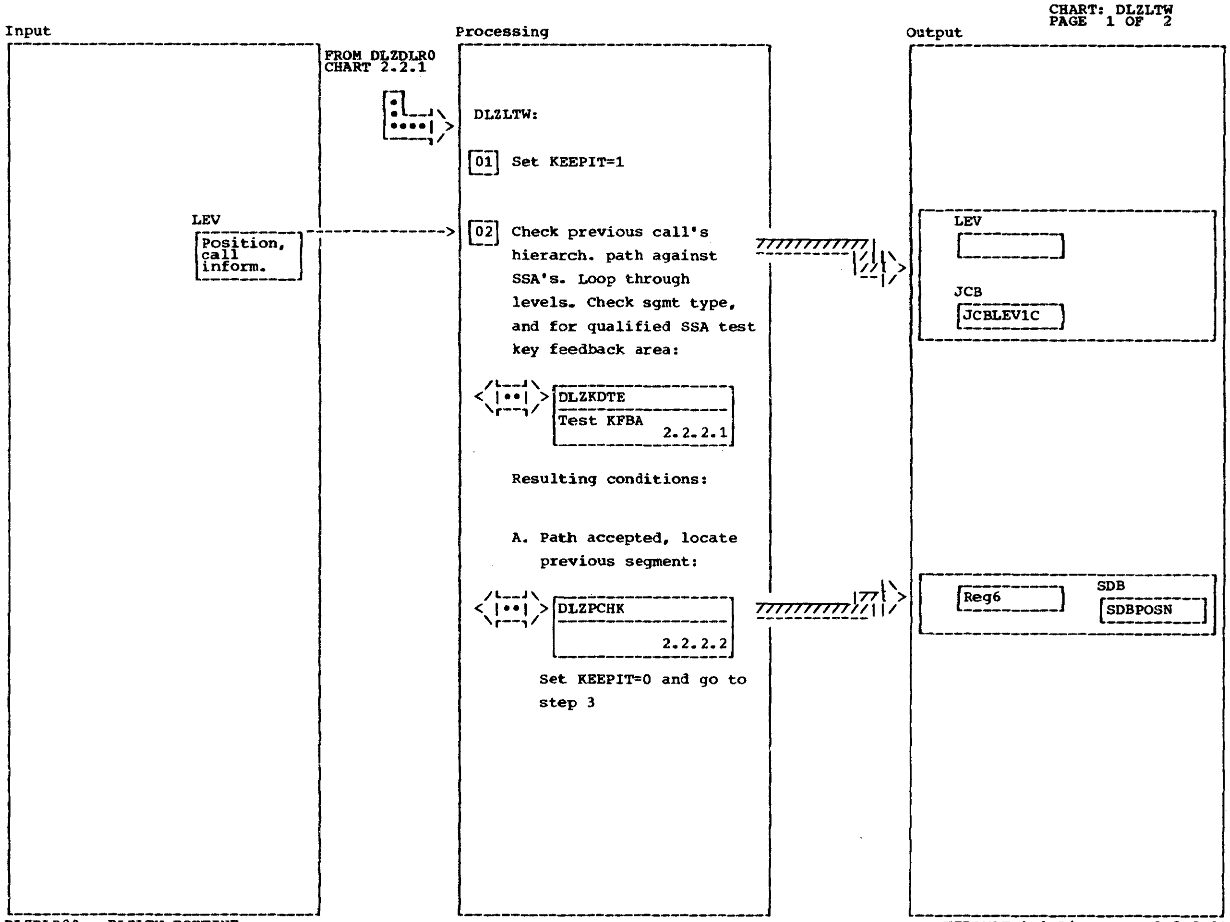


DLZDLR00 - RETRIEVE MODULE HIPOMAT 1.1 Diagram - 2.2.1-02

Notes	Routine	Label	Ref
06 DLZAREJ calls DLZSSA, if necessary, to find insert position for key. Control is then passed to DLZISRT for insert positioning. Return is to DLZDLR0.		RETURNI	
A. PSTSEG is address of data, PSTSEGL gives its length.			
For ISRT calls, DLZGETS does only housekeeping, no data moving. DLZGETS will pass control to DLZRETN and DLZDLR1 to exit.		ARETURNA	
For a segment with logical relationship DLZGETS will call DLZLOGR, ID(2.2.5.1), for data move /insert positioning.			
07 If call-type = GET go to ANALYZER and if call-type = ISRT go to LOAD/INSERT.			

Notes	Routine	Label	Ref

CHART: DLZLTW
PAGE 1 OF 2



DLZDLR00 - DLZLTW ROUTINE

HIPOMAT 1.1 Diagram - 2.2.2-01

Notes	Routine	Label	Ref
01 KEEPIT=1 means: try to use previous position. KEEPIT=0 means: DLZLTW has been left. Other values have special meanings.			
02 DLZKDTE is invoked via DLZSSA which is called by return to DLZDLR0 and back to DLZLTW. Logically, this is part of DLZLTW as indicated by KEEPIT=1.			
Qualified SSA test: after entering several routines return to DLZLTW, entries LTWSSACA, LTWSSAF, or LTWSSAG		LTWSSAQ	
Lowest level found valid is stored in JCBLEV1C		LTWSSACA	
A. Set code for exit: entry UNQLA in DLZSSA for GU or ISRT, entry SSAEVALH for GN			
DLZPCHK loads buffer location of previous segment into Reg6 (exception: HD, GN call) and -for HD- loads available SDBPOSN positions.			

Notes	Routine	Label	Ref

DLZDLR00 - DLZLTW ROUTINE

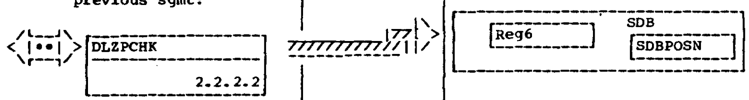
HIPOMAT 1.1 Diagram - 2.2.2-01

Input | Processing | Output

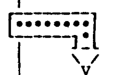
B. Discrepancy at root level: set KEEPIT=0 and go to step 3

C. Discrepancy at lower level: set KEEPIT=0

If GN call, position to previous sgmt:



03 Exit



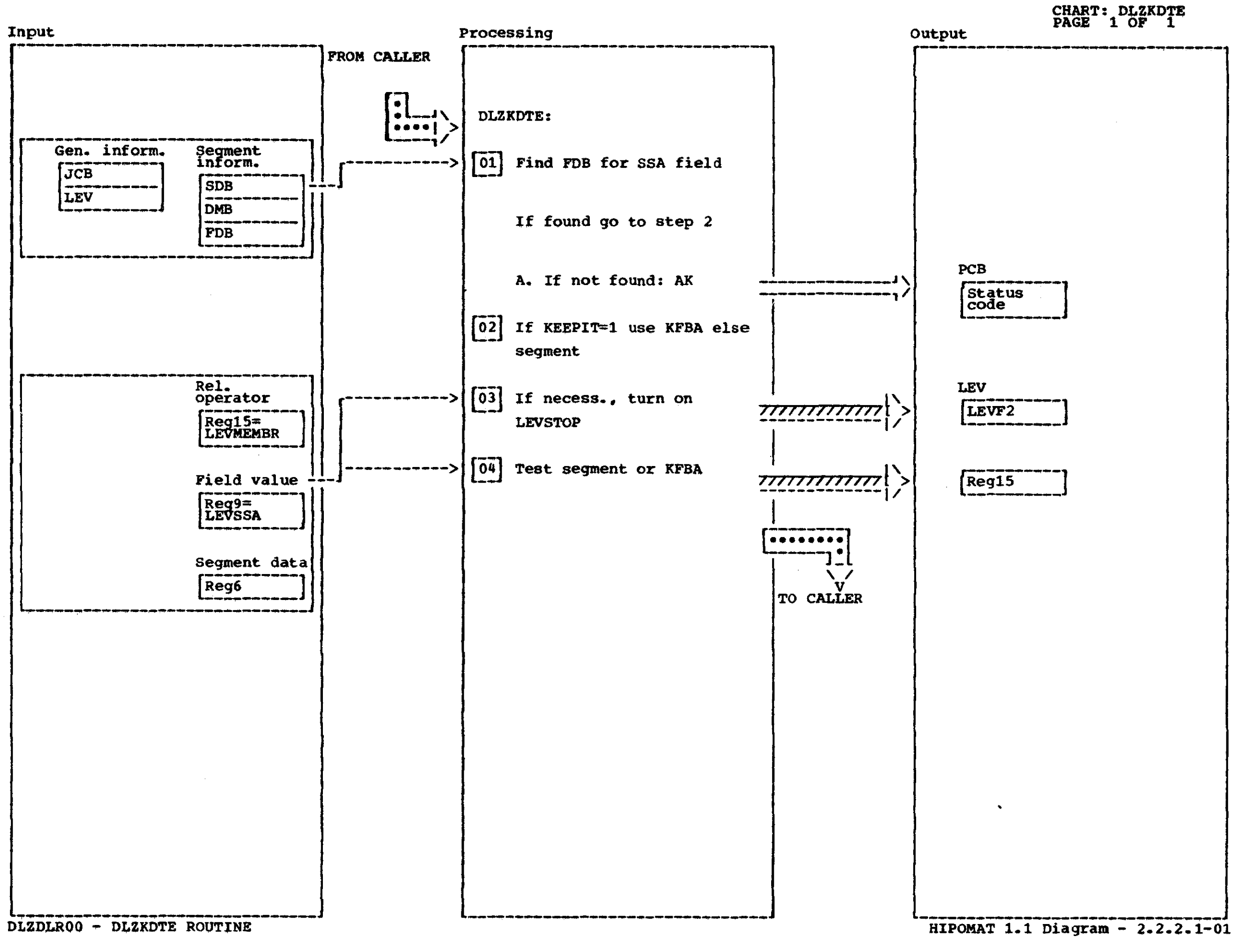
DLZDLR00 - DLZLTW ROUTINE

HIPOMAT 1.1 Diagram - 2.2.2-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
B. Set code for exit to entry MTWISSA in DLZTAG: get new root by key.							
C. Set exit code for entry SSAEVALL in DLZSSA.							

DLZDLR00 - DLZLTW ROUTINE

HIPOMAT 1.1 Diagram - 2.2.2-02

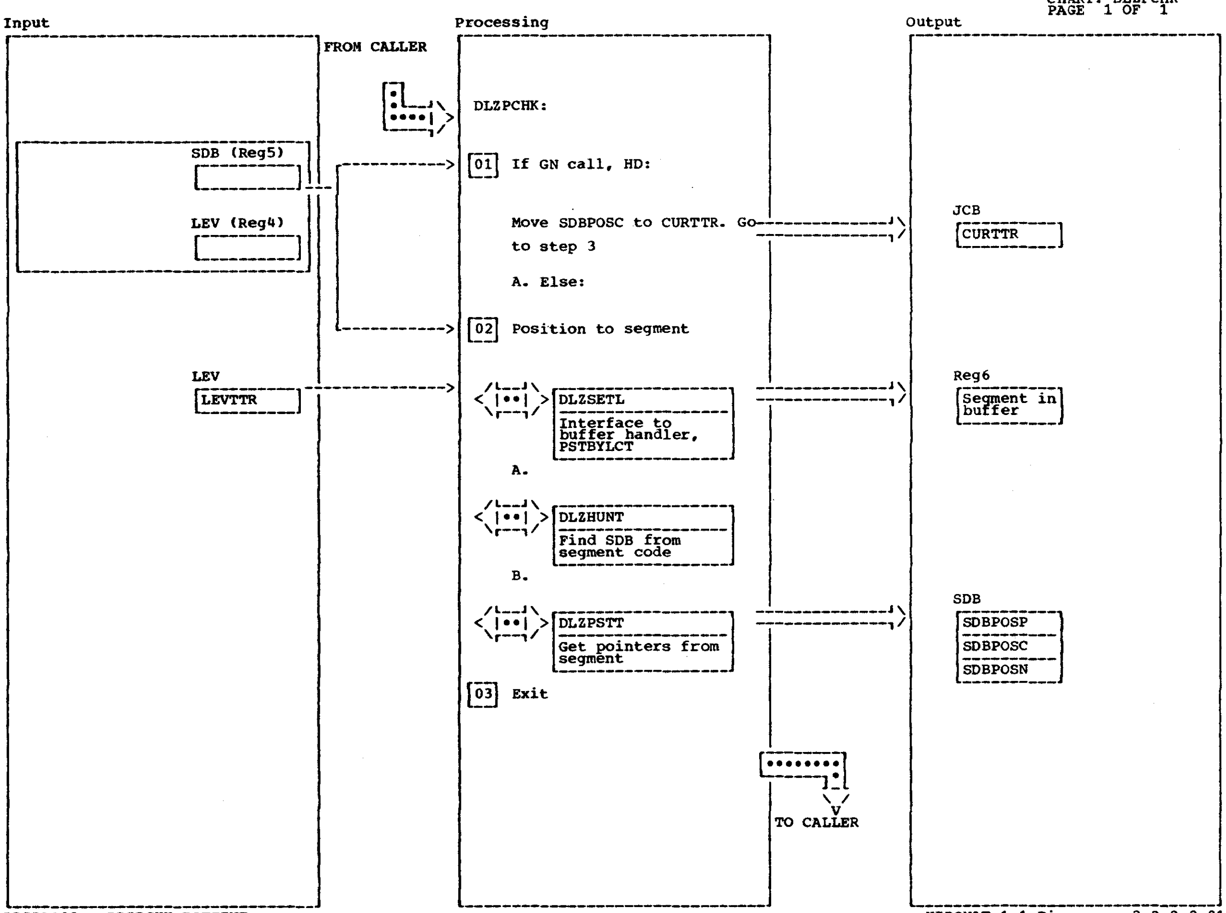


Notes	Routine	Label	Ref
01		KDTESTI	
A.		KDTESTC	
02 Test either segment or key feedback area against field value in SSA.		KDTESTBR	
If log. relationship, build concatenated segment.	DLZKDTL		
If variable length, build data.	DLZVLRT		
03 If qualification is on key, rel. op. is > or =, and key <= SSA.		KDTESTHA	
04 If accepted, R15=0, else R15=4.		KDTESTE	

DLZDLR00 - DLZKDTE ROUTINE

HIPOMAT 1.1 Diagram - 2.2.2.1-01

CHART: DLZPCHK
PAGE 1 OF 1



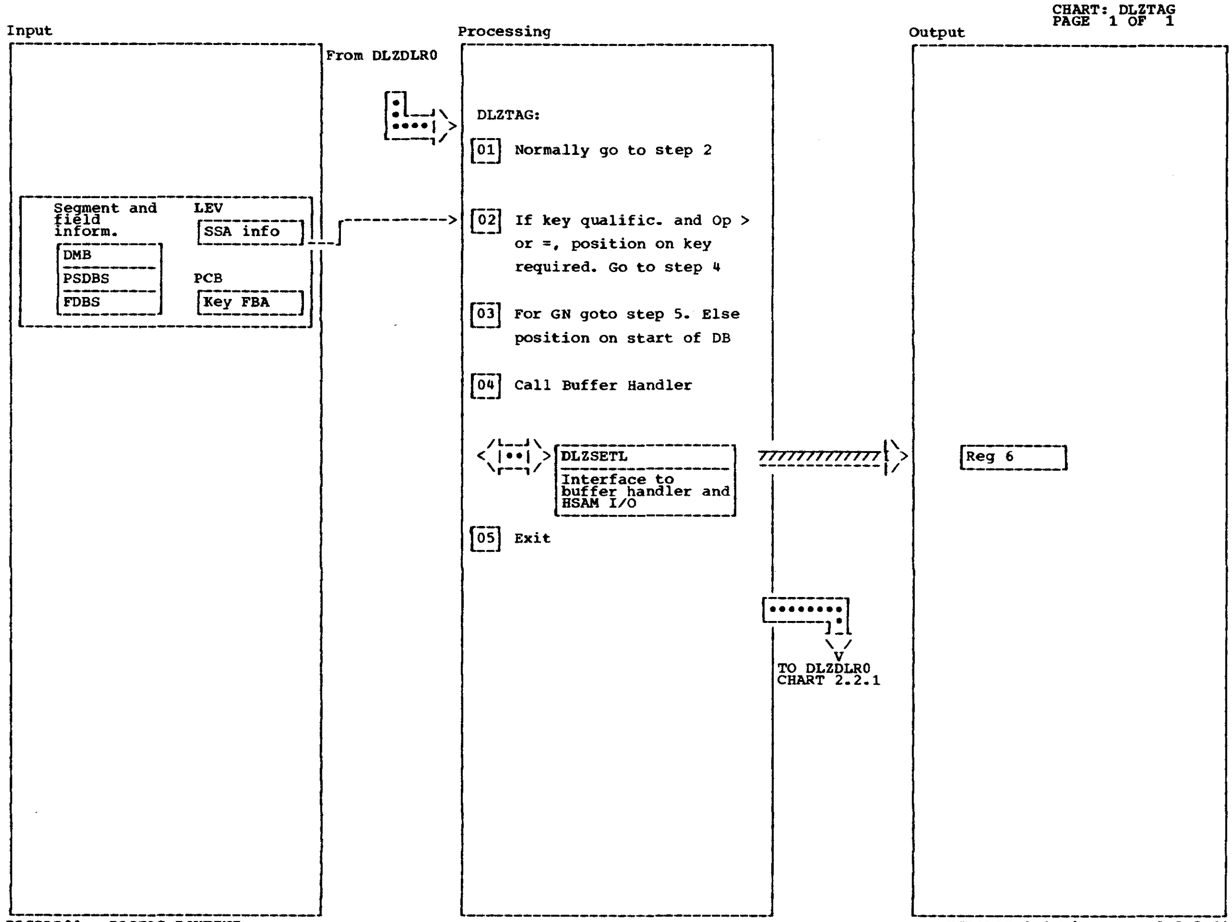
DLZDLR00 - DLZPCHK ROUTINE

HIPOMAT 1.1 Diagram - 2.2.2.2-01

Notes	Routine	Label	Ref
01			
A. For HSAM, more than 1 PCB: restore position.		POSCHKA	
For HISAM: take care of control interval splits.		POSCHKA2	
02		POSCHKB	
A. If not found (segment not sensitive): turn on LEVDLET, go to step 3.			
B. For HS: rel. record no. and offset moved to SDBPOSC, SDBPOSN already by DLZSETL.			
For HD: post twin pointers.	DLZPSTN		
Clear dependent positions (SDBPOSP, ...C, ...N). For HD: post child pointers.	DLZPSTA		
For HD, log. rel. with inverted structure: post child pointers. Subroutine called by DLZPSTA.	DLZAPST		
Clear SDBPOSP, ...C, ...N in preceding sibling SDBs unless multi- positioning.	DLZPOSA		

DLZDLR00 - DLZPCHK ROUTINE

HIPOMAT 1.1 Diagram - 2.2.2.2-01



DLZDLR00 - DLZTAG ROUTINE

HIPOMAT 1.1 Diagram - 2.2.3-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Depending on entry code and other input							
02 Set code PSTSTLEQ for DLZSETL. Set exit code for entry SSAEVAL in DLZSSA.		MTWISSA					
03 Set code PSTSTLBG for DLZSETL. Set exit code for entry SSAEVAL in DLZSSA.		NOLL					
For GN set exit code for entry SSAEVALM in DLZSSA.		KPURTC					
04 DLZSETL branches to subroutines according to DB organization. Reg 6 points to segment in buffer pool.							

DLZDLR00 - DLZTAG ROUTINE

HIPOMAT 1.1 Diagram - 2.2.3-01

Input

First part of LEV
- Position and status conditions set by RETRIEVE in previous or current call

Start level
- Reg4: level where the search has to start

Second part of LEV
- SSA descript. set by ANALYZER

Processing

FROM CALLER

DLZSSA:

01

A. If SSA unqualified, goto step 2

DLZKDTE
Check acceptability of a segment 2.2.2.1

If segment is not accepted goto step 5 → 05

02

DLZUPDT
Update level table

03 Goto next lower level

04 If level not qualified goto step 6.

A. Retrieve segment of specified type

DLZSKPG
Skip segments 2.2.4.1

B. Goto step 1 → 01

Output

First part of LEV
- Descript. of last acceptable sgmt. including its hierarch. path

DLZDLR00 - DLZSSA ROUTINE — HIPOMAT 1.1 Diagram - 2.2.4-01

Notes	Routine	Label	Ref
04			
A. Prepare input (segment type etc.) before entering the central DLZSKPG routine.	DLZSKPG	SKIPGENA	

Notes	Routine	Label	Ref

Input | Processing | Output

05 Segment not acceptable

A. If segment is in same DB-record goto step 5c, else: → 5C

B. Get next root segment

DLZSETL — Interface to buffer handler and HSAM I/O

5C → C. Skip to next segment of same type under present root

DLZSKPG — Skip segments 2.2.4.1

D. Goto step 1 → 01

06 Exit

RETURN TO CALLER

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

Input

Reg15
Option code

SDB
SDBORGN
SDBPOSC
SDBPOSN

Reg6
Buffer loc. of old segment

JCB
JCBCODE, bit JCBRDREQ

FROM CALLER

Processing

DLZSKPG:

[01] Test Option. If 'Skip to next' goto step 2 (JCBLVT will be 0)

A. Prepare control input

[02]

DLZSKPS
Skip to next segment

A. If skip failed, goto step 4

[03] Option 'Skip to next':

DLZHUNT
Test if segment sensitive

If not sensitive, goto step 2

A. Option 'Skip to specif. segment': test segmt. level and segmt. code

If accepted goto step 4

Output

JCB
JCBLVT

DLZDLR00 - DLZSKPG ROUTINE | HIPOMAT 1.1 Diagram - 2.2.4.1-01

Notes	Routine	Label	Ref
[01] Major options: (1) Skip to next segment, Reg15 >= 0. (2) Skip to specified segment, Reg15 = -1.			
A. JCBLVT = X'02', req. sgmt. code, sgmt. level in physical DB, parentage level.		SKIPGENA	
[02] For JCBRDREQ off, current segment is examined first.		SKIPGEN	
DLZSKPS calls general skip routine DLZSKPE which calls specific skip routines:			
for HS	DLZSKPD		
for HD, using SDBPOSN	DLZSTLA		
In some cases (HS, skip to first child of current segment), DLZSKPD is called directly from DLZSKPS.			
A. Fail: e.g., if end of ESDS chain reached for HISAM.			
[03]			

Notes	Routine	Label	Ref

Input	Processing	Output
	B. Else, if position still before segment searched: goto step 2 Else indicate failure: make Reg6 negative 04 Exit TO CALLER	

DLZDLR00 - DLZSKPG ROUTINE

HIPOMAT 1.1 Diagram - 2.2.4.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
B. If sgmt. code of sgmt. found is not larger than that required.							

DLZDLR00 - DLZSKPG ROUTINE

HIPOMAT 1.1 Diagram - 2.2.4.1-02

Input | Processing | Output

FROM DLZDLR0
CHART 2.2.1
STEP 6A

DLZGETS:

Input: LEV: LEVLEV; DMB

[01] Turn on LEVDATA. Turn on LEVHELD if get hold call.

A. Save lowest level number

Output: LEV: LEVF1 (..1. 1...); JCB: KEEPIT+2

[02] If logical relationship

DLZLOGR — build concat. segment — 2.2.5.1

If variable length segment

DLZVLRT — build/ expand segment

[03] Move segment to I/O area

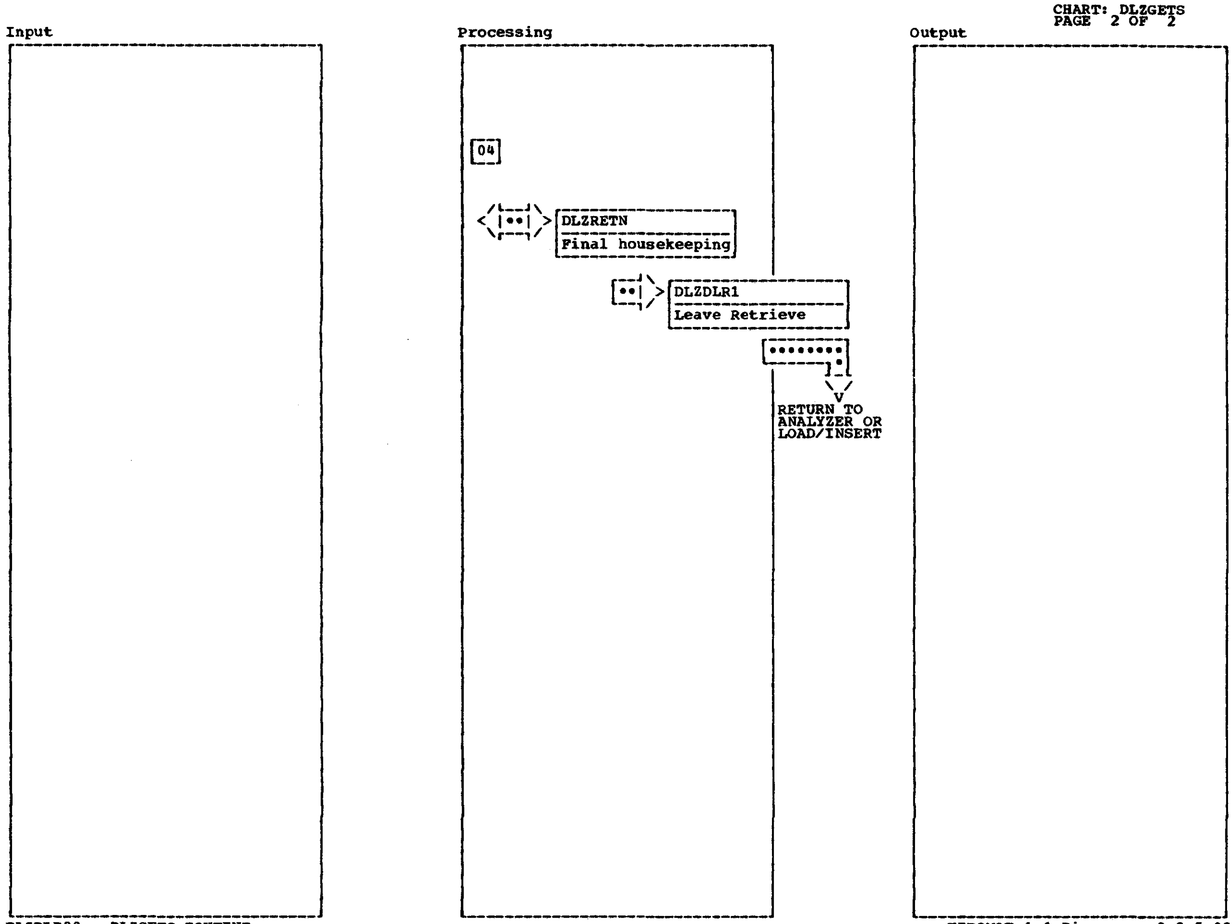
Input: DMB: DMBDL; Data: Segment in buffer pool

DLZMOVA

Give segment location and length

Output: PST: PSTSEG, PSTSEGL; I/O area

Notes	Routine	Label	Ref
[03] If batch or only one task active, and if segment is fixed length and not involved in logical relationship, segment data are not moved but left in buffer pool. Same is true for Insert calls. For a path call (*D command) data have already been moved in DLZUPDT, are not moved here. Address of I/O area is in PSBIOAWK.			



DLZDLR00 - DLZGETS ROUTINE

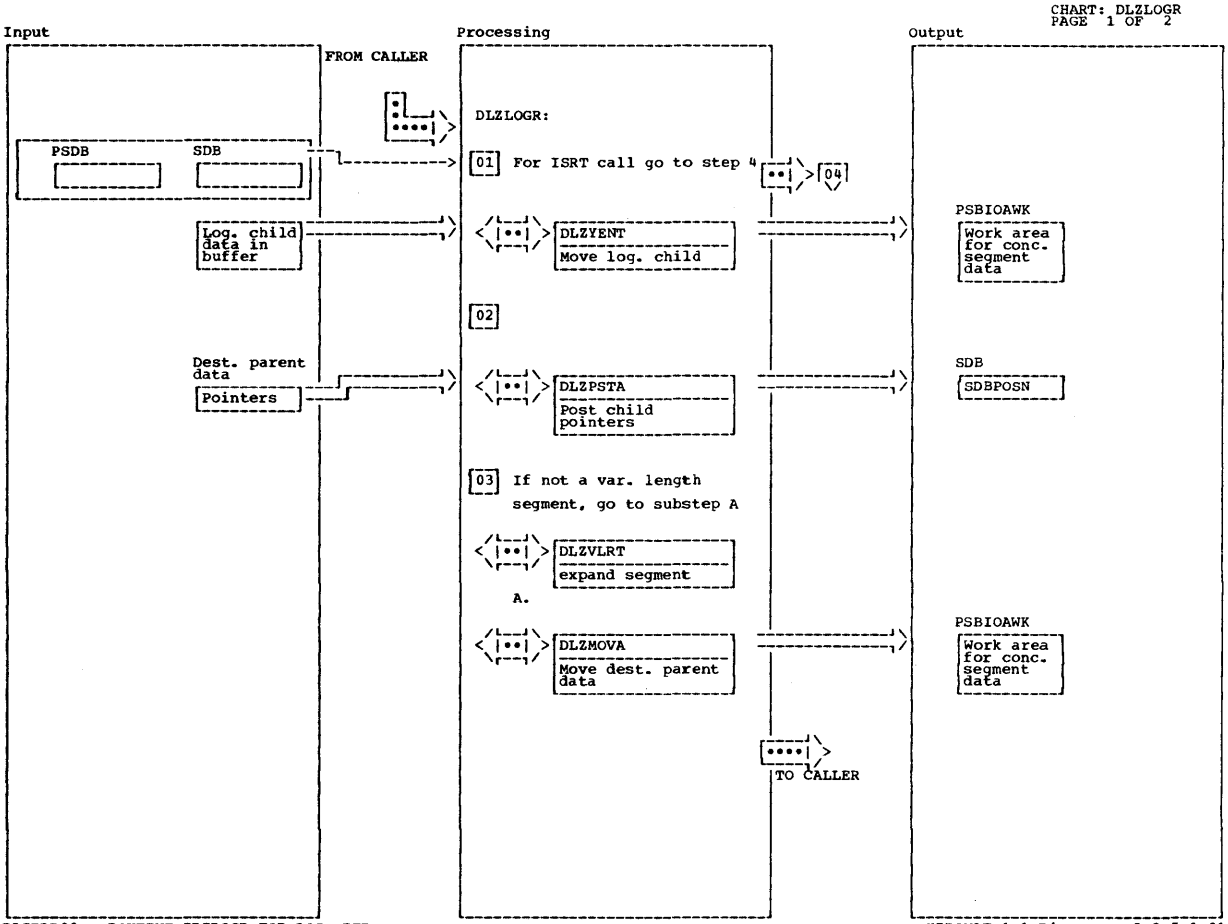
HIPOMAT 1.1 Diagram - 2.2.5-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
04 For insert calls, return is to Load/ Insert							

DLZDLR00 - DLZGETS ROUTINE

HIPOMAT 1.1 Diagram - 2.2.5-02

CHART: DLZLOGR
PAGE 1 OF 2

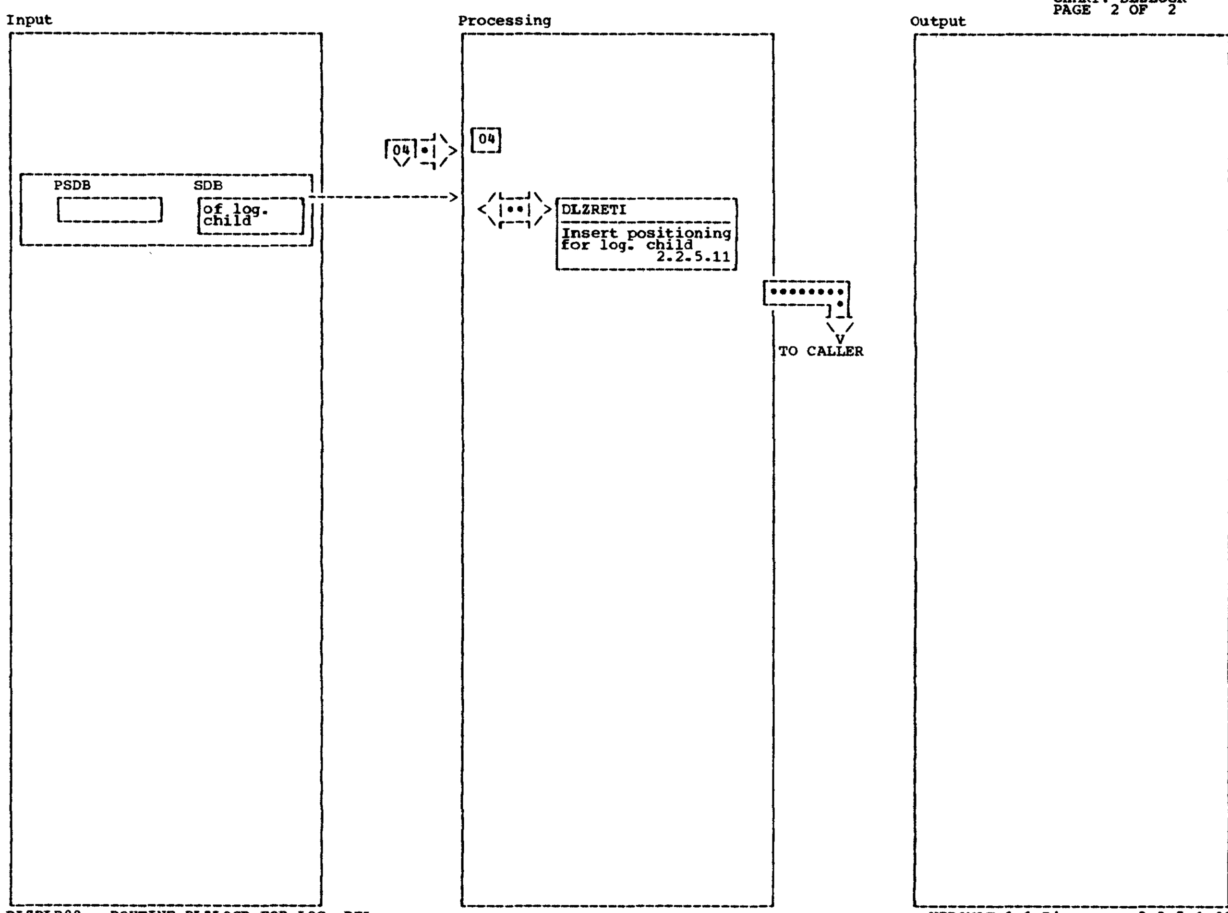


DLZDLR00 - ROUTINE DLZLOGR FOR LOG. REL. HIPOMAT 1.1 Diagram - 2.2.5.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Destination parent concatenated key and logical child data.	DLZYENT						
02 Get child pointers from destination parent. If physical parent is dependent in logical data base (inverted structure), get physical parent pointer.	DLZPSTA						

DLZDLR00 - ROUTINE DLZLOGR FOR LOG. REL. HIPOMAT 1.1 Diagram - 2.2.5.1-01

CHART: DLZLOGR
PAGE 2 OF 2



DLZDLR00 - ROUTINE DLZLOGR FOR LOG. REL.

HIPOMAT 1.1 Diagram - 2.2.5.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
04 Destination parent exists. Position segment on alternate twin chain.							

DLZDLR00 - ROUTINE DLZLOGR FOR LOG. REL.

HIPOMAT 1.1 Diagram - 2.2.5.1-02

Input

SDB: of log. child

PSBIOAWK: Log. child data including concaten. key

LEV: LEVUSEOF

DMB: PSDB's, secondary lists

Dest. parent in buffer

FROM CALLER

Processing

DLZRETI:

[01] Retrieve destination parent using concatenated key

DLZRETK

[02] For virtual log. child go to substep B

A. Find pointer number

Go to step 3

B. Find pointer numbers

[03] Get pointers from destin. parent

If no key and rule 'FIRST', go to exit, step 5

Output

PSBIOAWK: Dest. parent data

Notes	Routine	Label	Ref
[01] LEVUSEOF indicates offset of key for this level in concatenated key.			LEV
Destination parent data are stored behind concatenated key and log. child.	DLZRETK		
[02] For virtual logical child, i.e. insert through logical path. Positioning on physical twin chain required.			
A. Find logical twin pointer number. Find logical child first and last pointers in logical parent. Find FDB for key of log. child, if present.		RETISRTF	
B. Find physical twin pointer number. Find physical child first and last pointers in parent. Find FDB for key of virtual logical child, if present.		RETISRTR	
Log. twin key is moved to key feedback area.	DLZUPDL		

Notes	Routine	Label	Ref
[03]		RETISRTU	

Input	Processing	Output
	[04] Follow alternate twin chain until key (if present) larger than key of inserted segment or to end of chain. Go to exit, step 5 Special case: matching key found in chain: A. Key unique: If segment deleted logically, go to exit, step 5 Else status code 'II' and go to exit, step 5 B. Key nonunique: If rule is 'FIRST', go to exit, step 5 Else follow twin chain until larger key found or end of chain [05] Exit RETURN TO CALLER	DBPCB DBPCBSTC

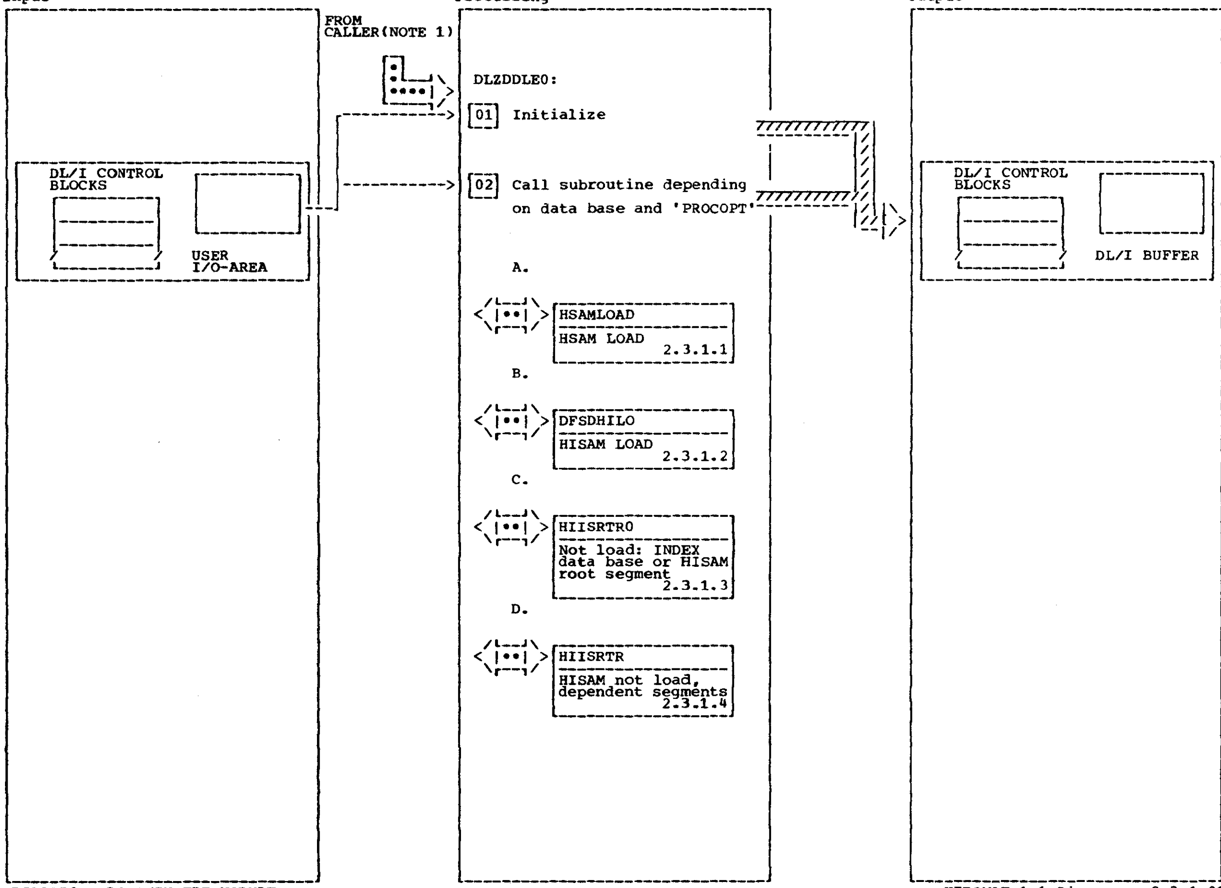
DLZDLR00 - DLZRETI ROUTINE

HIPOMAT 1.1 Diagram - 2.2.5.11-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[04] Alternate means: logical twin chain if entering from physical path, physical if entering from logical path.		RETISRTE					
If sequence field is in destination parent concatenated key - possible only for virtual log. child - the virtual area (phys. parent conc. key and log. child data) is built in PSBIOAWK calling routines DLZYSTC and DLZMOVA. As an indication, first byte of PSTWRKT5 is set X'FF'.		RETIVK					
A. For logically deleted segment, turn on bit JCBDEFDL in field JCBCODE.							

DLZDLR00 - DLZRETI ROUTINE

HIPOMAT 1.1 Diagram - 2.2.5.11-02



DLZDDLE0 - LOAD/INSERT MODULE — HIPOMAT 1.1 Diagram - 2.3.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 DLZDDLE0 is called from DLZDLA00 the call analyzer or from DLZDLR00 the retrieve module.							

DLZDDLE0 - LOAD/INSERT MODULE — HIPOMAT 1.1 Diagram - 2.3.1-01

Input

DL/I CONTROL BLOCKS

USER I/O-AREA

Processing

E.

DFSDHDL0
HDAM/HIDAM LOAD 2.3.1.5

F. Do special checking for logical child segments

DFSDHDI0
HDAM/HIDAM NOT LOAD 2.3.1.6

03

A.

DFSDNXT0
Ending routine : NOT LOAD mode 2.3.1.7

B.

HIISNXLV
Ending routine : LOAD mode 2.3.1.8

TO DLZDLA00 CHART 2.1.1

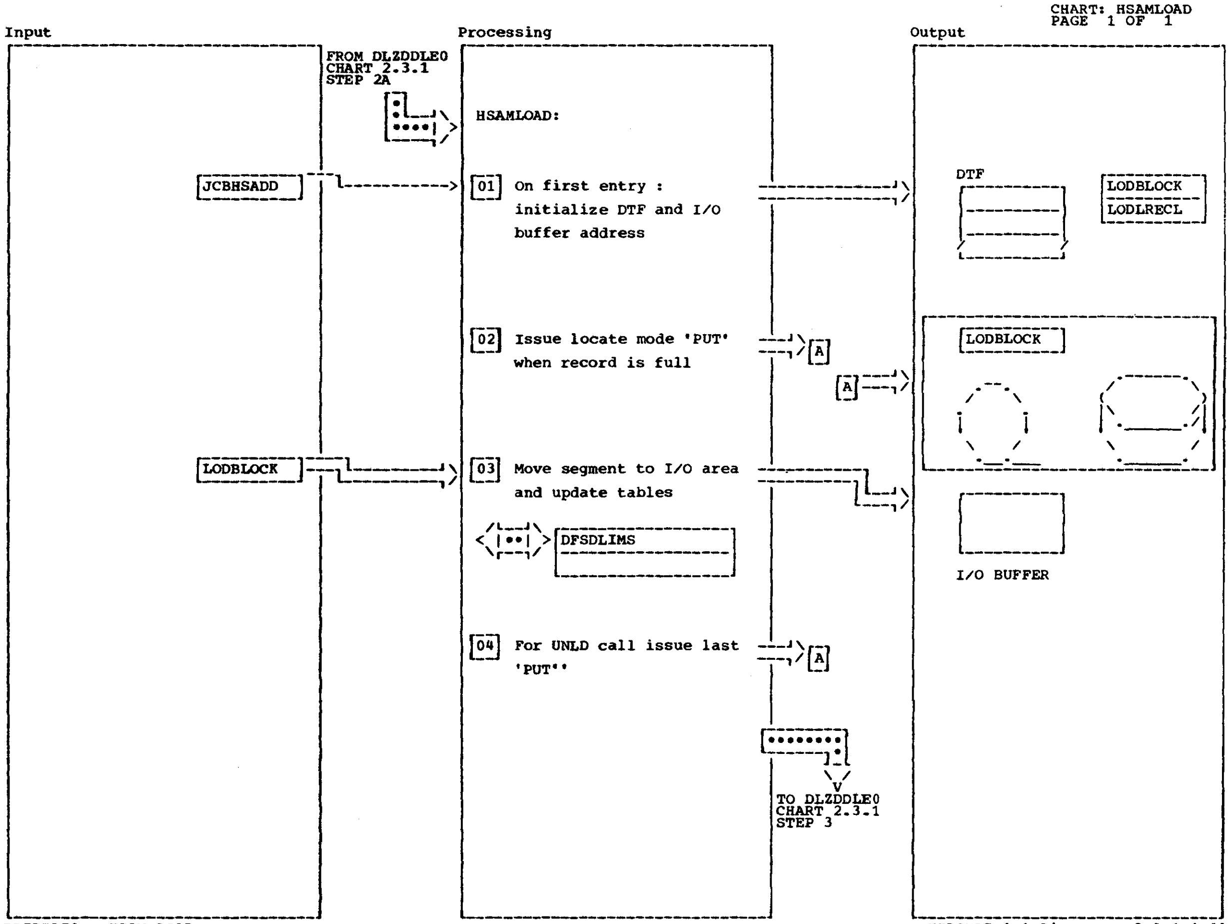
Output

DL/I CONTROL BLOCKS

DL/I BUFFER

DLZDDLE0 - LOAD/INSERT MODULE HIPOMAT 1.1 Diagram - 2.3.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref



DLZDDLE0 - HSAM LOAD HIPOMAT 1.1 Diagram - 2.3.1.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 DLZDLOC0 stores the I/O area address in the JCB. With every 'PUT' it is updated. The record size is taken from the DTF, the error exit address in the DTF updated -QSYNAD-.		HSAMFRST					

CHART: DFSDHILO
PAGE 1 OF 1

Input	Processing	Output
	FROM DLZDDLE0 CHART 2.3.1 STEP 2B	
	DFSDHILO:	
JCB: JCBPRESF; JCB: LODBLOCK, LODOFFSET, LODLRECL; SDB	01 a).. Root segments Write previous KSDS record and get buffer for new one	
PSDB: DMBPRSZ, DMBDL; JCB: LODBLOCK, LODOFFSET, LODLRELL	b).. Dependent segments If no more space in ESDS write previous ESDS and get buffer for new one	JCB: LODBLOCK, LODOFFSET, LODLRECL
	02 Move segment to buffer	
	DFSDLIMS	
JCB: LODBLOCK, LODOFFSET, LODLRECL	03 UNLD - call Write previous KSDS and ESDS record. write new KSDS record with root key of FF's	
	TO DLZDDLE0 CHART 2.3.1 STEP 3	

DLZDDLE0 - HISAM-LOAD

HIPOMAT 1.1 Diagram - 2.3.1.2-01

Notes	Routine	Label	Ref
01			
A. Record length, buffer address, offset into buffer is stored in the JCB and passed from call to call.	WRITEOLD		
When a call for a new root segment is made, the buffer handler is called to write the previous KSDS record and to get bufferspace for the new one.	DLZDBH00		
B. If there is space left in the ESDS records, continue with step 2. Else the RBA of the next ESDS record is calculated, the pointer of the current ESDS record updated, and the buffer-handler called to write the ESDS. Another call to DLZDBH00 is made to get buffer space for a new ESDS record.	WRITEOLD NEWRBA DLZDBH00	NEEDOSAM	
ABEND 855 is given if VSAM returns an RBA different from the calculated one.			

Notes	Routine	Label	Ref
02 The segment is moved, the PCB key feed back and the level-table updated.	DFSDLIMS		

DLZDDLE0 - HISAM-LOAD

HIPOMAT 1.1 Diagram - 2.3.1.2-01

Input	Processing	Output
	FROM DLZDDLE0 CHART 2.3.1 STEP 2C	
	HIISRTR0:	
PST: PSTUSER, PSTBYTNM	01 Call DLZDBH00 to get segment with key EQ or HI	PSTDATA; KSDS record
PSTDATA; ACB extension	02 If key of returned segment is equal, then update SDB and level table. If it is not equal go to step 7	SDB; LEVTAB
	03 Return II status, when segment was not deleted	DBPCB: 'II'
	TO DLZDDLE0 2.3.1 STEP 4	
	else log old segment	
I/O AREA	04 Move segment, update PCB and level table	KSDS record; PCB; LEVTAB
	DFSDLIMS	
	05 Indicate only one segment in record and log new record	

DLZDDLE0 - HISAM-ROOT INSERT

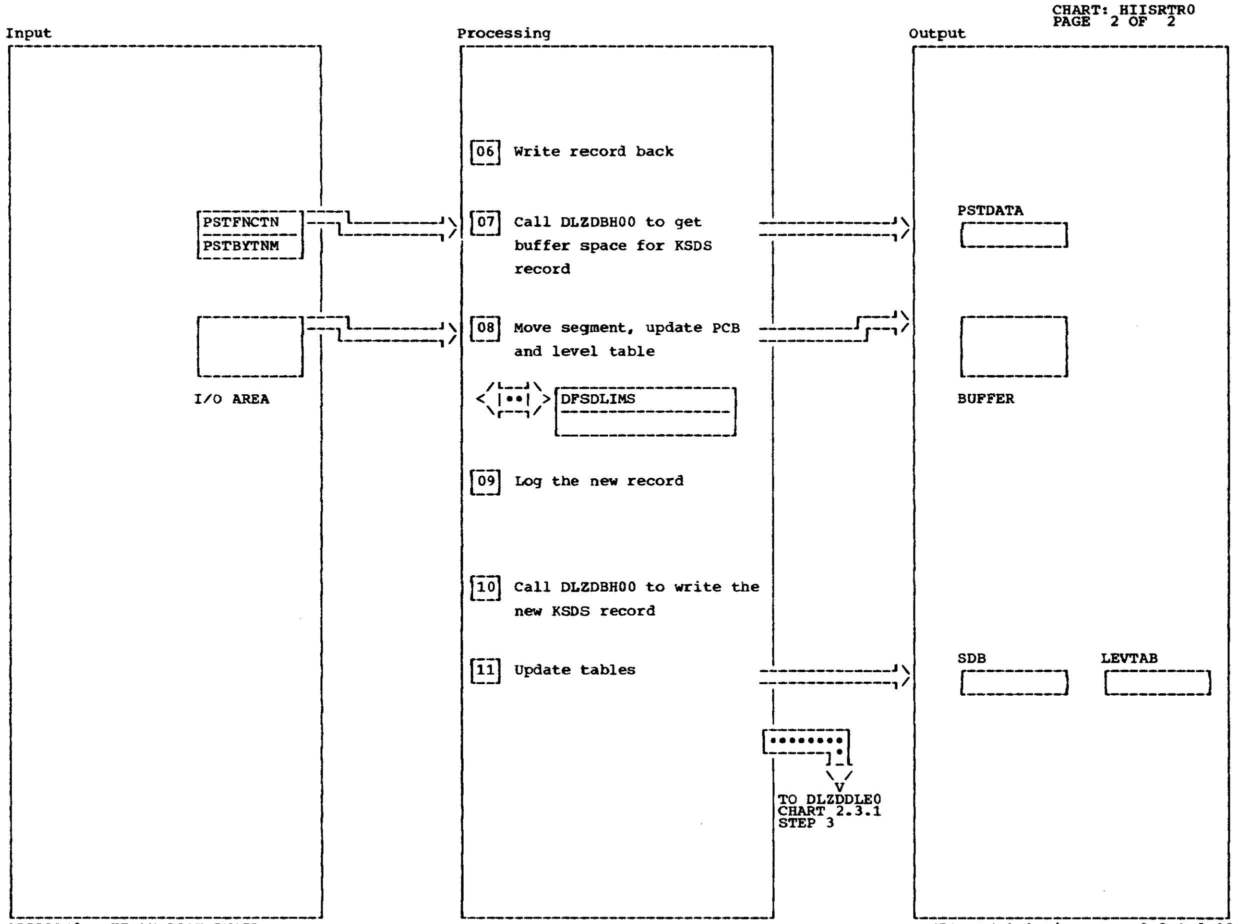
HIPOMAT 1.1 Diagram - 2.3.1.3-01

Notes	Routine	Label	Ref
01 The buffer-handler is called with 'PSTSTLEQ' to get a segment with key equal or higher than the one to be inserted.	GOTOFUNC	HIIRTR0	
02 If the key returned is higher, processing continues with step 7.		ISNOTEQ	
03 When the delete-flag is not on in the segment returned, an II-status code is returned to the caller.		NONOI	
The data base log module is called to log the old KSDS record.	DLZRDBL0	ISDELETE	
04 The new root segment is moved to the KSDS record. PCB key feed back area and level table are updated.	DFSDLIMS	LOGAFTNX	
05 The pointer to the ESDS record is cleared and '00' moved to the KSDS record behind the root segment. The data base log module is called to log the new KSDS record.			

DLZDDLE0 - HISAM-ROOT INSERT

HIPOMAT 1.1 Diagram - 2.3.1.3-01

CHART: HIISRTR0
PAGE 2 OF 2

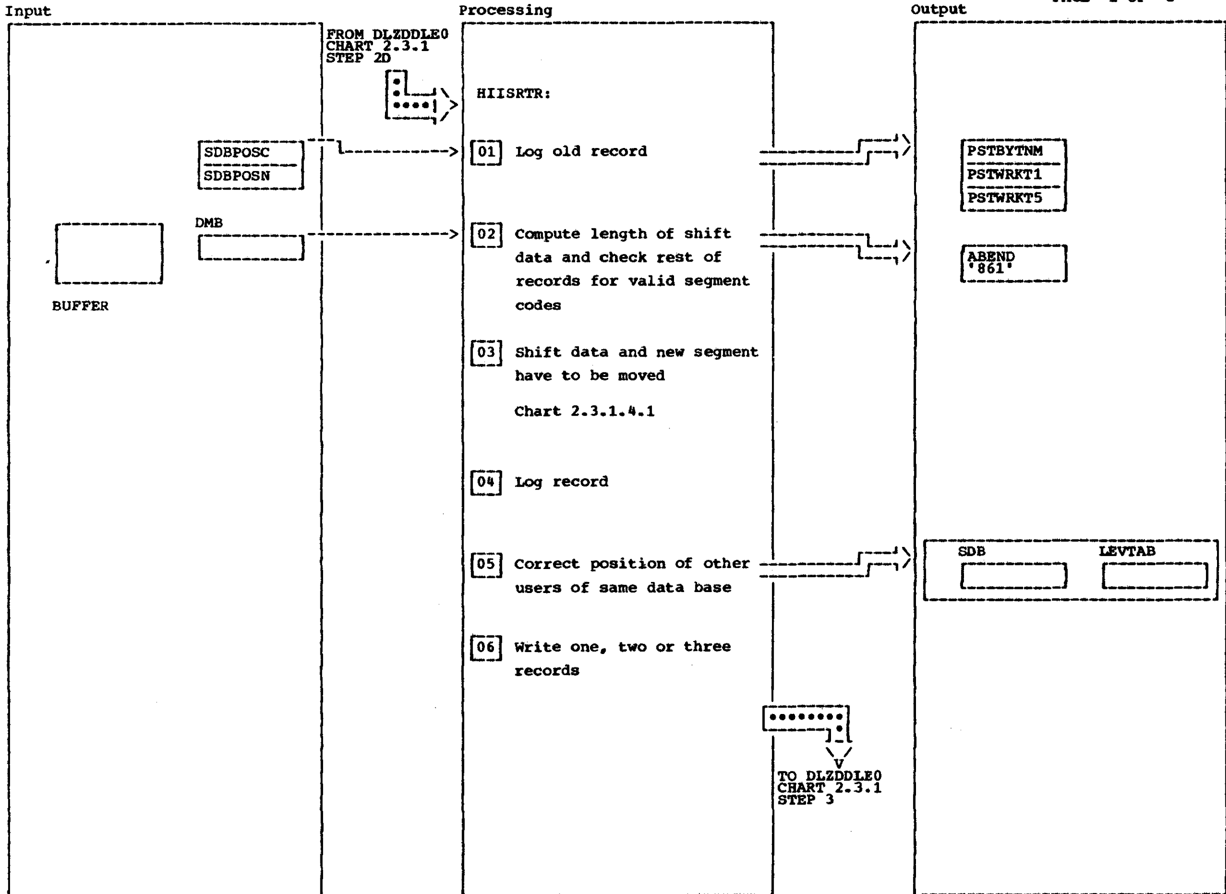


DLZDDLE0 - HISAM-ROOT INSERT

HIPOMAT 1.1 Diagram - 2.3.1.3-02

Notes	Routine	Label	Ref
06 The buffer-handler is called to write the KSDS record back -PSTBFALT-.			
07 The buffer-handler is called -PSTGBSPC- to get buffer space for one KSDS record.	GOTOFUNC	ISNOTEQ	
10 -PSTPUTKY- is used to write the new KSDS record.	GOTOFUNC		

Notes	Routine	Label	Ref



DLZDDLE0 - HISAM DEPENDENT SEGMENT INSERT

HIPOMAT 1.1 Diagram - 2.3.1.4-01

Notes	Routine	Label	Ref
01 DLZDLR00 has positioned within an KSDS or ESDS record, where the new segment has to be inserted. The old record is logged from insert point on to the right.			
02 The record is inspected from the insert point to the right. The segment code is checked and the length of the remaining segments added to give the 'shift data'.		HAVELREC COMPSHFT ABENDB61	
03 Chart 2.3.1.4.1 describes in more detail what has to be done to move segment and 'shift data'.			
04 Log the old record from insert point to the right.	DLZRDBL0	LOGLEVCO	
05 SDB's and level tables of other PCB's that are positioned in the same record are updated to show the shifted position of the segments.		INSADJUS	

Notes	Routine	Label	Ref
06 DLZDBH00 is called to write back the old record, and to write one or two new ESDS records.		KNNDONEX	

Input | Processing | Output

FROM HIISRTR
CHART 2.3.1.4
STEP 3

NOTSC:

Input: DMB, PSTUSER, I/O AREA

01 Segment and 'shift data' fit in old record :

A. Move 'shift data' right

B. Move segment to buffer, update tables

Output: old record

02 Segment fits in old record, but not 'shift data''

A. Calculate RBA of new ESDS record

B. Get buffer space for one ESDS record

C. Chain old and new record and log chain

D. Move 'shift data' to new ESDS record

E. Log new record

F. Move segment to old record

Output: old record, new record

DLZDDLE0 - HISAM DEPEND. SEGM. INSERT

HIPOMAT 1.1 Diagram - 2.3.1.4.1-01

Notes	Routine	Label	Ref
01 When both the new segment and the shift data fit in the old record, the shift data are moved right by segment length. The segment is moved to the record and PCB and level table are updated.	DFSDLIMS	OVERLAPL	
02 A new ESDS record has to be built.	GETNESDS	SEGTOOLD	
	LOGCHAIN		
	COMMOVE		
	LOGNEWOS		
	DFSDLIMS		
	GOTOFUNC		
	DLZDBH00		
	DLZRDBL0		

Input

DMB

I/O AREA

Processing

[03] Segment and 'shift data' to new ESDS

A. Calculate RBA of new ESDS record

B. Get buffer buffer space of new ESDS record

C. Chain the two records and log the chain

D. move segment to new record, update tables

E. Move 'shift data' to new record. If they do not fit repeat 3 a,b,c

F. Log one or two new ESDS records

TO HIISRTR
CHART 2.3.1.4
STEP 4

Output

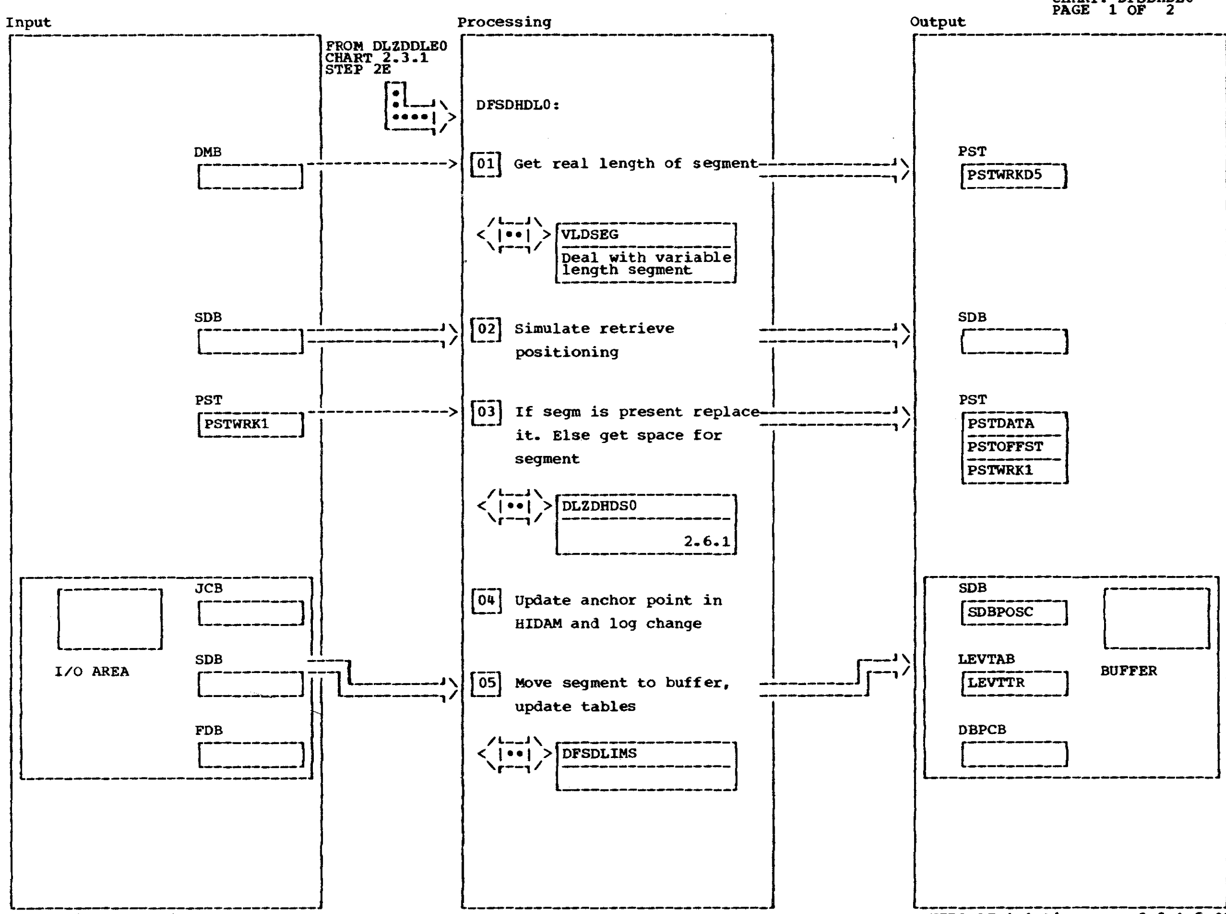
old record

new record(s)

DLZDDLE0 - HISAM DEPEND. SEGM. INSERT — HIPOMAT 1.1 Diagram - 2.3.1.4.1-02

Notes	Routine	Label	Ref
[03] Neither segment nor 'shift data' fit in the old record. a new ESDS record has to be built. if it does not have room for the segment and 'shift data' another new ESDS record has to be built. The records are chained and logged.	GETNESDS	SEGTONEW	
	LOGCHAIN	SHIFT00	
	DFSDLIMS	SHIFTOS2	
	COMMOVE		
	LOGNEWOS		
	NEWRBA		
	GOTOFUNC		
	DLZDBH00		
	DLZRDBL0		

Notes	Routine	Label	Ref



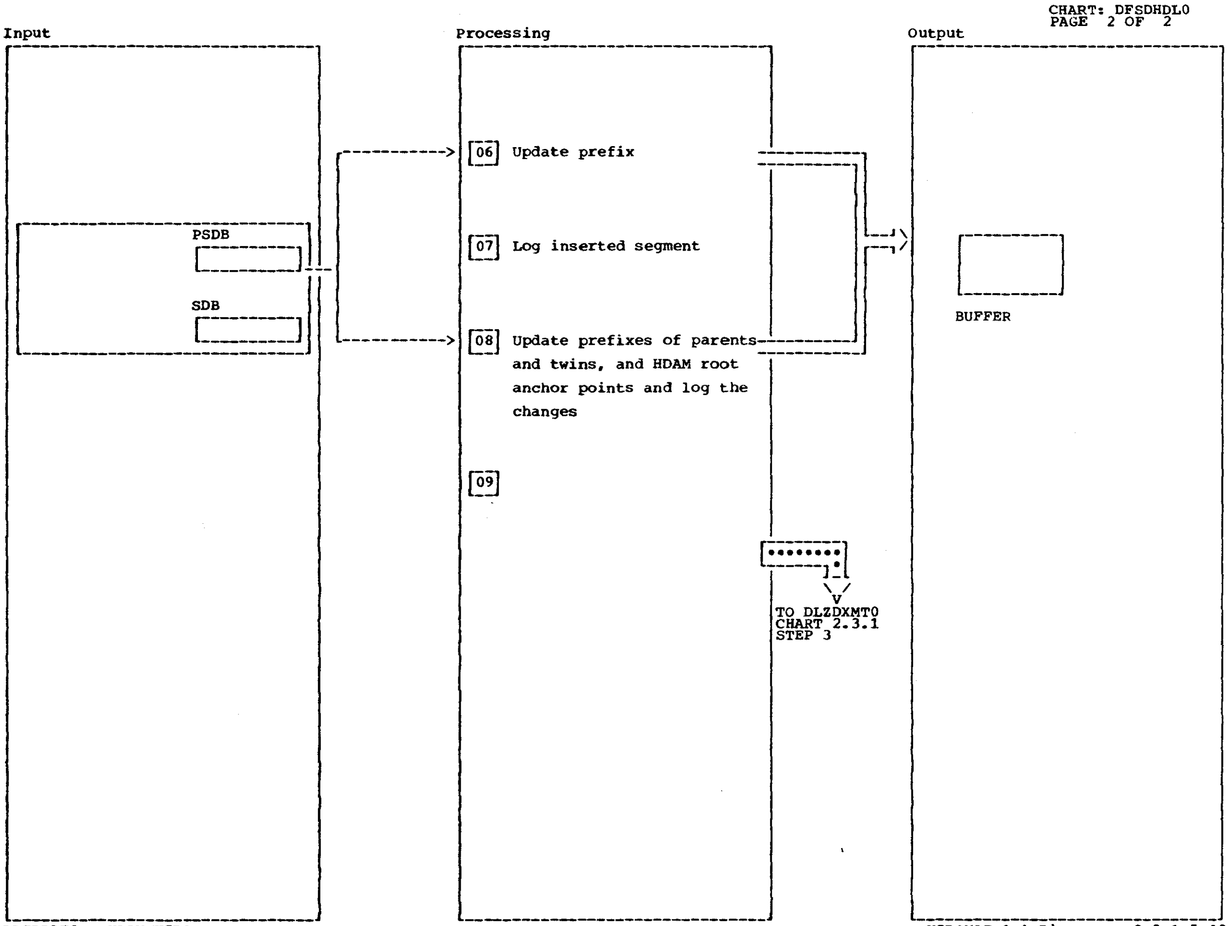
DLZDDLE0 - HDAM/HIDAM

HIPOMAT 1.1 Diagram - 2.3.1.5-01

Notes	Routine	Label	Ref
01 The subroutine VLDSEG takes the length from the PSDB for fixed length segments. For variable length segments from the user's I/O area. The compaction exit routine is called, if it exists.		VLDSEG	
ABEND '863' is given when the compaction routine changes the sequence field.			
02 For HDAM root segments DLZDLR00 did the positioning. For other segments it is done here.			
03 Space management is called to get space for the segment. If the segment was deleted in one path only, i.e. it was not removed by DLZDLD00, the segment is replaced with the new data.	DLZDHDS0	GETSPACE SPACEOUT POSTPST SPACEOK	
04 HIDAM root segments without PTB-pointers are chained off the anchor point in chronological sequence.			
05 Move segment to buffer, update PCB key feed back, update level table.	DFSDLIMS	ANCHOROK MVVL1 GETKEY NOFIELDS	

DLZDDLE0 - HDAM/HIDAM

HIPOMAT 1.1 Diagram - 2.3.1.5-01



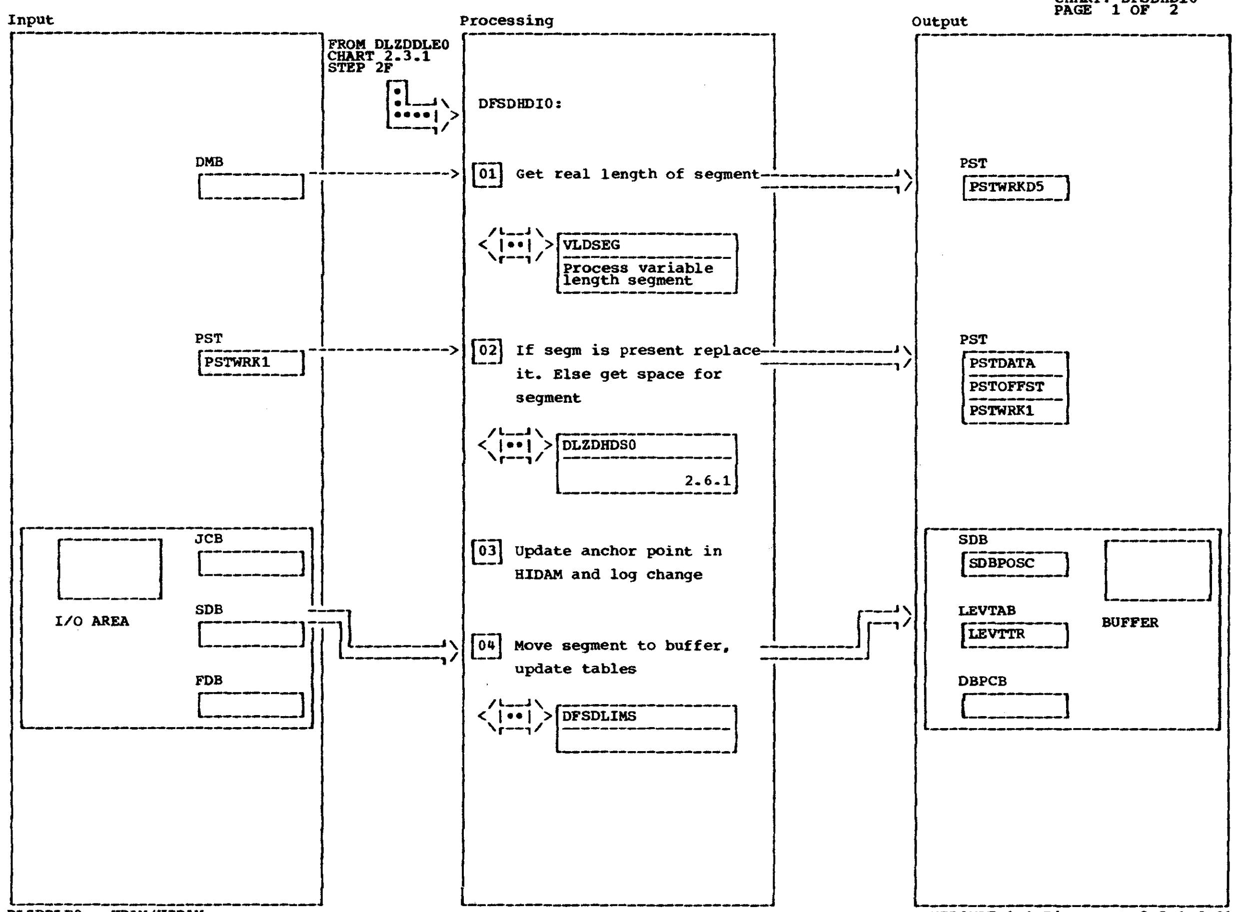
DLZDDLE0 - HDAM/HIDAM

HIPOMAT 1.1 Diagram - 2.3.1.5-02

Notes	Routine	Label	Ref
06 The prefix of the segment is updated : physical twin ptrs, physical parent ptr, logical parent ptr, logical twin ptrs.			
07 The data base log module is called to log the inserted segment.		MYPREOK	
08 Call space management to update the bitmap if required: update prefix of physical twins, logical twins and physical parent. Update anchor point for HDAM root segments, call the data base log module to log all changes.	TOSPACE DLZDHDS0 UPPARENT UPPREFIX	UPBITMAP BITMAPOK HDDANCOR	

Notes	Routine	Label	Ref

CHART: DFSDHDI0
PAGE 1 OF 2



DLZDDLE0 - HDAM/HIDAM

HIPOMAT 1.1 Diagram - 2.3.1.6-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 When this entry to this routine is used, DLZDLR00 had done the positioning.		DFSDHDIO					
02 Space management is called to get space for the segment. If the segment was deleted in one path only, i.e. it was not removed by DLZDLD00, the segment is replaced with the new data.	DLZDHDS0	GETSPACE SPACEOUT POSTPST SPACEOK					
03 HIDAM root segments without PTB-pointers are chained off the anchor point in chronological sequence.							
04 Move segment to buffer, update PCB key feed back, update level table.	DFSDLIMS	ANCHOROK MVVL1 GETKEY NOFIELDS					

DLZDDLE0 - HDAM/HIDAM

HIPOMAT 1.1 Diagram - 2.3.1.6-01

CHART: DFSDHDI0
PAGE 2 OF 2

Input

PSDB

SDB

Processing

[05] Update prefix

[06] Log inserted segment

[07] Update prefixes of parents and twins, and HDAM root anchor points and log the changes

[08]

TO DLZDDLE0
CHART 2.3.1
STEP 3

Output

BUFFER

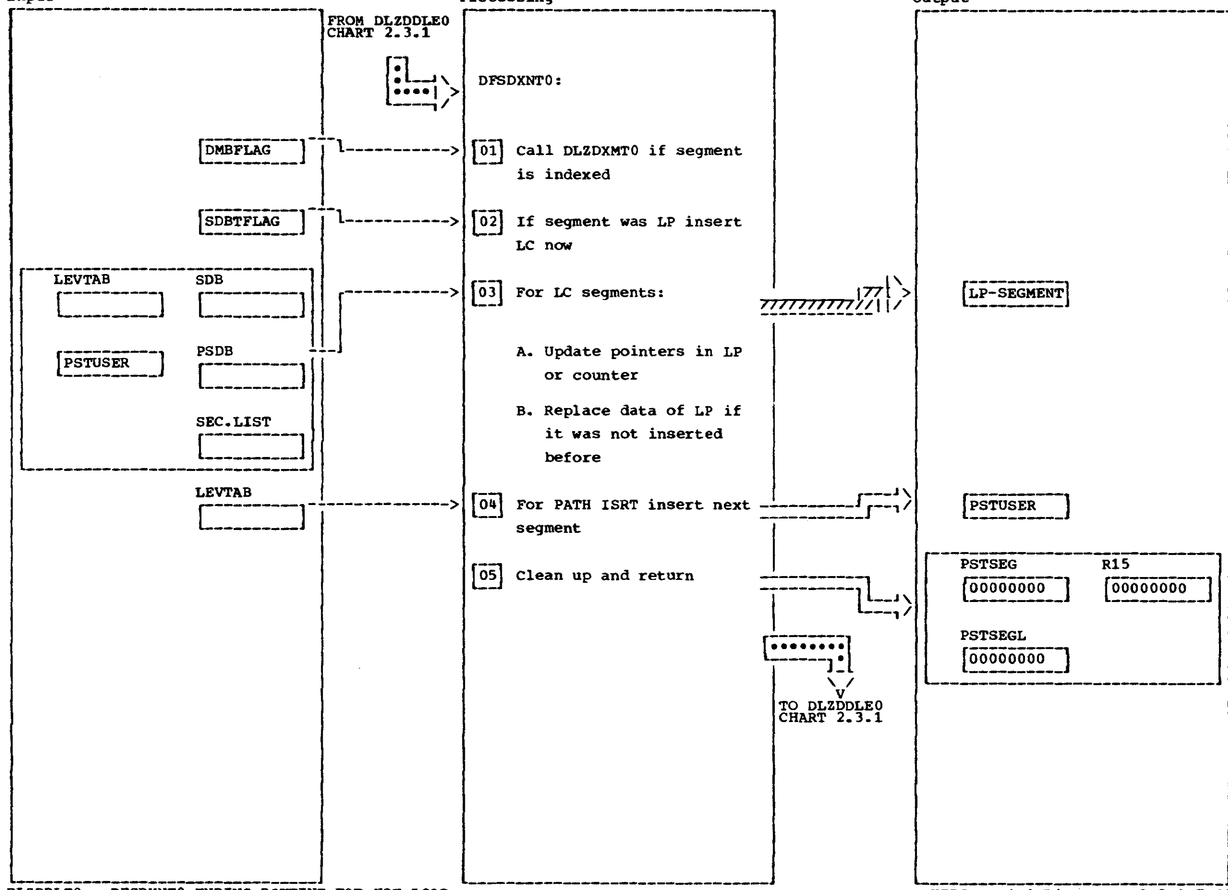
DLZDDLE0 - HDAM/HIDAM

HIPOMAT 1.1 Diagram - 2.3.1.6-02

Notes	Routine	Label	Ref
[05] The prefix of the segment is updated : physical twin ptrs, physical parent ptr, logical parent ptr, logical twin ptrs.			
[06] The data base log module is called to log the inserted segment.		MYPREOK	
[07] Call space management to update the bitmap if required: update prefix of physical twins, logical twins and physical parent. Update anchor point for HDAM root segments, call the data base log module to log all changes.	TOSPACE DLZDHDS0 UPPARENT UPPREFIX	UPBITMAP BITMAPOK HDDANCOR	

Notes	Routine	Label	Ref

CHART: DFSDXNT0
PAGE 1 OF 1



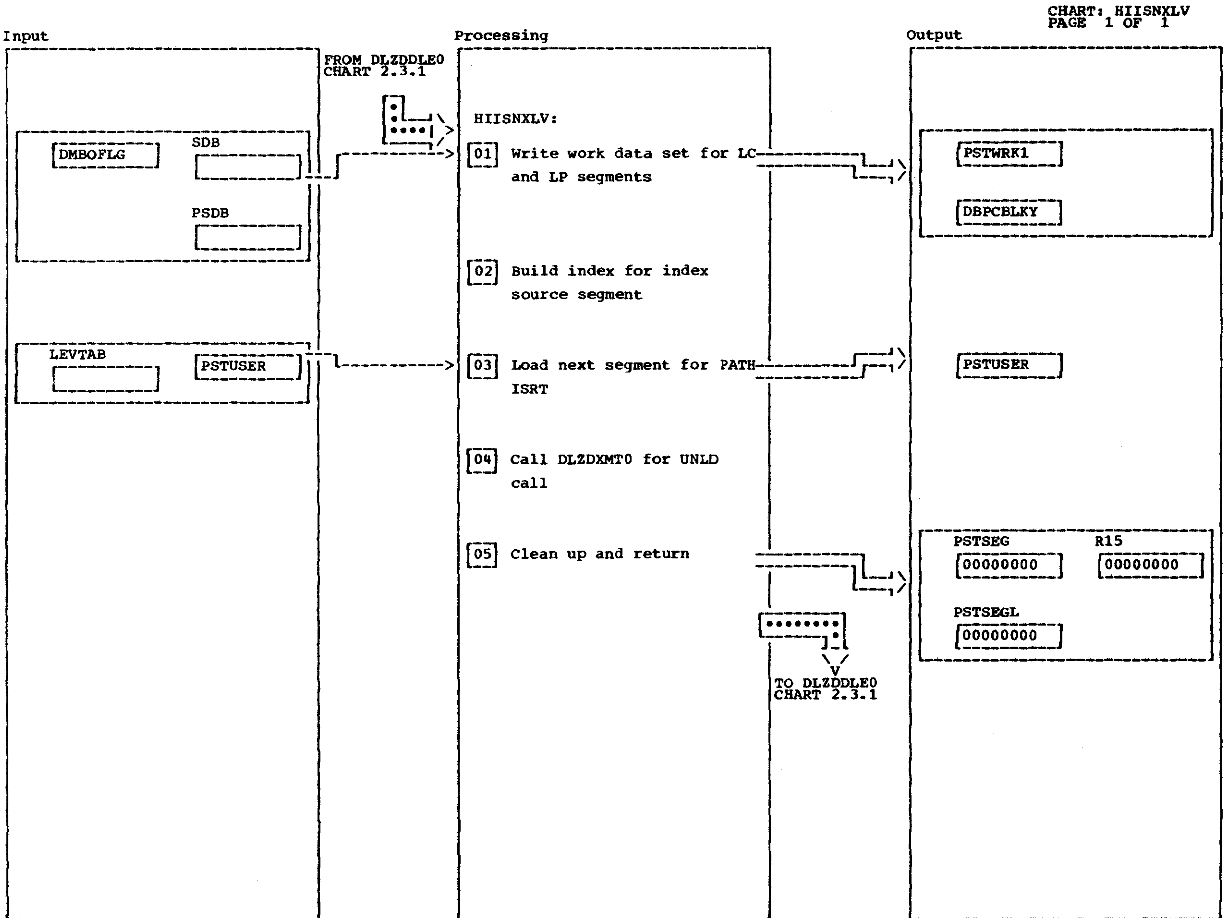
DLZDDLE0 - DFSDXNT0 ENDING ROUTINE FOR NOT LOAD

HIPOMAT 1.1 Diagram - 2.3.1.7-01

Notes	Routine	Label	Ref
01 Index Maintenance is called to build the primary or secondary index for an index source segment.			
02 If the ISRT call was for a concatenated segment, the destination parent was inserted first - if it did not exist before the ISRT call - . The next step is to insert the logical child segment.The insert process is repeated from Chart 2.3.1 step 2F on.		NXTLEVIS	
03			
A. The 'logical child first' and the 'logical child last' pointers in the logical parent segment are updated, or the counter, if relationship is unidirectional.		NOTTARG STLPADDR	
B. If the ISRT rule of the destination parent is virtual and this segment existed already, then the data of the destination parent is replaced. DLZDXMT0 is called to replace the index, if the destination parent is an index source segment 'PSTXMRPL' .		NOADD FIXREP	
04 If there are more segments to be inserted in a PATH, then point to next segment in I/O area and continue with Chart 2.3.1 step 2.			

DLZDDLE0 - DFSDXNT0 ENDING ROUTINE FOR NOT LOAD

HIPOMAT 1.1 Diagram - 2.3.1.7-01



DLZDDLE0 - HIISNXLV ENDING ROUTINE LOAD MODE

HIPOMAT 1.1 Diagram - 2.3.1.8-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 If the segment just loaded was a logical child or a logical parent segment, DLZDSEH0 is called to write the work data set. If opening of the work data set failes due to 'ASSGN SYS013,IGN' and the segment was an LP processing continues. On any other open failure ABEND 864 is given		CALLERN CALLWORK					
02 If the segment is an index source segment DLZDXMT0 is called. It writes the work data set or writes the index pointer segment directly		NOTLOAD NCALLNDX					
03 For PATH ISRT the pointer to the I/O area is updated and processing continues with Chart 2.3.1 step 2		NOINDEX2 NXTLEVLD					
04 DLZDXMT0 is called to inspect all PSDB's of that DMB for index source segments and builds an FF-key index pointer record for it							

DLZDDLE0 - HIISNXLV ENDING ROUTINE LOAD MODE

HIPOMAT 1.1 Diagram - 2.3.1.8-01

Input

FROM DLZDLA00 CHART 2.1.1.1

DBPCB
DBPCBKFD

USER I/O AREA

JCBPRESF

Processing

DLZDLD00:

[01] Initialize addresses and check that key field has not been changed

[02] If call is REPL

REPLACE
Process replace 2.4.1.1

[03] If call is DLET and data base is HISAM

Process HISAM delete 2.4.1.2

[04] If call is DLET and data base is HIDAM or HDAM

Process HD/HID delete 2.4.1.3

[05] Return to DLZDLA00 with return code in R15

TO DLZDLA00 CHART 2.1.1.1

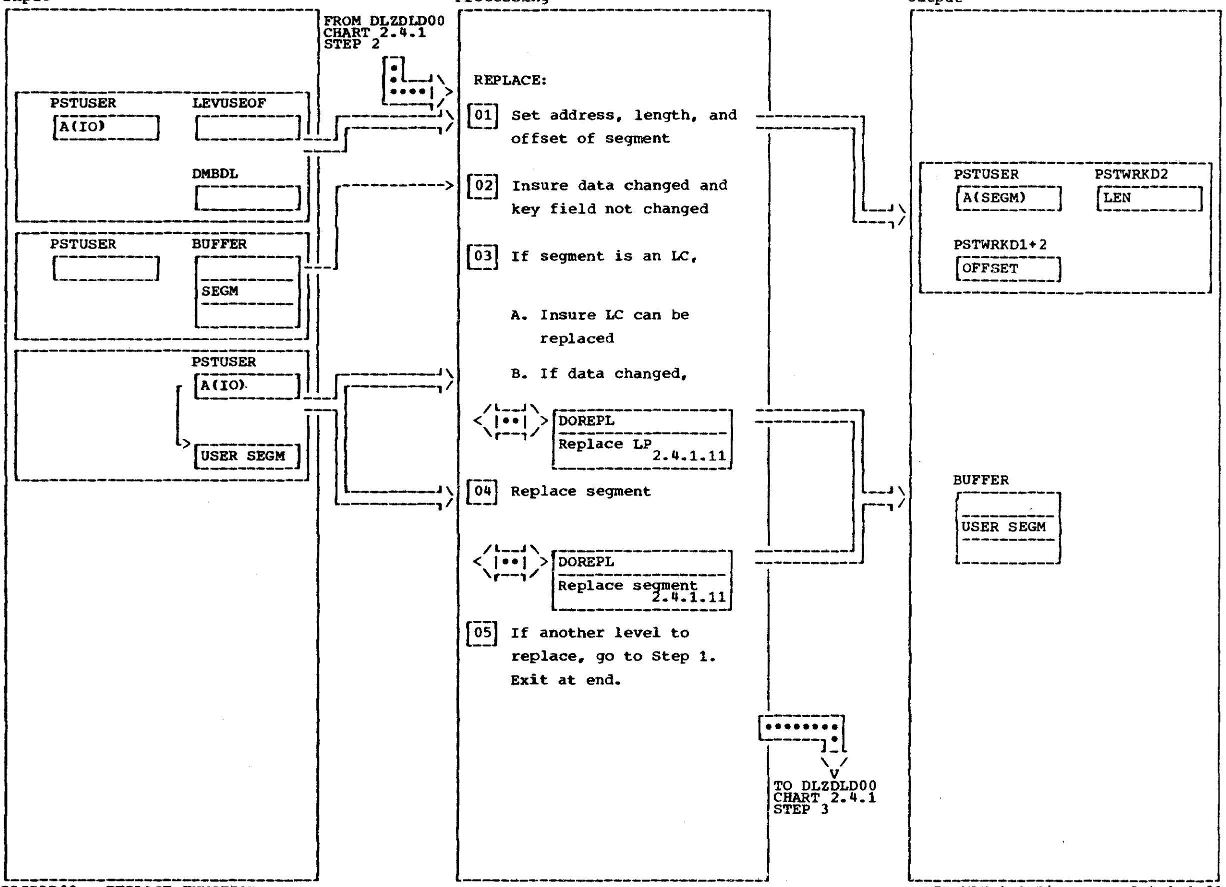
Output

R15
RTN CODE

DLZDLD00 - DLET/REPL-MODULE

HIPOMAT 1.1 Diagram - 2.4.1-01

Notes	Routine	Label	Ref
[01] The segment to be deleted or replaced is identified by contents of JCBLEV1C. Position is established by DLZDLR00 in previous call.	DLZDLD00	REPCK01	
[02]		REPLACE	
[03]		DELETE	
[05] If a user error occurred, DBPCBSTC has return code. If abend, PSTERCD1 has abend code and registers are saved at SCDABSAV+8.		REPLDONE	



Notes	Routine	Label	Ref
01 PSTUSER will have new value if path call had been made. The length is taken from the first two bytes of I/O area if segment is variable length. The offset is from PSTDATA to the segment data in the buffer.		REPLEV03 REPLEV07	
02 Additional logic is needed if segment is variable length or if PROCSEQ is specified.		CHKREPL CHKREPL1	
03			
A. A1 : The following checks are made for the LC :		CHKREPFF CHKRLP	
a) Neither the physical nor logical key fields can be changed (DA status)			
b) If LC retrieved from logical path and rule is physical, RX status code. If rule is logical, no replace and blank status code. If rule is virtual, OK to replace LC			

Notes	Routine	Label	Ref
c) If LC retrieved from physical path, OK to replace LC			
A2 : The following checks are made for the destination parent :			
a) If data didn't change, no replace			
b) If replace rule is physical, RX status. If logical, no change and blank status. If virtual, the key of the LP can not be changed (DA status). The segment can be replaced			
B. This replaces the LP, if LC will also be replaced.		REPPAR01	
04 This replaces normal segment, LC, or LP if LC not changed.		REPFINAL	
05 Another segment in hierarchy can be replaced, if path call.		LEVDONE	

Input		Processing	Output
DMBFLAG .... 1... DELETE BYTE .... ..1.	FROM REPLACE CHART 2.4.1.1	DOREPL:	
		01 If segment is indexed and not marked physically deleted	
		A. Go build work area for Index Maint	DELETE WORK AREA
		B. Call XMT	
		DLZDXMT0 — REPL index pointer(s) 2.5.1	
DBPCBSTC		C. If blank or NE status, continue	
DMBVLDFG .... ..1.		02 If segment is variable length	
		A.	
		REPVLS00 — Replace variable 2.4.1.12	
		B. Go to Step 5 → 05	

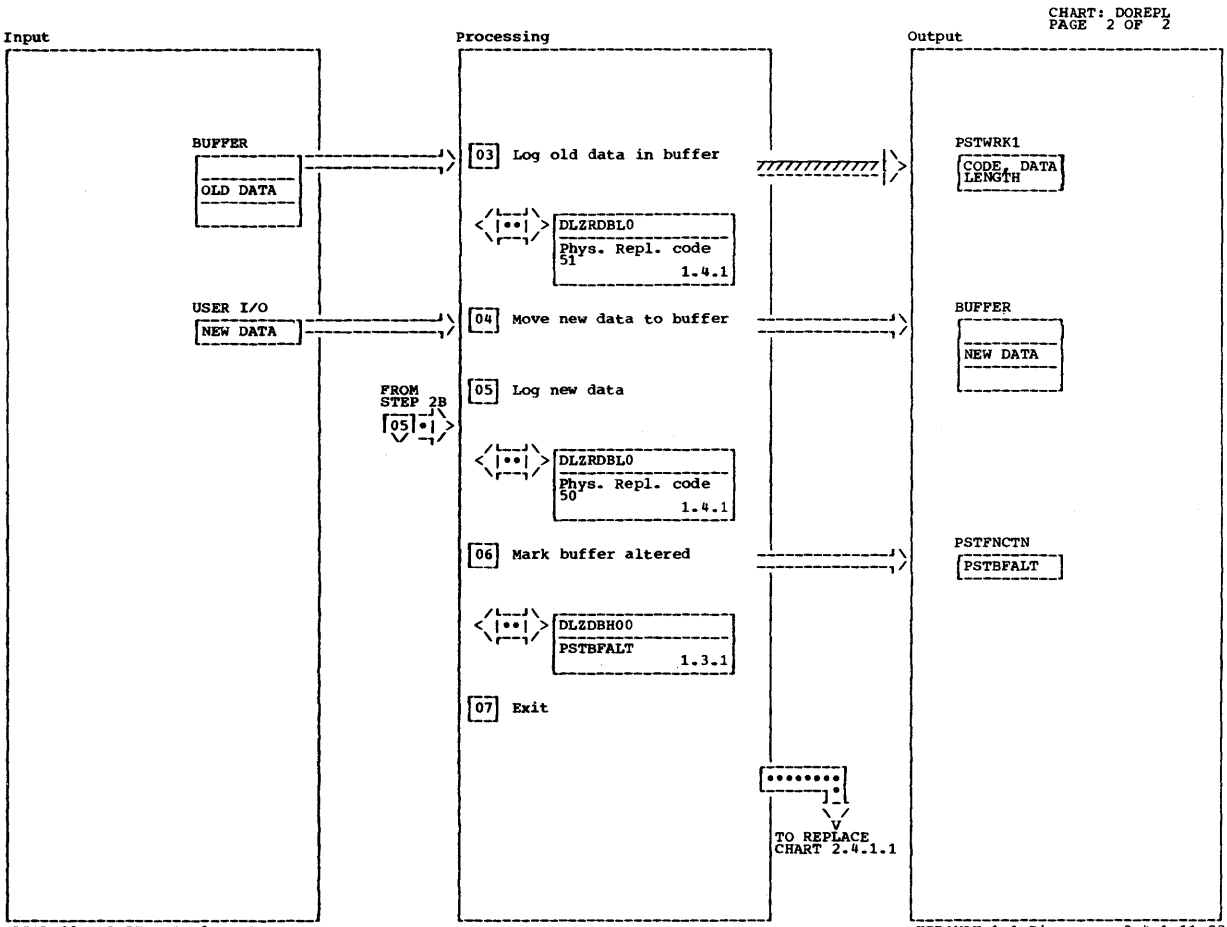
DLZDLD00 - DOREPL REPLACE DATA

HIPOMAT 1.1 Diagram - 2.4.1.11-01

Notes	Routine	Label	Ref
01 Index Maintenance needs the actual concatenated key of this segment. If return code is NE, we continue processing because index is now set as per new data. Work area is freed.		DOREPL DOREPL6	
02		DOREPL10	

DLZDLD00 - DOREPL Replace Data

HIPOMAT 1.1 Diagram - 2.4.1.11-01



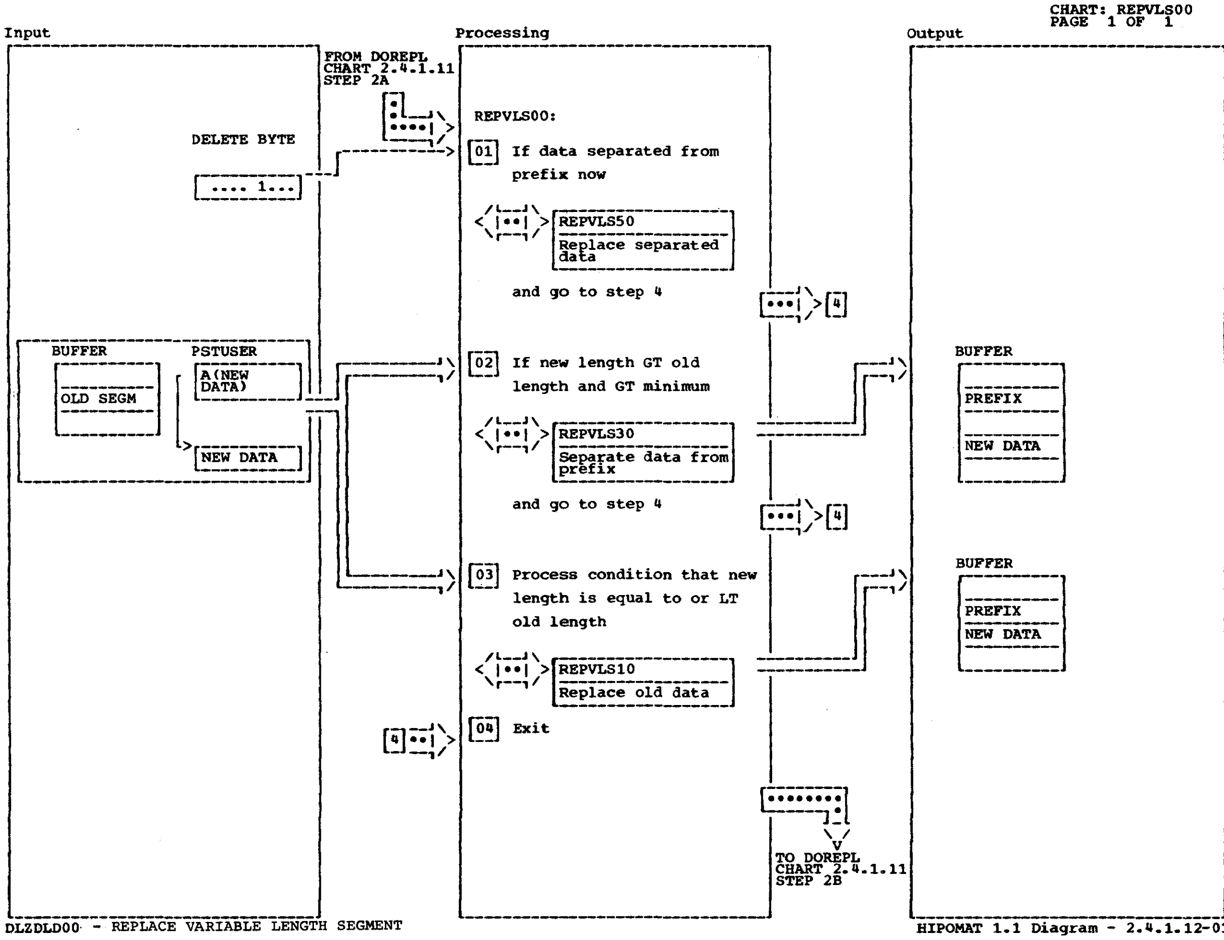
DLZDLD00 - DOREPL Replace Data

HIPOMAT 1.1 Diagram - 2.4.1.11-02

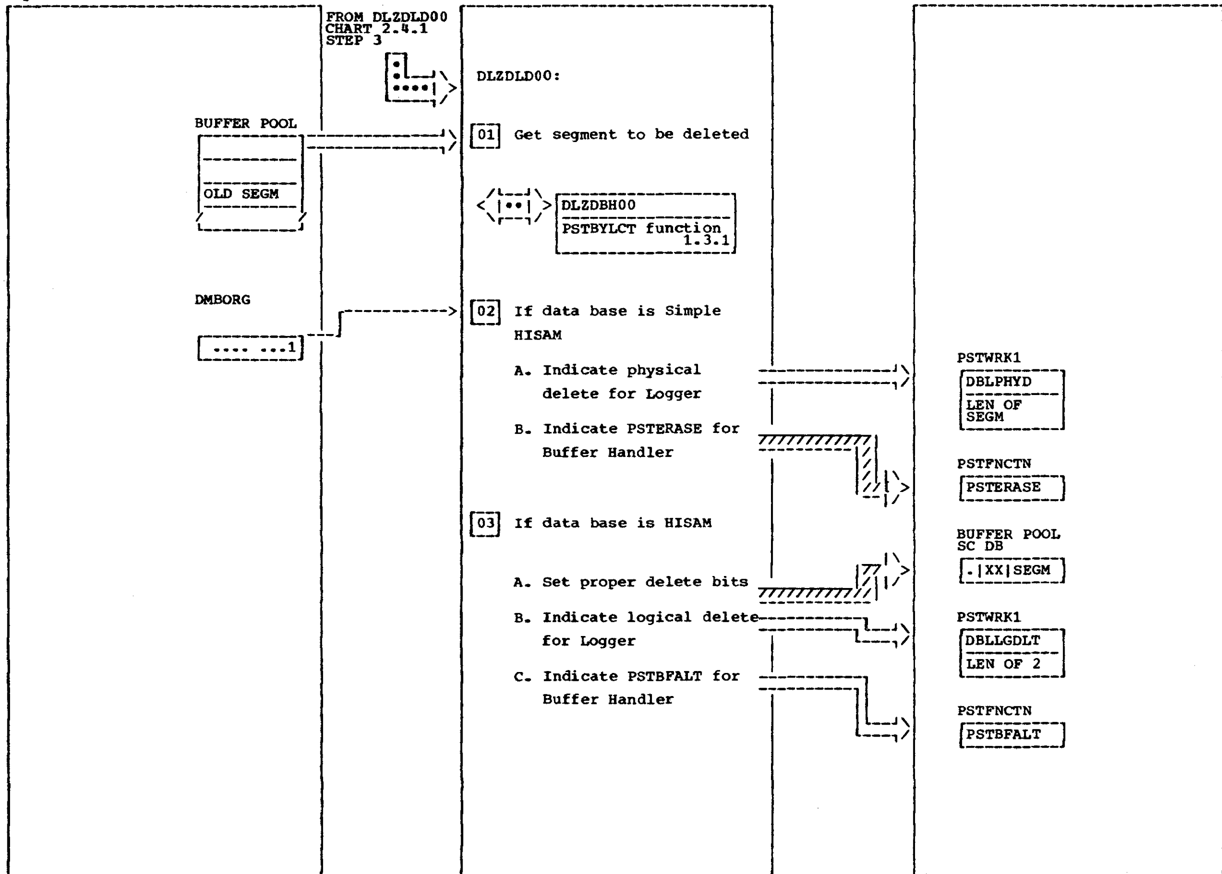
Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
03 DBLPHYR+DBLPHYRO is set in first byte of PSTWRK1.		DOREPL12 DOREPL13					
04 The address of the user's I/O area is in PSTUSER.		DOREPL15					
05 DBLPHYR is set in PSTWRK1 with the length of the segment.		DOREPL92 REPL18					

DLZDLD00 - DOREPL REPLACE DATA

HIPOMAT 1.1 Diagram - 2.4.1.11-02



Notes	Routine	Label	Ref
01 When the data is previously separated and the new data length is LT the old length, an attempt is made to relocate the new data adjacent to the prefix.	DLZDLDR0	REPVLS01	
02 When the old segment size is not large enough for the new segment, the data is separated from the prefix. A pointer overlays the first 4 bytes of the old data and will be used to find the new.		REPVLS03	
03 When the new data will fit in the old location, it is moved over the old data with any excess bytes being freed.		REPVLS10	
04 All changes to the data base have been logged.		REPVLS38	



DLZDLD00 - HISAM DELETE PROCESSING

HIPOMAT 1.1 Diagram - 2.4.1.2-01

Notes	Routine	Label	Ref
01		DELT40	
02 The entire segment to be erased is logged.		SHISAM	
03 Only the segment code and delete byte are logged.		DELT41 LOGDLT	

Notes	Routine	Label	Ref

Input	Processing	Output
	[04] Log the change	LOGOUT
	DLZRDBL0 1.4.1	
	[05] Update the data base	
	DLZDBH00 PSTFNCTN 1.3.1	
	TO DLZDLD00 CHART 2.4.1 STEP 5	

DLZDLD00-HISAM DELETE PROCESSING

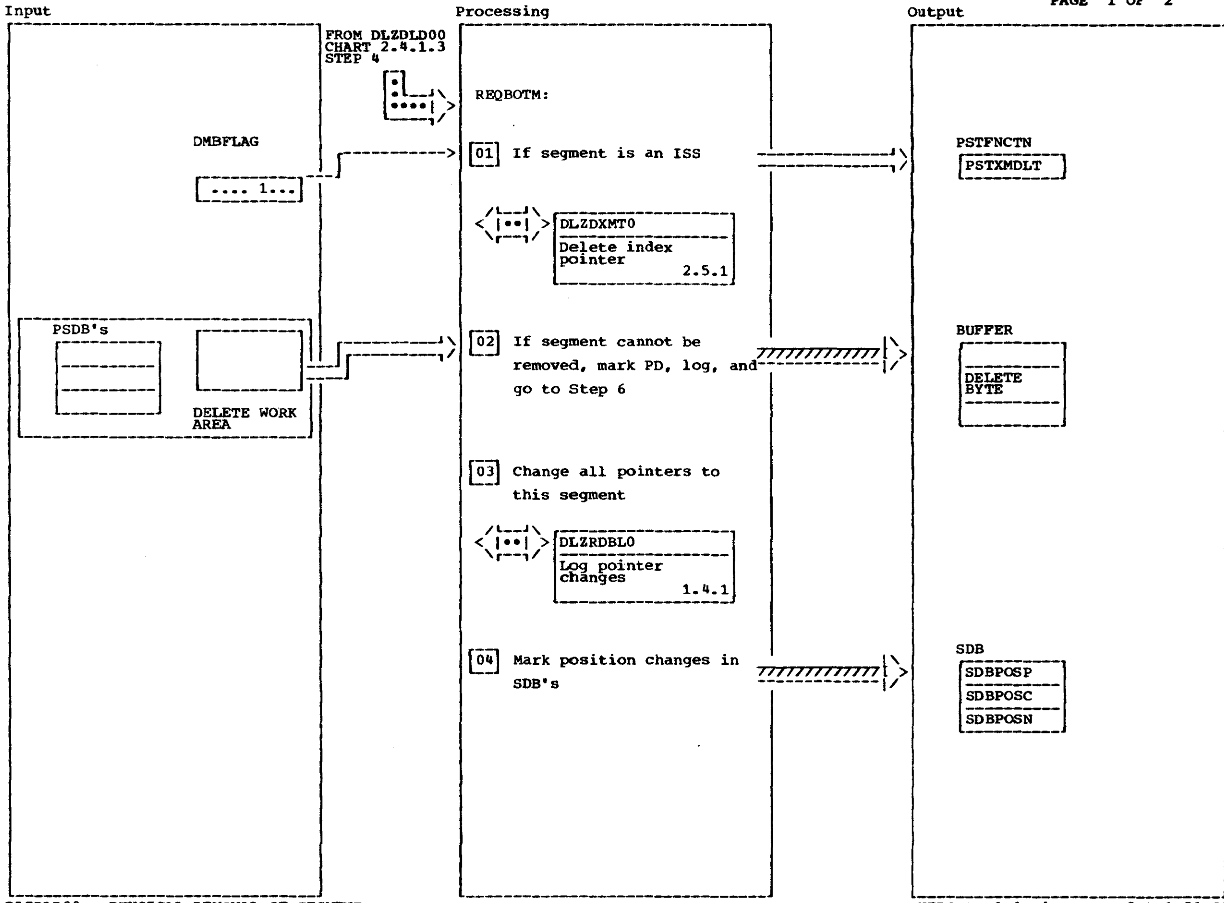
HIPOMAT 1.1 Diagram - 2.4.1.2-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

Input	Processing	Output
PSDB's	FROM DLZDLD00 CHART 2.4.1 STEP 4	
	DLZDLD00:	
	01 Scan PSDB's looking for LC or LP	
	A. If starting segment is an LC retrieved from logical path, mark him LD, if possible	
	B. Insure no violations of the physical delete rule.	
	02 Build workarea for path	DELETE WORKAREA
BUFFER SEGM	03 Read and process all segments from top to bottom. Determine how to delete LC or LP	
	04 At bottom,	BUFFER
	REQBOTM Delete segment(s) 2.4.1.31	
	05 Exit	
	TO DLZDLD00 CHART 2.4.1 STEP 5	

DLZDLD00 - HD DELETE PROCESSING HIPOMAT 1.1 Diagram - 2.4.1.3-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01							
A. LC will be marked logically deleted (LD) if delete rule = physical or logical and segment not PD (physically deleted).	DLZDLD00	PHYSCAN DELT13CK					
B. A logical parent can have no active logical children. An LC must not be accessable by his logical path.							
02 This is needed to remember where we are during scan of data base and to build concatentated keys.		SCANDMB NEWDMB					
03 LCF and LCL pointers in logical parents, LTF and LTB pointers in logical children will be updated now.		DOWN SCNHD REQDOWN					
04 Segments may be only marked deleted, not physically removed.		REQBOTM					
05 All work areas are freed.		ENDLTSCN					



DLZDLD00 - PHYSICAL REMOVAL OF SEGMENT

HIPOMAT 1.1 Diagram - 2.4.1.31-01

Notes	Routine	Label	Ref
01 If the index source segment (ISS) has been marked physically deleted (PD), no index maintenance is performed. Delete processing continues with blank or 'NE' status from DLZDXMT0.	DLZDLDD0	REQB01	
02 A segment will not be physically removed if still required because of a logical relationship. Note that the delete work area and DL/I blocks (primarily PSDB's) are used as input to every step.		REQB02	
03 If segment is an LC or LP, the logical relationship pointers (LC,LP,LT) have already been changed.	DLZDLDD0 DLZDLDA0	FREESPCE FRSPC00	
04 The current position (SDBPOSC) is marked 'lost' in this caller's PCB. If any other PCB has position on this segment, the position should be changed to bypass this segment.	DLZDLDA0	FRSPC05 MARKSDB	

Notes	Routine	Label	Ref

CHART: REQBOTM
PAGE 2 OF 2

Input	Processing	Output
	[05] Free segment's space	PSTFNCTN PSTFRSPC
	DLZDHDS0 Free space 2.6.1	
	[06] If starting segment not yet deleted, back up to parent and go to Step 1	
	[07] Exit	
	TO DLZDLD00 CHART 2.4.1.3 STEP 5	

DLZDLD00 - PHYSICAL REMOVAL OF SEGMENT

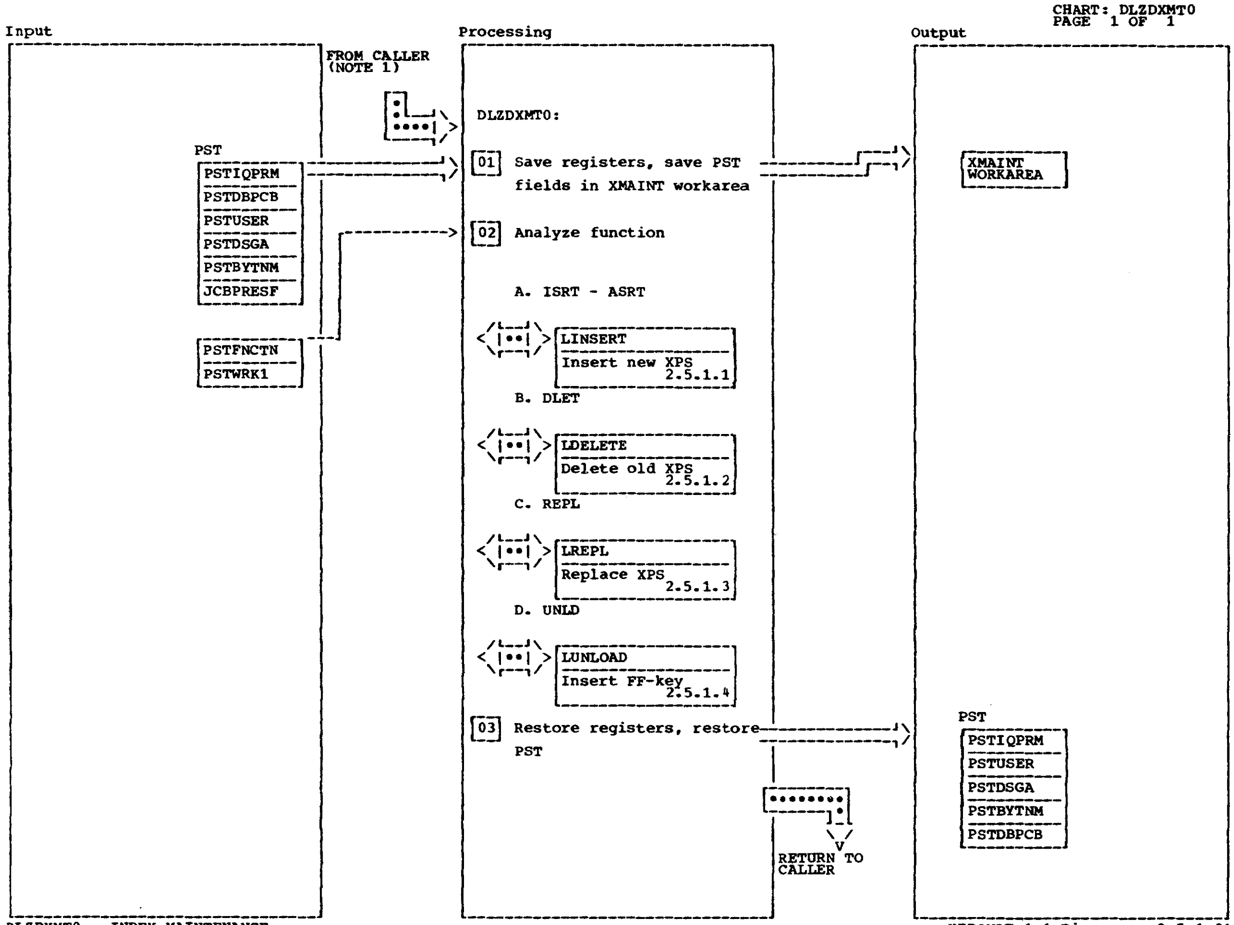
HIPOMAT 1.1 Diagram - 2.4.1.31-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[05] DLZDHDS0 makes the log calls for the physical delete.	DLZDLDA0	FRSPC05G					
[06]	DLZDLD00	BOTM1A					
[07] At end, a final log call is made to DLZRDBL0 which signifies delete is finally accomplished.	DLZDLD00	ENDLTSCN					

DLZDLD00 - PHYSICAL REMOVAL OF SEGMENT

HIPOMAT 1.1 Diagram - 2.4.1.31-02

CHART: DLZDXMT0
PAGE 1 OF 1



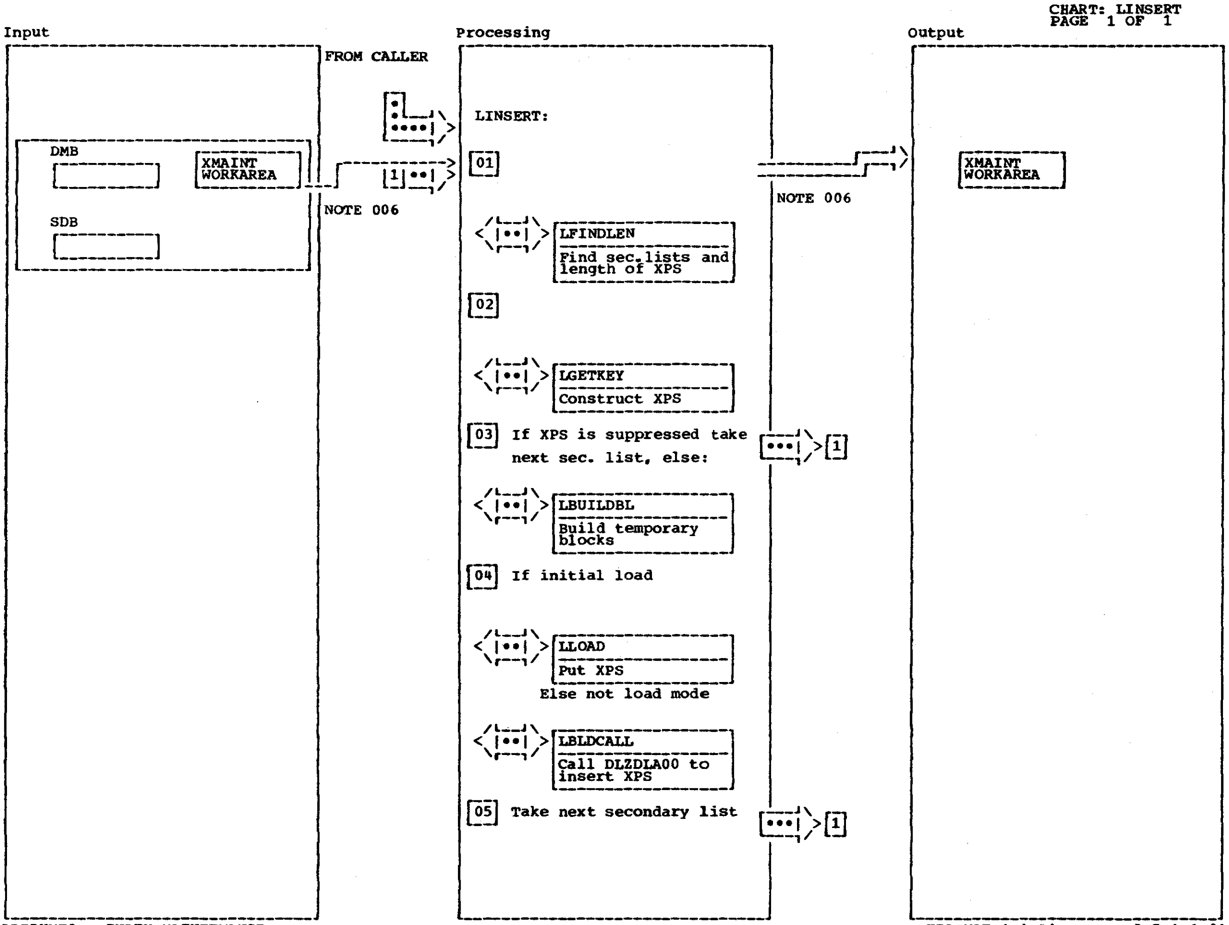
DLZDXMT0 - INDEX MAINTENANCE

HIPOMAT 1.1 Diagram - 2.5.1-01

Notes	Routine	Label	Ref
01 DLZDXMT0 is called from DLZDDLE0 and from DLZDLD00. Index target segment will be abbreviated - XTS - index source segment - ISS - index pointer segment - XPS. 02 Functions are ISRT - ASRT - UNLD - REPL when called from DLZDDLE0. REPL - DLET when called from DLZDLD00. PSTWRK1 contains the PSDB address of the ISS for DLET, else LSDB address of the ISS. A. Construct and insert all XPSs for this ISS, that should not be suppressed . B. Construct and delete all old XPSs existing for this ISS. C. Construct all old and new XPSs that can be constucted from that ISS. Depending on the data changed and the status of suppression -only delete old XPS, -only insert new XPS, -delete old and insert new XPS, - replace data of XPS. D. Write XPS with all FF-keys for all index data bases belonging to this PCB, if DLBL card is provided.			

DLZDXMT0 - INDEX MAINTENANCE

HIPOMAT 1.1 Diagram - 2.5.1-01



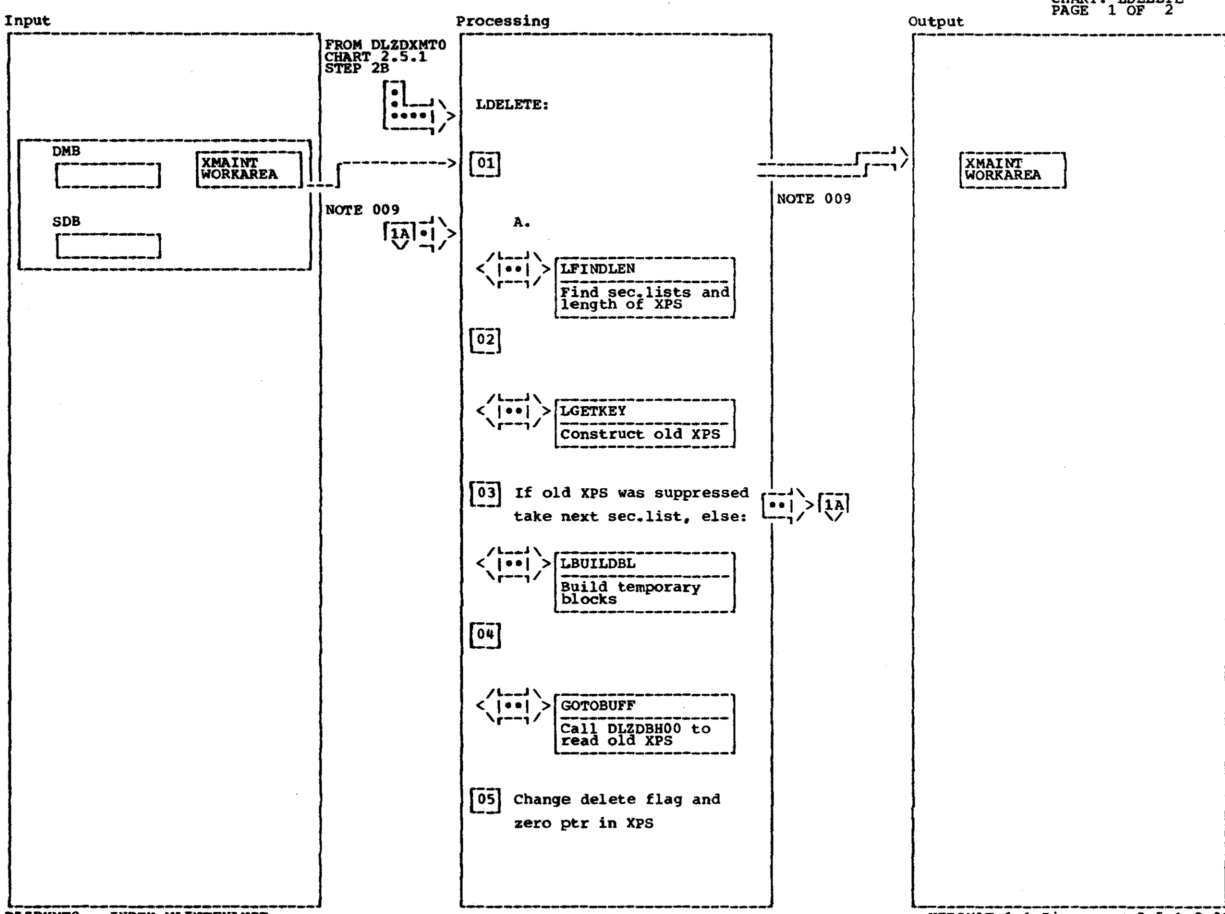
DLZDXMT0 - INDEX MAINTENANCE

HIPOMAT 1.1 Diagram - 2.5.1.1-01

Notes	Routine	Label	Ref
01 Find SECLISTs and PSDBs of ISS,XTS, XPS save their address in XMAINT workarea. Decide if primary or secondary index has to be built. Find length of XPS, sequence field, segment length and protected data length. When last secondary list is reached, exit is to - RETURN1 -. On error in sec. lists exit is to - LABND772 - abend code 772.	LFINDLEN RETURN1 LABND772		
02 For primary indexes move HIDAM root sequence field from user I/O area to workarea. For secondary indexes construct SRCH, SUBSEQ and DDATA fields.	LGETKEY LSYSRLD LSUPRESS LCALLEX GOTOBUFF		
03 When the index entry has to be suppressed due to SRCH equal to NULLVALUE or due to exit routine return code, the XPS is not inserted.			
Build temporary blocks: SDB - segment name = sequence field name of XPS update XMAINT JCB and DSG.	LBUILDBL		
04 Write XPS to index data base, if DLBL cards are provided. Else to workfile, call DLZDLOC0 to open index data base, if not open yet, or DLZDSEH0 to open workfile.	LLOAD LWORKTAP GOTOBUFF DLZDLOC0		
NOT LOAD mode : Prepare DL/I call list to call DLZDLA00 with an *X call.	LBLDCALL DLZDLA00		
06 These DL/I control blocks are used in all process steps as input. XMAINT workarea is referenced and modified in all steps as well.			

DLZDXMT0 - INDEX MAINTENANCE

HIPOMAT 1.1 Diagram - 2.5.1.1-01



DLZDXMT0 - INDEX MAINTENANCE

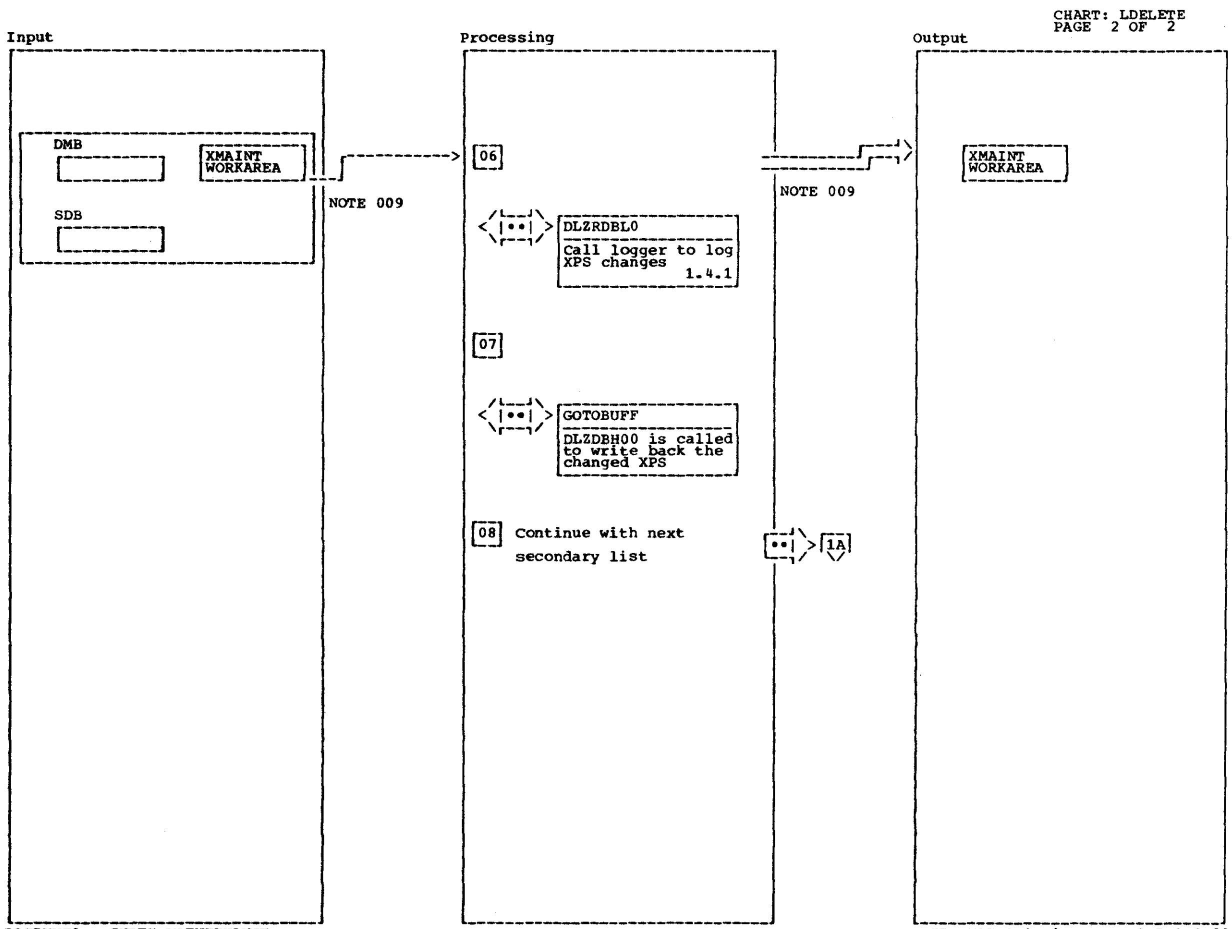
HIPOMAT 1.1 Diagram - 2.5.1.2-01

Notes	Routine	Label	Ref
01 Find SECLISTs and PSDBs of ISS, XTS, XPS. Save their addresses in XMAINT workarea. Decide if primary or secondary index has to be built. Find length of XPS, sequence field, segment length and protected data length. When last secondary list is reached, exit is to - RETURN1 -. On error in sec. lists, exit is to - LABND772 - abend code 772.	LFINDLEN RETURN1 LABND772		
02 For primary indexes move HIDAM root sequence field from DLZDLD00's workarea. For secondary indexes construct SRCH, SUBSEQ and DDATA fields.	LGETKEY LSYSRLD LSUPRESS LCALLEX GOTOBUFF		
03 When the old index entry was suppressed due to SRCH equal to NULLVAL or due to exit routine return code, process continues with step1.			
Build temporary blocks: SDB - segment name = sequence field	LBUILDBL DLZDLOC0		

Notes	Routine	Label	Ref
name of XPS. Update XMAINT JCB and DSG. Open index data base if not yet open calling DLZDLOC0.			
04 The buffer handler is called - PSTSTLEQ- to find the old XPS. If it is not found, or it is already deleted, or the pointer or key are not correct, an - NE- status code is returned to the caller.			
05 Delete flag is set to - C0-.			

DLZDXMT0 - INDEX MAINTENANCE

HIPOMAT 1.1 Diagram - 2.5.1.2-01



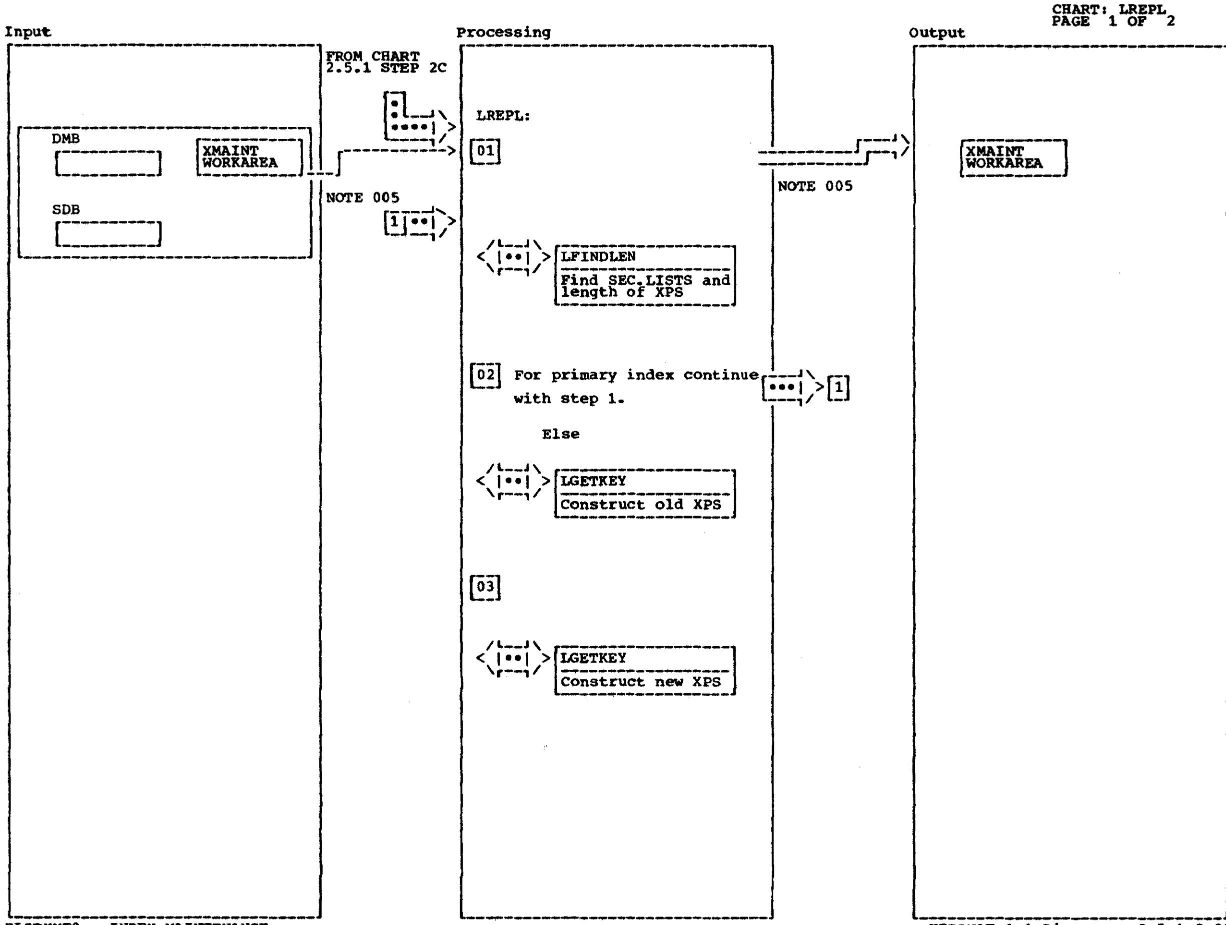
DLZDXMTO - INDEX MAINTENANCE

HIPOMAT 1.1 Diagram - 2.5.1.2-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
06 Chain maintenance and logical delete calls are made to data base log module.	DLZRDBL0						
07 The buffer handler is called - PSTBFALT- to write the changed XPS back.							
09 These DL/I control blocks are used in all processing steps. The XMAINT workarea is referenced or modified in all steps as well.							

DLZDXMTO - INDEX MAINTENANCE

HIPOMAT 1.1 Diagram - 2.5.1.2-02



DLZDXMT0 - INDEX MAINTENANCE — HIPOMAT 1.1 Diagram - 2.5.1.3-01

Notes	Routine	Label	Ref
01 Find SECLISTS and PSDBS of ISS,XTS, XPS save their address in XMAINT workarea. Decide if primary or secondary index has to be built. Find length of XPS, sequence field, segment length and protected data length. When last secondary list is reached, exit is to - RETURN1 -. On error in SEC.LISTS exit is to - LABND772 - abend code 772	LFINDLEN RETURN1 LABND772		
02 Construct old XPS from SRCH, SUBSEQ and DDATA fields	LGETKEY LSYSRLD LSUPRESS LCALLEX GOTOBUFF		
03 Construct new XPS from SRCH, SUBSEQ and DDATA fields. Entry to LGETKEY is at LKEY1C	LGETKEY LSYSRLD LUPRESS LCALLEX GOTOBUFF	LKEY1C	

Input

DMB

XMAINT WORKAREA

SDB

NOTE 005

Processing

[04] Replace XPS

NOTE 005

A. If old and new XPS are suppressed, take next SEC.LIST → [01]

B. Old XPS was suppressed - only insert new XPS

LINSERT
Insert new XPS

C. Old XPS was not suppressed, SRCH and SUBSEQ fields not changed - replace XPS

D. The old XPS was not suppressed and SRCH or SUBSEQ fields were changed

LDELETE
Delete old XPS

LINSERT
Insert new XPS

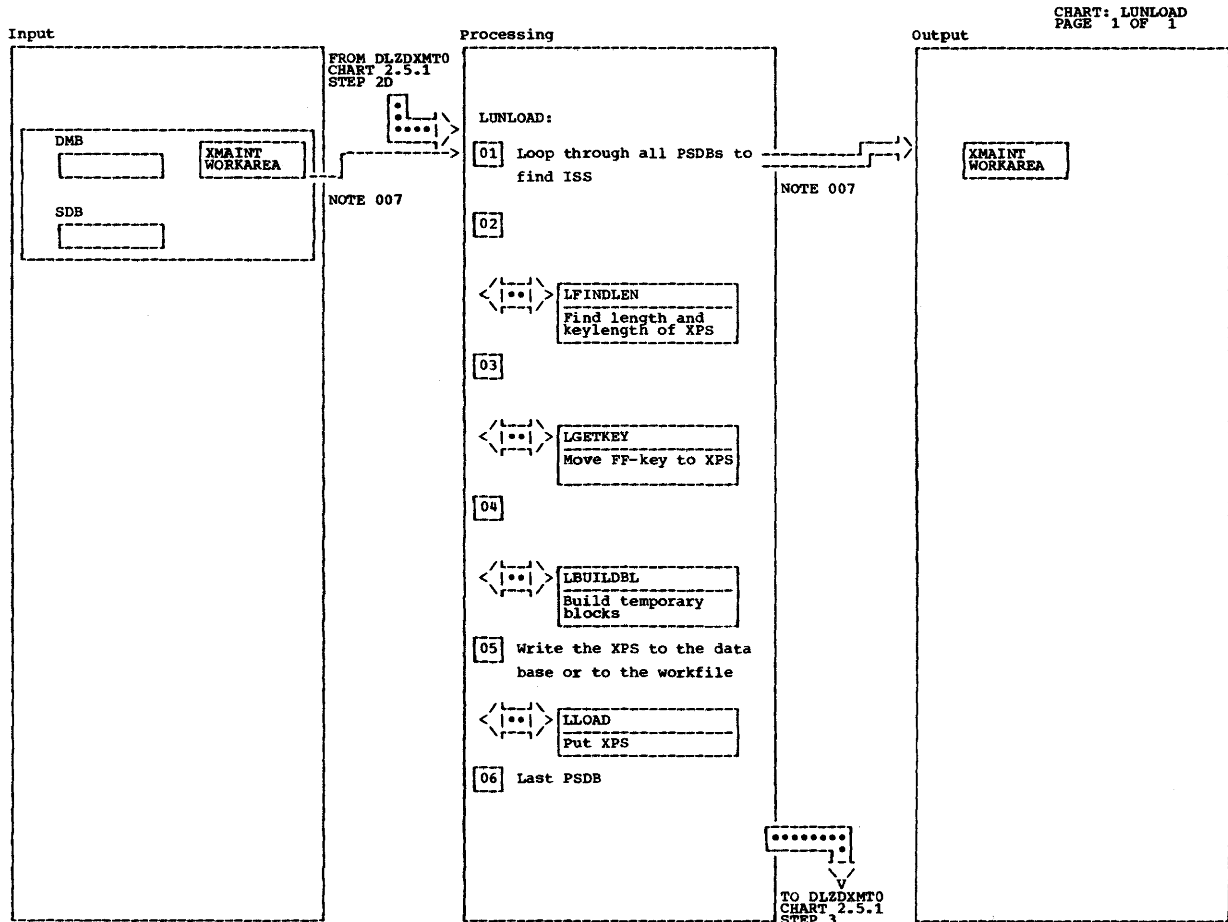
[05] Continue with next SEC.LIST → [01]

Output

XMAINT WORKAREA

Notes	Routine	Label	Ref
[04] Replacing of XPS is done in different ways, depending on suppression of old and new XPS			
A. When both old and new XPS are suppressed no action takes place			
B. Continue with insert subroutine, label - LINSERT3	LINSERT	LINSERT3	
C. DLZDBH00 is called to read the old XPS. On errors - NE - is returned. The data base log module is called, to log the old XPS and after the change of the DDATA fields, the new XDS. DLZDBH00 is called again, to write the XPS back - PSTBFALT	DLZDBH00 DLZRDBL0 LDELETE	LDELETE2 LREPLR1	
D. The LDELETE subroutine is called to delete the old XPS. Entry is at -LDELETE2-. Then the - LINSERT -routine is called to insert the new XPS, entry at - LINSERT3	LDELETE LINSERT	LDELETE2 LINSERT3	

Notes	Routine	Label	Ref
[05] These DL/I control blocks are used in all processing steps. The XMAINT workarea is referenced and/or modified in all steps as well			



DLZDXMT0 - INDEX MAINTENANCE

HIPOMAT 1.1 Diagram - 2.5.1.4-01

Notes	Routine	Label	Ref
01 DLZDDLE0 passes the LSDB address of the root segment with an UNLD call. The DDIR address is used and all PSDBs in that DMB are inspected if an index exists.			
02 Find length of XPS and its key length. Decide if primary or secondary index has to be built.	LFINDLEN	LFINDL1	
03 Move FF's in the length of the XPS sequence field to the XPS.	LGETKEY		
04 Build temporary blocks: SDB - segment name = sequence field name of XPS. Update XMAINT JCB and DSG.	LBUILDBL		
05 Write XPS to index data base, if DLBL cards are provided. Call DLZDLOC0 to open index data base, if not yet open.	LLOAD LWORKTAP GOTOBUFF DLZDLOC0		
07 These DL/I control blocks are referenced in all process steps.			

Notes	Routine	Label	Ref
XMAINT workarea is referenced and modified in all steps as well.			

DLZDXMT0 - INDEX MAINTENANCE

HIPOMAT 1.1 Diagram - 2.5.1.4-01

CHART: DLZDHDS0
PAGE 1 OF 1

Input

FROM CALLER

DSG
DSGDMBNO

PST
PSTWRK1

PST
PSTFNCTN

Processing

DLZDHDS0:

[01] Initialize work fields in the PST

[02] What function is requested?

A. Get Space (chart 2.6.1.1)

B. Get Space close to root Anchor Point (chart 2.6.1.1)

C. Free Space (chart 2.6.1.2)

D. Modify the Bit Map (chart 2.6.1.3)

E. Backout Get Space (chart 2.6.1.4)

F. Backout Free Space (chart 2.6.1.2)

G. Backout Modify Bit Map (chart 2.6.1.3)

[03] Exit

RETURN TO CALLER

Output

PST
PSTDMBNM
PSTACBNM
USPCE
UMAX

DLZDHDS0 - SPACE MANAGER   HIPOMAT 1.1 Diagram - 2.6.1-01

Notes	Routine	Label	Ref
[01] PSTWRK1 contains the length of the space to be obtained or freed.			
[02]			
A. Get space in a data base CI for the specified segment as close as possible to a specified base RBA. The caller passes the address of the involved segment's PSDB in reg. 5 and the base RBA in PSTBYTNM.	DLZGGSPC		
B. Get space in a data base CI for the specified segment as close as possible to a root anchor point. The caller passes the address of the involved segment's PSDB in reg. 5 and the CI nr /RAP nr (in the format BBBR) of the involved root anchor point in PSTBYTNM.	DLZGGSPC		
C. Free space that has been allocated for the specified segment in a data base CI. The caller passes the address of the involved segment's PSDB in reg.5.	DLZFRSPC		
D. Turn on or off the bit in the Bit Map representing the specified CI of a data base. The caller specifies the CI nr in PSTBLKNM.	DLZDHDS0	FIXBTMP	
E. Backs out a previously processed 'Get Space' call.	DLZDHDS0		
F. Backs out a previously processed 'Free Space' call.	DLZFRSPC		
G. Backs out a previously processed 'Modify Bit Map' call.	DLZDHDS0	FIXBTMP	

DLZDHDS0 - SPACE MANAGER   HIPOMAT 1.1 Diagram - 2.6.1-01

Input | Processing | Output

FROM DLZDHDS0
CHART 2.6.1
STEP 2 A,B

DLZGGSPC:

PST
PSTBYTNM

01 Get the CI into the buffer pool, which is pointed to by the base RBA

DL/I BUFFER

DLZDBH00
1.3.1

DATA BASE CI
FSE
FSE

02 Check if there is enough space in this CI

CI NR 1
BIT MAP

If enough space is avail., store RBA of space.If nec., update the Bit Map.Go to step 4

If enough space is not avail., continue with the next foll. substep of step 3

PST
PSTBYTNM
CI NR 1
BIT MAP

03 Locate another data base CI and get it into the buffer pool using the following strategy:

BUFFER PREFIXES

A. A CI on the same track that is in the buffer pool. Go to step 2

B. A CI on the same cyl that is in the buffer pool. Go to step 2

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
For the functions 'Get Space', 'Get Space close to RAP' ,the following Csects are used: Main routine: DLZDHDS0 DLZDHDS0 calls DLZGGSPC DLZGGSPC calls DLZRCHBK DLZLLCLC DLZRRHPL DLZRRHMP DLZMMLCT DLZMMUDT DLZRRHPL calls DLZRCHBK DLZRRHMP calls DLZMMLCT 02 To check,if enough space is available in a CI,the FSE's in this CI are checked.If there are more than one FSE in a CI,the free space with the largest of the three following values that will not cause a Bit Map change is taken: the size itself,the size+minimum segm length, the size*2. A Bit Map change is necessary, if the data base CI cannot accomodate the				maximum size segment any more, when an available space is used.The Bit Map update is performed by routine DLZMMUDT. 03 The calculation of the CI nr's for a given range is done by routine DLZLLCLC. A. The searching thru the buffer prefixes is done by routine DLZRRHPL.			

Input

CI NR 1
BIT MAP ----> [B]

Processing

[B] ----> C. A CI on the same track that has a 1-Bit in the Bit Map. Go to step 2

[B] ----> D. A CI in the same cylinder that has a 1-Bit in the Bit Map. Go to step 2

[A] ----> E. A CI being within the DELTA CYLINDERS that is in the buffer pool. Go to step 2

[B] ----> F. A CI being within the DELTA CYLINDERS that has a 1-bit in the Bit Map. Go to step 2

G. The next available CI at the end of the data base. Go to step 2

H. If no space is found, store error code ----> PST PSTRTCDE

[04] Exit

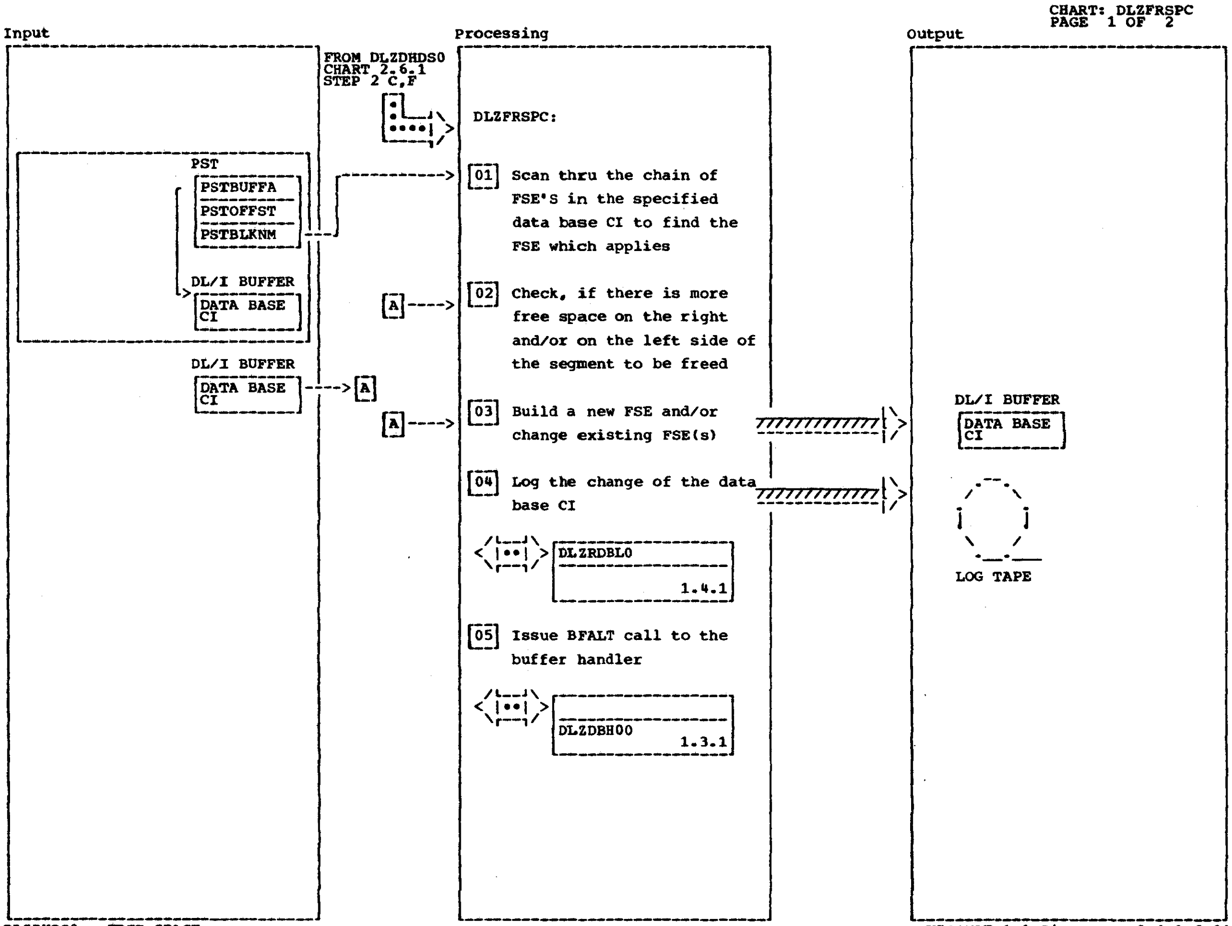
TO DLZDHDS0
CHART 2.6.1
STEP 3

Output

PST
PSTRTCDE

DLZDHDS0 - GET SPACE HIPOMAT 1.1 Diagram - 2.6.1.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
C. The searching thru the Bit Map(s) is done by routine DLZRRBMP.							
H. A return code of X'0C' will be returned to the caller.							



DLZDHDS0 - FREE SPACE HIPOMAT 1.1 Diagram - 2.6.1.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
For the functions 'Free Space' and 'Backout Free Space' the following csects are used: Main routine: DLZDHDS0 DLZDHDS0 calls DLZFRSP0 DLZFRSP0 calls DLZMMLCT and DLZMMUDT 01 The scan will be finished when an FSE with a higher offset than the one in PSTOFFST is reached or when the end of the FSE chain is reached. 02 The purpose of this check is to find out whether there will be a contiguous piece of free space after processing the current free space call.							

Input

CI NR 1

BIT MAP

Processing

06 If a Bit Map update is necessary update the Bit Map

07 Exit

TO DLZDHDS0
CHART 2.6.1
STEP 3

Output

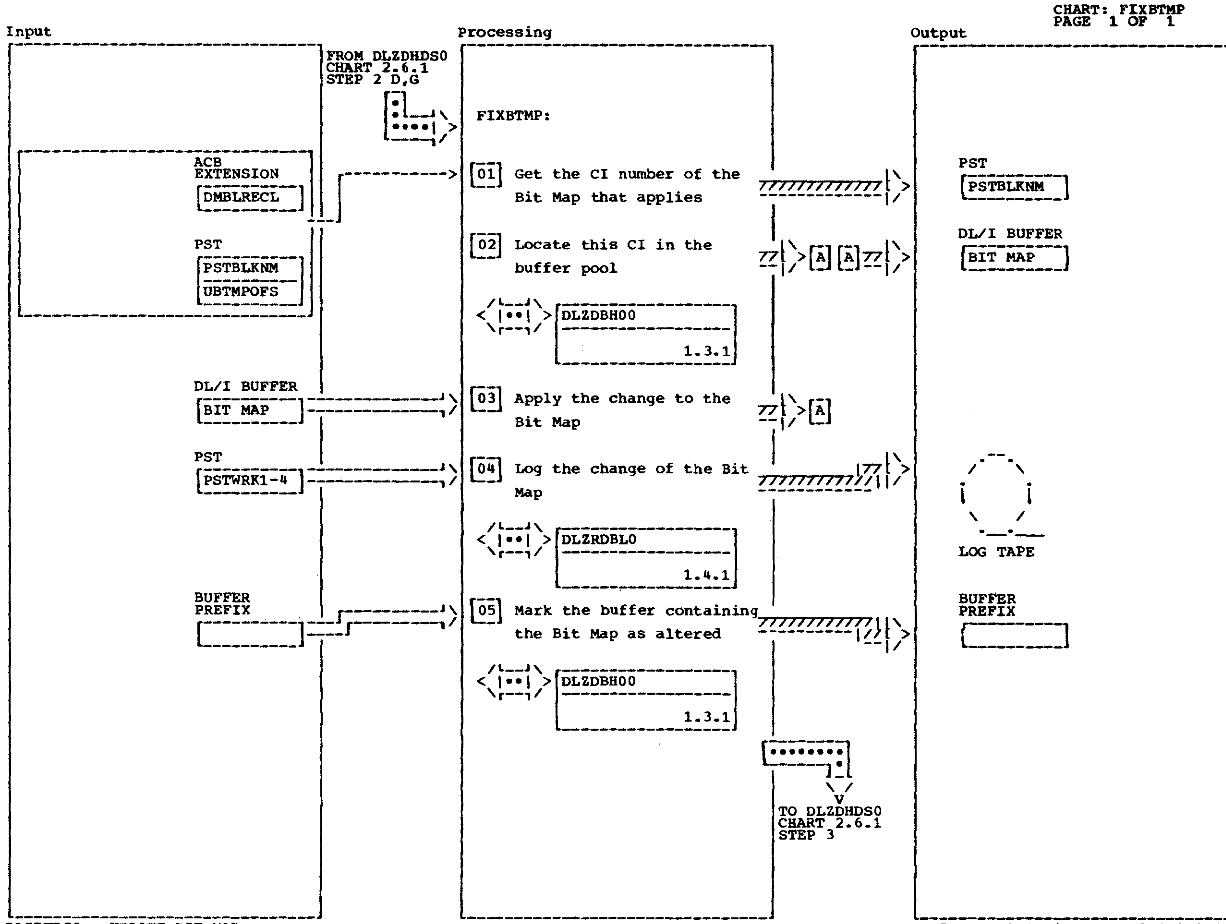
CI NR 1

BIT MAP

DLZDHDS0 - FREE SPACE

HIPOMAT 1.1 Diagram - 2.6.1.2-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
06 A Bit Map change is necessary, if the data base CI can accomodate the maximum size segment after processing the Free Space call. In this case the appropriate bit in the bit map has to be turned on. The bit map update is performed by routine DLZMMUDT.							



DLZDHDS0 - UPDATE BIT MAP

HIPOMAT 1.1 Diagram - 2.6.1.3-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
For the functions 'Fix Bit Map' and 'Backout Fix Bit Map' the following csects are used:							
Main routine : DLZDHDS0							
DLZDHDS0 calls DLZMMLCT and DLZMMUDT							
01 This step is performed by routine DLZMMLCT.							
02 A 'Byte Locate' call is issued.	DLZDBH00		CHART 1.3.1				
03 The update of the Bit Map is performed by routine DLZMMUDT.							
05 A 'Buffer Alter' call is issued.	DLZDBH00		CHART 1.3.1				

DLZDHDS0 - UPDATE BIT MAP

HIPOMAT 1.1 Diagram - 2.6.1.3-01

Input

FROM DLZDHDS0 CHART 2.6.1 STEP 2 E

PST: PSTBUFFA, PSTOFFST

DL/I BUFFER

PST: PSTOFFST, AOFF

DL/I BUFFER

Processing

[01] Scan thru the FSE's of the given CI and find the FSE that applies

[02] If the previously freed space cannot be found,store error code and go to step 9

[03] If the previously freed space is not in the middle of a freed area,go to step 5

[04] Create a dummy FSE

[05] Change the FSE(s) to reflect the acquisition of the space

[06] Prepare the information for logging the data base change

Output

AOFF

PST: PSTRTCDE

DL/I BUFFER

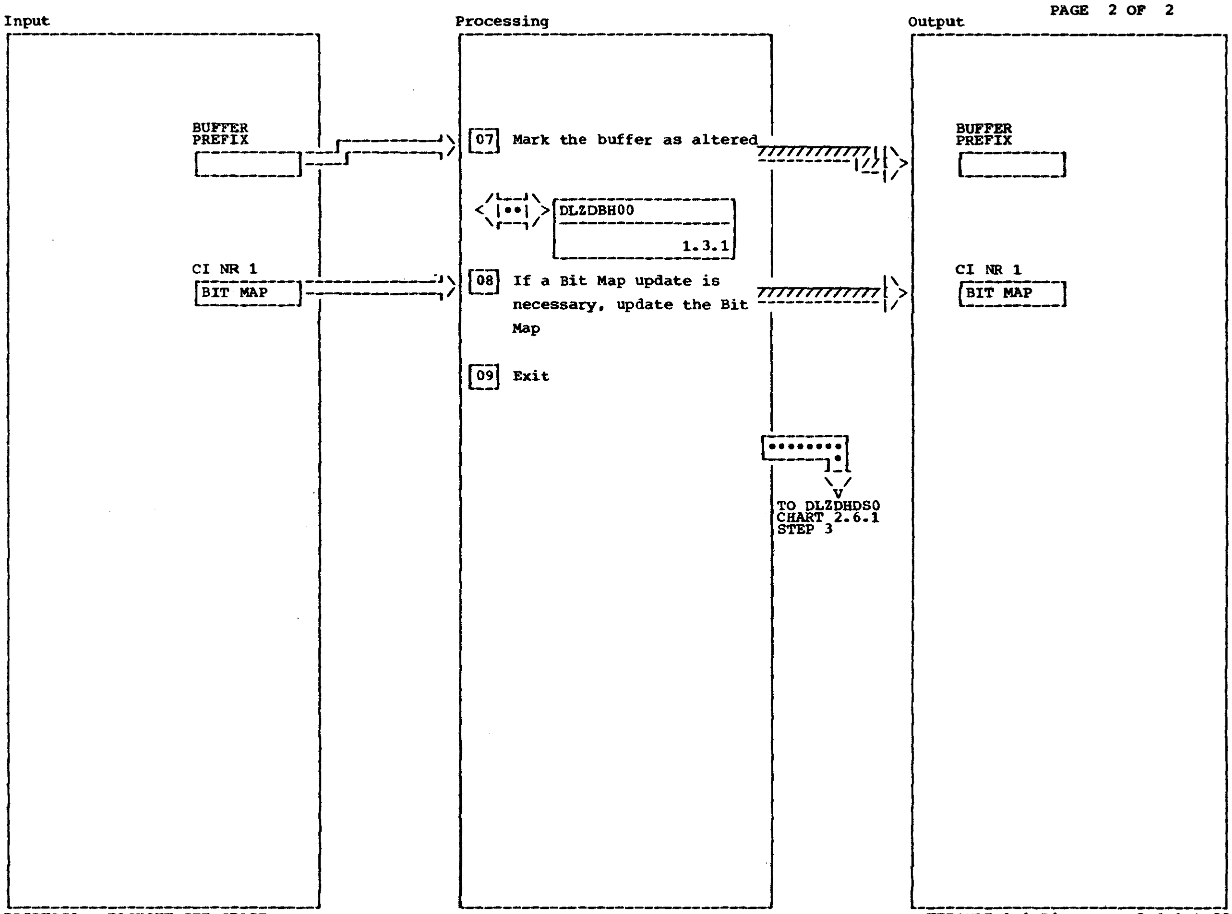
PST: PSTWRK1-4

DLZDHDS0 - BACKOUT GET SPACE

HIPOMAT 1.1 Diagram - 2.6.1.4-01

Notes	Routine	Label	Ref
[00] This function backs out a Free Space call processed previously. The following csects are used: Main routine : DLZDHDS0. DLZDHDS0 calls DLZRCHBK.			
[02] PSTOFFST contains the offset to the part of the data base CI which was freed during the Free Space call to be backed out.			
[03] A returncode of X'0C' is stored in PSTRTCDE.			

Notes	Routine	Label	Ref

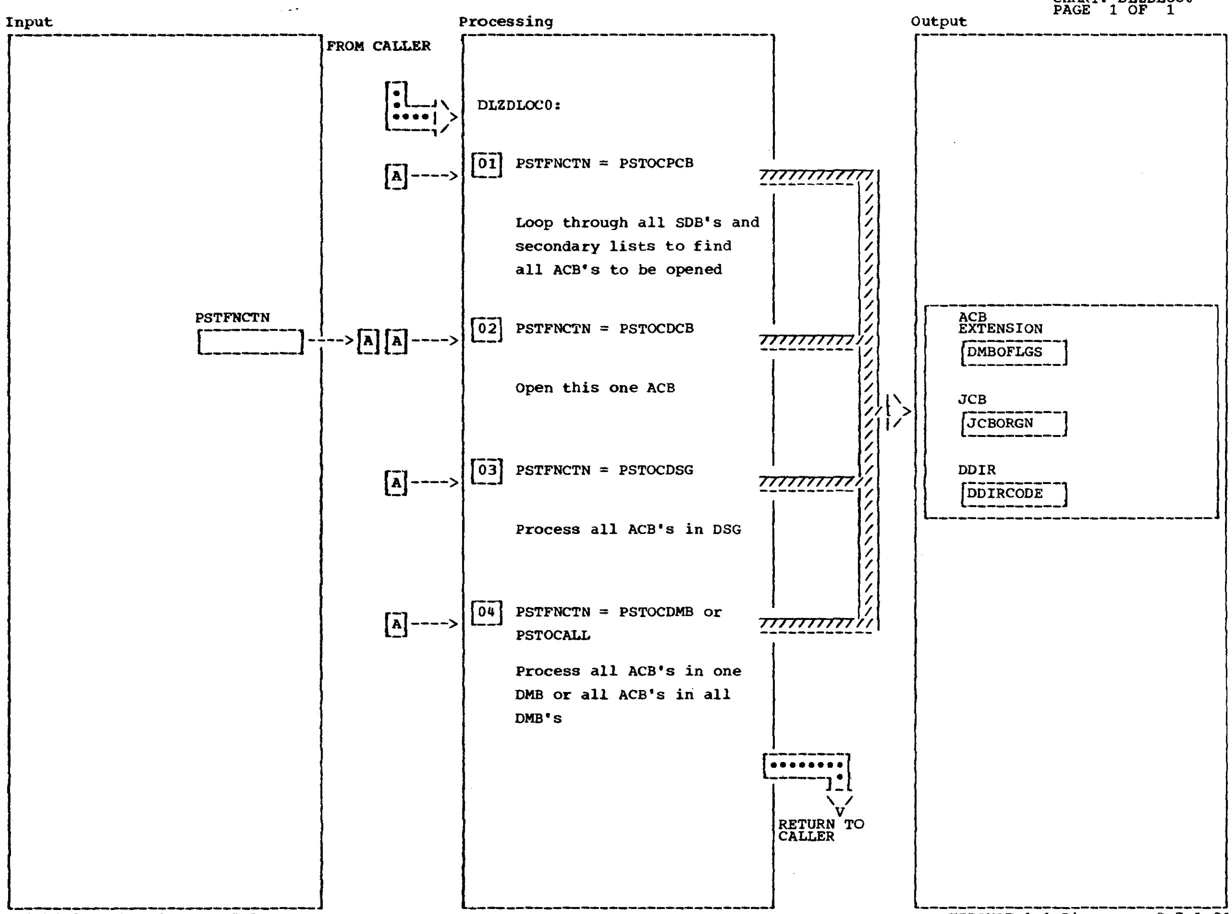


DLZDHDS0 - BACKOUT GET SPACE

HIPOMAT 1.1 Diagram - 2.6.1.4-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
07 A 'Buffer Alter' call is issued to the buffer handler.	DLZDBH00						
08 A Bit Map update is necessary,if the data base CI cannot accomodate the maximum length segment any more after backing out the previously processed Free Space call.The Bit Map update is performed by routine DLZMMUDT.							

CHART: DLZDLOC0
PAGE 1 OF 1



DLZDLOC0 - OPEN/CLOSE MODULE

HIPOMAT 1.1 Diagram - 2.7.1-01

Notes	Routine	Label	Ref
In all steps the subroutine DOCDCB chart 2.7.1.1 is called.	DOCDCB		
01 This function is used by DLZDLA00 :	PCENTRY PSROUT DOCDCB		
When the first data base call to a not open data base is issued (batch only). For PROCOPT=L only one data base is opened. For other processing options all related data bases are opened as well (index data bases, logically related data bases). This call is also used by the utilities DLZRDBC0 and DLZURGP0.			
02 DLZDLR00 uses this function for positioning an HSAM data base at the start point. It is also used by DLZURDB0. It opens only one ACB, i.e. for HISAM only KSDS or ESDS.	ACBENTRY DOCDCB		
03 DLZDLD00 uses this function when it finds a logically related data base, that is not opened (this can happen because of delete sensitivity propagation).	DGENTRY DOCDCB		
04 PSTOCALL + PSTOCOPN : DLZOLI00 uses this call to open all data bases in the system eligible for initial opening (online only).	DENTRY DROUTINE DOCDCB		
PSTOCALL + PSTOOCLS : is used to close all ACB's in the DL/I system (e.g. DLZDLA00).			
PSTOCDMB : is used by DLZOLI00 for deferred opening (online), by DLZDXMT0 and by data base utilities. It opens/closes one ACB but two ACB's for HISAM.			

Input	Processing	Output
R11: ACB extension / ACB	FROM DLZDLOC0 CHART 2.7.1	
	DOCDCB - OPEN	
	01 Issue DOS open	ACB EXTENSION: DMBOFLGS
	02 Issue 'SHOWCB' and compare DMB entries to VSAM define entries	
	03 Issue 'MODCB' to update the exit list	
	04 For an empty ESDS file: write control record	
	05 Log the open record	
	06 For HISAM KSDS go back to step 1 and open ESDS	
	07 Call compression routine if necessary	
	TO DLZDLOC0 CHART 2.7.1	

DLZDLOC0 - DOCDCB OPEN

HIPOMAT 1.1 Diagram - 2.7.1.1-01

Notes	Routine	Label	Ref
01 This part is called from all steps of chart 2.7.1 for opening. If the data base is open, return immediately, also when not planned for initial opening and call is PSTOCALL. Unsuccessfull opens have return error code in PST and flag in JCB. 'DLZ020' is issued.	DOCDCB	DOCOPEN	
02 Control interval size, relative key position and key length of DMB is compared to VSAM catalog entries. MISMATCH : DLZ025, DLZ027, and DLZ028. For HISAM the number of logical records in VSAM catalog has to be zero for PROCOPT=L and greater than zero for PROCOPT=L. For HD the high used RBA is inspected. Message 'DLZ023' is issued for conflicts.	DOCSHOW		
03 The exit list is updated with the address of the error handling routines of DLZDBH00.	DOCMOD		
04 The first control interval is written. (For HISAM as many records as fill one CI). It contains DL/I control information. For HD the ACB is closed and opened again to simulate 'NOT LOAD' to VSAM.	DOCFIRST DLZDBH00	DOCOPEN1	
07 All PSDBs are inspected to determine if a compaction routine with 'INIT' specified exists.	DOCVARI		

Input

FROM DLZDDLE0 CHART 2.3.1.7

PARM LIST(R3)

- CTR
- LTF PTR
- LTB PTR
- LP PTR
- A(LPCK)
- A(CDS REC)

DMBSCDE

1... ....

DMBSEC

CONTROL D.S.

Processing

DLZDSEH0:

01 Perform initialization functions

Initialization and opens 2.8.1.1

DLZURGS0 CHART 4.6.1.2

02 If DLZURGS0 is caller,

A. Initialize dummy reload prefix header with parameters

B. Store its address

03 If end of secondary list, go to Step 7 (07)

04 Match control d.s. entry with LC or LP sec. list entry

Output

IOAREA

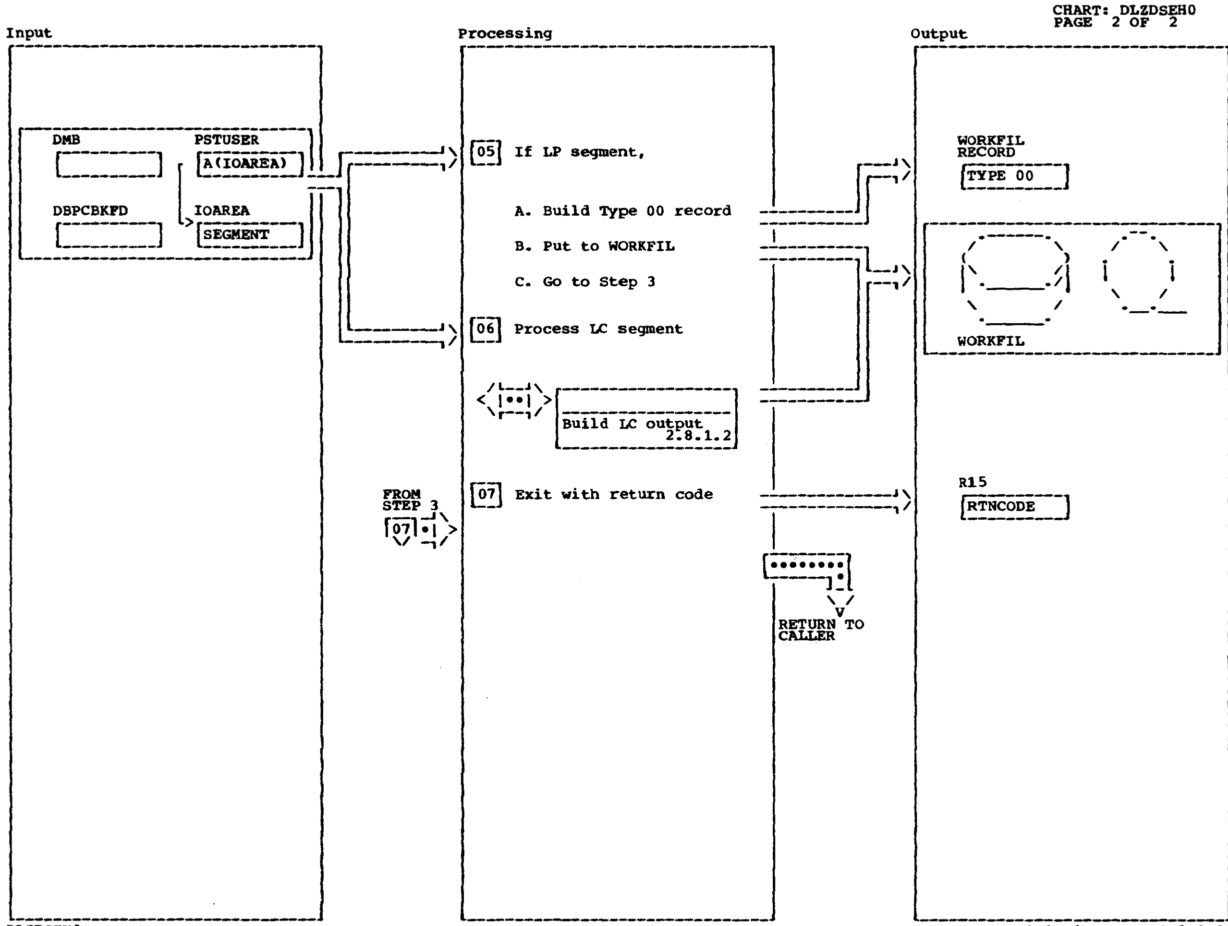
PSTUSER

A(IOAREA)

DLZDSEH0 HIPOMAT 1.1 Diagram - 2.8.1-01

Notes	Routine	Label	Ref
01 This primary entry point is used by load/insert when a data base is being initially loaded or reloaded. There are 7 fullwords of addresses immediately preceding this entry point used by modules that interface with DLZDSEH0. A logical parent or logical child record is input to this module.	DLZDSEH0	INIT	
02 This is the primary entry point for the scan utility.		TEST	
03 This routine must be re-entered when input segment is an LP because it could have more than 1 LC type.		TLISTEND	
04		TESTC	

CHART: DLZDSEH0
PAGE 2 OF 2

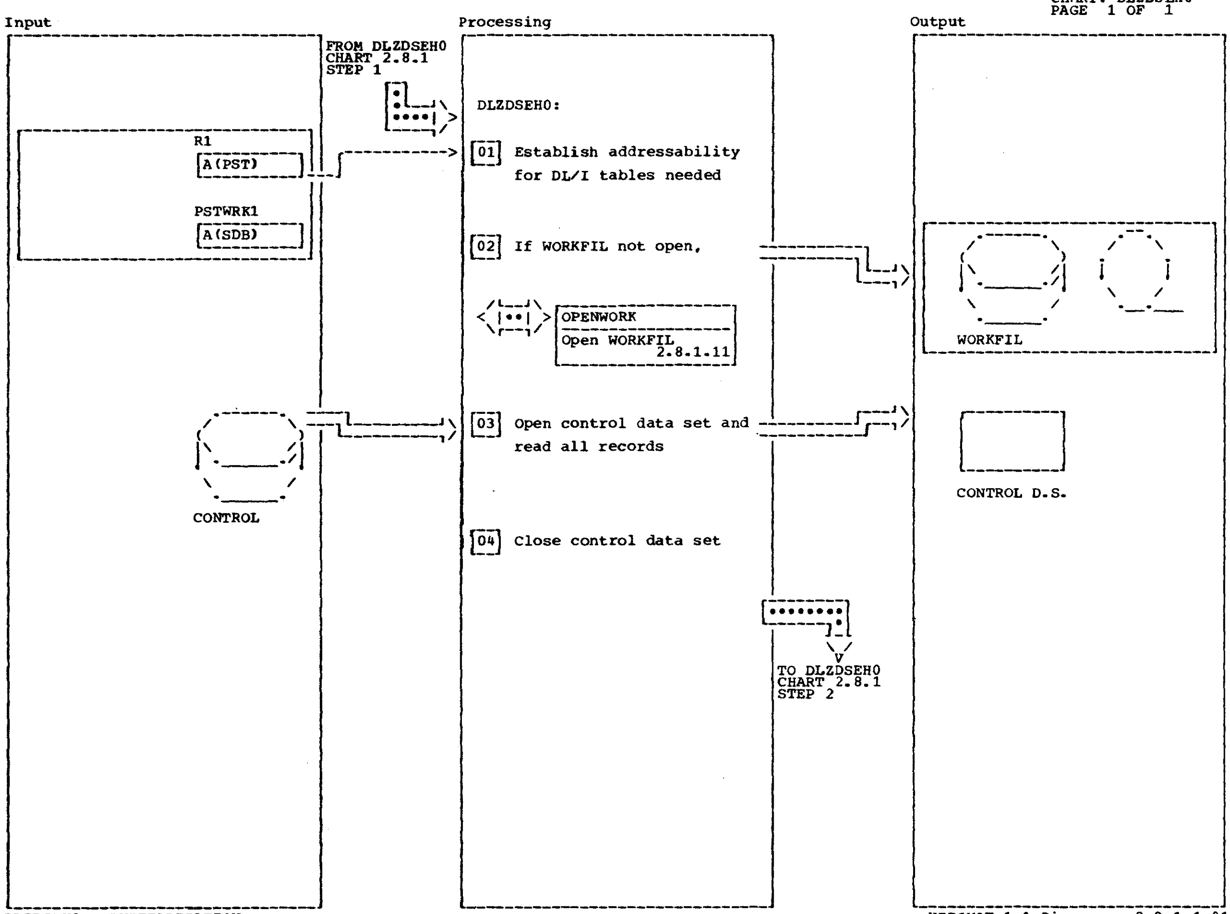


DLZDSEH0 HIPOMAT 1.1 Diagram - 2.8.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
05 Description of WORKFIL record can be found in DLZURWF1 DSECT.	DLZDSEH0	LP1					
06		CHILD					
07 If any error occurred, call DL/I error message module to write DLZ007 message on console with return code.		RETURN					

DLZDSEH0 HIPOMAT 1.1 Diagram - 2.8.1-02

CHART: DLZDSEH0
PAGE 1 OF 1

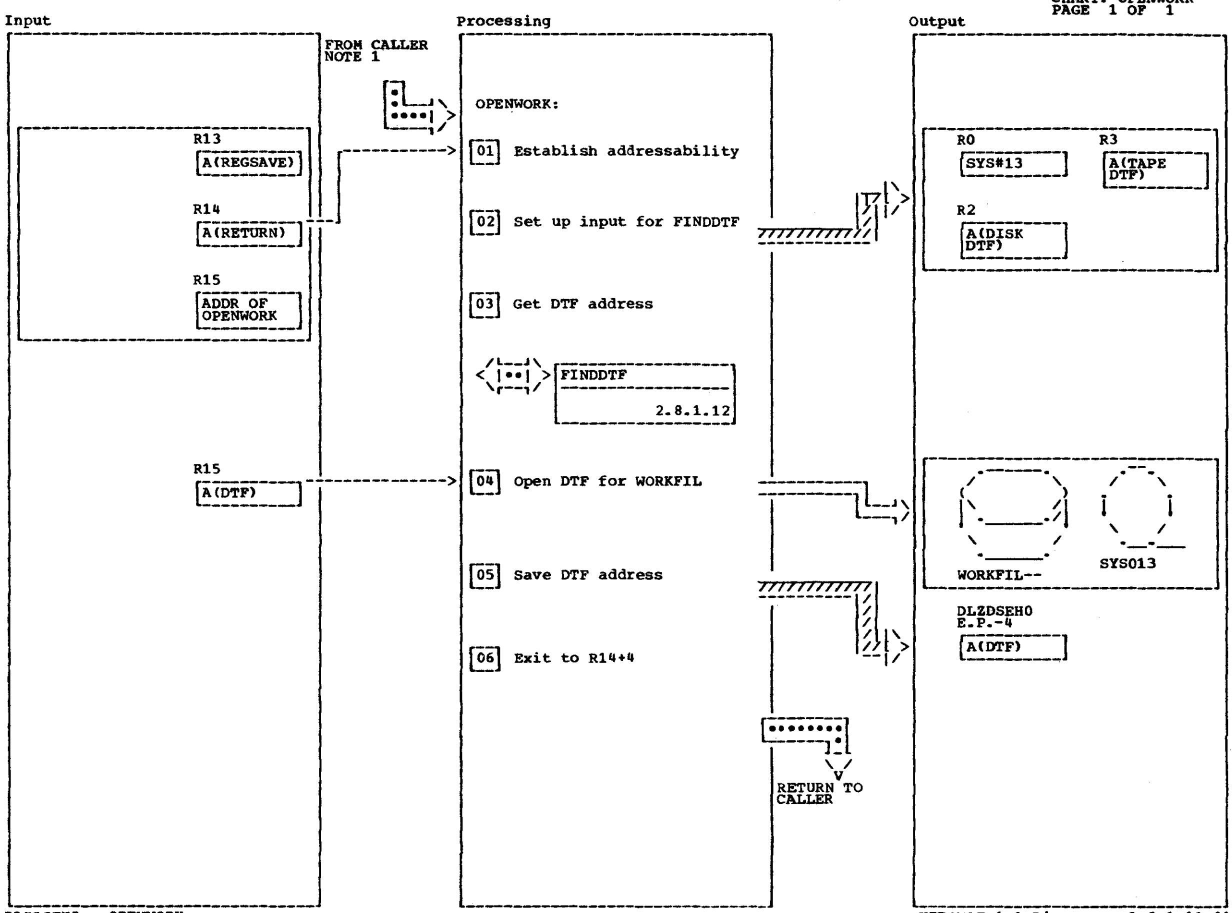


DLZDSEH0 - INITIALIZATION

HIPOMAT 1.1 Diagram - 2.8.1.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 The secondary list entries for the input segment are the primary source of information from the DL/I blocks.	DLZDSEH0	INIT					
02 The address of the DTF is found in the address list at the beginning of DLZDSEH0. If it is 0, this workfile must be opened.							
03 This open is done only once. The 'FINDDTF' routine is used to determine the correct DTF. If more than 1 record exists on the CDS, a GETVIS is done to hold the entire file in core at one time.		LPLCA					

CHART: OPENWORK
PAGE 1 OF 1



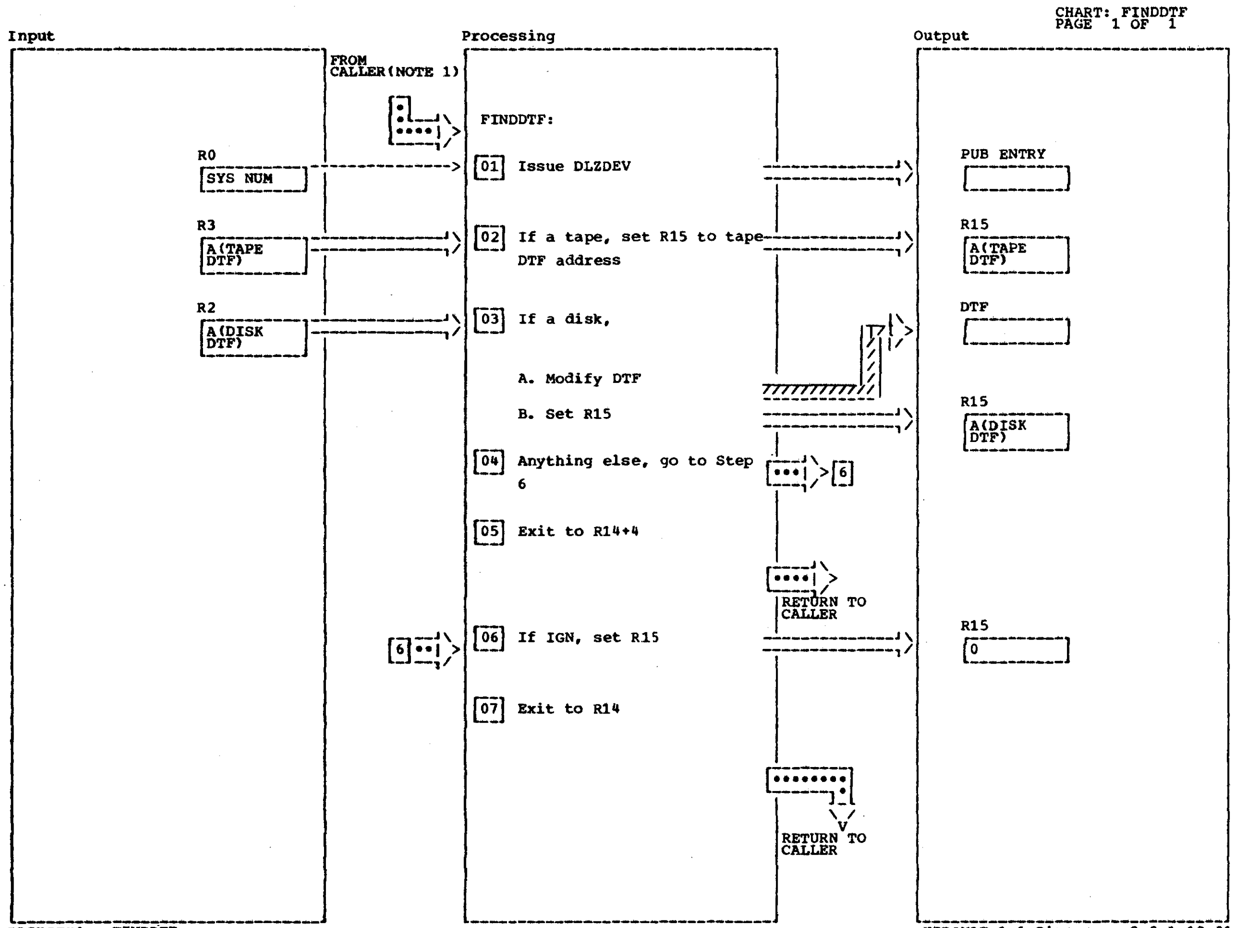
DLZDSEHO - OPENWORK

HIPOMAT 1.1 Diagram - 2.8.1.11-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 This routine is called by DLZDSEHO, DLZDXMTO, and DLZURGSO.	OPENWORK	OPENWORK					
03 If control is returned to address in R14, an error occurred. R14+4 is normal return.							
04 R15 has address of correct DTF as returned by FINDDTF.							
05 When the WORKFIL is open, the address is saved in the address list at the beginning of DLZDSEHO CSECT.							
06 If an error was detected, control is returned to the address in R14. Normal return is R14+4.		OPENEXIT					

DLZDSEHO - OPENWORK

HIPOMAT 1.1 Diagram - 2.8.1.11-01



DLZDSEHO - FINDDTF

HIPOMAT 1.1 Diagram - 2.8.1.12-01

Notes	Routine	Label	Ref
01 This subroutine is called by OPENWORK, DLZDSEHO, and DLZURGS0. DLZDEV macro finds PUB entry for given programmer logical unit and the device type byte is used to determine further processing.	OPENWORK	FINDDTF	
02 2400, 3410, and 3420 are supported.		FINDTF0	
03 2314, 3330, 3333, 3340A & B are supported.		FINDTF1 FINDTF2	
05 Normal return.		FINDEXIT	
06 This allows DLZDXMT0 to build secondary index entries during load.		FINDERRX	
07 This is the error exit.		FINDERRU	

Notes	Routine	Label	Ref

Input

PSTUSER
A(IOAREA)

DMB

DBPCBKFD

CONTROL D.S.

Processing

FROM DLZDSEH0
CHART 2.8.1
STEP 6

DLZDSEH0:

01 Build Type 10 record and write it to WORKFIL

02 If LTF pointers and non-unique sequence field, build and write Type 20 record

03 If LTB pointers and non-unique sequence field, build and write Type 30 record

TO DLZDSEH0
CHART 2.8.1
STEP 7

Output

WORKFIL RECORD
TYPE 10

WORKFIL RECORD
TYPE 20

WORKFIL RECORD
TYPE 30

DLZDSEH(- BUILD LC WORKFIL

HIPOMAT 1.1 Diagram - 2.8.1.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01		CHILD					
		LC1					
02		LC120A					
03		LC130B					

Input

INPUT

PARTS PCB
CTL.-PCB
PAR. PART
COMP. PART

PARTS DB

Processing

This module is entered from an application program via CALL.

01 Obtain and adjust input data.

02 Read parent and component part. if not found, go to step 9.

03 Increment the LLC of the parent part by 1 to obtain the initial LLC. Set actual LLC to initial LLC. if the actual LLC is not higher than the LIC of the component part, no processing is required and control is passed to step 9.

04 Insert root segment LLCTL with key = X to start the control data base.

05 Insert dependent segments into the control data base for parent part and for component part.

Output

LLC

INITIAL
ACTUAL

CONTROL DB

000 - MAINTENANCE OF LOW - LEVEL CODES

HIPOMAT 1.1 Diagram - 3.1.1-01

Notes	Routine	Label	Ref
01 The calling application program uses three different entry points for Assembler, COBOL or PL/I. A parameter list consisting of 6 pointers identifies 6 fields, 4 of them containing input data, 2 of them expecting output data.		DLZNNCA DLZNNCC DLZNNCP	
05 The original LLC of the component is saved in an UPDMASTR segment. A PARTBEXP segment for continuity check control with a key composed of hexa zeros plus the key of the parent part is inserted. The continuity check itself is explained in note 6 of 002 - VERTICAL EXPLOSION CONTROL. A PARTBEXP segment for explosion control with a key composed of the actual LLC plus key of the component part is inserted.		PARTBEXP	

Notes	Routine	Label	Ref

Input

Processing

[06] Replace the old LLC in the component part by the actual LLC.

[07] Execute LLC/CC in DL/I.

002
VERTICAL EXPLOSION CONTROL
3.1.3

[08] Remove all control information from the control data base by deleting the root segment LLCTL with key = X.

[09] Set up return information. Return to calling application program.

Output

PARTS DB

CONTROL DB

OUTPUT

RETURN CD.
LOOP KEY

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[09] Return information is obtained from the status bits of the LECB and from the internal loop key field.		DLZNNEC					

CHART: DLZNN
PAGE 1 OF 2

Input

INPUT
- PARTS PCB
- CTL-PCB
- PART

PARTS DB

PARTS DB

Processing

This module is entered from an application program via CALL.

[01] Obtain and adjust input data.

[02] Read part. If not found, go to step 9.

[03] Test the LLC of the part. If it is not 0, go to step 8. Else set initial and actual LLC to 0.

[04] The part is tested whether it has component parts. If no components are found, control is passed to step 9.

[05] Insert root segment LLCTL with key = X to start the control data base.

[06] Insert a segment PARTBEXP with key composed of packed zeros, i.e., actual LLC plus part key.

Output

LLC
- INITIAL
- ACTUAL

CONTROL DB

001 - INITIAL GENERATION OF LOW - LEVEL CODES

HIPOMAT 1.1 Diagram - 3.1.2-01

Notes	Routine	Label	Ref
[01] The calling application program has three entry points for Assembler, COBOL or PL/I. A parameter list consisting of 5 pointers identifies 5 fields, 3 of them containing input data, 2 of them expecting output data.		DLZNNGA DLZNNGC DLZNNGP	
[04] A bit is set in the LECB to indicate that no component part exists.		LECBSNOC	

Notes	Routine	Label	Ref

001 - INITIAL GENERATION OF LOW - LEVEL CODES

HIPOMAT 1.1 Diagram - 3.1.2-01

Input	Processing	Output
	07 Execute LLC/CC in DL/I.	
	002 VERTICAL EXPLOSION CONTROL 3.1.3	
	08 Remove all control information from the control data base by deleting the root segment LLCTL with key = X	CONTROL DB
	09 Set up return information. Return to calling application program.	OUTPUT: RET.-CODE, LOOP KEY

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
09 Return information is obtained from the status bits of the LECB and from the internal loop key field.		DLZNNEC					

Input

LLC
ACTUAL
INITIAL

CONTROL DB

Processing

01 The actual low-level code is used to identify the next part to be processed. A segment PARTBEXP is read with key equal or greater to LLC plus hex zeros.

02 If no segment PARTBEXP is found , the actual LLC is exhausted. Control is passed to module 005.

005
NEXT PARENT ON HIGHER LEVEL.
3.1.3.3

03 Upon return from 005, the actual LLC is tested. If it is higher than the initial LLC, control is passed to step 1, else processing is completed and control is passed to step 8.

Output

002 - VERTICAL EXPLOSION CONTROL

HIPOMAT 1.1 Diagram - 3.1.3-01

Notes	Routine	Label	Ref
01 Vertical explosion control is performed by means of PARTBEXP segments. Each time a new component part is encountered with a low-level code which needs replacement, a PARTBEXP segment - key = LLC + part key - is created. When going down a product-structure tree, this step of LLC/CC in DL/I identifies a new component part to become a parent part within the recursive process of explosion. Explosion proceeds on a FIFO basis.		PARTBEXP	
02 During previous explosions, no component part was found requiring the replacement of its current low-level code, or no component part was found at all. Therefore, no segment PARTBEXP was inserted.			
03 The initial low-level code was established either in module 000 or in module 001, resp.			

Notes	Routine	Label	Ref

Input	Processing	Output
PARTS DB	04 If a segment is found, read the root segment of the part.	PART: ACTUAL
	05 Increment the actual LLC by 1 to post into the components of the new part.	LLC: ACTUAL
	06 Perform continuity check and go to step 9. If the check fails, restore old LLC status in 006.	CONTROL DB
	006 ERROR RECOVERY HANDLER 3.1.3.4	OUTPUT DATA: LOOP KEY, ERROR INFO
	07 Upon return from 006, control is passed to step 12.	

002 - VERTICAL EXPLOSION CONTROL

HIPOMAT 1.1 Diagram - 3.1.3-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
06 The continuity check is performed using the segment type PARTBEXP. Each time a new part is becoming exploded, a segment is inserted which only consists of the part key preceded by 2 bytes hexa zeros. If a part occurs twice in a particular hierarchical path, DL/I will reject the request for insertion because a segment with same key is already existing. LLC/CC in DL/I tests this condition and signals continuity check. Insertion is processed here. However if in updating mode, LLC/CC in DL/I inserts a PARTBEXP segment of this type for the part identified by PARM3 already in 000, step 5.		PARTBEXP					

Input | Processing | Output

Processing

[08] If the continuity check did not indicate a loop, the actual part will be exploded into its component parts.

003
EXPLOSION OF A PART 3.1.3.1

[09] Upon return from 003, a test is made to detect whether the actual part has had component parts at all. If components were found, control is passed back to step 1.

[10] Else, 004 is employed.

004
NEXT PARENT ON SAME LEVEL 3.1.3.2

[11] Upon return, go to step 1.

[12] Go back to higher level module 000 or 001

002 - VERTICAL EXPLOSION CONTROL

HIPOMAT 1.1 Diagram - 3.1.3-03

Notes	Routine	Label	Ref
[09] A switch in the LECB is used to transfer information whether a part has component parts. The switch is turned off before entering 003, i.e., it is assumed that the part has components. Upon return from 003, the status of this switch is tested. If the switch is on, 003 has indicated that the part does not have components.		LECBSNOC	

Notes	Routine	Label	Ref

Input

PARTS DB

LLC
ACTUAL

Processing

01 Read the first or next component segment of the actual part. If not found, processing is completed and control is passed to step 6.

02 Compare actual LLC and LLC in the component. If the actual LLC is not higher, no further processing is required and control is passed back to step 1.

03 Save the old LLC of the component in an UPDMASTR segment.

04 Replace the old LLC of the component by the actual LLC.

Output

CONTROL DB

PARTS DB

003 - EXPLOSION OF A PART

HIPOMAT 1.1 Diagram - 3.1.3.1-01

Notes	Routine	Label	Ref
01 If the no-component-found LECBSNOC condition was raised when retrieving the first segment, a switch indicates to 002 that the actual part does not have any component parts at all and another part has to be selected for explosion.		LECBSNOC	

Notes	Routine	Label	Ref

Input	Processing	Output
	[05] Insert a segment PARTBEXP with key composed of actual LLC plus PARTKEY of the component. Go back to step 1. ⟹	CONTROL DB
	[06] Go back to module 002.	

003 - EXPLOSION OF A PART

HIPOMAT 1.1 Diagram - 3.1.3.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

Input	Processing	Output
PART ACTUAL	01 Remove the actual part form the hierarchical path.	
CONTROL DB	02 Delete segment PARTBEXP with key composed of hex zeros and the key of the actual part.	
LLC ACTUAL	03 Decrement actual LLC by 1.	LLC ACTUAL
CONTROL DB	04 Remove the first segment PARTBEXP with key = actual LLC + hex zeros since it has been completely exploded.	
	05 Return to module 002.	

004 - NEXT PARENT ON SAME LEVEL

HIPOMAT 1.1 Diagram - 3.1.3.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
02 A part may occur multiple times within a product-structure tree. However, it must not pccur twice within a hierarchical path. Therefore, if a hierarchical path is left or is modified, all PARTBEXP seqments for continuity check related to branches which have become obsolete will be removed.							
04 When returning to step 1 in module 002, the next part on the same level will be read. Step 3 in 004 neutralizes step 4 in 002.							

004 - NEXT PARENT ON SAME LEVEL

HIPOMAT 1.1 Diagram - 3.1.3.2-01

Input	Processing	Output
LLC ACTUAL	01 Decrement the actual LLC by 1.	LLC ACTUAL
CONTROL DB; LLC ACTUAL	02 Delete the first segment PARTBEXP identified by a key composed of the actual LLC and of hex zeros.	PART KEY
CONTROL DB; PART KEY	03 Delete the segment PARTBEXP identified by a key composed of hex zeros and the part key retrieved in step 2.	
	04 Return to module 002.	

005 - NEXT PARENT ON HIGHER LEVEL

HIPOMAT 1.1 Diagram - 3.1.3.3-01

Notes	Routine	Label	Ref
01 This allows to continue in module 002 at step 1 on the next higher, i.e., numerically lower level.			
02 A part may occur multiple times within a product-structure tree. However, it must not occur twice within a hierarchical path. Therefore, if a hierarchical path is left or is modified, all PARTBEXP segments for continuity check related to branches which have become obsolete will be removed.			
03 Since this hierarchical path is exhausted, the control segment for explosion is deleted.			

005 - NEXT PARENT ON HIGHER LEVEL

HIPOMAT 1.1 Diagram - 3.1.3.3-01

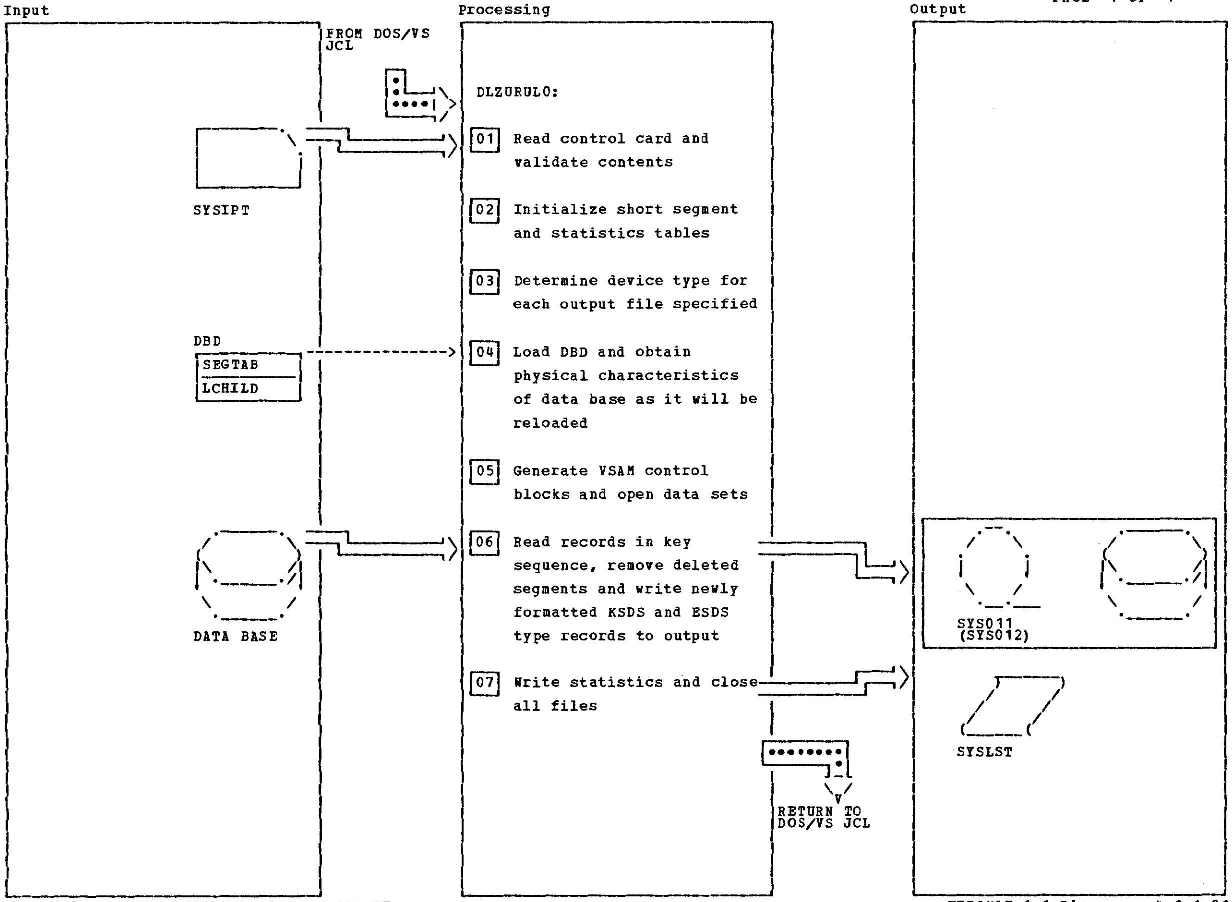
Input	Processing	Output
CONTROL DB	01 Read sequentially all UPDMASTR segments.	
PARTS DB	02 Restore all LLCs of parts referenced by an UPDMASTR segment to its original value.	PARTS DB
	03 Return to module 002.	

006 - CONTINUITY ERROR HANDLER

HIPOMAT 1.1 Diagram - 3.1.3.4-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

CHART: DLZURULO
PAGE 1 OF 1



DLZURULO - HISAM REORGANIZATION UNLOAD UT.

HIPOMAT 1.1 Diagram - 4.1.1-01

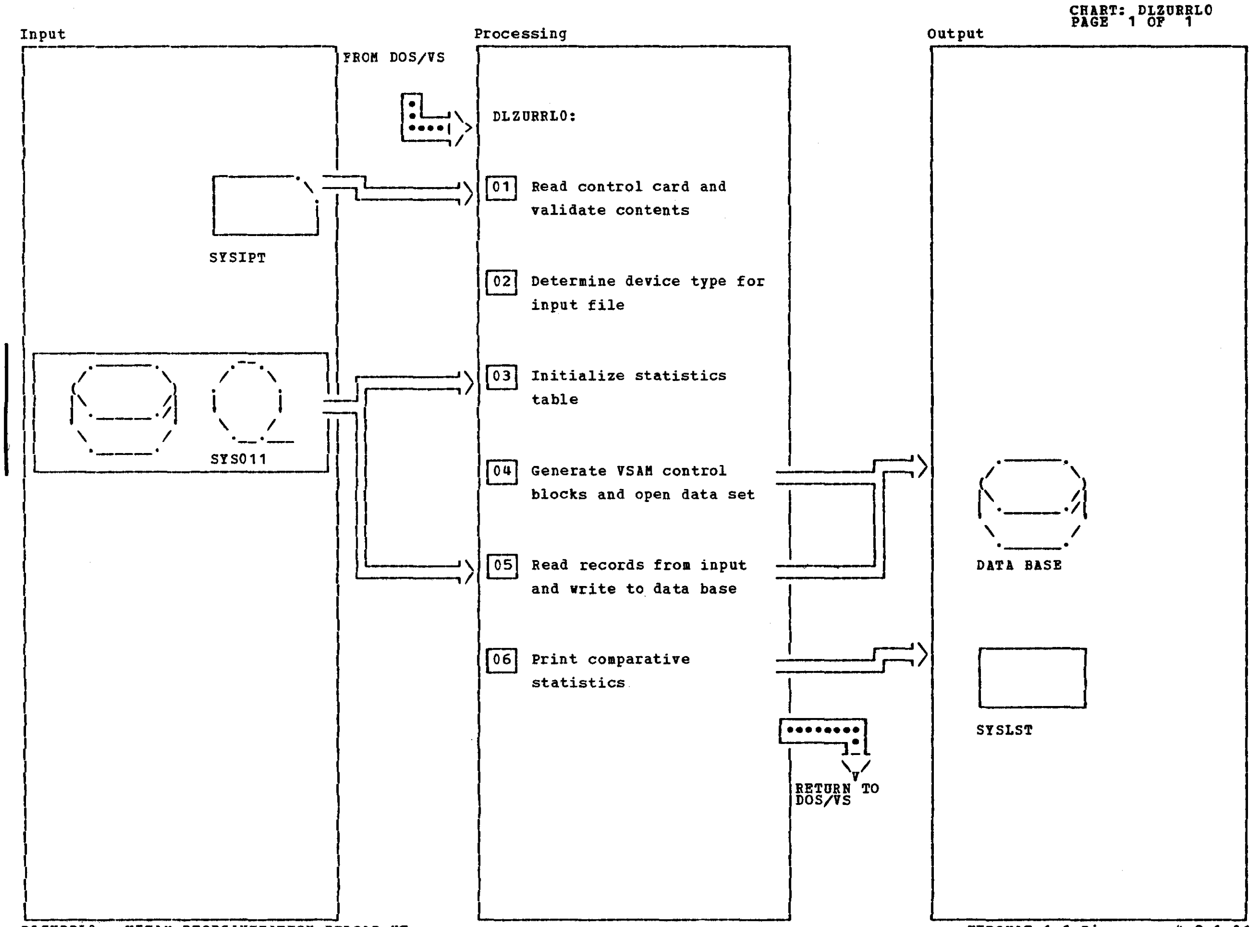
Notes	Routine	Label	Ref
01 Validate DBD name, KSDS name, output file name(s) and number of O/P files.			
03 DLZDEV macro obtains data from PUB. Device type may be TAPE or DASD.			
05 Issue GENCB for ACB, RPL and EXLST. Open KSDS and ESDS unless ACCESS=SHISAM (KSDS only).			
06 Processing as follows:			
A. Read KSDS records in key sequence - bypass if deleted. ESDS records containing overflow dependent segments are read by RBA.			
B. Format work area like KSDS record with new attributes.			
C. Move as many segments as will fit into KSDS work area, bypassing deleted segments. Calculate ovflw RBA. Write image of KSDS to output.			

DLZURULO - HISAM REORGANIZATION UNLOAD UT.

Notes	Routine	Label	Ref
D. Format work area like ESDS record with new attributes.			
E. Move as many dependent segments as will fit into ESDS work area, bypassing deleted segments. Calculate RBA for next record, if required. Write image of ESDS to output.			
07 Statistics also written to SYS011 to be used for comparative purposes during reload.			
Processing will continue if additional input cards.			

HIPOMAT 1.1 Diagram - 4.1.1-01

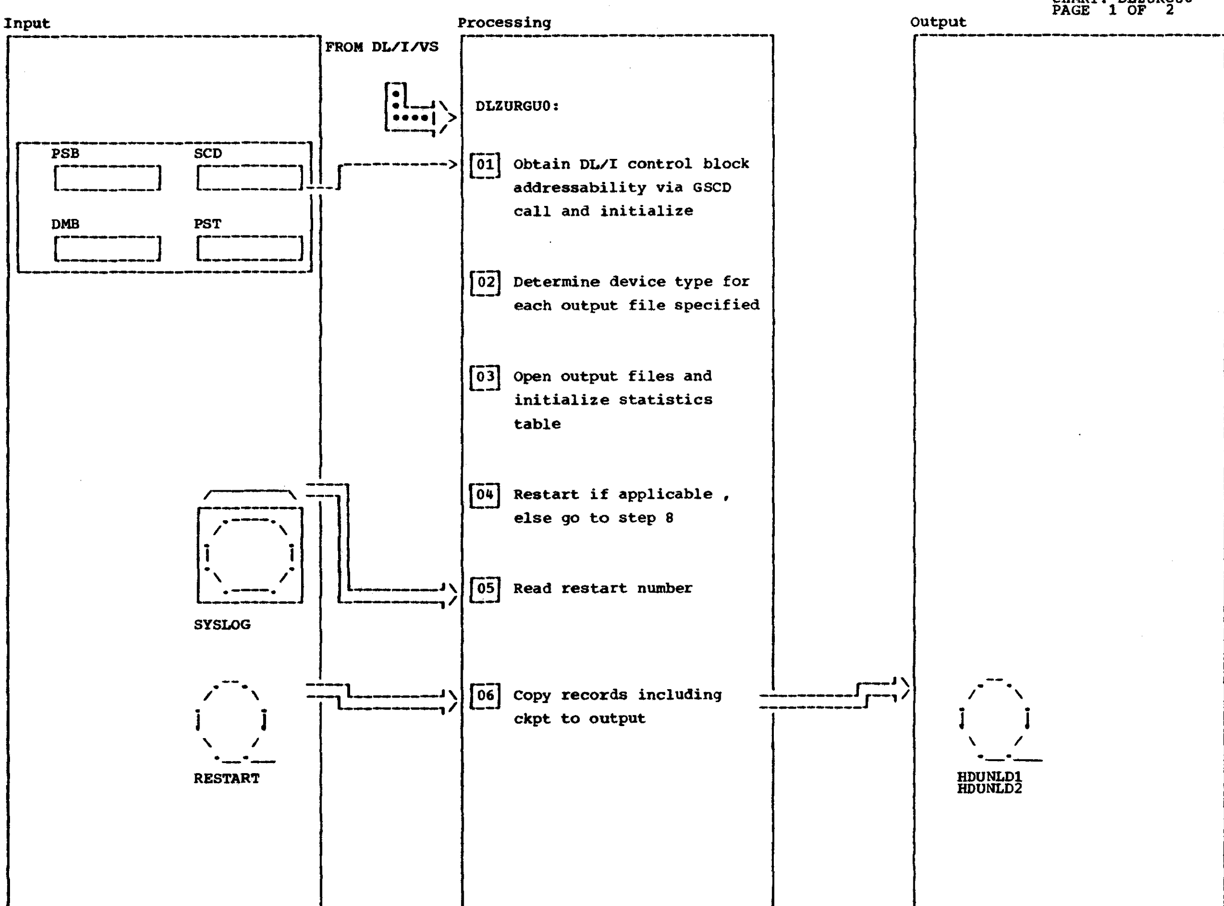
CHART: DLZURRLO
PAGE 1 OF 1



DLZURRLO - HISAM REORGANIZATION RELOAD UT. HIPOMAT 1.1 Diagram - 4.2.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Validate input filename.							
02 DLZDEV macro obtains data from PUB.Device may be TAPE or DASD.							
03 The first record on the input file contains a statistics table initialized to zero. Included is the segment code and length for all segment types in the data base.							
04 Issue GENCB for ACB, RPL and EXLST. Open KSDS and ESDS unless ACCESS=SHISAM (KSDS only).							
05 KSDS image records written to KSDS as key sequence records. ESDS image records written ESDS as address sequence records.							

DLZURRLO - HISAM REORGANIZATION RELOAD UT. HIPOMAT 1.1 Diagram - 4.2.1-01

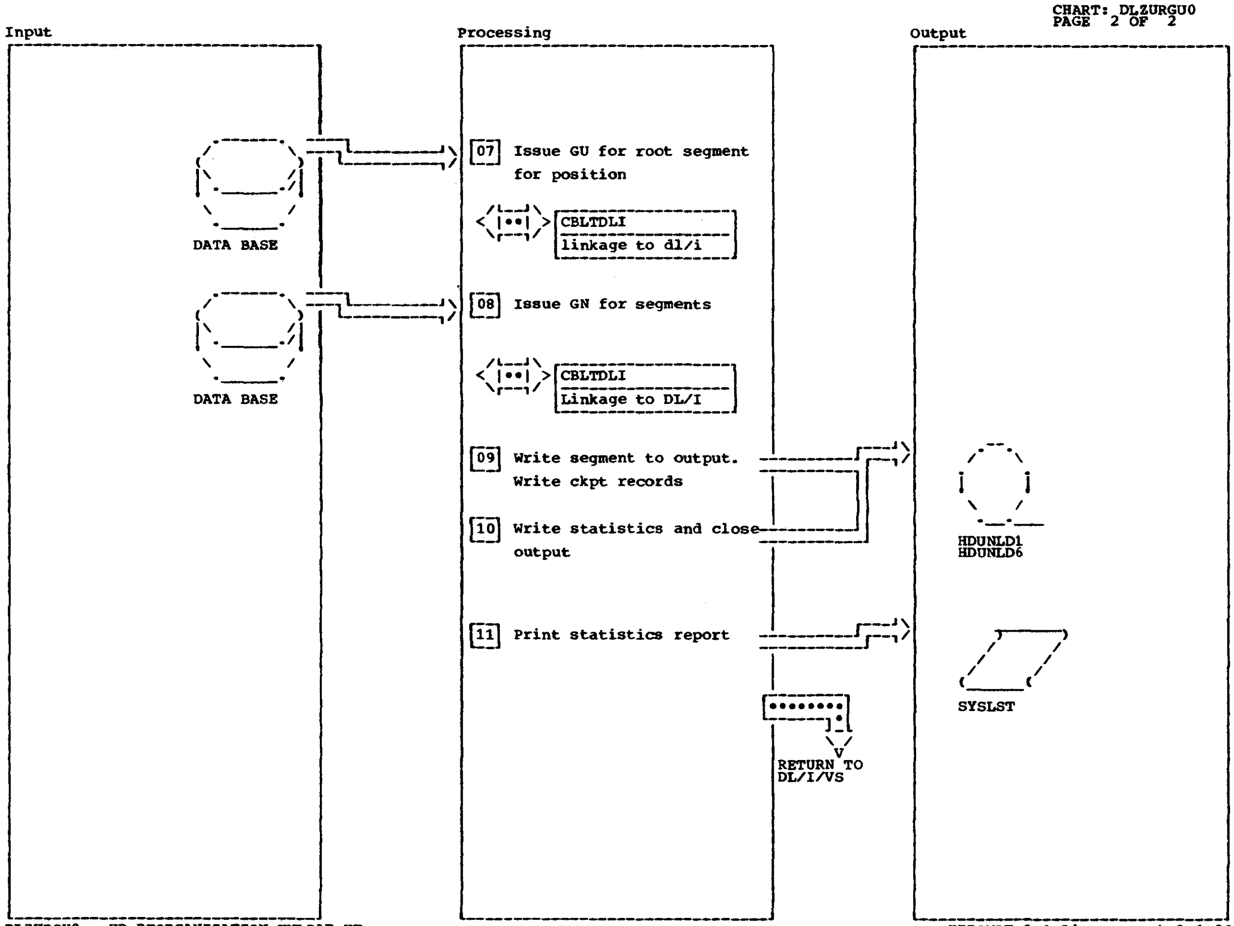


DLZURGU0 - HD REORGANIZATION UNLOAD UT. HIPOMAT 1.1 Diagram - 4.3.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
02 DLZDEV macro obtains data from PUB. Device type may be TAPE or DASD.							
03 Table contains segment code and length for all segment types in data base.							
04 If SYS010 not IGN, restart takes place.							
05 Program writes DLZ0318I message to SYSLOG requesting restart number and reads response.							
06 RESTART = SYS010, HDUNLD1 = SYS011, HDUNLD2 = SYS012 .							

DLZURGU0 - HD REORGANIZATION UNLOAD UT. HIPOMAT 1.1 Diagram - 4.3.1-01

CHART: DLZURGU0
PAGE 2 OF 2



DLZURGU0 - HD REORGANIZATION UNLOAD UT.
HIPOMAT 1.1 Diagram - 4.3.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
07 If HISAM issue *'C' call, otherwise issue *'T' call.							
09 DL/I prefix is attached to segment. Ckpt records are written at first root segment after every 5000 segments.							

DLZURGU0 - HD REORGANIZATION UNLOAD UT.
HIPOMAT 1.1 Diagram - 4.3.1-02

Input	Processing	Output
PSB, SCD, DMB, PST (From DL/I/VS)	DLZURGLO:	
	01 Obtain DL/I control block addressability via GSCD call and initialize.	
	02 Determine device type for input.	
	03 If not restarting, go to Step 11.	
	04 If restarting, ask operator for checkpoint restart number.	
HDUNLD	05 Locate checkpoint record.	
DATA BASE	06 Position data base (GU call).	
	CBLTDLI — Linkage to DL/I	
	07 Find end of data (GN calls).	
	CBLTDLI — Linkage to DL/I	

DLZURGLO - HD Reorganization Reload Utility — HIPOMAT 1.1 Diagram - 4.4.1-01

Notes	Routine	Label	Ref
02 DLZDEV macro obtains data from PUB. Device may be TAPE or DASD.			
03 If the HD Reorganization Reload Utility program fails, the reload restart capability allows you to restart from a checkpoint record. Before resubmitting the job for reload restart, change the parameter card from ULU to ULR.			
04 The number of the last valid checkpoint record on the unloaded file is found in console message DLZ381I. Valid checkpoint numbers are decimal values between 1 and 9999.			

Notes	Routine	Label	Ref

Input

HDUNLD

RSTFILE

Processing

08 Read corresponding records from input.

09 If a work file is present, copy old to new.

10 Reset processing options.

11 Read input and build SSA for each segment.

12 Issue ASRT call to load segments.

CBLTDLI
Linkage to DL/I

13 Print comparative statistics report.

Return to DL/I/VS

Output

WORKFIL

DATA BASE

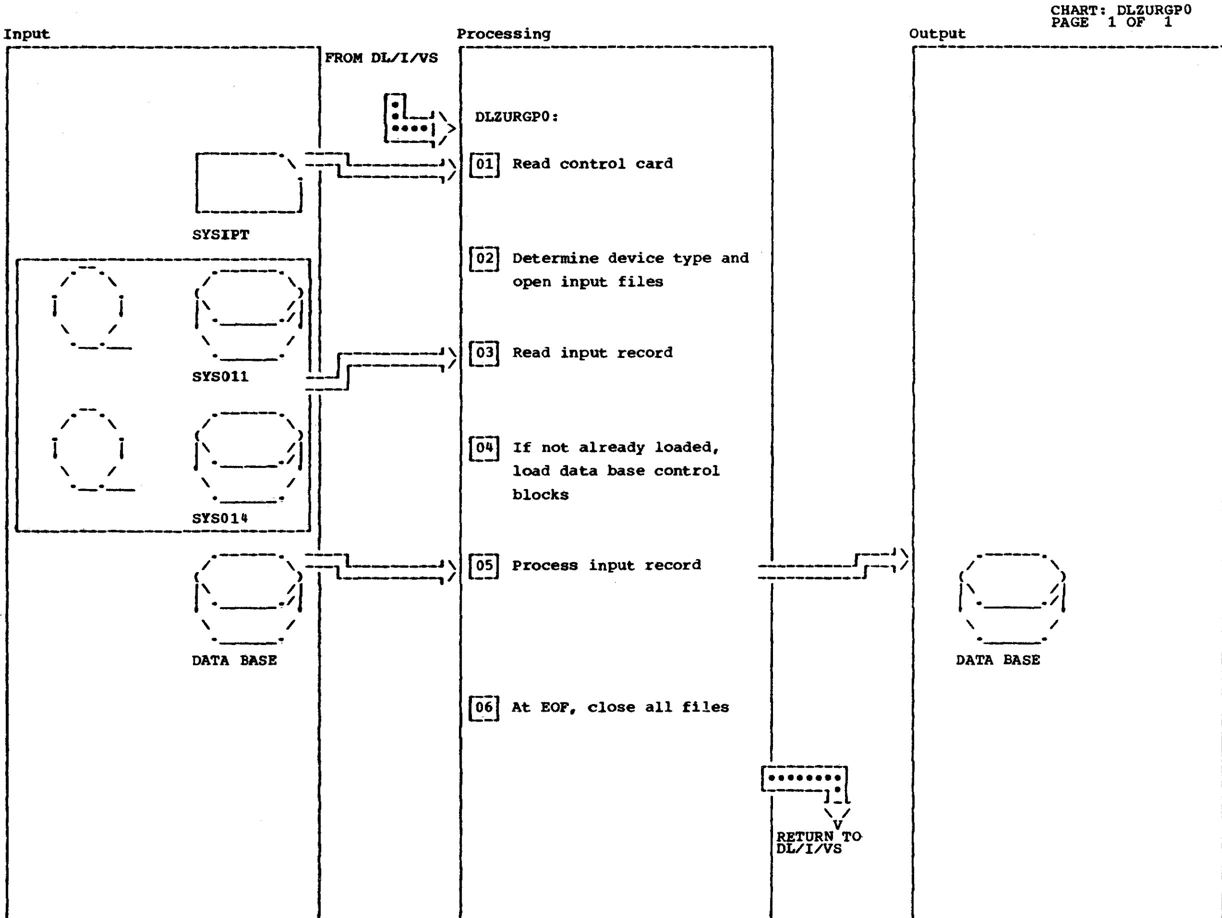
SYSLST

DLZURGL0 - HD Reorganization Reload Utility

HIPOMAT 1.1 Diagram - 4.4.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
09 If a work file (for logical relationships or secondary indices) was being created during the reload, the partially created work file should be submitted as input to the restarted job assigned as SYS010 with a filename of RSTFILE.							

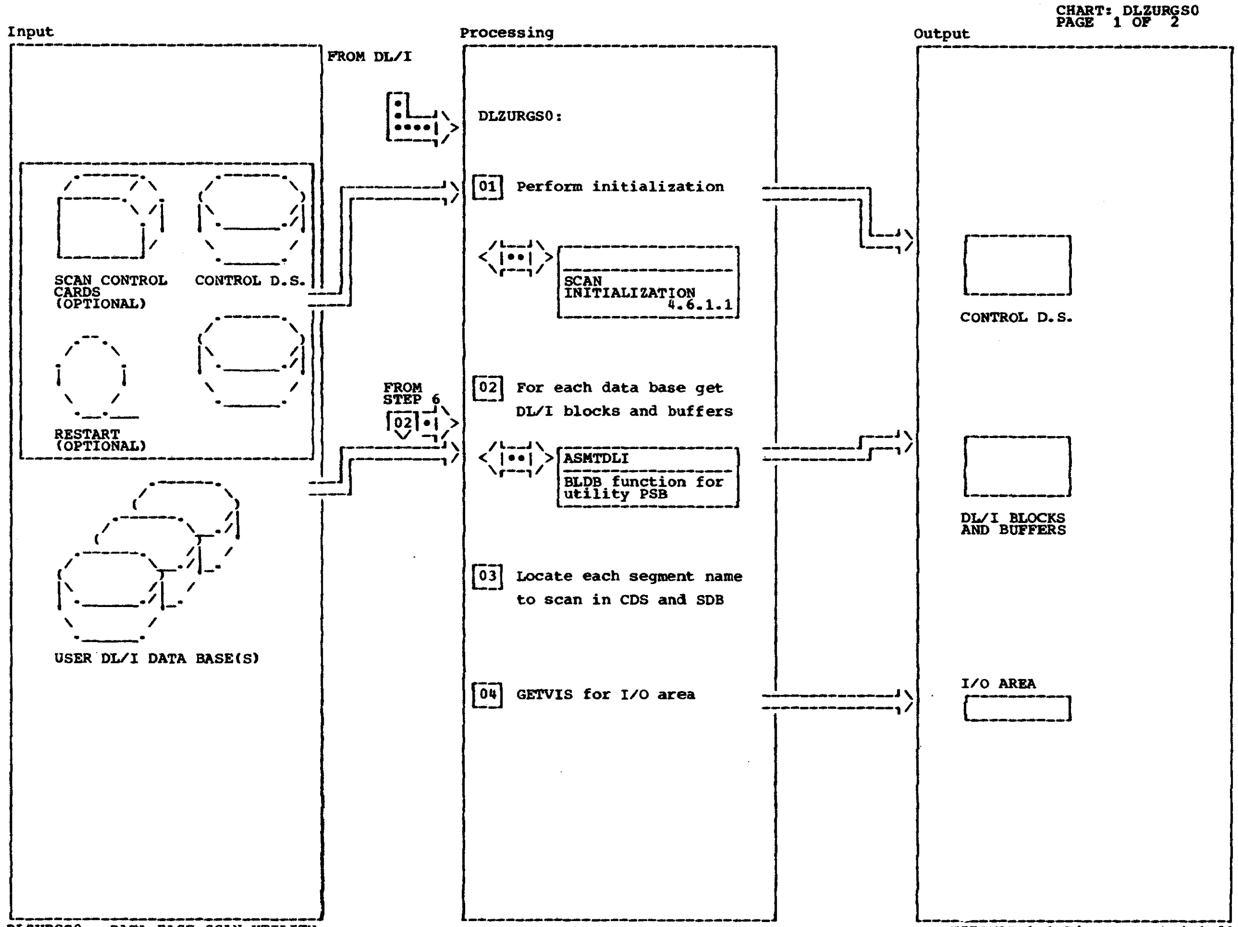
CHART: DLZURGP0
PAGE 1 OF 1



DLZURGP0 - PREFIX UPDATE UTILITY HIPOMAT 1.1 Diagram - 4.5.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01		OPEN1					
02 DLZDEV macro obtains data from PUB. Device type may be TAPE or DASD.		OPENINP					
04 DLZBLKLD macro is used to load DB blocks dynamically.		BLDBLKS					
05 TYPE 0 and TYPE 1 records (LC/LP) are processed by buffer handler calls. TYPE 4 records (SI) are processed by DL/I INSERT/UPDATE calls.		TYPE0 TYPE1					

DLZURGP0 - PREFIX UPDATE UTILITY HIPOMAT 1.1 Diagram - 4.5.1-01



DLZURGS0 - DATA BASE SCAN UTILITY

HIPOMAT 1.1 Diagram - 4.6.1-01

Notes	Routine	Label	Ref
01 This utility executes as 'ULU' under DL/I control. No blocks or buffers have been loaded yet. Only the nucleus exists.	DLZURGS0		
02 The 'BLDB' call loads all blocks for PSB specified and allocates buffers. The Utility PSB for this data base is used.		NXTDB BLDBLKS NEWDB	
03 The Control Data Set (CDS) entries are modified to save SDB and PSDB addresses.		NXTSEG NXTSGFND	
04 The size is the longest needed for this data base. Any previous I/O area is freed.		LNGSEG	

Notes	Routine	Label	Ref

Input

Processing

05 Scan data base

PROC
Write workfile record
4.6.1.2

06 If another data base to scan, go to Step 2

02

07 Close files and exit

RETURN TO DL/I

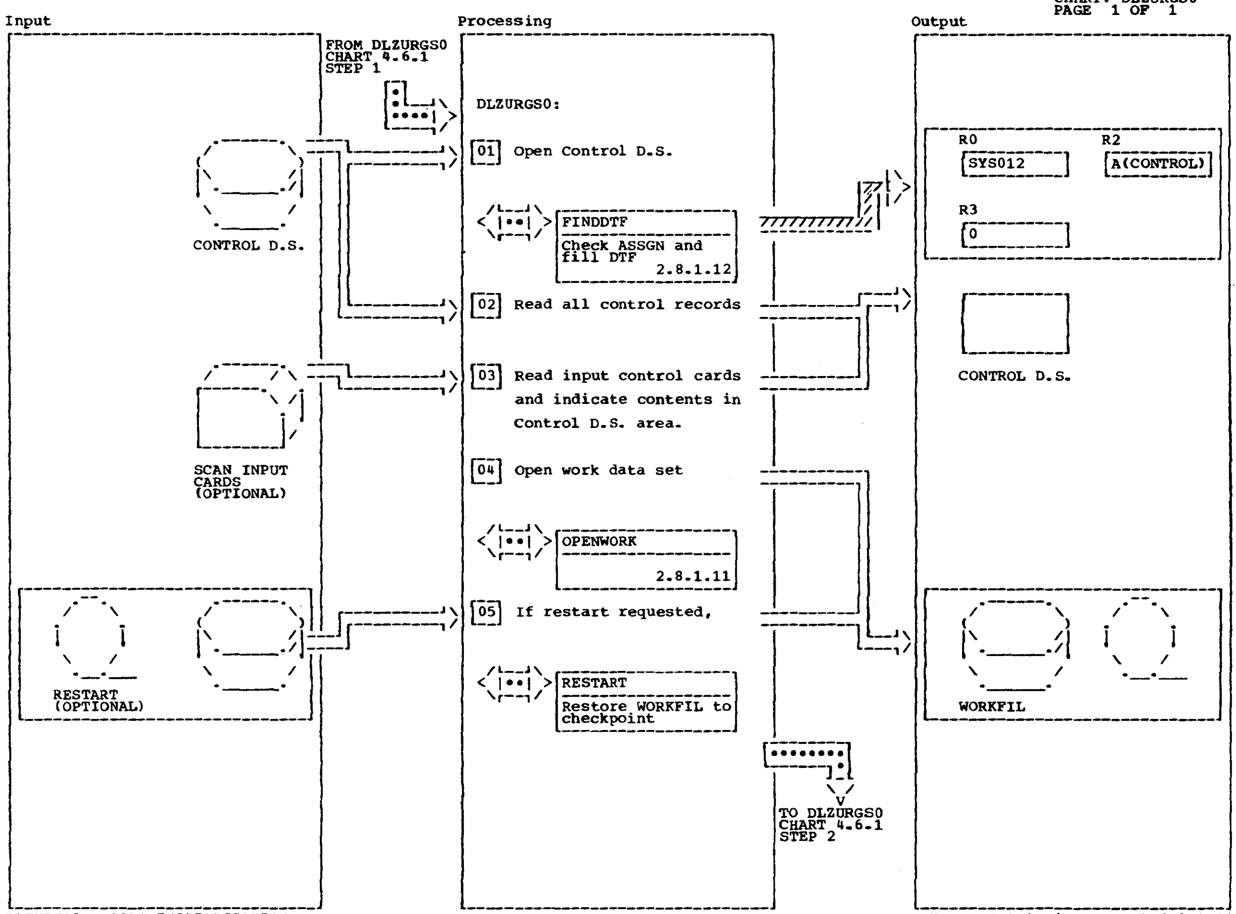
Output

WORKFIL

DLZURGS0 - DATA BASE SCAN UTILITY

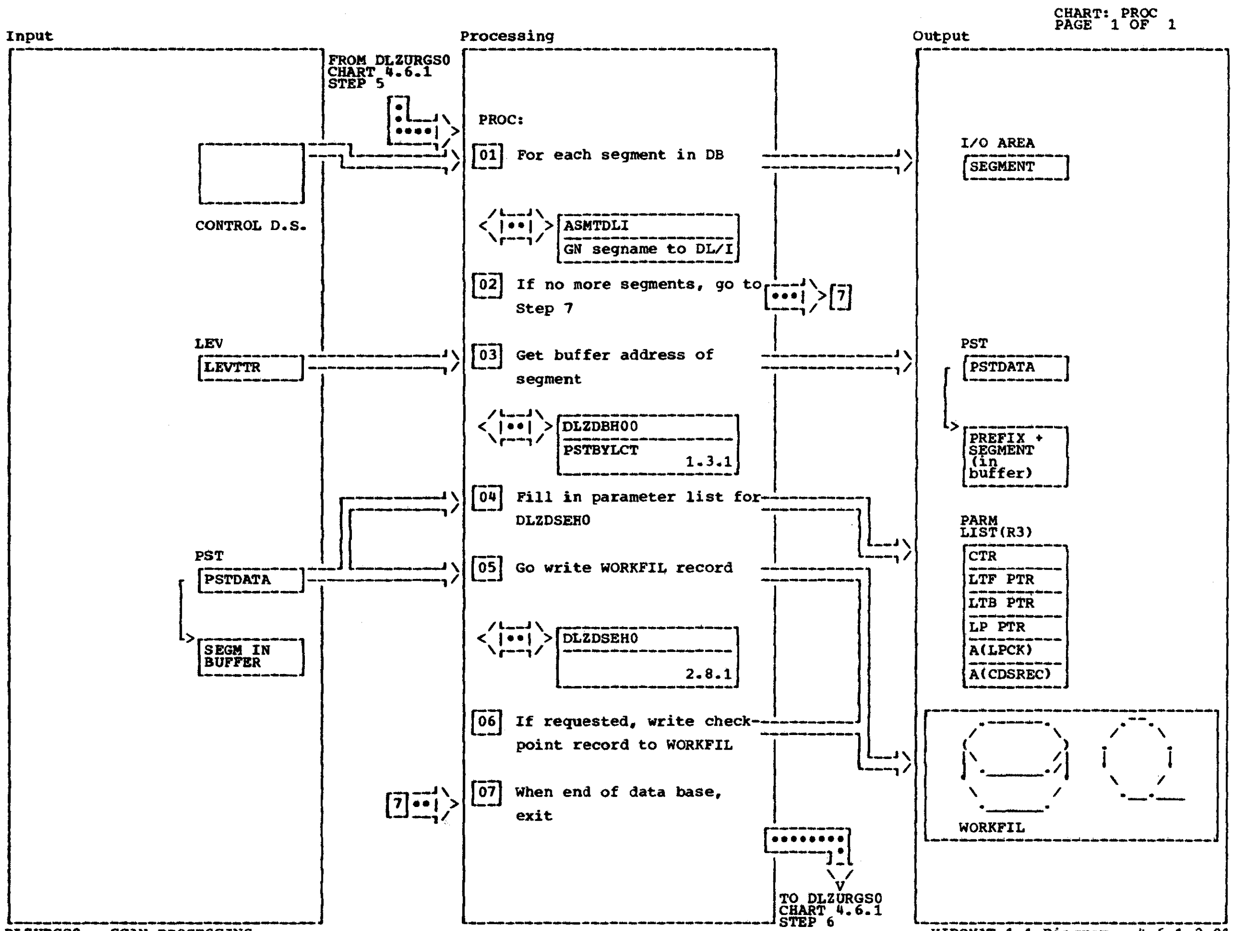
HIPOMAT 1.1 Diagram - 4.6.1-02

Notes	Routine	Label	Ref
05 Every LC or LP segment to be scanned is read and a workfile record created for it.		PROC	
07 Files are closed. Return to DL/I to close last scanned data base.		NXTDBDN TERM	



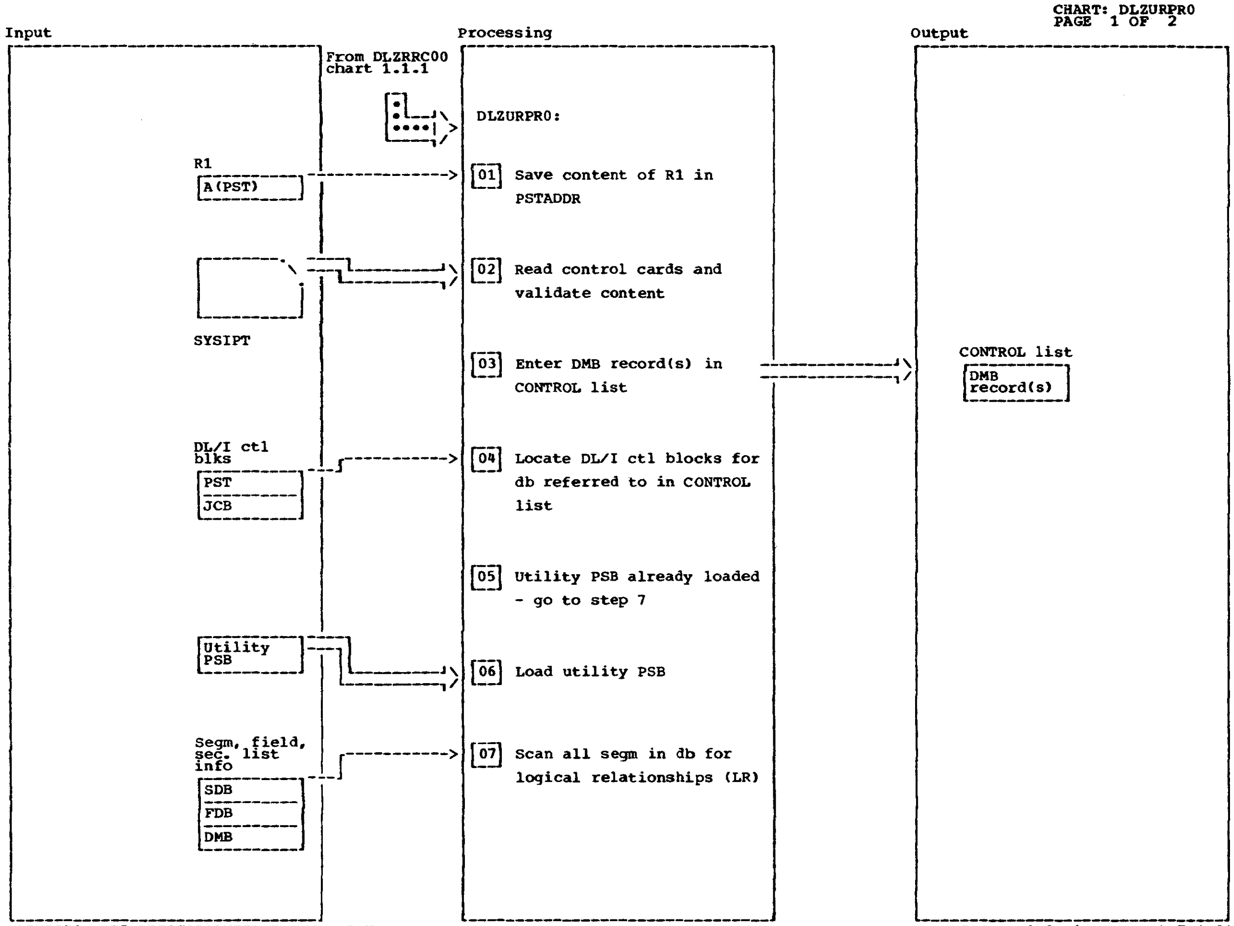
DLZURGS0 - SCAN INITIALIZATION HIPOMAT 1.1 Diagram - 4.6.1.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 The printer, reader, and console are also opened. The 'FINDDTF' subroutine is used to check that SYS012 is properly assigned to disk and to fill correct device type in DTF.	DLZURGS0	PROCCTL					
02		GETCDS					
03 Input on 'DBS=' card is used to modify Control d.s. in core. 'RSTRT=' and/or 'CHKPT=' specify checkpoint/restart capabilities. 'ABEND' card used for testing.		NXTCR					
04 This routine resides with DLZDSEH0. Its address is found in the DLZDSEH0 prefix.		SCAN					
05 Restart records are copied from the previous WORKFIL to the new WORKFIL until the specified checkpoint record is found. An SSA is set to do qualified GU on last segment to reestablish position.		RSTRT32					



DLZURGS0 - SCAN PROCESSING HIPOMAT 1.1 Diagram - 4.6.1.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 This chart shows steps taken for each occurrence in a data base of the segment types that are scanned.		PROC					
02 Scan completed for data base - message written to SYSLST.		PROC05					
03 Scan must have the prefix information to give to DLZDSEH0.		TESTRSTA					
04 In addition, R11 has address of WORKFIL DTF, R1 has PST address, PSTWRK1 has 'FUNCIHPS' and SDB address.		LCLPOFF					
05 The 'TEST' entry point is used and a register save area provided.		TESTRT					
06 A checkpoint record is written after every 'n' work file records. 'N' is specified on the CHKPT input card. A message is written to the console giving the current checkpoint record number for later reference.		CHKPT					



DLZURPR0 - DB PREREORGANIZATION UTILITY

HIPOMAT 1.1 Diagram - 4.7.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
02 Control card contains identifier as DBIL (initial load), DBR (reorganize), OPTIONS and DBDnames. In case of control card errors, message is printed and job terminates.		NXTCR					
03 DMB records contain DMB names of db and user options specified in ctl cards.							

Input	Processing	Output
	[08] Enter segm records in CONTROL list	CONTROL list: DMB records / segm records / sec. list records
	[09] Enter sec. list records in CONTROL list	
	[10] Write CONTROL list to CONTROL data set	CONTROL data set (SYS012)
	[11] Print (and punch) DBS cards, if requested by OPTIONS	SYSLST, SYSPCH
	Return to DOS/VS	

DLZURPRO - DB PREREORGANIZATION UTILITY — HIPOMAT 1.1 Diagram - 4.7.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[08] Segm records contain segm names involved in LR.							
[09] Sec. list records contain DMB names which refer to logically related data bases.							
[11] DBS indicates db must be scanned using SCAN utility (DLZURGS0).							

Input		Processing	Output
	FROM DOS/VS	DLZUACB0:	
		01 Initialize PSB name table	
		DLZUSCH0 — Entry point is OPENSRCH 4.8.1.1	
BUILD CARDS		02 Create PSB name table	
		A. Read control cards until EOF	
		B. Add valid PSB name to table	
		DLZUSCH0 — Entry point is INSRCH 4.8.1.2	

DLZUACB0 - APPLICATION CONTROL BLOCK CREATION UTILITY HIPOMAT 1.1 Diagram - 4.8.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 DLI-nucleus loaded							
02 Control cards are checked for valid input format. The output device is set at this time (SYSLNK or SYSPCH). If the initialized PSB table is not large enough, an extension is created by DLZUSCH0.		READCARD					

DLZUACB0 - APPLICATION CONTROL BLOCK CREATION UTILITY HIPOMAT 1.1 Diagram - 4.8.1-01

Input

CIL (PSB's and DBD's)

Processing

[03] For each PSB name in table

A. Build and write blocks

DLZDLBL0 — 4.8.1.3

B. Write completion message

DLZUMSG0 — 4.8.1.4

[04] Free PSB table

DLZUSCH0 — Entry point CLOSESCH — 4.8.1.5

[05] Close files and exit

RETURN TO DOS/VS

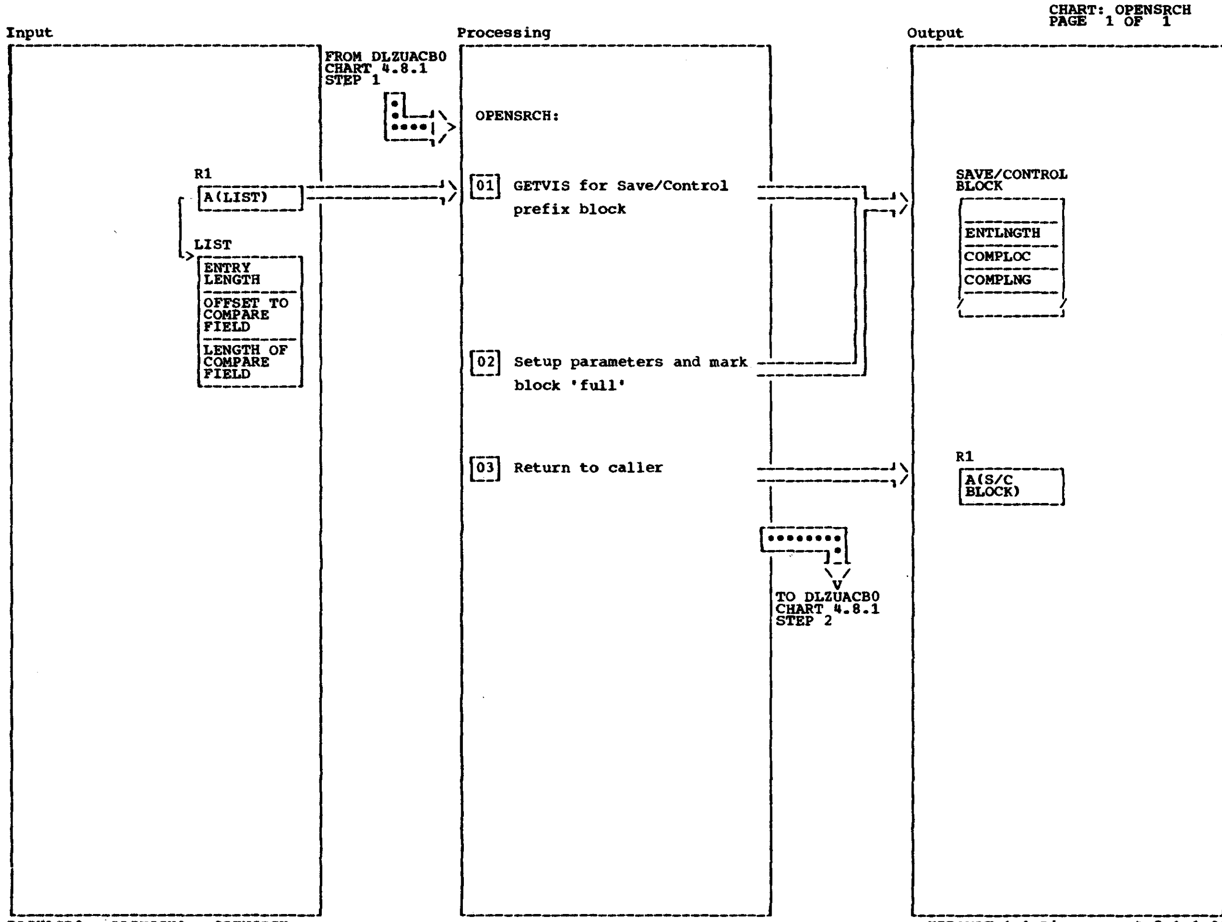
Output

SYSLNK (DMB's and PSB's) or SYSPCH

SYSLST

DLZUACB0 - APPLICATION CONTROL BLOCK CREATION UTILITY — HIPOMAT 1.1 Diagram - 4.8.1-02

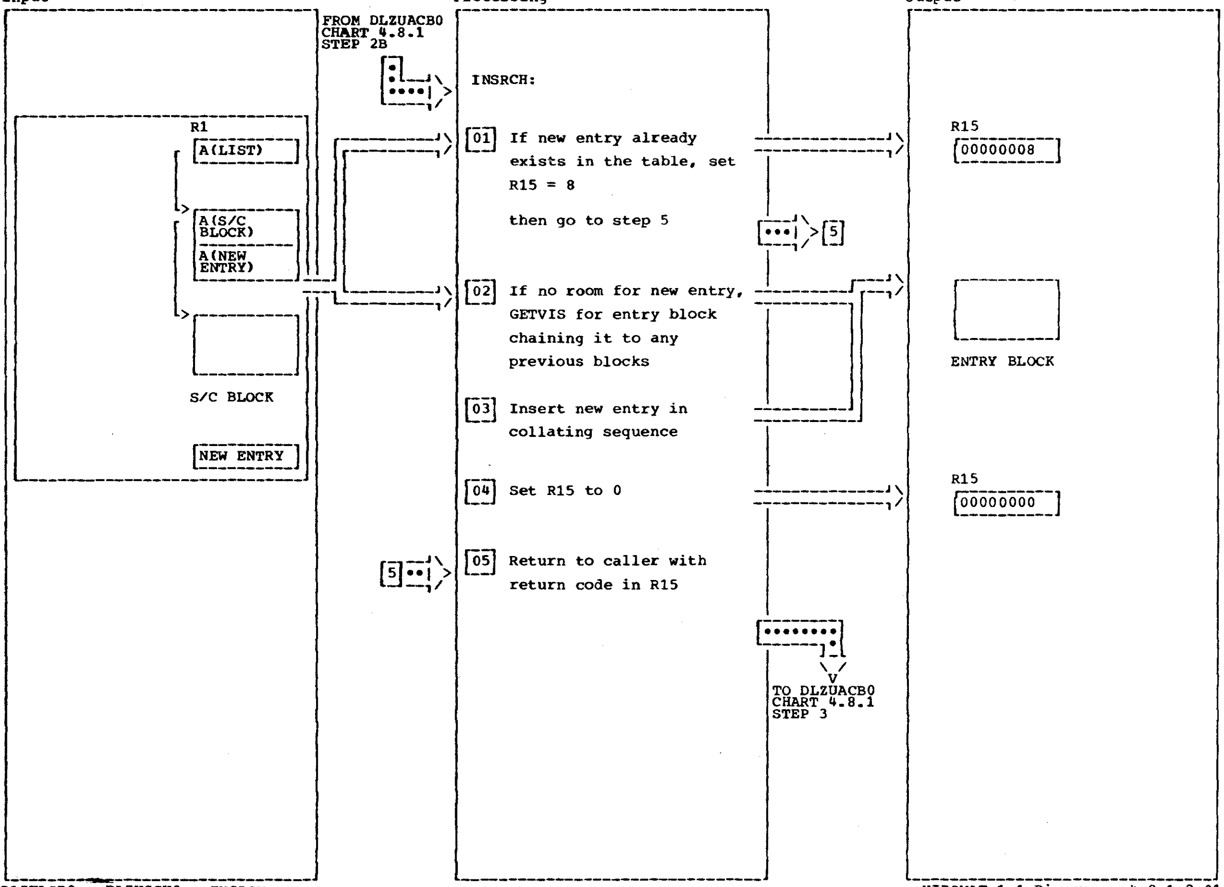
Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[03] The completion message is normal unless a non-zero return code is found from DLZDLBL0.		CALLBB					
[04]		BLDDONE					
[05]		CLOSE					



DLZUACB0 - DLZUSCH0 - OPENSRCH

HIPOMAT 1.1 Diagram - 4.8.1.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Length is 128 bytes.	OPENSRCH						
02 Block now contains information needed to build entry block. The first (or only) entry is obtained before the first actual insert.							
03 The address of the created block is returned to the caller. NOTE : This routine is very generalized and could be used for other purposes, but is only used by DLZUACB0 to build the PSB name table.							



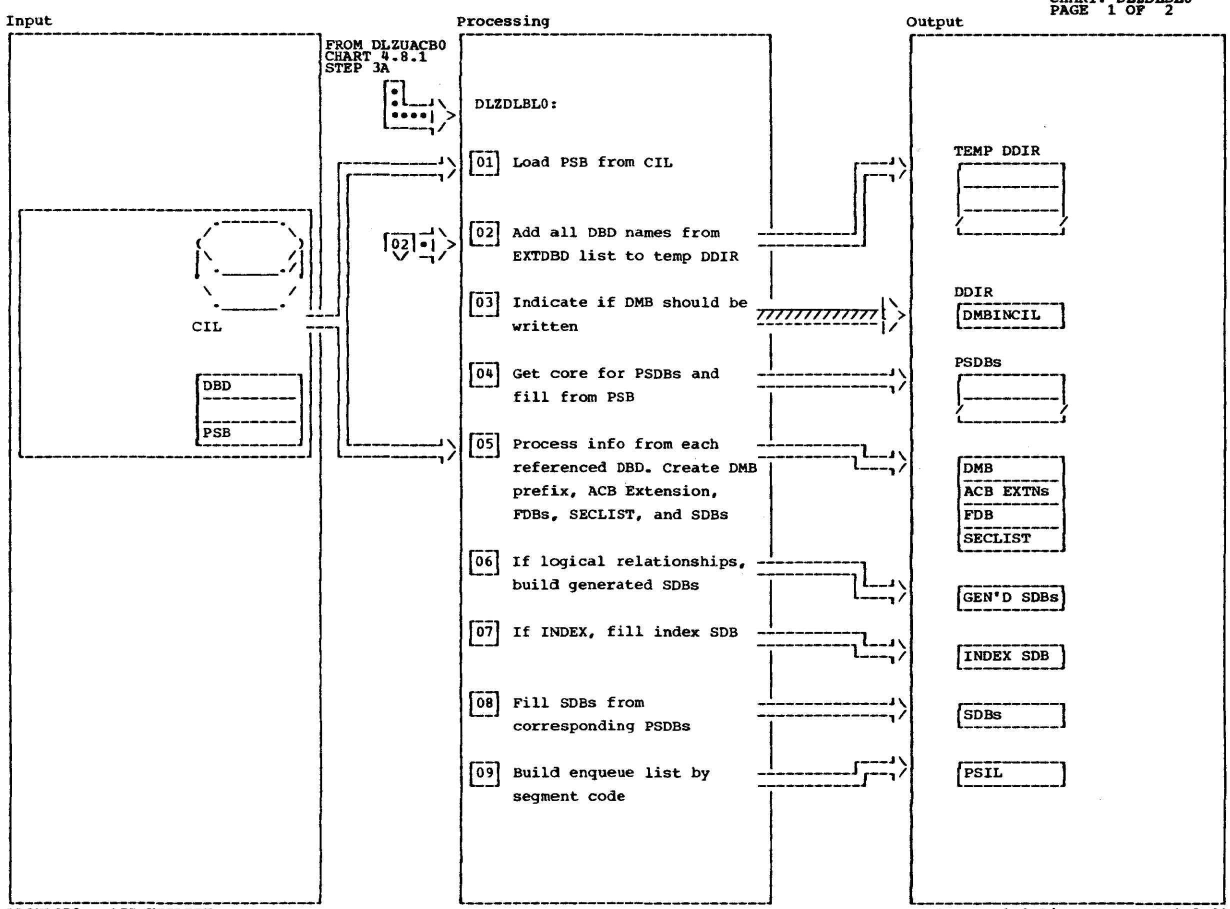
DLZUACB0 - DLZUSCH0 - INSRCH

HIPOMAT 1.1 Diagram - 4.8.1.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01	INSRCH	INSRT1					
02 Enough room for 16 entries is acquired.		NOTFND					
03		MVOK					
04		OKGO					
05		NOGO					

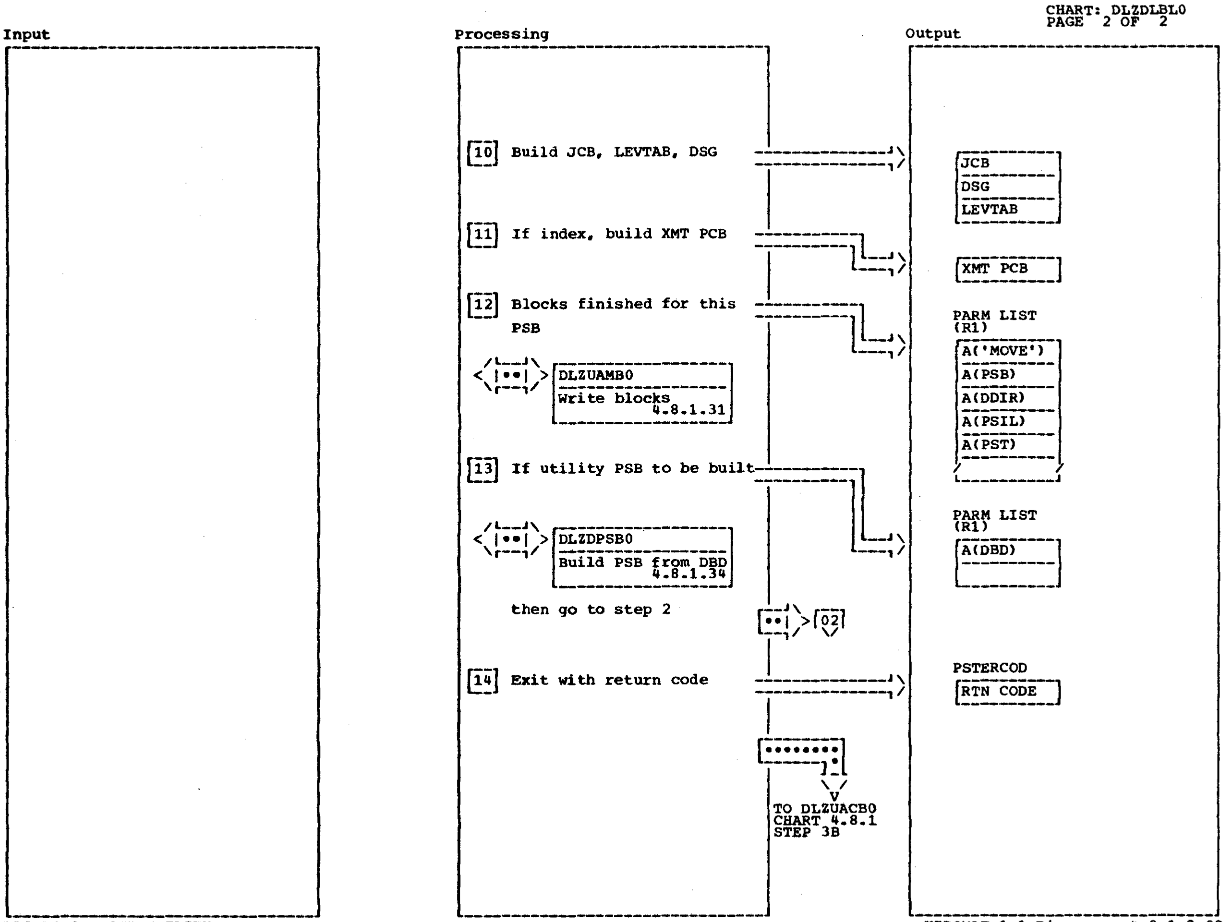
DLZUACB0 - DLZUSCH0-INSRCH

HIPOMAT 1.1 Diagram - 4.8.1.2-01



Notes	Routine	Label	Ref
01 The PSB name is found in the PDIR at PDIRSYM.			
02		UPSBENT DUMDIRA	
03 All DMB names are also kept in an internal table, so we know which DMBs have been built during this run for a previous PSB and need not be built again. DMBs are also not built if they are found in the CIL or are LOGICAL.		DUMDIRD1	
04 SDBs are partially created at this time also.		SGTABGS LOADDBD	
05 LCHDTAB entries and INDXTAB entries are converted to secondary list entries. Connections are made during construction. Because either Version 1.0 or 1.1 DBDs are accepted, special logic exists to remove VSAM ACB(s) from the DBD if V 1.0.		SIGMAA EPSILONA TYP64RES BLDSECNX EPSILOZZ	
06 Various checks are made to insure all logical connections are correct.		BLDLOGA BLDSDBP	
07 Special processing when PROCSEQ specified		SUMA	
08 Parentages established and checked here.		SUM SUMSPB PARNTST	
09 Sensitivity is determined by user's PROCOPT specification and special rules for logical relationships and indexes. Output will be the PSIL (Program Segment Intent List) and is for DMBs - not PSBs.	DLZDLBA0	SUMSPMN INTPROP	

CHART: DLZDLBL0
PAGE 2 OF 2



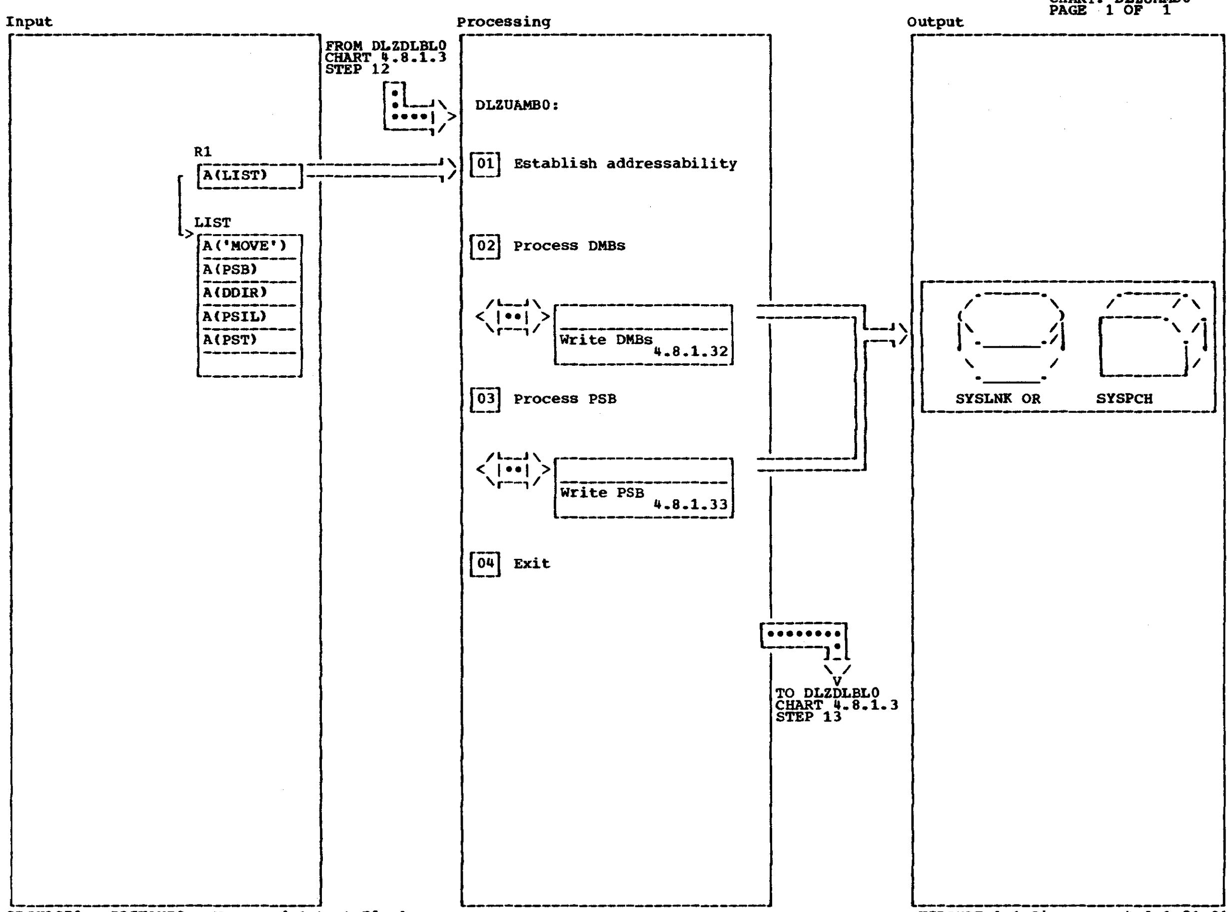
DLZUACB0 - ACB-UTILITY HIPOMAT 1.1 Diagram - 4.8.1.3-02

Notes	Routine	Label	Ref
10 At this point, checks are made to insure the PCB structure is valid.	BLDJCB	BLDJCB HIERCK01 SUMSPE1 SUMSPH SUMSPO	
11 This is required when any index is involved. The PCB contains a DBPCB, JCB, 2 DSGs, 2 SDBs, and 2 LEVTABs. Sizes of required work areas are set.		BLDEND BLDNONDX	
12 Block Mover (DLZUAMB0) called with register 1 containing address of parameter list.		SIGMA	
13 Utility PSB is built for every HISAM, HDAM, HIDAM, and secondary index DMB that was just written by the Block Mover.		NXTFLAG SETUPSB	
14 PSTERCOD = 0 if OK and non-zero if not. If an error was detected, DLZLBLM0 is called to set up the message parameter list for the specified error message.		FREEDBD	

Notes	Routine	Label	Ref
DLZUACB0 can call DLZDLBL0 at its primary entry point to process another PSB name.			

DLZUACB0 - ACB-UTILITY HIPOMAT 1.1 Diagram - 4.8.1.3-02

CHART: DLZUAMB0
PAGE 1 OF 1



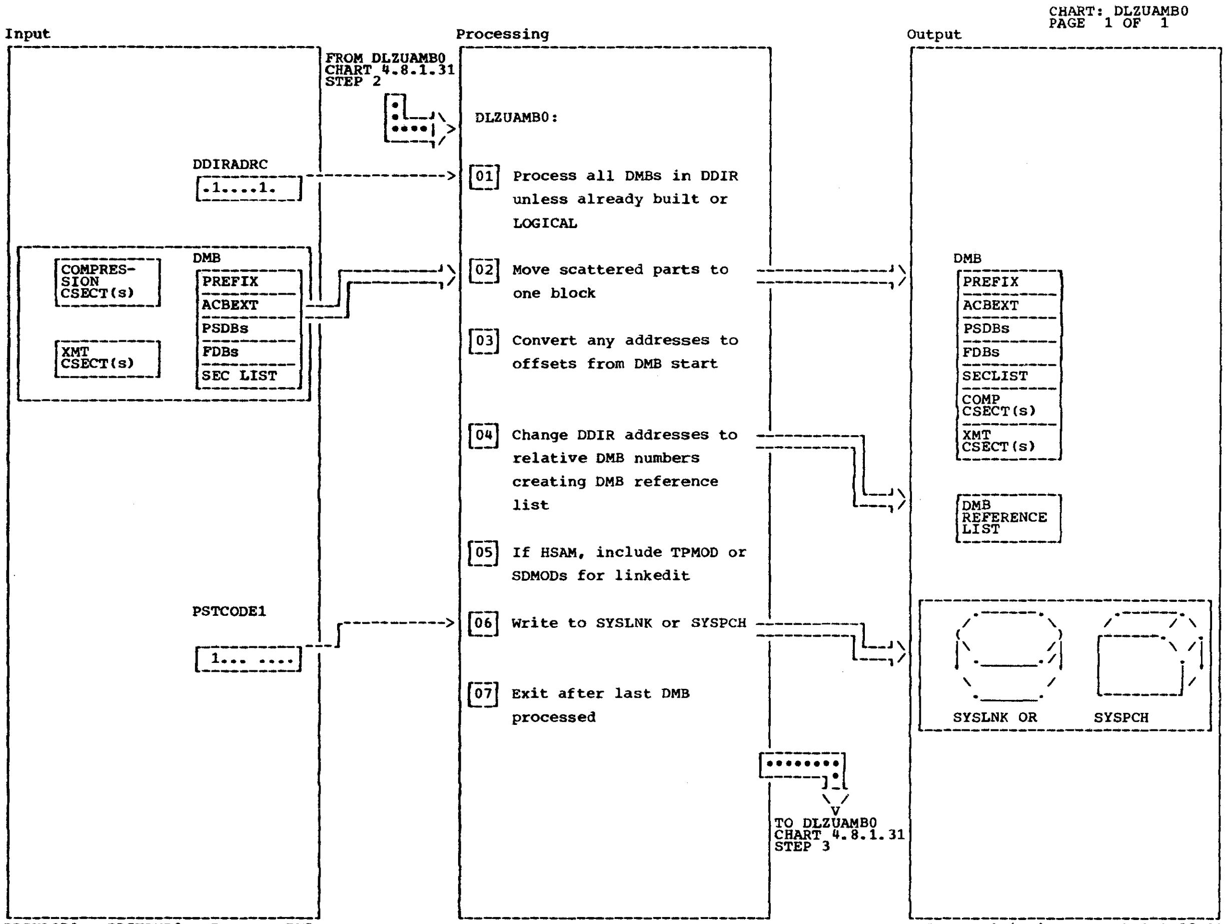
DLZUACB0 - DLZUAMB0 - Move and Output Blocks

HIPOMAT 1.1 Diagram - 4.8.1.31-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 A parameter list is passed to this module via Register 1. From this all blocks can be found.	DLZUAMB0	SAVE					
02		LOOPDDIR					
03		PROCPSB					
04 If an error occurred, PSTERCOD is non-zero.		RETURN ERRET					

DLZUACB0 - DLZUAMB0 - Move and Output Blocks

HIPOMAT 1.1 Diagram - 4.8.1.31-01



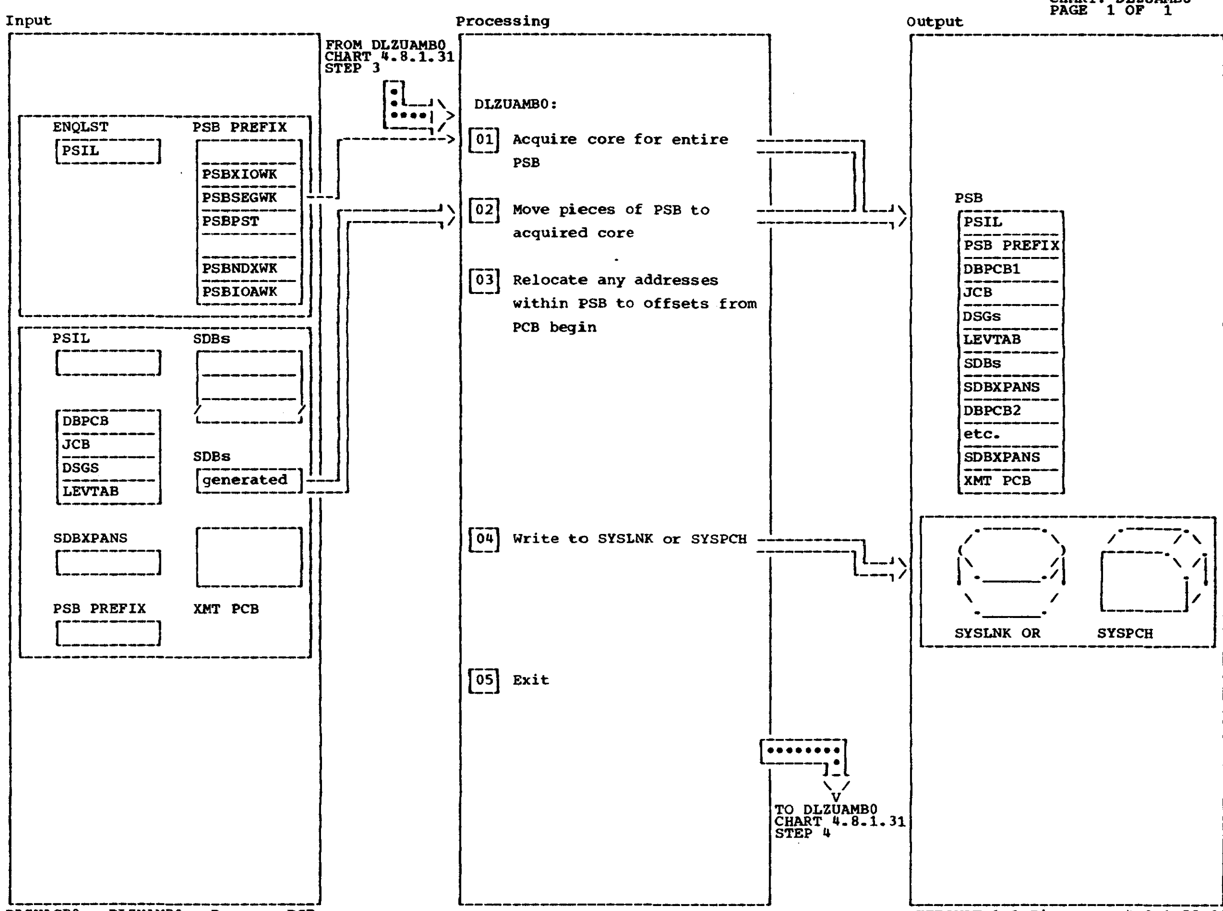
DLZUACB0 - DLZUAMB0 - Process DMBs HIPOMAT 1.1 Diagram - 4.8.1.32-01

Notes	Routine	Label	Ref
01 This chart is performed for all DMBs.		LOOPDDIR	
02 Any addresses which do not fall within DMB are set to zero.		PROPSET NXTCSECT NXSECLST	
04 The reference list is the last part in the DMB.		COD1	
05 The names in the relocatable library of the required mod's are DLZTAPE or DLZDISKI and DLZDISKO.		ISHSAM	
06 The same subroutine is used for the PSB also.		PUTLNK	
07		LOGICAL	

Notes	Routine	Label	Ref

DLZUACB0 - DLZUAMB0 - Process DMBs HIPOMAT 1.1 Diagram - 4.8.1.32-01

CHART: DLZUAMB0
PAGE 1 OF 1



DLZUACB0 - DLZUAMB0 - Process PSB

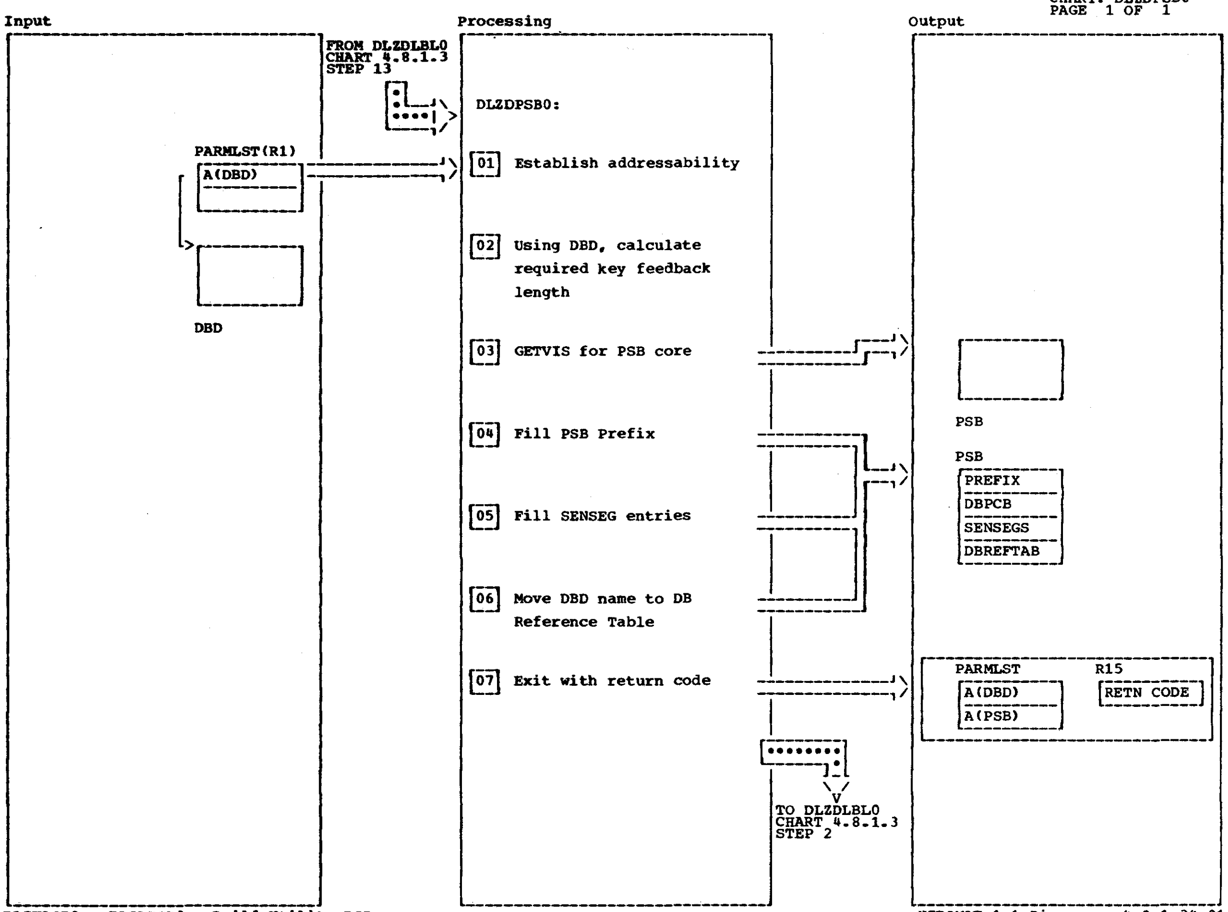
HIPOMAT 1.1 Diagram - 4.8.1.33-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 GETVIS is used. The size calculation formula is PSBPST - PSBXIOWK - PSBSEGWK - PSBNDXWK - PSBIOAWK + len of PSIL.		PROCPSB					
02 In reality, steps 2 and 3 are performed simultaneously.		MVDBPCB MVDBPCB1 MVDBPCB2 CHKTRGT CKEXPM					
04		PUTLNK					
05 FREEVIS issued to free up PSB core acquired.							

DLZUACB0 - DLZUAMB0 - Process PSB

HIPOMAT 1.1 Diagram - 4.8.1.33-01

CHART: DLZDPSB0
PAGE 1 OF 1



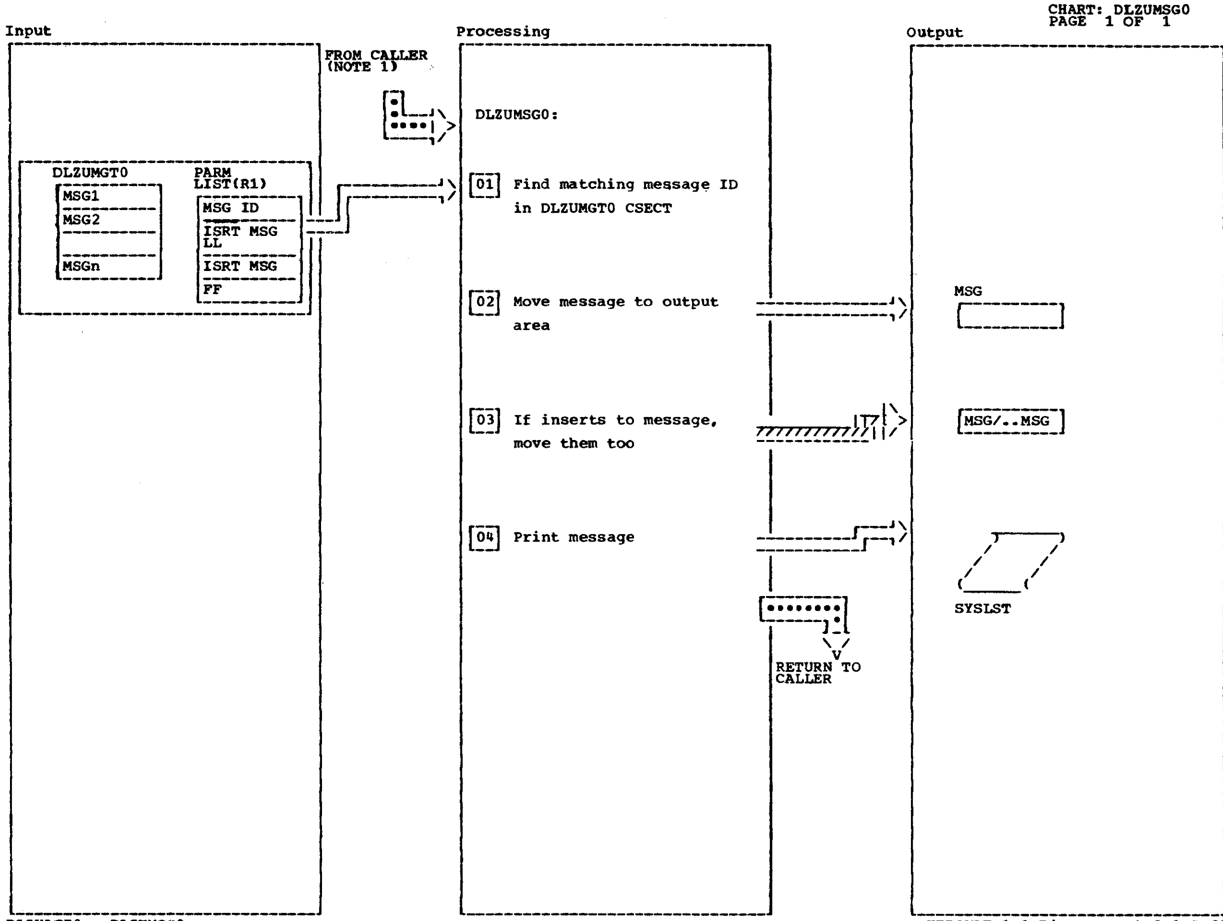
DLZUACB0 - DLZDPSB0 - Build Utility PSB

HIPOMAT 1.1 Diagram - 4.8.1.34-01

Notes	Routine	Label	Ref
01 Parameter list containing DBD address is passed in Register 1. The contents of this DBD are used to create the utility PSB.	DLZDPSB0	INIT	
02 Result will be stored in PSB Prefix.		SEGLOOP GETKEYSZ	
03 The area is also cleared to zeros.		USECURR1	
04 PROCOPT of 'A' is set for all DBD's except secondary index where 'LS' is set.		CLRDONE	
05 Same PROCOPT as Note 04.		PSEGLOOP	
06 In addition, no SORTAB is indicated.		SETDBREF	
07 The address of the built utility PSB is returned to the caller in the parameter list.		RETURN	

DLZUACB0 - DLZDPSB0 - Build Utility PSB

HIPOMAT 1.1 Diagram - 4.8.1.34-01



DLZUACB0 - DLZUMSG0 HIPOMAT 1.1 Diagram - 4.8.1.4-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 This routine can be called by DLZUACB0, DLZLBLM0, DLZUAMB0, or DLZDPSB0.	DLZUMSG0	TESTID					
02 DLZUMGT0 also has information concerning message (length, inserts allowed, length of inserts).		IDEQU					
03 These values are set by the calling program at execution time.		NXTIN					
04 Actually a subroutine in DLZUACB0 CSECT is used.							

Input | Processing | Output

FROM DLZUACB0
CHART 4.8.1
STEP 4

CLOSESCH:

R1
A(S/C
BLOCK)

S/C BLOCK

[01] FREEVIS the save control block and all entry blocks

[02] Return to caller

TO DLZUACB0
CHART 4.8.1
STEP 5

Notes	Routine	Label	Ref
[01] All entry blocks can be found beginning from the save/control block.		FRNEXT	
[02] No return-code.		TEXTGRP	

Notes	Routine	Label	Ref

Input	Processing	Output
CONTROL CARDS	FROM DOS/VS	
	DLZUCUM0:	
	01 Get time and date of this execution	
CONTROL CARDS	02 Call DLZUCCT0	
	DLZUCCT0 — Read and process control cards 4.9.1.1	
	03 If no CUMOUT but LOGOUT, call DLZUC150 at entry point DLZUEX15	LOGOUT
	DLZUC150 — Write LOGOUT 4.9.1.2	
LOGIN01-99; CUMIN (OPTIONAL)	04 If CUMOUT, loaded sort program (SORT) will call DLZUC150 at its Exit15 and DLZUEX35 at Exit35	CUMOUT; LOGOUT (OPTIONAL)
	A. Sort LOGINxx	
	SORT	
	B. Go to Step 6 on completion of sort → 06	

DLZUCUM0 - CHANGE ACCUMULATION UTILITY

HIPOMAT 1.1 Diagram - 4.9.1-01

Notes	Routine	Label	Ref
01 Header line is printed on SYSLST.		TIMEDEC	
02 Three returns as follows:		READCD	
A. Error - issue error message		BADEND	
B. No cum output, call DLZUEX15. Then issue successful run message.		GOODEND	
C. Cum output, call SORT.		SORT	
04 SORT is invoked by LOAD and BALR. At exit 35, DLZUC350 is called at entry point DLZUEX35. See chart 4.9.1.3 .		SORT	

Notes	Routine	Label	Ref

Input	Processing	Output
	FROM DLZUC150 4.9.1.2 → 05 Entry if error detected in DLZUC150. Establish addressability	
PROCFLAG 1	STEP 4B → 06 Write completion message	MESSAGE
	DLZUCER0 Write messages 4.9.1.4	
	07 Exit	
	RETURN TO DOS/VS	

DLZUCUM0 - CHANGE ACCUMULATION UTILITY

HIPOMAT 1.1 Diagram - 4.9.1-02

Notes	Routine	Label	Ref
05 This entry point is necessary because DLZUC150, not knowing who called him (DLZUCUM0 or SORT), must return to this module if an error was detected.		DLZERRTN	
06 May be OK message or error message from SORT or DLZUC150 or DLZUC350. If PROCTERM X'01' bit on in PROCFLAG, an error occurred.		CLOSE	

DLZUCUM0 - CHANGE ACCUMULATION UTILITY

HIPOMAT 1.1 Diagram - 4.9.1-02

CHART: DLZUCCT0
PAGE 1 OF 1

Input | Processing | Output

FROM DLZUCUM0 CHART 4.9.1

DLZUCCT0:

SYSIPT

DMB FROM CORE IMAGE LIBRARY

01 Read control cards and validity check

02 Save appropriate control card data

03 Issue GETVIS for DB and date/time table, if required

04 Check for valid DMB in CIL. Prepare DB table and date/time table in alphabetical order

05 Test for valid combination of cards or no input (use default values)

06 Return to DLZUCUM0

RETURN TO DLZUCUM0 CHART 4.9.1

CUMCONST

DB-TABLE

DATE/TIME TABLE

DLZUCUM0 - Input card processor DLZUCCT0 HIPOMAT 1.1 Diagram - 4.9.1.1-01

Notes	Routine	Label	Ref
01 Possible card types are :		GETCARD	
A. 'ID' specifies db number, max key length, number of sort work and log files.			
B. 'DB0' describes records to be accumulated from input and written to CUMOUT.			
C. 'DB1' describes records be written to to new log file.			
D. Error card - call DLZUCER0 to write appropriate error message.		ERROR	
02 Data from control card(s) are saved in a CSECT residing in DLZUCUM0 addressable by all modules in this utility. DSECT name is DLZUCUMC.			
03 Tables are not required if *ALL was specified. The number of entries in each table is equal to the number of data bases as specified on the ID control card or default of 16.		GETMAIN	

Notes	Routine	Label	Ref
04 This information is filled from the DB0 and/or DB1 card(s) if present.		DDNUMCHK	
05 If any errors in Steps 3,4 or 5, call DLZUCER0 to write error message and exit. See Chart 4.9.1.4			

DLZUCUM0 - Input card processor DLZUCCT0 HIPOMAT 1.1 Diagram - 4.9.1.1-01

Input	Processing	Output
PARAMETER LIST: A(RECORD), A(RTNCODE)	FROM SORT OR DLZUCUM0 CHART 4.9.1	
	DLZUC150:	
	01 Establish addressability and save sort parameter list address	
	02 If initial entry, open LOGIN and LOGOUT (if needed). Obtain workspace via GETVIS	
LOGIN01-99	03 Read log records bypassing all but '2F' and '50'	
	04 The '50' record is bypassed under certain conditions	
	05 Write the selected '50' record to output log file	LOGOUT

DLZUCUM0 - CHANGE ACCUMULATION UTILITY

HIPOMAT 1.1 Diagram - 4.9.1.2-01

Notes	Routine	Label	Ref
01 Program has two entry points as follows:			
A. DLZUC150 - E.P. from SORT.			
Entered when sort wants another input record			
B. DLZUEX15 - E.P. from DLZUCUM0.			4.9.1.
03 On EOF the file is closed. If more input specified, xx (LOGINxx) in the DTF is incremented by 1 and the next log file is opened.			
04 Bypass '50' record for following:			
A. *ALL and log date/time < purge date/time			
B. Dbname match and log date/time < purge date/time			
C. No dbname match and *OTHER not specified			
D. No dbname match and log date/time < purge date/time			
05 Write log record for following '50' :			
A. *ALL on DB1 card			
B. Dbname match and dbname on DB1 card			
C. No dbname match and *OTHER on DB1 card			

DLZUCUM0 - CHANGE ACCUMULATION UTILITY

HIPOMAT 1.1 Diagram - 4.9.1.2-01

Input	Processing	Output
	[06] Pass selected log records to SORT	LOG RECORD
	SORT Sort Module	
	[07] If any error occurs, exit to DLZUCUM0 at entry point DLZERRTN TO DLZUCUM0 4.9.1	
	[08] After all input records are processed return to caller RETURN TO CALLER	

DLZUCUM0 - CHANGE ACCUMULATION UTILITY

HIPOMAT 1.1 Diagram - 4.9.1.2-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[06] Pass the following to SORT: A. '2F' after high date/time moved in B. '50' and *ALL on DB0 card C. '50' - dbname match and dbname on DB1 card D. '50' no dbname match and *OTHER on DB0 card							
[07] A special entry point is needed to return control to DLZUCUM0 immediately after an error condition is detected.			4.9.1				

DLZUCUM0 - CHANGE ACCUMULATION UTILITY

HIPOMAT 1.1 Diagram - 4.9.1.2-02

Input

PARAMETER LIST
A(LOGREC)
A(RETNCOD)

SORTED LOG RECORD

SORTED LOG RECORD

CUMIN

Processing

FROM SORT

DLZUC350:

01 Receive log record from SORT

02 Initial entry processing as follows :

A. Process '2F' record and obtain workspace via GETVIS

B. Determine device type for CUMIN (if specified) and CUMOUT

03 Sorted log records are merged with matching CUMIN records (if specified) and records written to CUMOUT

04 When next log record needed, return to SORT

05 When both input files at EOF, return to SORT

RETURN TO SORT

Output

CUMOUT

PARAMETER LIST
A(RETNCOD)

Notes	Routine	Label	Ref
01 SORT returns at EOF with an indication that no more records exist.		DLZUEX35	
02 DLZDEV macro obtains data from PUB. Device type may be TAPE or DASD.		TSTEODDB	
03 The following merging logic is used for comparison of LOGIN and CUMIN to create CUMOUT.			
A. For every new DMB name - data set id, a cum header record is written either from the CUMIN record or created from the '2F' record.			
B. Every CUMIN record is purge checked by date/time as specified by the user. The DB table as modified by DLZUC150 is used for a specific DMB or the *ALL/*OTHER purge date is used as applicable.			
C. If a matching log record is found, all log records with the same data id will be merged with the CUMIN record and written to CUMOUT.			
D. If there is no matching log record, the CUMIN record is written to CUMOUT unchanged.			
E. If log records exist but no CUMIN, the log records are accumulated by data-id and written to CUMOUT.			
04 A 'delete' return code is given to SORT so that sort does not further process the current record. SORT will prepare the next input record and enter this program at Step 1.			
05			
A. Free all work areas, close CUMIN and CUMOUT.		ENDJOB	
B. Indicate 'no return' to SORT		ENDSORT	

Input

FROM CALLER

DLZCUMM0

MSG1

MSG2

MSGn

R2

R1

MSG NUM

Processing

DLZUCER0:

[01] Obtain address of the message CSECT (DLZCUMM0) and output DTF

[02] If multi part message and first time through,

A. Calculate address of message

B. Set Reg 2 to next print position

C. Return to caller → 5

[03] Calculate address of message if single part message

[04] Write output message

5 → [05] Exit

RETURN TO CALLER

Output

R2

A(MSG)

SYSLST OR SYSLOG

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[01] This module can be called by DLZUCUM0, DLZUCCT0, DLZUC150, or DLZUC350.			4.9				
The address of the output DTF which has already been opened is found in the CUMCONST table.		INITSV					
[02] Multi part message indicated by negative Reg 2		TESTR2					
[03] Reg 1 contains message number		MSGCOMM					
[04] Output can be to SYSLST or SYSLOG. In reality, only SYSLST is used.		MSGWRT					
[05]		RETURN					

Input

FROM DL/I

PST

SCD

LOGIN

Processing

DLZBACK0:

[01] Obtain DL/I control block addressability via GSCD call. Open log input for read backwards

[02] Read and deblock DL/I log record. Only accept records for specified PSB

[03] Terminate processing if termination '07' or scheduling '08' record

[04] Bypass if '50' record and physical replace

[05]

A. Process all other records for this PSB

DLZRDBC0
Process log record 4.10.2

B. If no error occurred go to step 2

Output

LOG RECORD

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[01] Initialize PSTDBPCB, PSTDGU, PSTDGN		INIT					
[02] At end of file, go to step 6.		READ NXTLREC					
[03]		CHKLOGT					
[04]		CHKDPHYR					
[05] The log record is placed in a work area (READAREA) whose address DLZRDBC0 obtains via a V-con.		OK CALLBO					

Input	Processing	Output
	06 Write appropriate completion message	SYSLST SYSLOG
	07 Return to DL/I to purge buffers, close DMBs and Logout file.	RETURN TO DL/I

DLZBACK0 - DATA BASE BACKOUT UTILITY

HIPOMAT 1.1 Diagram - 4.10.1-02

Notes	Routine	Label	Ref
06 The input log file is closed.		EOF	
The message texts are found in DLZBACM0 CSECT.		MSGGEN	

Notes	Routine	Label	Ref

Input	Processing	Output
BUFFER POOL: SEGMENT	FROM DLZBACK0 CHART 4.10.1 STEP 5A DLZRDBC0: 01 Initialize dummy DSG and PST fields and open DMB if not open DLZDLOC0 — Open DMB 2.7.1 02 Read data base record containing segment data to be changed DLZDBH00 — Read record 1.3.1 03 If simple HISAM back out log record Chart 4.10.2.1. 04 If HISAM or INDEX, back out log record Chart 4.10.2.2	DUMMY DSG: DSGDMBNO, DSGINDA, DSGDCBA, DSGDCBNO PST: PSTDSGA, PSTACBNM

DLZBACK0 - DATA BASE BACKOUT PROCESSING HIPOMAT 1.1 Diagram - 4.10.2-01

Notes	Routine	Label	Ref
01		INIT	
02 The following calls are made to the buffer handler:		LOCDCB CALLBFRH	
A. If HISAM KSDS issue PSTSTLEQ call		SETISAMC	
B. If HISAM ESDS issue PSTBYLCT call		LOCBLK	
C. If HD ESDS issue PSTBKLCT call		SETBLKLT	

Notes	Routine	Label	Ref

Input	Processing	Output
	[05] If HD data base back out log record Chart 4.10.2.3	
	[06] Log data base change DLZRDBL0 DATA BASE LOGGER 1.4.1	LOGOUT
	[07] Write data base record DLZDBH00 Write record 1.3.1	DATA BASE
	TO DLZBACK0 CHART 4.10.1 STEP 5B	

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[06] Output log records contain the 'opposite' function which was on the input log.		CALLLOG LOG					
[07] The return code is checked and appropriate action is taken depending on call and return code.		WRITEBFR					

Input

LOG RECORD

Processing

FROM DLZRDBC0
CHART 4.10.2
STEP 3

DLZRDBC0:

[01] No action if key found and log record is physical delete or key not found and log record is physical insert or replace

[02] If physical delete get buffer space and move old data.

DLZDBH00
Get buffer space
1.3.1

Then set buffer handler function to add new key (PSTPUTKY)

[03] If physical insert, set buffer handler function to erase key (PSTERASE)

[04] If physical replace, replace data in buffer with old data

TO DLZRDBC0
CHART 4.10.2
STEP 4

Output

BUFFER POOL
New segm

BUFFER POOL
Updted seg

DLZBACK0 - SIMPLE HISAM BACKOUT PROCESSING

HIPOMAT 1.1 Diagram - 4.10.2.1-01

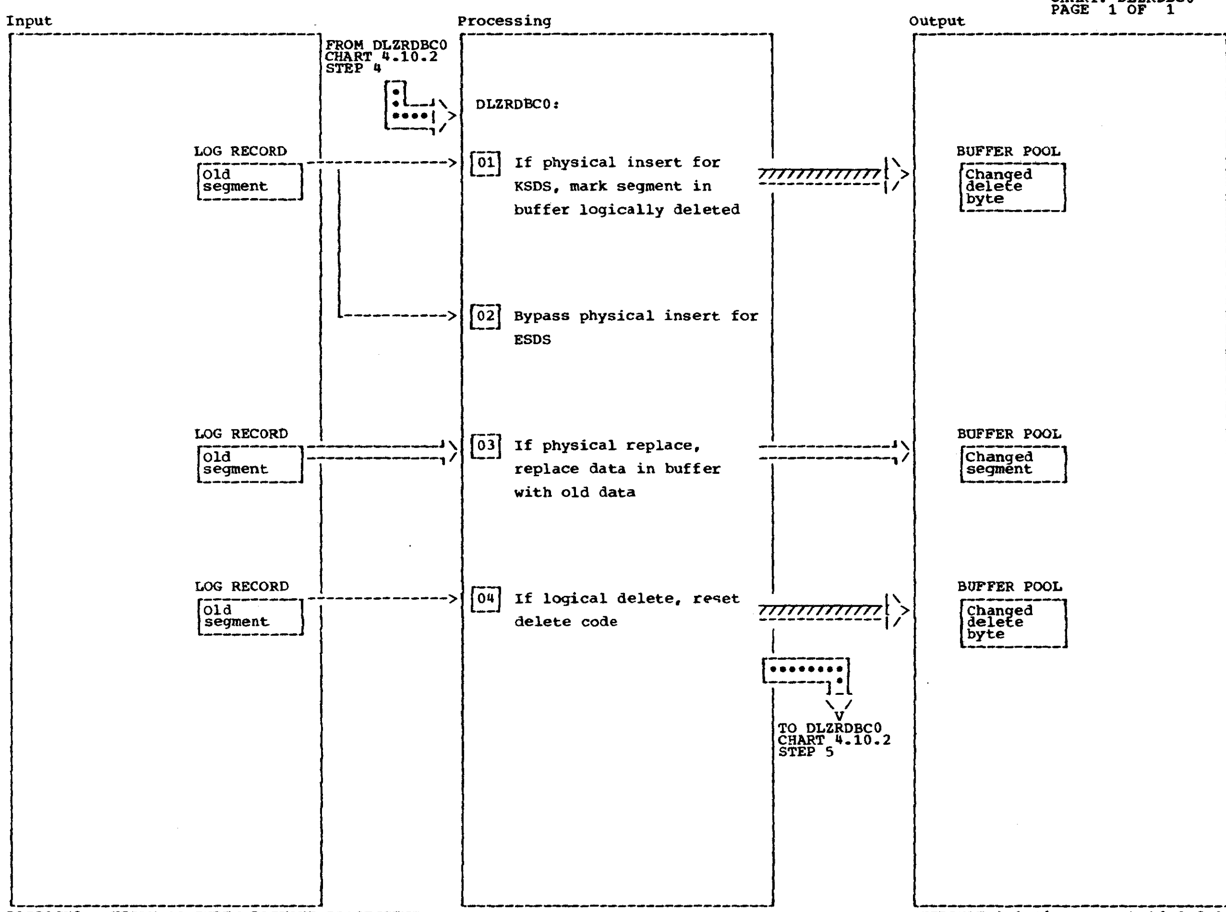
Notes	Routine	Label	Ref
[01] The address of the log record is input to this routine.		KEYNOTFD CKSHISAM	
[02]		KEYNOTFD	
[03]		CHKSHISM	
[04]		CALLREP	

Notes	Routine	Label	Ref

DLZBACK0 - SIMPLE HISAM BACKOUT PROCESSING

HIPOMAT 1.1 Diagram - 4.10.2.1-01

CHART: DLZRDBC0
PAGE 1 OF 1



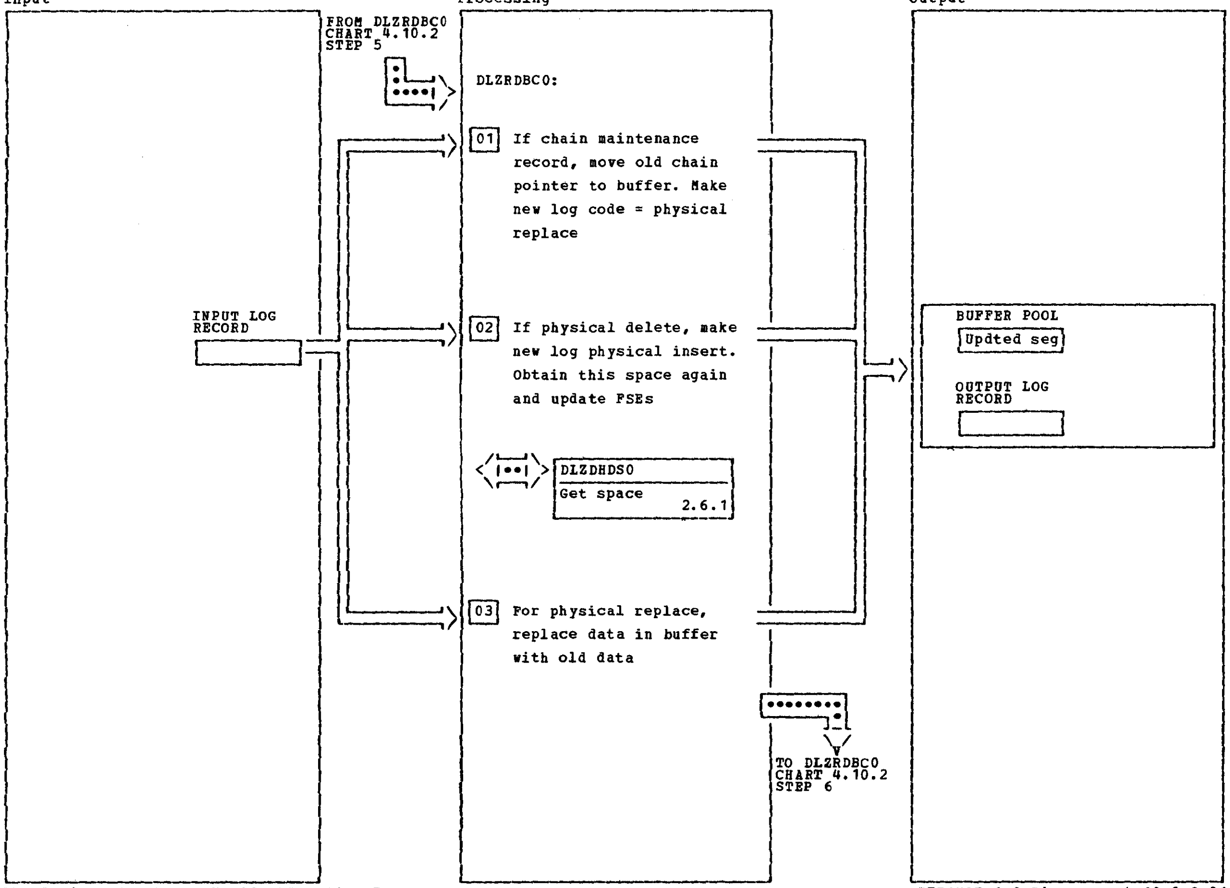
DLZBACK0 - HISAM OR INDEX BACKOUT PROCESSING

HIPOMAT 1.1 Diagram - 4.10.2.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 If segment is in an INDEX data base (primary or secondary), the pointer to the index target segment is also zeroed.		CHKUSERI LOGDLET SETPHYRP					
02 Chain maintenance log records for KSDS effectively back out physical insert to ESDS.		CHKUSERI					
03		CKDICALL					
04		CHKLGDLT					

DLZBACK0 - HISAM OR INDEX BACKOUT PROCESSING

HIPOMAT 1.1 Diagram - 4.10.2.2-



DLZBACK0 - HD DATA BASE BACKOUT PROCESSING

HIPOMAT 1.1 Diagram - 4.10.2.3-01

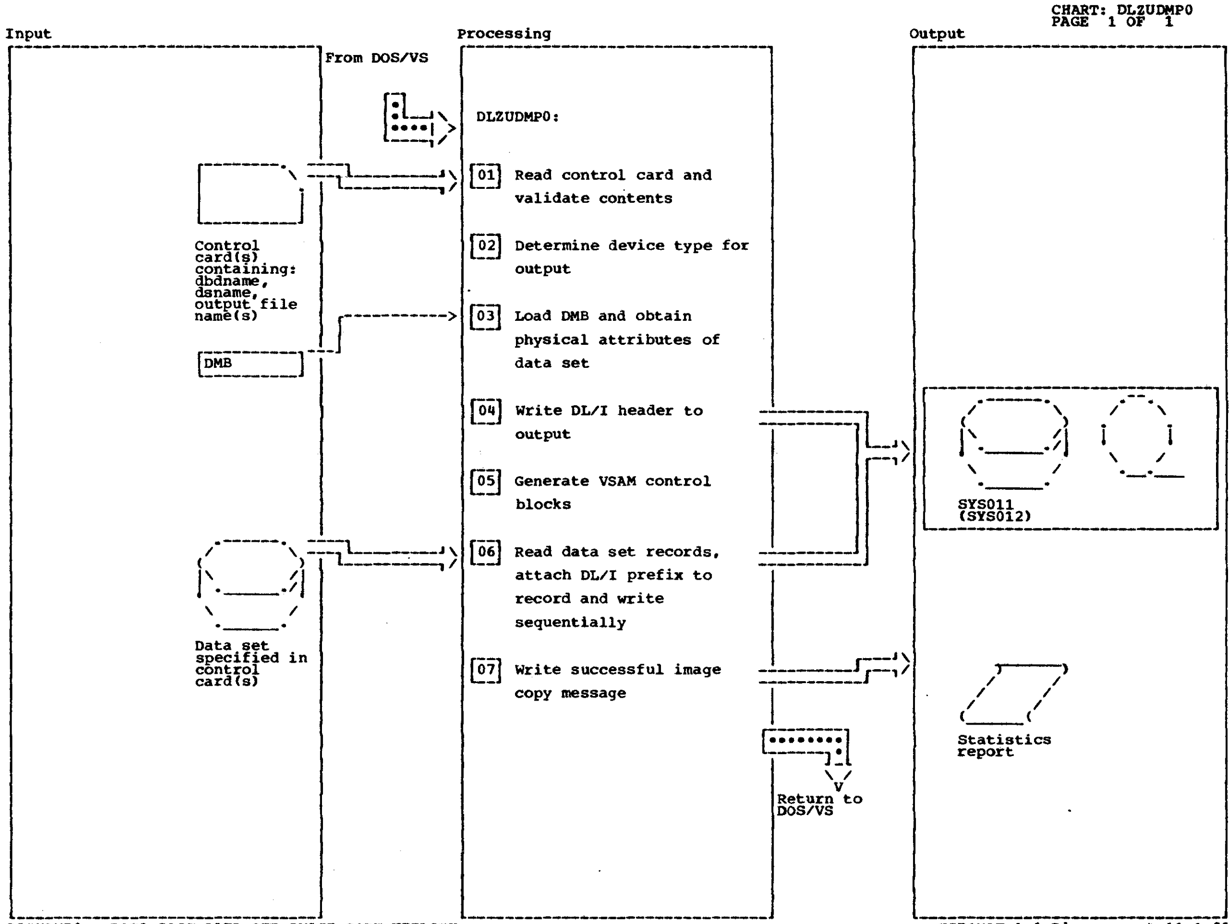
Notes	Routine	Label	Ref
01		CHKCHAIN	
02		NEXTFSE	
		NOTINFS	
		LASTCOMP	
03		NOTINFS	
		CHKREPPC	

Notes	Routine	Label	Ref

ZBACK0 - HD DATA BASE BACKOUT PROCESSING

HIPOMAT 1.1 Diagram - 4.10.2.3-01

CHART: DLZUDMP0
PAGE 1 OF 1

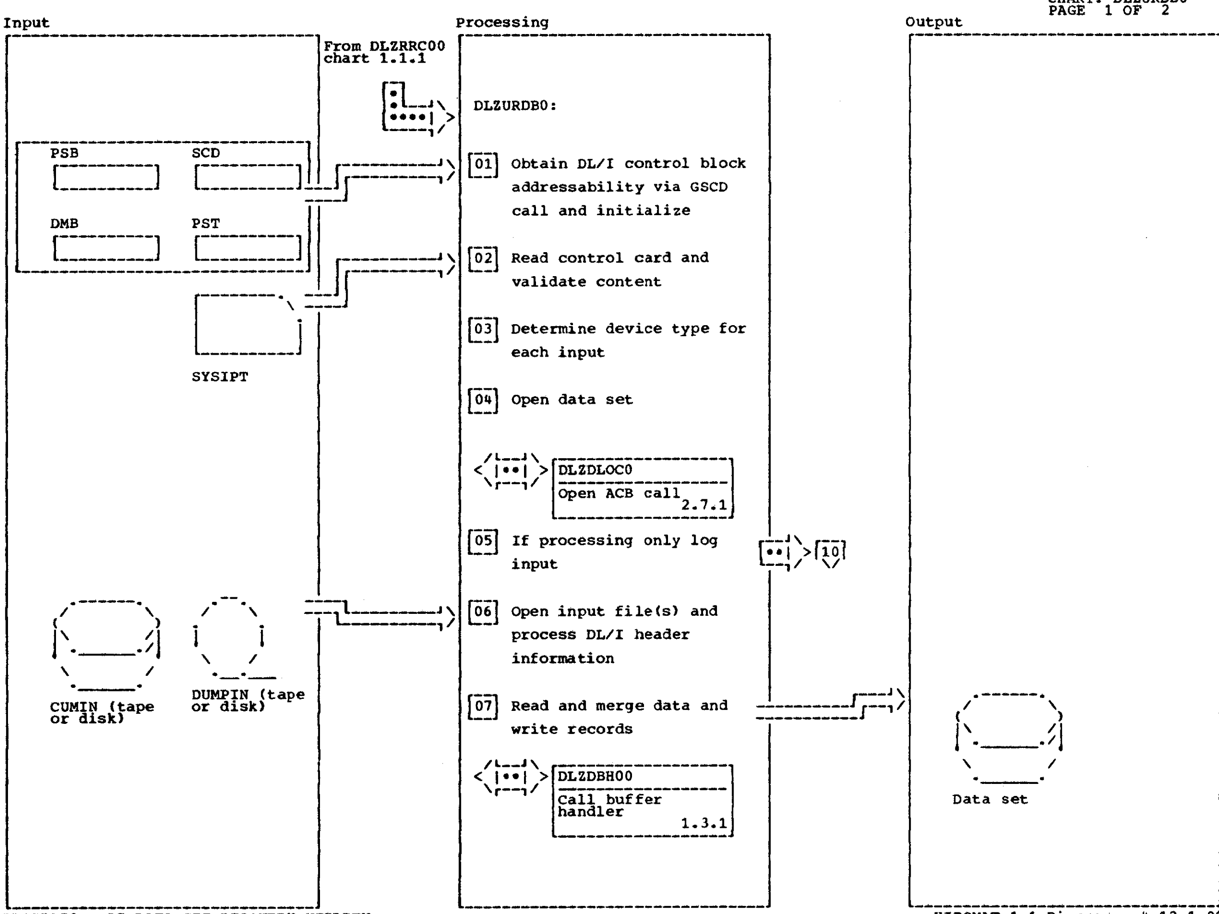


DLZUDMP0 - DATA BASE DATA SET IMAGE COPY UTILITY HIPOMAT 1.1 Diagram - 4.11.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Validate DBD name, input data set name, output filename(s) and number of output files.							
02 DLZDEV macro obtains data from PUB. Device type may be tape or dasd.							
05 Use GENCB to generate ACB, RPL and EXLST.							

DLZUDMP0 - DATA BASE DATA SET IMAGE COPY UTILITY HIPOMAT 1.1 Diagram - 4.11.1-01

CHART: DLZURDB0
PAGE 1 OF 2



DLZURDB0 - DB DATA SET RECOVERY UTILITY

HIPOMAT 1.1 Diagram - 4.12.1-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
01 Fields initialized are PSTDSGA, PSTACBNM, PSTFNCTN, DSGDMBNO.							
02 Validate CTL card identifier, DBD name, output data set name and number of log files.							
03 DLZDEV macro obtains data from PUB. Device type may be TAPE or DASD.							
06 DUMPIN file is mandatory and may be output from DLZUDMP0 or DLZURUL0. CUMIN file is optional and is output from DLZUCUM0.							
07 Records are read from DUMPIN and CUMIN (via GET calls) and written in ascending order (compare by key if KSDS, by RBA if ESDS). Proper PSTFNCTN is supplied for call of buffer handler.		SETFLOW					

Input

LOGIN01 to LOGINXX

Processing

[08] Close input files

[09] If no log input to be processed → [13]

[10] Open LOGIN file

[11] Process all log records for this data set

DLZDBH00
Call buffer handler
1.3.1

[12] Close LOGIN file

[13] Close data set

DLZDLOC0
Close ACB call
2.7.1

Return to DOS/VS

Output

Data set

Notes	Routine	Label	Ref
[09] LOGIN is optional.			
[11] LOGIN01 to LOGINXX files are processed sequentially.		PROCLOGS	

Notes	Routine	Label	Ref

Input | Processing | Output

FROM DOS/VS JCL

DLZURG10:

SYSIPT

[01] Read control card

[02] Determine device type and open first input workfile and output files

SYS012

[03] Read control data set

[04] Determine device type and open first input workfile and output files

SYS013

[05] Execute SORT/MERGE

SYS014

SYS010

DLZURG10 - PREFIX RESOLUTION UTILITY

HIPOMAT 1.1 Diagram - 4.13.1-01

Notes	Routine	Label	Ref
[02] DLZDEV macro obtains data from PUB. Device type may be TAPE or DASD.		CDSIN	
[04] See note 2.		OPENRT1	
[05] Sort is by (13,255,A,5,1,A). Exits E15 and E35 are described in chart 4.13.1.1 and 4.13.1.2 .		SORT1	

Input	Processing	Output
SYS010	[06] If logical relationships, close intermediate file and reopen it. Open LR output file and execute SORT/MERGE	SYS011
	[07] Print statistics and message summary if requested	SYSLST
	[08] Close all files	
	RETURN TO DOS/VS JCL	

DLZURG10 - PREFIX RESOLUTION UTILITY

HIPOMAT 1.1 Diagram - 4.13.1-02

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[06] Sort is by (29,16,A,5,1,A). Exits E15 and E35 are described in chart 4.13.1.3 and 4.13.1.4 .		SORT2					
[07] Control data set contains user options as specified in DLZURPR0.		SUMM					

Input

SYS013

FROM DOS/VS SORT/MERGE

Processing

SORT E15 EXIT DLZX15S1:

[01] Read workfile records and pass to SORT/MERGE

[02] At EOF, close workfile and open next one, if any. Otherwise indicate end to SORT/MERGE

RETURN TO DOS/VS SORT/MERGE

Output

DLZURG10 - SORT E15

HIPOMAT 1.1 Diagram - 4.13.1.1-01

Notes	Routine	Label	Ref
[01] Record length is changed to 270 to force it being greater than control fields. Original record length is saved in last 2 bytes of LRECL field.		EXIT15S1	

Notes	Routine	Label	Ref

DLZURG10 - SORT E15

HIPOMAT 1.1 Diagram - 4.13.1.1-01

Input		Processing		Output
	FROM DOS/VS SORT/MERGE	SORT E35 EXIT DLZX35S1:		
		[01] Get records from SORT/MERGE		
		[02] Process record depending on record type		
		[03] If secondary indexing, write to SI output file		SYS014
		[04] If logical relationships, write to intermediate file		SYS010
			TO DOS/VS SORT/MERGE	

DLZURG10 - SORT E35

HIPOMAT 1.1 Diagram - 4.13.1.2-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[01] Original record length is restored before processing.		EXIT35S1					
[03] This is final output for secondary index relationships.		TYPE04RT					
[04] This file used as input for second SORT/MERGE.		ESTTYPE					

DLZURG10 - SORT E35

HIPOMAT 1.1 Diagram - 4.13.1.2-01

Input

FROM DOS/VS SORT/MERGE

SYS010

Processing

SORT E15 EXIT DLZX15S2:

[01] Get records from intermediate file and pass to SORT/MERGE

[02] At EOF, indicate end to SORT/MERGE

TO DOS/VS SORT/MERGE

Output

DLZURG10 - SORT E15

HIPOMAT 1.1 Diagram - 4.13.1.3-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref
[01] This file was written during first SORT.		EXIT15S1					

DLZURG10 - SORT E15

HIPOMAT 1.1 Diagram - 4.13.1.3-01

Input | Processing | Output

FROM DOS/VS SORT/MERGE

SORT E35 EXIT DLZX35S2:

01 Get sorted records

02 Write to LR output file

SYS011

TO DOS/VS SORT/MERGE

DLZURG10 - SORT E35

HIPOMAT 1.1 Diagram - 4.13.1.4-01

Notes	Routine	Label	Ref	Notes	Routine	Label	Ref

DLZURG10 - SORT E35

HIPOMAT 1.1 Diagram - 4.13.1.4-01

REGION CONTROL

DL/I BATCH INITIALIZATION - DLZRRC00

This module receives control from DOS/VS job management and serves as the initialization routine for batch DL/I initialization. Its responsibilities are to:

- Establish base register addressability
- Read required PARM information from SYSIPT or SYSLOG based on the UPSI byte setting as follows:

Bit	Value	Meaning
0	0	Read PARM information from SYSIPT
	1	Read PARM information from SYSLOG
1-4		Reserved for Application Program use
5	0	Storage dump at STXIT ABEND if STXIT is active
	1	No storage dump at STXIT ABEND if STXIT active
6	0	Record data base modifications on system log
	1	Bypass system log
7	0	STXIT ABEND processing if abnormal termination
	1	Bypass STXIT ABEND processing

- Determine load address for batch nucleus module (DLZBNUC0)
- Provide a DL/I message subroutine (DLZERRMS)
- Branch to region control interface (DLZRRC10)

Entry Interface - DLZRRC00

Receives control from DOS/VS job management

Exit Interface - Region Control (DLZRRC10)

Passes control through branch to DLZRRC10.

Register Contents:

R2	Batch nucleus (DLZBNUC0) load address
R3	Address of PARM information
R6	Address of SYSLOG DTF
R7	Address of DLZERRMS

Entry Interface - DLZERRMS

Receives control through BALR from DL/I action module

Register Contents:

R1	PST address
R13	Save area address
R14	Return address
R15	Entry point address (DLZERRMS)

Exit Interface - Calling Module

Passes control through branch on register 14

REGION CONTROL PRIMARY INTERFACE - DLZRRC10

This routine receives control from the DL/I initialization routine and serves as the primary interface for all DL/I program executions. Its responsibilities are:

- Save input parameters
- Load batch nucleus module (DLZBNUC0)
- Establish SCD and PST addressability
- Invoke parameter analysis (DLZRRA00)
- Branch to application program control module (DLZPCC00)

Entry Interface - DLZRRC10

Receives control through branch from DLZRRC00

Register Contents:

R2	Batch nucleus (DLZBNUC0) load address

Exit Interface - Parameter Analysis

Passes control through branch to DLZRRA00

Register Contents:

R2	Address of SCD
R3	Address of PARM information
R9	Address of PST
R13	Save area address

USER PARAMETER ANALYSIS - DLZRRA00

Associated with each PARM format is a generalized specification table which describes the input PARM string format, conversion and length, and output parameter format. The input PARM string and its associated specification table are processed by DLZRRA00.

The PARM specification table is used to process element by element through the PARM value. For each parameter value there is a specification table element. The element identifies each parameter value as fixed, positional, required, not required, or last, and specifies whether binary conversion is to be performed.

If the PARM character string is of valid length, it is classified by format, and certain functions common to format groups are performed. The internal parameter areas are contained in the PST in the batch nucleus module (DLZBNUC0).

These parameter areas are used as work areas during parameter analysis and for region and program control, and contain internal representations of the PARM values, remote supervisor parameter lists, register save areas, and application program parameter lists.

If parameter string analysis is unsuccessful, a request is made to the console operator to re-enter parameter information or terminate initialization.

Layout and Description of PARM Field

	xxx,aaaaaaaa,bbbbbbb,ccc,keyword operands
xxx	PARM identifier in columns 1-3. DLI = Data base program to be executed. UDR = Data base recovery utility to be executed. ULU = Data base reorganization or logical relationship resolution program to be executed. ULR = HD reorganization reload utility to be restarted from checkpoint record.
aaaaaaaa	One- to eight-character name of the application program to be executed.
bbbbbbb	One- to seven-character name of the program specification block (PSB) as specified in the PSB generation. If PARM is UDR, ULU, or ULR, one- to seven-character name of the data base description (DBD) as specified in the DBD generation.
ccc	Number of data base buffer sub-pools required for job execution.
keyword operands	HDBFR, HSBRD, ASLOG, and TRACE.

Entry Interface

Receives control through branch from DLZRRC10

Register Contents:

R2	Address of SCD
R3	Address of PARM information
R9	Address of PST
R13	Save area address
R14	Return address

Exit Interface

Passes control through branch to DLZPCC00

Register Contents:

R2	Address of SCD
R9	Address of PST

APPLICATION PROGRAM CONTROL

APPLICATION PROGRAM CONTROL - DLZPCC00

This routine is used only in the batch regions. It performs some functions analogous to those performed by the CICS scheduler in the online control program. It is responsible for the following functions:

- Initializing the storage management routine
- Invoking the application control blocks loader/relocator (DLZDBLM0)
- Invoking the control program initialization routine
- Loading the application program
- Initializing the PL/I region (if PL/I)
- Invoking the application program
- Issuing an unload call in behalf of the application program upon termination
- Writing the application program termination record on the DL/I log
- Closing the DL/I log.

Control Blocks - DLZPCC00

PST

Entry Interface - DLZPCC00

Receives control through branch from DLZRRC10

Register Contents:

R2	Address of SCD
R9	Address of PST
R13	Save area address
R14	Return address
R15	Entry point

Exit Interface - Block Loading

Passes control through BAL to DLZPINIT (entry point in DLZDBLM0)

Register Contents:

R2	Address of SCD
R9	Address of PST
R14	Return address

Exit Interface - Application Program

Passes control through BAL to application program

Register Contents:

R1	Address of PCB address list
R13	Save area address
R14	Return address
R15	Entry point

Exit Interface - Unload Call

Passes control through branch to call analyzer (DLZDLA00) (batch control program)

Register Contents:

R1	Address of PST
R13	Save area address
R14	Return address
R15	Entry address of call analyzer obtained from the analyzer entry point field in the SCD

Exit Interface - DB Log Call

Passes control through BAL to data base logger (DLZRDBL0)

Register Contents:

R1	Address of PST
R13	Save Area Address
R14	Return Address
R15	Entry point of forced write log routine obtained from SCD at label SCDREENT

Exit Interface - DOS Supervisor

Issues an SVC 2 Normal EOJ Supervisor Call.

APPLICATION CONTROL BLOCKS LOAD AND RELOCATE - DLZDBLM0

This routine performs the functions of loading and relocating DL/I application control blocks. Once the blocks are loaded and offsets resolved to actual addresses, the SDBs in the PCBs are connected to the appropriate PSDBs in the DMBs. The JCB data sets in the data base are connected to the appropriate ACBs in the DMBs, and control is returned to the calling routine.

The module first checks to determine if this is a 'DLI', 'ULU', 'UDR', or 'ULR' execution. If 'DLI' execution, the PSB name extracted from the PARM card is moved to the PSB directory and the PSB is loaded. The address of the PSB segment intent list and the PSB are stored in the PSB directory. The index work area (if required) is allocated and addresses are resolved. Next the intent list is scanned and the DMB directory is constructed from it. The DMB directory entries are scanned and the DMBLOADR subroutine (see below) is called to load and relocate the DMBs in the directory. Upon completion, the SDBs are connected to their corresponding PSDBs, the JCB DSGs are connected to their ACBs, and return is made to the caller.

If 'ULU', 'ULR', or 'UDR' execution, a special utility PSB is loaded from the DOS/VS Core Image Library. The PSB name is generated from the DBD name in the PARM statement. The remaining processing is the same as for 'DLI'. For the following utilities there is no PSB name in the parameter information:

DLZURPR0 - Data base prereorganization
DLZURGS0 - Data base scan
DLZURGP0 - Data base prefix update

These utilities perform dynamic block loading using the DLZBLKLD macro (see Chapter 13 for a description of this macro).

The DMBLOADR subroutine performs the loading and relocation of DMBs. The DMB directory is accessed and the DMB name extracted from it. A load is issued for the DMB and, if HDAM, the randomizing module extracted from the DMB is loaded. Next, the DMB directory entry is updated with a buffer size indication. For HD, this value is the control interval size of the data set; for HISAM, it is the logical record size. Then all offsets are relocated to addresses, and control is passed to DLZCPI00.

INTERFACES - DLZDBLM0

Register Contents:

R9	PST address
R13	Address of one of a set of prechained save areas
R14	Return address
R15	Entry point

CONTROL PROGRAM INITIALIZATION - DLZCPI00

This routine receives control from the application control blocks load and relocate module and completes the intialization of the DL/I batch system. It is responsible for:

- Allocation of the buffer pool
- Formatting the buffer pool prefix, one or more subpool prefixes, and the buffer prefixes
- Loading all required DL/I action modules
- Initializing the SCD
- Opening the DL/I log
- Writing the application program scheduling record on the DL/I log

Entry Interface - DLZCPI00

Receives control through branch from routine DLZDBLM0.

Register Contents:

R2	Address of SCD
R3	Address of PDIR
R9	Address of PST
R13	Save area address
R14	Return address
R15	Entry point

Exit Interface

Through return to DLZPCC00

R1	Address of PST
R2	Address of SCD

LANGUAGE INTERFACE

LANGUAGE INTERFACE - DLZLI000

The language interface provides communication between the application program and the program request handler. A copy of this module is link edited with user application programs.

The language interface has responsibility for:

- Storing the user's registers in the save area provided.
- Providing a specific entry for Assembler, COBOL, and PL/I application programs.
- Locating the entry point of the program request handler.
- Passing control to the program request handler

Size of Module - DLZLI000

This module contains approximately 32 bytes of code.

Entry Interface - DLZLI000

Receives control through branch from application program

Register Contents:

R1	Call parameter list of implicit or explicit format
R13	Save area address
R14	Return address
R15	Entry point

Exit Interface

Passes control to program request handler through branch from DLZLI000

Register Contents:

R0	Language identifier code
R1	Parameter list
R2-14	As entered from application program
R15	Entry point of program request handler

BATCH NUCLEUS

PROGRAM REQUEST HANDLER - DLZPRHB0

The interface between the application program and the DL/I batch or control program is managed by the program request handler routine (DLZPRHB0) in module DLZBNUC0. It accepts parameters passed to it by the language interface module (DLZLI000), validates them, and passes a parameter list to the call analyzer.

The program request handler accepts three call list formats: implicit direct, explicit direct, and explicit indirect. COBOL and Assembler-language programs may use either the implicit direct or explicit direct call list formats. Since special provisions are made for PL/I in handling the explicit indirect call list, it may be used only by PL/I language programs.

The first parameter (argument 0) of the DL/I CALL determines whether the list is explicit or implicit. If the argument contains the address of the parameter count (count of the number of arguments that follow), this list is an explicit list. If the argument contains the address of the DL/I CALL function, this list is an implicit list.

The responsibilities of this routine are to:

- Verify parameter list addresses aligned and within the dynamic area of the machine
- Reformat explicit parameter lists to implicit prior to submission
- Reset PL/I STXIT PC processing
- Provide caller's parameter list to the call analyzer
- Return data to application program work areas
- Maintain PL/I variable-length character string dope vector
- Identify abnormal termination condition
- Return directly to application program

Control Blocks - DLZPRHB0

PDIR
PST

Entry Interface - DLZPRHB0

Receives control through branch from DLZLI000

Register Contents:

R0	Language indicator - zero if COBOL or Assembler; nonzero if PL/I
R1	Address of embedded parameter list in application program format
R13	Save area address
R14	Return (to application program)
R15	Entry point address

Exit Interface - to Call Analyzer DLZDLA00

Passes control through branch using entry point from SCD

Register Contents:

R1	Address of PST
R13	Save area address
R14	Return address
R15	Entry point of call analyzer

Exit Interface - to STXIT ABEND DLZABEND

Passes control through branch using entry point from SCD

Exit Interface - Return to Application Program

Passes control through branch using return address

Register Contents:

R2-12	Restored to contents upon entry from application program to language interface, DLZLI000
R14	Return address of application program

STXIT ABEND - DLZABEND

Abnormal terminations invoked through the DOS/VS STXIT or terminations requested by DL/I action modules are handled by DLZABEND. Responsibilities are as follows:

- Close the DI/I log.
- Issue an UNLD call to write the last records for Simple HSAM, HSAM, Simple HISAM and HISAM or write all buffers altered by the user. The UNLD call also closes the data base.
- If a dump is requested, write a formatted dump of DL/I control blocks.
- Cancel the partition.

Entry Interface

Receives control through DOS/VS STXIT PC interface or branch from the program request handler (DLZPRHB0).

Exit Interface - to DOS/VS CLOSE

Passes control through SVC 2 ($$BCLOSE) for data base logger

Register Contents:

R1	Address of DL/I log DTF

Exit Interface - UNLD Call

Passes control through branch to call analyzer (DLZDLA00)

Register Contents:

R1	Address of PST
R13	Save area address
R14	Return address
R15	Entry address of call analyzer obtained from SCD

Exit Interface - to DOS/VS

Passes control through SVC 6 (CANCEL) or SVC 2 ($$BDUMP)

DL/I IWAIT - DLZIWAIT

This module receives control when a DL/I action module requires DOS/VS wait linkage.

Entry Interface - DLZIWAIT

Receives control through BALR from a DL/I action module

Register Contents:

R2	Address of event control block
R14	Return address of caller
R15	Entry point of DLZIWAIT

Exit Interface

Passes control through SVC (WAIT) to DOS/VS.

Exit Interface

Passes control through branch on register 14 to the calling program.

MULTIPLE PARTITION SUPPORT (MPS)

START MPS TRANSACTION - DLZMSTR0

This module is invoked by the user via a specific transaction code (CSDA) to start multiple partition support (MPS). The responsibilities of this module are to:

- Check if the DL/I nucleus is loaded.
- Check if MPS is already active.
- Attach the master partition controller (DLZMPC00).

Size of Module

Approximately 500 bytes.

Control Blocks Addressed

CSA Common System Area (CICS/VS) SCD System Contents Directory

Register Contents

R13 Contains CSA address

MASTER PARTITION CONTROLLER (MPC) - DLZMPC00

The master partition controller (MPC) is attached by the start transaction module (DLZMSTR0).

The functions performed by the master partition controller are:

- Initialize the MPC partition table (DLZMPTCT).
- Define all XECBs required for cross partition communication.
- Process all start batch partition controller (BPC) requests and attach a BPC for a specific batch partition.
- Process all stop partition requests.
- Process the abend condition if the batch partition controller attach fails.
- Process the stop transaction request to terminate MPS.
- Return control to CICS/VS after all activity is completed.

Size of Module

Approximately 1956 bytes.

Control Blocks Addressed

MPCPT	MPC Partition Table
SYSCOM	System Communication Region
CSA	Common System Area (CICS/VS)
SCD	System Contents Directory
MPCECBLT	CICS ECB Pointer List
COMREG	Partition Communications Region
TCA	Task Control Area

Register Contents

R12 Contains TCA address (at entry)
R13 Contains CSA address (at entry)

Macros Used

```
DFHKC    TYPE=WAIT
DFHKC    TYPE=ATTACH
DFHKC    TYPE=RETURN
XECBTAB    TYPE=CHECK
XECBTAB    TYPE=DEFINE
XECBTAB    TYPE=DELETE
XPOST
```

BATCH PARTITION CONTROLLER (BPC) - DLZBPC00

The batch partition controller (BPC) is attached by the master partition controller (MPC) when a start request has been made by a batch partition. The functions performed by the batch partition controller are:

- Define XECB for cross partition communication with the MPS batch initialization (DLZMINIT), MPS batch program request handler (DLZMPRH), and MPS batch termination (DLZMTERM).
- Issue the DL/I scheduling call on behalf of the batch partition.
- Process all DL/I calls on behalf of the batch partition.
- Process ABEND conditions occurring in the batch partition.
- Return control to CICS/VS for normal and abnormal conditions

This module must be link-edited with the language interface module, DLZLI000.

Size of Module

Approximately 1140 bytes.

Control Blocks Addressed

MPCPT	MPC Partition Table
TCA	Transaction Control Area
TWA	Transaction Work Area
PST	Partition Specification Table
PPST	Prefix PST

Register Contents

R12 Contains TCA address (at entry)
R13 Contains CSA address (at entry)

Macros Used

```
DFHKC  TYPE=WAIT
DFHKC  TYPE=ATTACH
DFHKC  TYPE=RETURN
XECBTAB  TYPE=CHECK
XECBTAB  TYPE=DEFINE
XECBTAB  TYPE=DELETE
XPOST
```

MPS BATCH - DLZMPI00

The MPS batch module is made up of the following five routines:
1. MPS Batch Initialization (DLZMINIT)
2. MPS Batch Termination (DLZMTERM)
3. MPS Batch Program Request Handler (DLZMPRH)
4. MPS Batch Abend (DLZMABND)
5. MPS Batch Message Writer (DLZMMSG)

Size of Module

The MPS batch module contains approximately 5300 bytes of code. This includes constants and other non-code areas that are not included in the 'size of module' given for each routine.

A separate description for each routine is given in the following text.

MPS Batch Initialization - DLZMINIT

This is one of five routines that make up module DLZMPI00 to support the batch part of MPS.

DLZMINIT reads the input parameter statement and checks it for validity. It then loads the user's program. Then it determines what to use as a partition identifier by checking the PIK in the BG COMREG. This value, modified and made printable, is put into each XECBTAB macro issued.

After saving the program name and PSB name for use by online, an XECB, DLZXCBn1, is defined in the batch partition for communicating with the online partition. The online partition XECB (DLZXCBn0, with n being the identifier) is XPOSTed. This lets the online partition know that there is an MPS batch job ready to run in this batch partition.

When the online partition completes its initialization, the batch routine sets up STXIT routines, finishes other initialization activities, and goes to the user program.

DLZMINIT is entered by DOS/VS job control at the start of the job.

Size of Module

Approximately 1270 bytes (excluding constants and other noncode areas).

Control Blocks Addressed

MPCPT	MPC Partition Table
TCA	Transaction Control Area
PST	Partition Specification Table
COMREG	Communication Region
XCB1	XECB DLZXCBn1 and data following it
DTFs for	SYSLST, SYSLOG, and SYSIPT
STXIT AB	Savearea
STXIT PC	Savearea
XECBs	DLZXCBn0, DLZXCBn2, DLZXCBn3

Register Contents (at Entry to Other Routines

- User Program
  - R1 PCB list if not PL/I; or a pointer to a list containing the following if PL/I:
    - address of PCB list
    - address of location containing size of dynamic storage
    - address of start of dynamic storage
  - R13 Save area
  - R14 Return address
  - R15 Entry address

- Message Writer (DLZMMSG)
  - R14 Return Address

- ABEND Routine (DLZMABND)
  - No special register values

Macro Used

```
XECBTAB  TYPE= DEFINE
XECBTAB  TYPE= DELETE
XECBTAB  TYPE= CHECK
XPOST
XWAIT
OPEN
CLOSE
GET
PUT
CANCEL
STXIT  PC
STXIT  AB
MVCOM
COMRG
LOAD
```

MPS Batch Termination - DLZMTERM

This is one of five routines that make up module DLZMPI00 to support the batch part of MPS.

The MPS batch termination routine is entered when the user program finishes. It tells the online partition to do termination activity, deletes its own XECB, and ends the job.

Size of Module

Approximately 100 bytes.

Control Blocks Addressed

XCB1 XECB DLZXCBn1 and the data following it

Register Contents

Registers have the same values at entry as when MPS batch initialization (DLZMINIT) completed.

Macros Used

XPOST
XWAIT
EOJ
XECBTAB TYPE=DELETE

MPS Batch Program Request Handler -DLZMPRH

This is one of five routines that make up module DLZMPI00 to support the batch part of MPS.

The MPS batch program request handler routine is entered on each call to DL/I made by the user program. The user call list is validated and set up for the online partition to use. Then the online partition is notified by an XPOST of XECB DLZXCBN2. When the call is complete, data is moved to the user's I/O area.

Size of Module

Approximately 646 bytes (excluding constants and other non-code areas).

Control Blocks Addressed

MPCPT	MPC Partition Table
TCA	Transaction Control Area
PST	Partition Specification Table
XCB1	XECB DLZXCB1

Register Contents

- At entry:
  - R0 If=1, PL/I; if=0, not PL/I and value is ignored
  - R1 If PL/I, points to list of pointers to parameters; if not PL/I, points to list of parameters
  - R13 Save area
  - R14 Return address
  - R15 Entry address
- Message Writer (DLZMMSG)
  - R14 Return address

Macros Used

STXIT PC
XPOST
XWAIT
XECBTAB TYPE=CHECK

MPS Batch ABEND - DLZMABND

This is one of five routines that make up module DLZMPI00 to support the batch part of MPS.

The MPS batch abend routine has three entries:

1. PC STXIT
2. AB STXIT
3. Other MPS batch routines that cause abnormal termination.

The first two each identify which way the abend routine was entered. They then send an error massage. Then the third entry joins them as the online partition is notified. All entries delete the batch XECB and cancel or dump.

When an abnormal termination situation has occurred, DLZMABND is entered by:

- DLZMINIT
- DLZMTERM
- DLZMPRH

Size of Module

Approximately 88 bytes.

Control Block Addressed

STXIT AB Save area
STXIT PC Save area

Register Contents

- At entry
  No special values except base registers initialized

- Message Writer (DLZMMSG)
  R14 Return address

Exits

JDUMP If dump requested
CANCEL If no dump requested

Entry Points

STXIT AB	If abnormal end entered by DOS/VS
STXIT PC	If program check determined by DOS/VS
XPOST Entry	Other abnormal end when BPC must be notified

Macros Used

XPOST
XECBTAB TYPE=DELETE
JDUMP
CANCEL

MPS Batch Message Writer - DLZMMSG

This is one of five routines that make up module DLZMPI00 to support the batch part of MPS.

The MPS batch message writer routine handles all messages issued by the MPS batch partition. At entry, a parameter list is set up. The first parameter is always a pointer to the message number. Other parameters, if any, are as needed for the message.

When a message is to be written to SYSLOG and SYSLST, the DLZMMSG routine is entered by:

- DLZMINIT
- DLZMTERM
- DLZMPRH
- DLZMABND

Size of Module

Approximately 130 bytes.

Control Blocks Addressed

DTFs for SYSLOG and SYSLST

Register Contents

- At entry:
  R14 Return address
  Base registers already initialized

- At entry to message table (DLZMMSGT):

R1	Points to parameter list
R4	Base register for DLZMMSGT
R5	Address of where message is to be placed
R7	Length of message set up before calling DLZMMSGT; after call, R7 has total message length

Exits

To calling routine via branch register 14

Macros Used

PUT

STOP MPS TRANSACTION - DLZMSTPO

This module is invoked when a user wants to stop MPS. The user inputs a specific transaction code (CSDD) defined to initiate the stop transaction processing. The module then notifies (XPOST) the particular XECB that causes the MPC to end the MPS environment.

After the XPOST, the MPC allows batch jobs already executing to complete, but will not allow any new ones to start.

This transaction should be started before CICS/VS non-immediate shutdown is initiated.

Size of Module

Approximately 250 bytes.

Macros Used

XECBTAB TYPE=CHECK

CALL ANALYZER - DLZDLA00

The call analyzer module is used for initiation of all data base calls. Under normal circumstances, it receives control from the DL/I online program request handler (DLZODP00) in the CICS-DL/I region or from the batch application program request handler (DLZBPR00). It receives control from application program control (DLZPCC00) at termination of a DL/I batch partition or online task termination (DLZODP01) in a CICS-DL/I region.

For internal DL/I calls to update an index data base, this module (DLZDLA00) receives control from the index maintenance module (DLZDXMT0).

The call types handled by the call analyzer module can be divided into two groups: (1) normal data base calls, and (2) special control calls, which are sometimes referred to as 'pseudo' calls. The special calls are GSCD, get SCD address; TERM, write all buffers altered by that user; and UNLD, write last records for simple HSAM, HSAM, simple HISAM, and HISAM load or write all HDAM and HIDAM data base buffers altered by that user and close all data sets in the system.

The primary responsibilities of the call analyzer are:

- Test the first parameter in the call list for a valid four-character function and encode this into a one-byte function code.
- Test the second parameter in the call list for a valid PCB address and store the PCB address in the PST.
- Store the third parameter in the call list in the PST. This is the user's I/O area address.
- Verify the format of all segment search arguments (SSAs) in the call list and fill in the corresponding level table entry for the SSA in the call.
- Do required checking based on call type and SSAs.
- Do sequence checking when loading a data base.
- Pass control to the proper action module to process the call.

If a data base call requires the VSAM control blocks or SAM DTF representing the files within a data base to be opened, the analyzer calls upon the DL/I open/close module (DLZDLOC0) to perform the data management open for all files which may be needed for that PCB. The DL/I open/close module is called when the UNLD call is received to close all DL/I data bases opened in the batch partition.

CONTROL BLOCKS - DLZDLA00

PST
PDIR
PSB
DDIR
DMB

PCB
JCB
Level table
SDB
FDB

SIZE OF MODULE - DLZDLA00

This module contains approximately 4000 bytes of code.

INTERFACES - DLZDLA00

Register Contents

R1 = PST address
R13 = Save area address
R14 = Return address
R15 = Entry point address

Receives control from DLZPCC00, DLZODP00, and DLZPRHB0.

Passes control to DLZDLR00, DLZDLD00, DLZDDLE0 (DL/I action modules):

These modules need not save the analyzer's registers. They can return to the analyzer's entry point plus an offset stored in the SCD.

Call to DLZDLOC0 - DL/I open/close:

PSTFNCTN has open function
PSTDBPCB has address of the PCB

Call to DLZDBH00 - buffer handler:

PSTFNCTN is PSTPGUSR (X'07')

DL/I OPEN/CLOSE MODULE - DLZDLOC0

The function of module DLZDLOC0 is to open and close the DL/I data bases in either the CICS online control region or the batch partition. DOS/VS open/close macros are used to open and close data sets. DLZDLOC0 opens/closes VSAM ACBs for all data base organizations besides HSAM and simple HSAM, where DTFs are used. For simplicity the term ACB is used in the following description where ACB or DTF would be correct. For a HISAM data base with all functions, except for PSTOCDCB, both the KSDS and ESDS are opened/closed.

The PSTFNCTN byte in the PST determines the type of operation to be performed by DLZDLOC0.

- PSTOCDCB (X'10') - Only one ACB is opened/closed. It is located by DSG address (PSTDSGA).

- PSTOCPCB (X'02') - For PROCOPT = L or LS one data base is opened.

  For PROCOPT ≠ L or LS:

  All SDBs of that PCB are scanned and all referenced data bases are opened, that is, index data bases and logically related data bases are opened/closed with this call.

- PSTOCDSG (X'40') - One or two (HISAM) data bases are opened/closed.

  The ACB is located by DSG address (PSTDSGA).

- PSTOCALI (X'04')

  - For open:

    All ACBs specified for initial opening are opened (CICS online control region only)

  - For close:

    All ACBs in the system are closed.

- PSTOCDMB (X'01') - The ACBs of one DMB are opened/closed. The DMB directory address is passed in register 2.

DLZDLOCO compares the following values specified in DBD generation with the VSAM catalog entries for a data base:

- Control interval size
- Key length (KSDS)
- Relative key position (KSDS)
- Highest RBA used in the data base based on the PROCOPT. For example, PROCOPT=L requires an empty data base (high RBA=0), while a data base must contain data if PROCOPT≠L (high RBA>0).

For HISAM, HIDAM, and HDAM data bases, the first control interval of the VSAM ESDS is reserved for the DL/I control record (see Chapter 12). DLZDLOCO maintains this record.

- If PROCOPT=L or LS, space is acquired for one control interval and the DL/I control record is constructed. The buffer handler (DLZDBH00) is called to write the DL/I control record.

An open record, code X'2F', is written to the log tape whenever a data base is opened. If the open call is successful, bit zero (JCBOPEN) of the JCBORGN byte equals one (PCB call); and bit zero (PSTOCBAD) of the PSTFNCTN byte equals zero.

All PSDBs of a DMB are scanned for variable length segments with the edit/compression routine. All edit/compression routines that have 'INIT' specified are called after "open" and before "close".

SIZE OF MODULE - DLZDLOCO

The DL/I open/close module contains approximately 3000 bytes of code.

INTERFACES - DLZDLOCO

Register Contents

R1 - PST address
R2 - DDIR address if it is a close DMB call
R13 - Save area address
R14 - Return address
R15 - Entry point address

- DL/I control record - DLZRECO

- PSTFNCTN field of the PST:

Bit	Value	Meaning
1	1	Process DSG
2	1	Open for load
3	1	Process specific ACB
4	0	Close call
	1	Open call
5	1	Open/close all DMBs
6	1	Open/close a PCB
7	1	Open/close a DMB

DL/I DELETE-REPLACE CALL FUNCTION - DLZDLD00

This module performs the logical actions involved in replacing or deleting segments in a DL/I data base for all organizations, except HSAM (which has no delete or replace).

The replace function checks to ensure that the key field of the segment was not inadvertently altered and that the replace rules were not violated. If the segment to be replaced is indexed, this module interfaces with the DL/I index maintenance module (DLZDXMT0).

The first check made upon entry is a key check of the contents of the PCB key feedback area to the key of the segment in the user's I/O area. If there are any changes, a 'DA' status code results. Next the segment is retrieved and the sequence fields are checked for any changes. If any changes occurred, a 'DA' status code again results. Then the remainder of the data is checked for changes. If there were no changes, a blank status code is returned. If there were changes, the data is replaced.

If the segment is a logical child, a check is made to insure that replace rules were not violated. A check for replace rules is done prior to the destination parent's retrieval. If the logical child has a physical replace rule, then data must not be changed if retrieved from the logical path, otherwise an 'RX' status code will result. If the replace rule is logical, no status code will be given and no data will be changed for the logical child if it is retrieved from the logical path. If the replace rule is virtual, then a replace will be done if the destination parent does not violate any rules. At this stage, a decision will be made whether or not the data is to be replaced. If the data is to be replaced, a check will be made of the other portion of the concatenated segment before actually accomplishing the replace. The destination parent is then retrieved and checks are made for keys, data, and rules on the destination parent. If all checks are met, the destination parent is replaced prior to replacing the logical child.

If the segment to be replaced is in an HDAM or HIDAM data base and the segment is variable length, the segment and its prefix may be separated. The separation of data is determined by the min-byte value of DBDGEN and the current size of the segment. Also in this regard, if the segment was previously separated from its prefix prior to a replace call, the replace will attempt to rejoin data and prefix.

The delete function for a HISAM data base includes a check of the key field of the segment to be sure this is the segment to be deleted. If the organization is simple HISAM, the buffer handler is called to issue a VSAM ERASE. Otherwise, the segment is deleted by setting the HISAM segment delete bit. In addition, if this is the root segment, the record delete bit is also set.

The delete function for HDAM or HIDAM data bases includes a check of the key field of the segment to ensure that the segment is truly the segment to be deleted. A check is also made to ensure that delete rules stated for the DMB will not be violated. If logically related segments with a physical delete rule exist in the data base within the physical hierarchy starting with the segment to be deleted, a scan is made of all the segments to ensure that they include no segment which has not been logically deleted.

A scan of the data base from the point of deletion is performed. During this scan, each segment is accessed twice: once on the way 'down', and again on the way 'up'. While scanning 'down', any segment in a logical relationship is inspected to determine its eligibility for deletion and to terminate as many logical relationships as possible. In some cases (for example, the last logical child for a logical parent which has already been deleted through its physical path), the deletion of all, or a portion of, the logically related data base record is required. In this case, the delete action is expanded to perform the total delete function (except for the checking) for the new data base record. Then the scan of the original data base record is continued at the point of exit.

When scanning 'up', an interface with index maintenance (DLZDXMT0) is made if the segment is indexed. Physical pointers are adjusted to bypass any removable segments (HDAM or HIDAM segments which are no longer required) whose space is released by interfacing with the space management module, DLZDHDS0. For nonremovable segments (segments required to remain because of existing logical relationships), a logical delete bit is set to indicate the status of the segment.

A work area is obtained from the DL/I buffer pool to maintain the concatenated key and position of segments in the data base record(s) being scanned during delete or for calls to index maintenance during replace.

INTERFACES - DLZDLD00

Register Contents at Entry

- R1 Contains the address of the PST
- R13 Points to the current save area
- R14 Contains the DL/I analyze call function module (DFSDLA00) return point
- R15 Contains the module entry point

Register Contents at Exit

- R1 Contains the PST address
- R13 Points to the current save area
- R14 Contains the DL/I analyze call function module (DFSDLA00) return point
- R15 Contains a return code (0)

Segment delete codes utilized in the second byte of the prefix of each DL/I segment:

1...	This segment has been deleted (HISAM only).
.1..	This data base record has been deleted (HISAM only).
..1.	This segment has been processed by delete.
...1	This variable-length segment has its data separated from the prefix.
.... x...	Reserved
.... .1..	This segment is no longer required by its physical parent.
.... ..1.	This segment is no longer required by its logical parent.
....1	This segment has been removed from its logical twin chain.
1111 1111	This segment contains the separated data of a variable-length segment.

DELETE/REPLACE WORK SPACE ACQUISITION AND THE WORK SPACE PREFIX

DLZDLD00 acquires space to build work area(s) from DLZDBH00 (buffer handler) via a PSTGBSPC call. The çalculated minimum size required is indicated in PSTBYTNM. If the space is available, the buffer handler returns the address of the selected buffer in PSTDATA and its size in PSTWFK1.

The first section of the work space contains a prefix whose format and contents are described in Chapter 12. Immediately following is the work area containing information concerning the segment to be deleted (or the index source segment to be replaced), its physical data base (HIDAM or HDAM), and other segments in that data base record.

If a second work area is needed because of logically related segments and the space remaining in the current work space is large enough, the next work area will be allocated in the same work space (buffer) immediately following the previous work area. Forward and backward chains are maintained. If the remaining space is not large enough, another buffer is obtained from the buffer handler and chained to and from the previous work space.

Except in the case of an error condition, work areas are freed in the reverse order in which they were allocated. When the work area freed was the first one in the work space, the buffer is freed via a PSTFBSPC call to the buffer handler.

INTERFACES - DLZDLD00

This module interfaces with the following modules:

DLZDBH00
DLZDHDS0
DLZRDBL0
DLZDXMT0

Register Contents on ABEND - in the SCD ABEND Save Area

```
R1      -   PST address
R2      -   SCD address
R3      -   SDB address
R4      -   DMB address
R5      -   PSDB address
R6/R10      Work registers
R11 -       Base - (subroutine CSECT)
R12 -       Base (main CSECT)
R13 -       Current save area
R14/R15 - Work registers
```

CONTROL BLOCKS - DLZDLD00

- Delete workspace prefix
- Delete work area.

DL/I LOAD/INSERT MODULE - DLZDDLE0

The function of DLZDDLE0 is to load HDAM, HIDAM, Simple HISAM, HISAM, Simple HSAM, and HSAM data bases (in batch only) and insert segments into HDAM, HIDAM, Simple HISAM, and HISAM data bases.

DLZDDLE0 is entered from the DL/I call analyzer (DLZDLA00) on load requests for HIDAM, Simple HISAM, HISAM, HSAM, and Simple HSAM segments, HDAM dependent segments, and insert requests for Simple HISAM and HISAM roots. It is also entered from the retrieve module (DLZDLR00) on load requests for HDAM root segments, and insert requests for HDAM, HIDAM, and HISAM dependent segments.

The module performs the following functions:

A. HDAM/HIDAM load/insert -

1. Normal segment:

- Positioning: retrieve positions for inserting and loading of HDAM roots. For all other loading, DLZDDLE0 simulates retrieve positioning.
- Space for new segment is acquired using the space management module, DLZDHDS0.
- The segment is moved from the user's I/O area to the buffer.
- Prefix pointers are updated.
- Actual write is performed by the buffer handler using VSAM.
- Prefix pointers of twins and parents are updated.
- The data base logger (DLZRDBL0) is called to write the new segment and the updated prefixes.
- If the segment is an index source segment, index maintenance (DLZDXMT0) is called.
- Exit is to the call analyzer.

2. Concatenated segment:

- If the destination parent already exists, and the insert rule is physical or logical: same as normal segment.

- If the destination parent exists and the insert rule is virtual: the logical child segment is inserted as for a normal segment, data of destination parent are replaced afterwards.

- If the destination parent does not exist and the rule is not physical, the destination parent is inserted as for a normal segment; afterwards the logical child is inserted as a normal segment.

B. HISAM and simple HISAM load

- Main storage for a logical record for key sequenced data set (KSDS) and for entry sequenced data set (ESDS) is acquired from the buffer handler.

- The root and all dependent segments that fit into one logical record are written to the KSDS, using the buffer handler. The remaining dependent segments are moved to one or more records of the ESDS.

- Pointers to those records are inserted.

C. HISAM and simple HISAM root insert

- A key equal to or greater than the request is made to the buffer handler. If the key exists and the delete bit is flagged (HISAM), the space is reused; otherwise a II status code is returned. If the key does not exist, main storage is acquired from the buffer handler and the new record is built and then inserted by VSAM through the buffer handler.

- Old (if deleted) and new records are logged.

D. HISAM dependent segment insert

- If the segment fits into the record for which retrieve (DLZDLR00) has positioned, it is inserted by shifting the segments beyond the insert point to the right. If the segment does not fit into the record, a new ESDS record is built. The segment and shifted data are inserted into the new record. If the shifted data does not fit into the record, a second new ESDS record is created.

- Pointers to the new records are created.

- Old and new records are logged.

E. HSAM and simple HSAM load

- The I/O areas allocated by batch initialization are used to move the segments from the user area. PUT locate is executed, whenever one I/O area is filled.

BLOCKS AND TABLES - DLZDDLE0

PST
DDIR
DMB
PCB
JCB

Level table
SDB
FDB
SCD

SIZE OF MODULE - DLZDDLE0

This module contains approximately 8500 bytes of code.

INTERFACES - DLZDDLE0

Registers on Entry and to All Called Modules

R1 = PST

This module calls the following modules:

DLZRDBL0	- Data base logger
DLZDBH00	- Buffer handler
DLZDHDS0	- Space management
DLZDXMT0	- Index maintenance

STATUS CODES - DLZDDLE0

II
AO
IX
LB

INDEX MAINTENANCE - DLZDXMT0

Module name = CSECT name = DLZDXMT0

Function:

The function of this module is to load - insert - delete the index pointer segment of a HIDAM data base and to load - insert - delete - replace the index pointer segment for secondary indexes of a HDAM or HIDAM data base.

Abbreviations used throughout the module are:

ISS	Index source segment
XDS	Index target segment (indexed segment)
XNS	Index pointer segment (indexing segment)

The following major functions are performed:

ALL CALLS

- Save PST information in XMAINT work area

LOAD
INSERT

- Build index pointer segment in work area

  For primary indexes - take key from user I/O area. For secondary indexes - construct segment from SRCH, SUBSEQ and DDATA fields. For

/CK fields use PCB-key feedback area or read parents of ISS using SDBPOSC or PP pointers. Call user suppression routine, if needed.

- Build temporary blocks SDB, JCB, DSG

INSERT

- Build call list and SSA
- Call analyzer
- Take next index relationship of this ISS

LOAD

- Open data base, if necessary, or work data set
- Call buffer handler to write index record or write work data set for secondary index
- Take next index relationship of this ISS

UNLD

- Write FF-key record to all index data bases belonging to this data base

DLET

- Call buffer handler to get old ISS
- Construct the old index pointer segment
- For /CK fields take CONCAT key from DLET work area
- Call user exit routine, to check for suppression
- Build temporary blocks
- Log POINTER_CHANGE and DEL.BYTE CHANGE
- Call buffer handler to change index
- Take next index entry

REPL

- First part = DLET
- Second part = ISRT

ALL CALLS

- Restore PST
- Return to calling module

Entries:

Receives control from DLZDDLE0 (load/insert) and DLZDLD00 (delete/replace)

Register Contents

R1 = PST address
R14 = Return address
R15 = Start address

PSTWRK1	LSDB of ISS for ISRT, ASTR, REPL calls	
	LSDB of ROOT for UNLD CALL	
	PSDB of ISS for DLFT call	
PSTFNCTN	'A0'	Delete
	'A1'	Replace
	'A2'	Insert
	'A3'	Unload
PSTBYTNM	RBA of index source segment	

Interface to called modules:

1. DLZDLA00 (analyzer)
   Called for insert, not load mode

   PSTIQPRM points to internal call list
   Segment name*X(keyvalue) is used as SSA

2. DLZBBH00 (buffer handler)

PSTFNCTN:	PSTMSPUT load HIDAM index
	PSTBYLCT get index target segment again
	PSTSTLEQ get index pointer segment
	PSTPUTKY index of HIDAM data base
	PSTBFALT update index of HIDAM data base
PSTBYTNM:	RBA of segment
or	Pointer to key to be inserted

3. DLZDIOC0 (open/close)

R2:	Address of DDIR
PSTFNCTN:	PSTOCOPN + PSTOCLD + PSTOCDMB
	PSTOCOPN + PSTOCDMB
	PSTOCCLS + PSTOCDMB

4. DLZRDBL0 (logger)

PSTWRK1:	DBLLGDLT	logical delete
	DBLNDXC + DBLCMC	XMAINT chain maintenance
PSTWRK2:	Old segment code and old delete byte	
	Old RBA pointer	
PSTOFFST:	Offset to new segment code	
	Offset to new RBA pointer	
PSTBYTNM:	RBA of record	

5. DLZDSEH0 (work data set module)

   Is called at entry point - 12 to open work file.
   Return is to BALR if open not successful,
   to BALR + 4 if open successful.

Exits:

Back to calling module.

CONTROL BLOCKS - DLZDXMT0

- Index work area - DLZXMTWA
- SSA for the XMAINT call to the analyzer.

DL/I RETRIEVE MODULE - DLZDLR00

The DL/I retrieve module is responsible for retrieval of all segments, independent of physical data base organization. When an application program requests the retrieval of a segment, this module (DLZDLR00) gains control from the DL/I call analyzer, DLZDLA00. The analyzer has validity-checked the parameters in the application program's retrieval request. The analyzer has also placed this parameter information for retrieval in the DL/I control blocks.

Based upon this information, the retrieve module calls the DL/I buffer handler module, DLZDBH00, which controls physical I/O operations, to read the block containing the desired segment. Once the desired block exists in the data base buffer pool, its presence is made known to the retrieve module.

It is the responsibility of the retrieve module to "deblock" segments within the block. Once the desired segment is located, the retrieve module places the location and length of the segment in the PST control block associated with the application making the retrieve request and returns to the DL/I call analyzer. Once a particular segment within a data base is retrieved for a particular application program, "position" is established within the data base for the application program. This "position" is subsequently used to move sequentially through the data base if the application program issues GN and GNP calls.

If the block containing the segment to be retrieved already exists in the data base buffer pool, the request from the retrieve module to the buffer handler results only in the address of the desired data being returned to the retrieve module. No physical I/O is performed. In the case of HISAM, if a retrieve request involves inspection of several segments within a record, the retrieve module requests only the first of these from the buffer handler and finds the remaining segments itself, utilizing position information. Positioning information for each application program and each data base is maintained in the DL/I control blocks which are an extension of the PCB (that is, JCB, LEVVTAB, and LSDB).

In addition to servicing all data base retrieval requests, the retrieve module performs "positioning" functions for all segment insertion. In this case, the retrieve module receives control from the DL/I call analyzer module on an insert call. Prior to the insertion of a new segment occurrence, DL/I must insure that the segment does not already exist in the data base. It is the responsibility of the retrieve module to retrieve the block where the segment to be inserted may already exist. If the segment does not already exist in the data base, the block retrieved is normally used for segment insertion. Once the desired physical block is retrieved and positioning for segment insertion within the block is established, control is passed to the DL/I load/insert module, DLZDDLE0. If the data base organization is Simple HSAM or HSAM, the retrieve module performs the I/O (Get/Put) rather than calling the buffer handler.

HIDAM root retrieval by key (qualified GU, GN), results in two buffer handling requests. The first retrieves the index segment as any HISAM root. The second uses the RBA of the HIDAM root in the index segment to get the corresponding root segment. The position of the index segment is saved in a special SDB.

Retrieval of segments addressed by secondary indexes is performed in the same manner, as far as possible, as the retrieval of a HIDAM primary root segment. (The SDBs are generated so that the index looks like a primary index and the index target segment like a HIDAM primary root.) The most important differences are:

- The layout of the index pointer segment is user dependent and is different from that of a primary index.
- The sequence field of a secondary index is not necessarily part of the target segment and may be in a dependent segment.

Variable length segments are handled by the routine VLRT which provides an exit to a user routine to handle any necessary data expansion after calling the normal buffer handler interface (SETL).

Retrieval of logically related segments requires special handling. The retrieved segment (the concatenated segment) consists of the logical child (that is the concatenated key and the intersection data) and the physical or logical parent (destination parent). Since the SDBs always reflect the user's view of the data base, the same program logic is used whether the segment to be concatenated to the logical child is a physical or a logical parent. The concatenated key of the destination parent is constructed using the physical or the logical parent pointer of the logical child and the physical parent pointer of the destination parent. For ISRT calls the concatenated key in front of the input data is used to position on the destination parent. All positions on the physical path to the destination parent and on the twin chain of the destination parent are maintained.

COMMAND CODES AFFECTING RETRIEVAL

D - The segment data is moved when the level table is updated and not at return to the analyzer.

L - The segment skip routine is employed to skip to the last occurrence.

T - The RBA specified in the SSA is moved to the next position pointer location in the appropriate SDB and an unqualified GN is performed.

F - For a GN (GNP) call, the same logic is employed to retrieve the first occurrence as for a GU call.

MODULE LAYOUT - DLZDLR00

This module consists of 60 subroutines, a main entry routine (DLZDLR0), a main exit routine (DLZDLR1), and a general linkage and maintenance support routine (DLZKLNKD), each of which is preceded by a description in the form input - processing - output. The subroutines are linked using macro DLZRLNK and the following macros (refer to the comments in the DLZRLNK source program listing):

DLZRHDR - First macro of a subroutine; generates DSECTs, EQU, and module identification.

DLZRTLR - Last macro of a subroutine; generates a LTORG statement.

DLZRCLL - Generates code to transfer control to a subroutine using DLZRLNK.

DLZREXT - Generates code to return control to a calling subroutine using DLZRLNK.

The module is supplied as eight files. The first seven, DLZDLRA0 to DLZDLRG0, contain the subroutines and the eighth, DLZDLNKD, contains the linkage and maintenance support routine that is generated using the macro DLZRINK. The second file, DLZDLRA0, also contains the routines DLZDLR0 and DLZDLR1. The distribution of the subroutines within the CSECTs contained in the files DLZDLRA0 to DLZDLRG0 is arbitrary and can be changed at will, necessitating only that the affected CSECTs be reassembled.

MAINTENANCE SUPPORT - DLZDLR00

The module DLZRLNKD contains facilities to dynamically dump control blocks and I/O buffer sections. The extent and frequency of the dumping is controlled by DLZRLNK macro parameters or control fields in the PST as described in the DLZRLNK source program listing.

SIZE OF MODULE - DLZDLR00

The load module contains approximately 25000 bytes of code.

INTERFACES - DLZDLR00

This module interfaces with the following modules:

DLZDDLE0 - Load/insert
DLZDBH00 - Buffer handler

Register Contents on Entry and Return

R0 = SCD
R1 = PST
R2 = PCB

Register Contents During Execution

R0 = Work
R1 = Work
R2 = PCB
R3 = JCB
R4 = LEVTAB
R5 = SDB
R6 = Segment address
R7 = PST
R8 = DSG part of JCB
R9 = Byte or record location of SEGM in data base
R11= Base register for linkage routine DLZRLNKD
R12= Base register
R13= Save area
R14= Work
R15= Work

SPACE MANAGEMENT - DLZDHDS0

Module DLZDHDS0 allocates and maintains free space on direct access storage devices for storage of DL/I segments in the hierarchical direct organizations (HDAM and HIDAM). This space is managed through the use of free space elements (FSEs) in each block of each data set of a data base and a bit map. The bit map describes blocks that have at least one FSE which can contain the largest segment in the data set. There is one bit map per data set consisting of one or more blocks distributed equidistant over the data set.

Module DLZDHDS0 consists of nine CSECTs which perform the following functions:

DLZDHD00 contains the entry point for the combined module. It saves registers, initializes the work words in the PST, and branches to the appropriate module.

DLZGGSP0 consists of a 'driver' for all subfunctions that may be invoked to find space. It uses one byte of the work space to control invocation. This CSECT also controls formatting for HDAM when the root anchor point is beyond the current end of the data set and formatting of new bit map blocks, if necessary.

DLZFRSP0 returns to free space the space occupied by a segment being deleted. It logs the deletion of the segment and updates the bit map if required.

DLZRCHB0 searches the block passed to it for an FSE that satisfies the current request. If none is found, control returns to the calling module. If the request can be satisfied, the return is directly to the invoker of DLZDHDS0.

DLZRRHP0 searches the DL/I buffer pool for a block in the range passed to it. If one is found, module DLZRCHB0 is called to search it. If the block is rejected, the search continues to the end of the pool, and control is returned to DLZGGSP0. To avoid changing the position of buffers on the buffer pool use chain, online and batch are treated differently. In a batch environment, the buffer to be searched is passed to DLZRCHB0 and may be used without being requested from the buffer handler. In a DL/I online environment, the buffer is passed to DLZRCHB0. If the request can be satisfied from it, the buffer is then requested from DLZDBH00 and again passed to DLZRCHB0 for actual alteration.

DLZRRHM0 searches the bit map for a bit that is a one and is also in the specified range. If one is found, its corresponding block number is returned to DLZGTSP0. If all bits are zero, PSTNOSPC is returned to DLZGGSP0. The map search functions include creation and formatting of new bit map blocks, if necessary. To further proximity of space for related segments, whenever possible, the search within a given range is done from the center to the outer ends of that range in both directions at the same time.

DLZLMCL0 calculates search limits for DLZGGSP0. A switch is used to determine the appropriate limit - track, control area, delta control areas. The limits of the previous scan are used to break the range into two subranges. This prevents the re-requesting of blocks that were rejected during earlier scans.

DLZMPLC0 determines the block number for the bit map block appropriate to the block number passed to it. It also determines the relative bit position in the bit map block of the block number passed to it.

DLZMMUD0 turns the appropriate bit ON or OFF according to the entry point involved. The log is also called to reflect the change.

SIZE OF MODULE - DLZDHDS0

This module contains approximately 4800 bytes of code.

INTERFACES - DLZDHDS0

The following modules are called by DLZDHDS0:

DLZDBH00 - Buffer handler
DLZRDBL0 - Data base logger

Calling Sequence

R1	PST address			
	PSTDSGA	DSG address for appropriate file (all calls)		
	PSTFNCTN			
		PSTGTSPC	01	Get space
		PSTFRSPC	02	Free space
		PSTBTMPF	03	Turn off bit in bit map
		PSTGTRAP	04	Get space close to root anchor point
	PSTRBN	RBN of segment to get space close to - PSTGTSPC RBN of segment to be deleted - PSTFRSPC BBBR - PSTGTRAP where BBB = relative block number, R = root anchor point number		
	PSTBLKNM	Block number whose bit is to be turned off - PSTBTMPF		
R5	DMBPSDB	Address of PSDB of subject segment		
R14	Return point			
R15	Entry point - DLZDHDS0			

On Return

PSTRTCDE	- PSTCALOK	Space obtained; RBN is in PSTRBN - PSTGTSPC, PSTGTRAP Space freed - PSTFRSPC
	- PSTBTMPF	Space obtained. After insert, call DLZDHDS0 to adjust bit map.
R15	- 0	For above return codes.
	- 4	Error has occurred; check PSTRTCDE
PSTRTCDE	- PSTGTDS	The RBN to get close to does not exist
	- PSTNOSPC	DLZDHDS0 could not find space in data set - PSTGTSPC, PSTGTRAP
	- PSTIOERR	See DLZDBH00
	PSTNPLSP	See DLZDBH00

BUFFER HANDLER - DLZDBH00

The primary functions of module DLZDBH00 are:

1. To satisfy requests for buffer space for the processing of the data blocks of HD data bases. For Simple HISAM and HISAM data bases and for the index of HIDAM data bases, the VSAM buffer management is used.

2. To issue I/O requests to VSAM whenever data must be read or written. Thus, the buffer handler provides an interface between the DL/I action modules and VSAM data sets.

3. Whenever possible, to satisfy requests for data base segments and or records from data currently available in its buffer pool without issuing an I/O request. For this purpose, data is retained in the pool as long as possible. Various features such as use chains and alteration flags are employed so that a centralized buffer management is facilitated for concurrent use by all application programs.

The buffer handler satisfies the following requests as indicated by PSTFNCTN:

1. For processing HDAM, HIDAM, or HISAM ESDS:

Symbol Function	Hex Function	Description
PSTBYLCT	02	If the request is issued for an HDAM or HIDAM data base, the buffer handler retrieves the control interval whose relative byte number is stored in PSTYBTNM. The relative byte number in PSTBYTNM is first converted to a VSAM control interval number and an offset within the control interval. If this control interval is not in the buffer pool, buffer space is obtained in the buffer pool, the buffer which will be used is written, and the control interval is read into this buffer by a VSAM get call. If the requested control interval is already in the buffer pool, no read is done and the address of the buffer containing this control interval is passed back to the caller. If the request is issued for a HISAM ESDS data base, the buffer handler only issues the proper VSAM call for retrieving the record identified by the RBA which has been passed to the buffer handler in PSTBYTNM.
PSTBKLCT	01	The same as PSTBYLCT for an HDAM or HIDAM data base except that a VSAM control interval number is passed to the buffer handler in PSTBLKNM.

PSTBYALT	06	A locate relative byte number (refer to PSTBYLCT) is done first and then the buffer which contains the control interval is marked as altered by this specific user.
PSTBFALT	05	If the request has been issued for an HDAM or HIDAM data base, the buffer whose prefix address is stored in PSTBUFFA is marked altered. If, however, the request applies to a HISAM ESDS, the proper VSAM call is issued to write the record immediately.
PSTGBSPC	03	A buffer with the length specified in PSTBYTNM (possibly rounded to the next multiple of 512 bytes) is provided to the caller.
PSTFBSPC	04	A buffer identified by a DMB number, ACB number, and control interval number in PSTDMBNM, PSTACBNM, and PSTBLKNM is freed, that is, it is marked empty and put on the bottom of the use chain.
PSTPGUSR	07	All the buffers which have been modified by a specific user are written. All nonreusable buffers held by this user are marked empty and put to the bottom of the use chain. The bit representing this user is turned off in the user mask of all permanent write error blocks.
PSTBFMPT	04	All buffers of one data base or certain buffers of a data base are marked empty and put on the bottom of the use chain.
PSTWRITE	08	A logical record is added to a HISAM ESDS.

2. For processing HIDAM index, Simple HISAM or HISAM KSDS:

(a) Accessed by VSAM RBA

Symbol Function	Hex Function	Description
PSTBYLCT	02	Retrieve the VSAM KSDS record by the RBA which is in PSTBYTNM.
PSTBFALT	05	Write the VSAM KSDS record by the RBA which is in PSTBYTNM.
PSTERASE	0A	Delete the VSAM KSDS record identified by the RBA which is in PSTBYTNM.

(b) Accessed by key

Symbol Function	Hex Function	Description
PSTSTLEQ	09	Retrieve the VSAM KSDS record whose key is equal to or greater than the key whose address is stored in PSTBYTNM.
PSTGETNX	0B	Retrieve the next sequential VSAM KSDS record.

PSTSTLBG	0C	Retrieve the first VSAM KSDS record in a data base.
PSTPUTKY	0D	Insert a record by key directly into a VSAM KSDS.
PSTMSPUT	0E	Insert a record which is in ascending key order into a VSAM KSDS.

The buffers which are used for satisfying these requests are provided by VSAM buffer management. The buffer handler provides VSAM control blocks (ACB, EXLST, and RPL) to VSAM data management when issuing the required VSAM action macro.

The module DLZDBH00 consists of three CSECTs:

DLZDBH00 Contains the code for the functions

- PSTBYLCT
- PSTBKLCT
- PSTBYALT
- PSTBFALT
- PSTGBSPC
- Maintenance of write chain and use chain

DLZDBH02 Contains the code for the functions

- PSTSTLEQ PSTMSPUT
- PSTGETNX PSTERASE
- PSTSTLBG PSTWRITE
- PSTPUTKY

Additionally, this CSECT conatins the code required for preparing and issuing of VSAM calls and for processing feedback information by VSAM.

DLZDBH03 Contains code for the functions

- PSTFBSPC
- PSTBFMPT
- PSTPGUSR

In addition, this CSECT contains the subroutines for providing an enqueue/dequeue function.

WRITE CHAIN

The new control intervals of a HIDAM or HDAM data base are chained together on a write chain in ascending order of their control interval numbers. If one of the buffers on the write chain has to be written, all buffers on the chain are written.

There is a write chain for every data base. It is maintained by storing the prefix numbers of the prefixes of the next higher and the next lower buffers in bytes 18 and 19 of the prefix. A bit switch in byte 7 of the prefix (X'80') is on if a buffer is on a write chain.

USE CHAIN

All buffers are chained together in the order of their usage. This use chain is physically separated from the buffer prefixes and consists of one-byte elements containing relative numbers of prefixes. The order of the buffers on the use chain is indicated by the physical order of these use chain elements.

There is one use chain area per subpool. Each use chain area has a maximum of 32 entries. The maintenance of the use chain involves putting a use chain element on the bottom or on the top of the use chain as follows. The contents of the use chain element which is to be moved are saved. Then all use chain elements located behind the element to be put on top, or located before the element to be put on the bottom, are moved to the address which is one byte lower than the load address (or one byte higher if an element is placed at the bottom). The saved element is then stored at the top or the bottom of the chain.

ENQ/DEQ SUBROUTINES

Since transactions in an online environment may be processed in multi-thread mode, the buffer handler may have to synchronize and/or delay requests for buffers and/or buffer space. This is accomplished in two subroutines which perform ENQ/DEQ type functions and an interlock check. The following fields are used by the ENQ/DEQ routine:

Function	Label	Control block
ENQ/DEQ existing control interval (CI) ID	BFFRPST	Buffer prefix
	PPSTEXCI	PST prefix
ENQ/DEQ pending CI ID	BFFRNPST	Buffer prefix
	PPSTPECI	PST prefix
	PPSTCHAI	PST prefix
ENQ/DEQ subpool	SUBNQFI	Subpool information table
	SUBNQLA	Subpool information table
	PPSTSUPO	PST prefix
ENQ/DEQ matrix	BFPLPSIL	Buffer pool prefix
	BFPLFSIF	Buffer pool prefix
	BFPLPSIL	Buffer pool prefix
	PPSTMATR	PST prefix

For interlock detection, the ENQ/DEQ routines use the contents of the following buffer pool prefix fields:

BFPLINMA	interlock detection matrix
BFPLINW1	work areas
BFPLINW2	

The ENQ/DEQ routines use the following fields in the buffer pool prefix as work space:

BFPLNQW1
BFPLNQW2

Normally, the resources to be enqueued are the existing contents of a buffer (existing CI ID) or planned contents of a buffer (pending CI ID). Under certain circumstances, other resources may be enqueued.

Enqueuing of a resource consists of the following steps.

If the resource is available:

1. Store the PST ID into a field of the resource reserved for this purpose (that is, BFFRPST, BFFRNPST, SUBNQF1, BFLPSIF).

2. Store the resource ID (for example, the buffer number) into a field in the PST reserved for this purpose (that is, PPSTEXCI, PPSTPECI, PPSTSUPO, PPSTMATR).

3. Indicate successful ENQ with a return code of 4 and return to caller.

If the resource is not available:

1. Find a position for the current PST in the interlock detection matrix.

2. Indicate by an appropriate entry that this PST is waiting and for which task.

3. Check whether this waiting would cause an interlock.

4. If no interlock possible:

   a. Chain with appropriate chain fields the current PST behind the last PST already waiting for this resource.

   b. Return with a return code of 8 to indicate that a wait condition exists.

5. If an interlock would occur if the current PST were to attempt to wait on this resource:

   a. Remove the entry made in 2 above from the interlock detection matrix.

   b. Indicate with a return code of 12 that an interlock would occur and return.

Dequeuing of a resource consists of the following steps.

1. Remove the resource ID from the appropriate field in the current PST.

2. Remove the PST ID from the appropriate field in the resource.

3. If the PST chain fields indicate that no other PST was waiting on this resource, return to caller.

4. If another PST was waiting on this resource:

   a. Move the waiting PST ID into the resource and remove the corresponding wait indication from the interlock detection matrix.

   b. Post the waiting PSTs and unchain the current PST.

   c. If, because of 4.a, certain rows and columns in the interlock detection matrix are free now, make these available for use by other PSTs and post those (see description of action taken on pseudo-interlock conditions).

   d. Return to caller.

For performance reasons, resources contain, in addition to the owning PST's ID, the ID of the last PST in the wait chain for this resource. These IDs are also maintained by the ENQ/DEQ routines.

The interlock detection matrix consists of a pair of eight-bit matrices. The first bit matrix indicates for up to eight PSTs which PST is waiting on which other PST. Rows and columns are dynamically allocated to PSTs as required. A one-bit in the appropriate row and column indicates a wait condition. The second bit matrix is the transpose of the first. An imminent interlock is detected by some simple logical operations executed against those two matrices. In the event that eight PSTs are occupying this matrix when further PSTs request service involving a wait condition, a code of 16, indicating pseudo-interlock, is returned and no enqueuing takes place.

The following types of ENQ requests may occur:

ENQ existing CI ID	When a task either wants to write a buffer or wants to get posted when reading into or writing a buffer is finished.
ENQ pending CI ID	When a task wants to reuse a buffer in the buffer pool or when a task wants to get posted when the creation of a pending (i.e., new) CI is finished.
ENQ subpool	When there is currently no buffer prefix in a subpool allowing a pending CI ID.
ENQ matrix	When a task wants to ENQ on a resource currently held by another task and no free row/column in the interlock detection matrix is available.

The following action is taken by the main routine of the buffer handler on a return code (RC) indicating nonsuccessful ENQ.

Condition	RC	Issue
Wait	8	Issue IWAIT macro.
Interlock	12	Dequeue all resources held by this PST and retry the current DL/I request.
Pseudo	16	Dequeue all resources held by this PST and enqueue on interlock detection matrix. This causes a wait condition. Issue IWAIT. Upon post, dequeue matrix and retry current DL/I request.

CONTROL BLOCKS - DLZDBH00

- PST
- PST prefix
- DDIR
- DMB
- DSG
- SCD
- Buffer pool prefix
- Buffer prefix

SIZE OF MODULE - DLZDBH00

This module contains approximately 7500 bytes of code.

INTERFACES - DLZDBH00

DLZDBH00 uses the PST for communication from and to the calling modules and for work space. The DSG is used to obtain the DMB number and ACB number of the data set which applies during a request. The address of the buffer pool prefix is obtained from the SCD. The address of the buffer prefix area is obtained from the buffer pool prefix. VSAM is invoked for all I/O.

In order to make sure that writing of log information is always ahead of updating a data base, the buffer handler may branch to a specific entry point of DLZRDBL0 or DLZRDBL1. (Refer to the description in the paragraph about DLZRDBL0 and DLZRDBL1.)

DLZDBH00 issues the RELPAG macro for buffers that are marked empty.

BUFFER HANDLER FUNCTIONS AND REQUIRED FIELDS

The following chart illustrates which fields must be supplied to the buffer handler (input) for each specific function and which fields are filled in by the buffer handler (output) on completion of the function.

1. Function used to access a HIDAM or HDAM data base

Function	Input		Output	
	Field	Contents	Field	Contents
PSTBYLCT	PSTBYTNM	Relative byte number of desired segment	PSTDATA	Core address of desired segment
			PSTOFFST	Offset of segment from beginning of control interval
PSTBKLCT	PSTBLKNM	RBA of desired segment	PSTDATA	Core address of desired segment
PSTBYALT		See PSTBYLCT		See PSTBYLCT
PSTBFALT	PSTBUFFA	Address of buffer prefix which is to be marked altered		
PSTGBSPC	PSTBYTNM	Number of desired bytes	PSTDATA	Address of provided buffer
PSTFBSPC/PSTBFMPT	PSTDMBNM PSTACBNM PSTBLKNM	DMB ACB Control interval RBA All or part of buffer identifier may be passed.		
PSTGUSR	PSTDMBNM PSTACBNM PSTBLKNM PPSTID	DMB ACB Control interval RBA User identifier Any or all of these may be passed.		

2. **Functions used to access a HISAM ESDS**

Function	Input		Output	
	Field	Contents	Field	Contents
PSTBYLCT	PSTBYTNM	RBA of the logical record to be read	PSTDATA	Address of the record within the buffer
PSTBFALT	PSTBYTNM	RBA of the logical record to be written		
PSTWRITE	PSTDATA	Address of work area containing the logical record	PSTBLKNM	RBA of the record added to the ESDS as calculated by VSAM
	PSTBUFFA	Prefix address		

3. **Functions used to access a KSDS by key (Simple HISAM, HISAM or HIDAM index)**

Function	Input		Output	
	Field	Contents	Field	Contents
PSTSTLEQ	PSTBYTNM	Address of the field which contains search argument	PSTBYTNM	RBA of the logical record retrieved
			PSTDATA	Core address of record
PSTSTLBG			PSTBYTNM	RBA of the logical record retrieved
			PSTDATA	Core address of record
PSTGETNX			PSTBYTNM	RBA of the logical record retrieved
			PSTDATA	Core address of record
PSTPUTKY	PSTDATA	Address of work area containing the logical record		
	PETBUFFA	Prefix address		
PSTMSPUT	PSTDATA	Address of work area containing the logical record		
	PSTBUFFA	Prefix address		

4. Functions used to access a KSDS by RBA (HISAM or HIDAM index)

Function	Input		Output	
	Field	Contents	Field	Contents
PSTBYLCT	PSTBYTNM	RBA of the logical record to be retrieved	PSTDATA	Address of the record within the buffer
PSTBFALT	PSTBYTNM	RBA of the logical record to be written		
	PSTDATA	Address of the record within the buffer		
PSTERASE	PSTBYTNM	RBA of the logical record to be erased		

Calling Sequence

R0 - SCD address
R1 - PST address
R14 - Return address to caller
R15 - Address of DLZDBH00

Fields Required (Independent of Function)

PSTFNCTN Hexadecimal code for desired function

PSTDSGA Address of associated DSG needed for: PSTBYLCT, PSTBKLCT, PSTBYALT

PSTBLKNM Identification of desired block needed for: PSTBKLCT, PSTBFALT, PSTFBSPC

PSTDMBNM Number of associated DMB needed for: PSTBKLCT, PSTBFALT, PSTFBSPC, PSTGBSPC

PSTACBNM Number of associated ACB needed for: PSTBKLCT, PSTBFALT, PSTFBSPC, PSTGBSPC

PSTBYTNM PSTBYLCT/PSTBYALT - relative byte address of desired segment - relative record number of HISAM ESDS (high-order byte = X'80')

PSTGBSPC - fullword size of requested space

PSTBUFFA Address of buffer prefix for block to be marked 'altered' - PSTBFALT

DSGDMBNO DMB number of the referenced data base

DSGDCBNO ACB number of the referenced data set

On Return

R15 0 Request satisfied
4 Warning or error condition

Fields Returned (Independent of Function)

PSTOFFST	Offset from PSTDATA back to first byte of block
PSTDMBNM	Address of associated DMB number
PSTACBNM	Address of associated ACB number
PSTDATA	Address of first byte of requested segment, record, or space
PSTBUFFA	Address of buffer prefix
PSTNUMRO	Number of reads done during this call
PSTNUMWT	Number of writes done during this call
PSTCLRWT	Bit 0 — This caller waited during request; 1-8 — Reserved
PSTRTCDE	

Return Code Function	Hex Function	Description
PSTCLOK	00	No error occurred during this request.
PSTGTDS	04	Record, CI, or segment requested is more than one CI beyond the end of the data set - returned on PSTBKLCT, PSTBYLCT, PSTBYALT
PSTIOERR	08	Requested CI, record, or segment could not be read successfully on a PSTBKLCT, PSTBYLCT, or PSTBYALT call or could not be written successfully on a PSTPUTKY, PSTMSPUT, PSTWRITE, or PSTBFALT call.
PSTNOSPC	0C	An out of space condition occurred on the data set DASD while processing this request.
PSTBDCAL	10	The byte at PSTFNCTN is not a valid function or the DMB/ACB/BLKID in the PST do not match corresponding fields pointed to in PSTBUFFA for a PSTBFALT call.
PSTNOTFD	14	A PSTSTLEQ call has been issued for a record whose key is higher than the highest key in the data set.
PSTNWBLK	18	The requested CI, record, or segment will go in the CI, one greater than the current end of the data set. Space has been allocated in the pool to hold the new CI. The address is at PSTDATA.
PSTNPLSP	1C	The pool does not contain enough space to satisfy the request.
PSTWROSI	20	A request (GBSPC) was issued for a buffer size which exceeds the highest buffer size handled by any subpool.

PSTENDDA	24	The end of data set has been reached on a PSTGETNX call.
PSTBYEND	28	A request has been issued with a key or RBA higher than the highest key or RBA in the data set.
PSTEOD	2C	End of data set has been reached on a request by DLZDLOC0.

DATA BASE LOGGER - DLZRDBL0 AND DLZRDBL1

The data base logger module logs the modifications made to a data base. These data base log records are written to the system log. This module is invoked by several of the DL/I modules associated with data base modifications.

The logging of data base modifications, additions, and deletions is done on a physical basis to facilitate a quick recovery procedure. Only calls that actually cause a change to be made to a data base are logged. Two sets of information are logged for each modification - a before set and an after set.

The before information is that required by the data base backout utility. It is used to back out a partially completed update series and to restore a data base to some prior point in time.

The after information is that required by the data base recovery routines to restore the data base from a previous backup copy.

There are five basic types of data base log records.

1. POINTER maintenance record
   When a segment is deleted or inserted and it causes a change in any of the pointers in other segments, each pointer is logged separately as a POINTER maintenance record. A POINTER maintenance record is indicated by bits 1, 2, and 3 of the DLOGFLG2 field of the log record being set to zero.

2. PHYSICAL INSERT record
   When a segment is physically added to the data base, a PHYSICAL INSERT record is written. This type of record is indicated by a one in bit 1 of the DLOGFLG2 field.

3. PHYSICAL DELETE record
   When a segment is physically removed from the data base, a PHYSICAL DELETE record is written. This type of record is indicated by a one in bit 2 of the DLOGFLG2 field.

4. PHYSICAL REPLACE record
   When a segment in a data base is modified, a PHYSICAL REPLACE record is written. This type of record is indicated by a one in bit 3 of the DLOGFLG2 field.

5. LOGICAL DELETE record
   When a DLET call is issued but the segment is not physically removed from the data base, a LOGICAL DELETE record is written. Only the segment code and delete bytes are logged. A logical delete record is indicated by bits 1 and 2 of the DLOGFLG2 field being set to a one.

Record types 1, 2, 3, and 5 contain the before and after information in the same record and have a log code of X'50'. Type 4 requires two records. The after record has a log code of X'50'; the before record has a log code of X'51'. Additionally, if a physical insert reuses space of a deleted record, log records X'50' and X'51' are written.

If the change is an insert or a delete, the before and after are part of the same record. On an insert, the new segment, including the prefix, is logged as the change data. On a delete, the old segment and prefix are the change data. In HD, both insert and delete cause changes to the free space elements (FSEs) within a block. The new FSEs and their offsets are logged following the change data and a count of the changes is placed in bits 4 through 7 of the DLOGFLG1 field.

The information needed to create the log record is retrieved from the various DL/I blocks. A small amount of additional information is passed as parameters from the DL/I action modules.

The data base log tape format is undefined records (UNDEF). The block size is 1024 bytes. Maximum record length is 512 bytes. If a segment cannot be logged into one record, it is internally spanned over two or more log records. The first record is logged with a data length adjusted to match the data it contains. The offset for the second record is incremented by the length of the first, and the second is written as a separate segment. The adjusting of data length and offset continues until the entire segment is written.

CONTROL BLOCKS - DLZRDBL0 AND DLZRDBL1

- Data base log record
- Application program termination record
- Application program scheduling record
- File open record.

SIZE OF MODULE - DLZRDBL0

This module contains approximately 3300 bytes of code.

INTERFACES - DLZRDBL0

<u>Register Contents</u>

R1 - PST address
R13 - Save area
R14 - Return address
R15 - Entry point address.

High-order byte of PSTWRK1 field in PST:

Bit	Value	Definition
0	1	Index maintenance call
1-3	000	Chain maintenance call
	001	Physical replace
	010	Physical delete
	100	Physical insert
	110	Logical delete

	111	Reserved
4	1	Last change for this user call
5	0	One FSE (physical delete or insert)
	1	Two FSEs
6	1	Old copy of physical replace
7	1	New block log call
4&6	1-1	No data - end of user call

PSTWRK1 - Physical SDB address (except new block call)
- Data length (low halfword) if new block call

PSTWRK2, PSTWRK3, PSTWRK4 - Old data on pointer maintenance and logical delete calls. FSE data on physical insert and delete calls.

Before a data base block is updated (that is, before the buffer handler issues the put for an updated block), the associated log information is first written to the log tape in the following manner.

After issuing a put to write a log block to the log tape, the log module updates the count of written log blocks in the field SCDLOCOU.

When the log module processes a log call, in which a data base buffer is involved, the current count of written log books is stored from SCDLOCOU into byte 7 of the buffer prefix in the case of HD, or into the field DMBACBLC in the ACB extension in the case of HISAM and HIDAM index.

Before issuing any put for updating a data base block, the buffer handler compares the value stored in the buffer prefix (HD) or in the ACB extension (HISAM, HIDAM INDEX) with the current value in SCDLOCOU. If the two values are unequal, the log information associated with the data base update has already been written out. If the two values, however, are equal, the buffer handler branches to entry point WRIAHEAD of DLZRDBL0 to force the current contents of the log I/O area to be written out immediately. If, however, asynchronous logging was requested by the user, the count comparison is bypassed, that is, no "write ahead" logging takes place.

LOGGING IN THE ONLINE SYSTEM

In the online system the put for the log blocks is issued in a separate, asynchronous subtask, which is attached at system initialization time. This subtask is a separate CSECT within the log module DLZRDBL0.

The purpose for this is to avoid losing tasks when the end of volume condition is encountered on the log tape.

The communication between the asynchronous log subtask, the logger, and the DL/I online nucleus (DLZODP00) is achieved by using three ECBs as follows:

1. System ECB (SCDESECB, in SCD extension), which is used for the communication between the log module (DLZRDBL0) and DLZODP00.

2. Log I/O ECB (SCDELECB, in the SCD extension), which is used for the communication between the log module and the asynchronous log subtask.

3. Private ECB (fullword in the log subtask CSECT), which is used for the communication between the asynchronous log subtask and the log module during the end of the I/O operation that was initiated by the log subtask.

Figure 4.1 shows the events which take place when a PUT for a log block becomes necessary in an online environment.

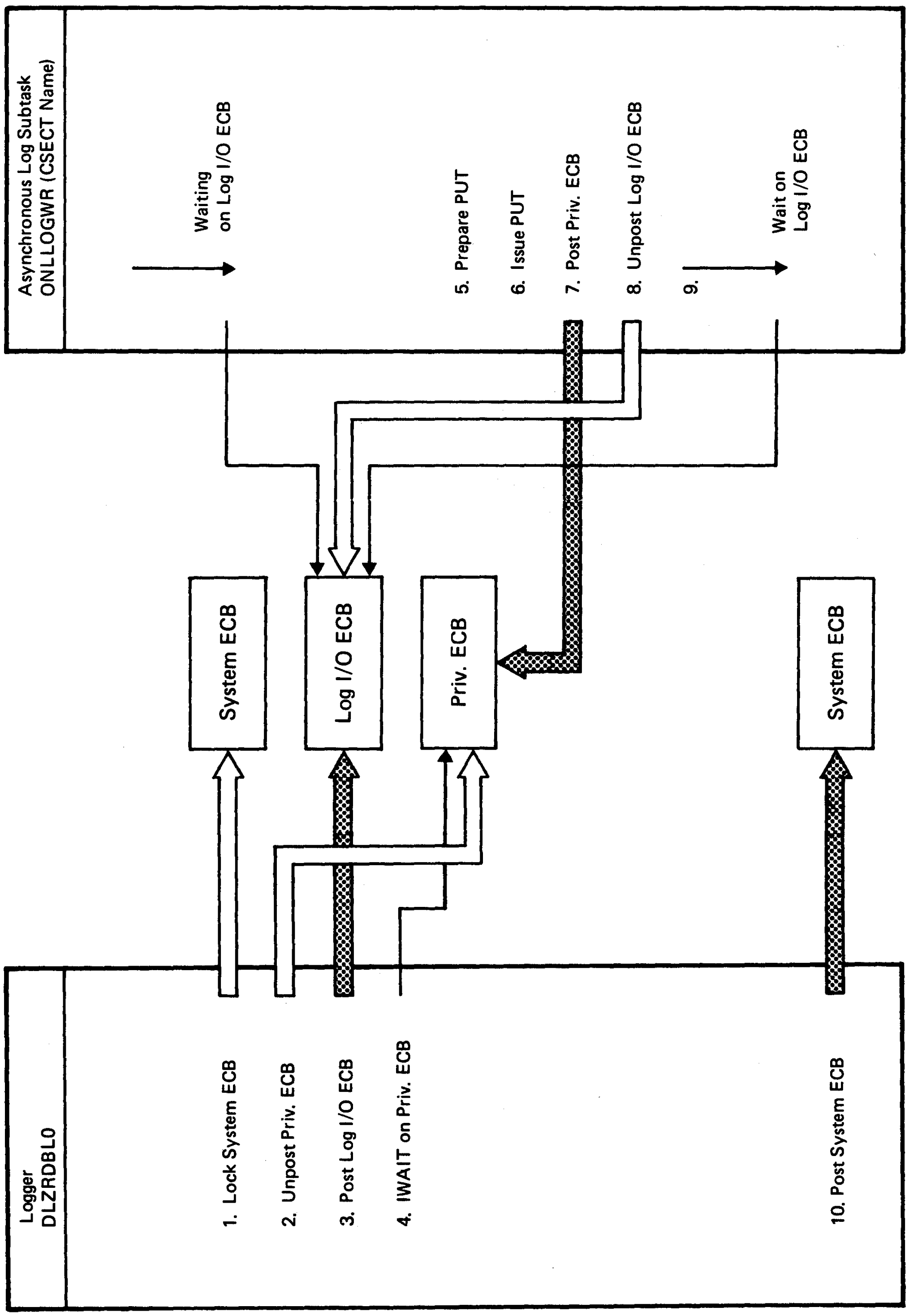


Figure 4.1. Online Log Block Put Operation

The relationship between all modules involved in the asynchronous log writing is as follows:

	DLZOBP00 PRH Schedul.Rout TERMIN. Rout MESSAGE Rout IWAIT Rout EXCPAD Rout	DLZOLI00	DLZRDBL0	CNLLOGWR
System ECB	Checks system ECB, if LOG subtask is active: 1 Before a call is processed (PRH branches to analyzer) 2 When a log request will be issued 3 Before branching back into a task after control was given up		When PUT has to be issued, unpost system ECB --- After log subtask is finished, post system ECB	
Log I/O ECB		Attach asynchronous log subtask	When PUT has to be issued, post log I/O ECB, get log subtask started	Waiting on log I/O ECB --- After put is finished, unpost log I/O ECB
Private ECB			When put has to be issued, lock private ECB (I/O is active) IWAIT on private ECB	After put, posts private ECB

LOGGING BY USING CICS JOURNALING (MODULE DLZRDBL1)

Besides writing log information in the standard way, logging in the online system can be done by using the journaling feature of CICS. That means the DL/I log information as described in the chapter about module DIZRDBL0 will go on the same tape as any CICS journal information.

This is possible because CICS uses different journal record IDs than DL/I (DL/I uses X'C7', X'08', X'2F', X'50', X'51'). Any DL/I utility which uses a journal tape will check the record ID and process only those records, which have record IDs used by DL/I.

The general structure of DL/I log records, CICS journal records and CICS journal blocks is as follows:

1. DL/I LOG RECORD

LL	ƀƀ	REC. ID	CONTINUED ACCORDING TO DSECT

0 2 4

2. CICS JOURNAL RECORD

SYSTEM PREFIX				USER PREFIX	JOURNALLED DATA
LL	ƀƀ	REC. ID	...		

0 2 4

3. LAYOUT OF A JOURNAL BLOCK

LL ƀƀ	CICS/VS LABEL RECORD	ANY COMBINATION
OF CICS/VS JOURNAL RECORDS AND		
DL/I LOG RECORDS		

If the user requests logging by CICS journaling (UPSI bits 6 and 7 = 0), DLZCLI00 loads module DLZRDBL1 instead of the standard log module DLZRDBL0. This module provides the following services:

- Build and write open records for each data base that has been opened. DFHJC TYPE=WRITE is issued to CICS.
- Build and write log records on request by the action modules. DFHJC TYPE=WRITE is issued.
- Write log records built by the sched/term. routine. DFHJC TYPE=WRITE is issued.
- Initiate a physical put to the journal tape on request of the buffer handler. DFHJC TYPE=WAIT is issued.

Before a journal call is issued to CICS, DLZRDBL1 checks if the task which is going to write a journal record already owns a JCA. If it does not, a GET JCA call is issued prior to issuing the DFHJC call.

Since DLZRDBL1 is not reentrant, no task can be allowed to enter this module while log I/O is being processed.

DLZRDBL1 unposts an ECB (SCDESECB) prior to any physical I/O. In various parts of DLZODP this ECB is checked, and, if it is locked, a CICS wait is issued before control is passed to any action module.

SIZE OF MODULE DLZRDBL1

This module contains approximately 2500 bytes.

When log information is written by using CICS journaling, the writing of log information is always ahead of updating the associated data base blocks. The scheme used is the same as with standard logging, the only difference being that the value for the number of written journal blocks (CICS ECN) is not manipulated by the log module but is taken out of the JCT.

CHAPTER 5. ONLINE DL/I PROCESSOR

Before attempting to use the information in this chapter you should be familiar with the Customer Information Control System/Virtual Storage (CICS/VS). References to the prerequisite publications are contained in the preface to this manual.

The online DL/I processor modules DLZOLI00 and DLZODP00 provide services in a CICS/VS-DL/I environment as follows:

a. DL/I system initialization
b. DL/I user task scheduling
c. Processing DL/I calls (online program request handler)
d. DL/I user task completion
e. DL/I normal system termination
f. DL/I abnormal system termination
g. DL/I online message writer
h. DL/I-VSAM-CICS synchronization via VSAM 'EXCP' Exit.

DL/I SYSTEM INITIALIZATION - DLZOLI00

In order to process DL/I applications in an on-line environment, a DL/I online nucleus must first be generated. The DL/I online nucleus generation procedure is described in *DL/I DOS/VS Utilities and Guide for the System Programmer*. The result of the procedure described in the publication is a DL/I online nucelus CSECT.

The generated nucleus, which is link-edited into a DOS/VS core image library, consists of a system contents directory (SCD), a table of partition specifications table prefixes (PPST), a PSB directory entry for each PSB specified, and an application control table (ACT).

The application control table (ACT) is used by DL/I online at CICS initialization to verify and load all PSBs and DMBs that can be referenced online. The ACT is used during scheduling to determine whether an online transaction is to use DL/I. It is also used by DL/I default scheduling to acquire a PSB to use with a DL/I application program if none was explicitly specified.

The ACT is produced from parameters specified in the following DLZACT macro instructions:

```
DLZACT TYPE=INITIAL
DLZACT TYPE=CONFIG
DLZACT TYPE=PROGRAM
DLZACT TYPE=BUFFER
DLZACT TYPE=FINAL
```

Each ACT program entry is generated from the DLZACT TYPE=PROGRAM statement. These statements define to DL/I which application programs can use DL/I online. They also define which PSB names can be used by each of the application programs. The ACT is located +4 from the beginning of the DL/I nucleus (DLZNUCxx). There is one ACT program for each DLZACT TYPE=PROGRAM statement used to generate the online nucleus. The application control table (ACT) has the following format.

Application Control Table

A.	B.	C.	
4	8	2	bytes

A. Buffer pool information address or 0
B. Storage layout control table name or 0
C. Number of HD DBDs in HDBFR operand

Generated from: DLZACT TYPE=PROGRAM

Program entry '1'

D.	E.	F.	G.	. . .	G.	
8	1	2	2		2	bytes

D. ACTNM ACT program entry name

E. ACTIND Entry indicator byte:
X'80' Program is a DL/1 program
X'40' Program name not in CICS PPT
X'20' ABEND option bit
X'02' Program is deferred-scheduled

F. ACTPCNT Count of PDIR (PSB) pointers for this program

G. ACTPPTR PDIR pointer(s). ACTPCNT indicates how many pointers are included here before the start of the next ACT entry.

Program entry 'n'

				. . .	

A maximum of 4095 DLZACT TYPE=PROGRAM statements and a maximum of 4095 unique entries (an entry consisting of program name and one PSBNAME) may occur in one ACT generation.

H.	
4	bytes

H. Delimiter (FF FF FF FF) indicating end of program entries

Generated from: DLZACT TYPE=BUFFER

(A)

HDBFR entry (subpool '1')

I.	J,	. . .	J.	K.	
2	8		8	2	bytes

I. Length of entry
J. DBD name
K. Number of buffers

HDBFR entry (subpool 'n')

		. . .		

HSBFR entry (DBD #1)

L.	M.	N.	O.	P.	
2	8	2	2	2	bytes

L. FF 00
M. DBD name
N. Number of index buffers
O. Number of KSDS buffers
P. Number of ESDS buffers

HSBFR entry (DBD #n)

Q.	
4	bytes

Q. Delimiter (FF FF FF FF)

DL/I initialization is performed during CICS/VS initialization just after loading the CICS/VS nucleus. The DL/I online nucleus module has been loaded by CICS/VS in the same manner as a CICS/VS nucleus module, and its address is placed in the CICS/VS CSA optional features list.

NUCLEUS AND TABLE INITIALIZATION

DL/I verifies the presence of the online nucleus by checking the CICS/VS optional features list DL/I entry for a non-zero value. Once verified, the program request handler entry point is moved to the DOS/VS COMREG using the MVCCM macro. Next, the application control table (ACT) is located and an indicator is set in each corresponding PPT entry for all application programs which will use DL/I. Each PSB name in the ACT is eight characters in length.

Next the PSB segment intent list is built. This is accomplished by loading each PSB defined in the ACT in ascending address space in the low end of the partition and moving the intent list, which is appended to the front of the PSB, to an entry in the PSB segment intent list table. The length of the PSB plus the length of the index work area, if required, are used to calculate how much storage to reserve. The segment intent list is overlaid during this process because its information is redundant. The PSB directory entry for each PSB is initialized with the address of the intent list, the PSB's storage address, and the amount of storage required.

The DMB directory is constructed. One DMB directory entry is created for each unique data base (DMB) defined in the PSB intent list entries. DMB names are eight characters in length and consist of the DBD generation name extended to seven characters by at-signs (@) if necessary. The eighth character is D. At this time, a validity check is performed to ensure that all required DMBs, defined by the PSB intent list, have been defined in the CICS/VS file control table (FCT). If any are missing, a message is written on the system console and the operator is given the option to continue or cancel. If initialization is to continue, PSBs which require the omitted DMB(s) are flagged to indicate this condition. Application programs which use these PSBs are not scheduled.

Initialization continues with the loading of all DMBs specified in the DMB directory. As each DMB is loaded, the corresponding entry in the DMB directory is initialized. A test is then made for HDAM and the defined randomizing routine is loaded. As the DMBs are loaded, they are initialized. After all DMBs have been loaded and initialized, the size of the buffer pool is determined. The size of the pool is calculated based on two variables. The first is a user-supplied parameter which defines the number of subpools. The second variable is the control interval size of each VSAM data set.

After the pool size is determined, the required address space is reserved. Then the buffer pool prefix in the online nucleus is initialized. Next the subpool prefixes are created and initialized. There are 32 prefixes for each subpool.

LOAD ACTION MODULES

Upon completion of the acquisition and initialization of the buffer pool and prefixes, the DL/I action modules are loaded. As the modules are loaded, their corresponding entry points are moved to the SCD. The modules are loaded in the following standard sequence if not otherwise specified by a storage layout control table:

DLZDBH00	-	Buffer handler
DLZDLR00	-	Retrieve
DLZDLA00	-	Call analyzer
DLZRDBL0	-	Data base logger
DLZDLD00	-	Delete/Replace
DLZDDLE0	-	Load/Insert
DLZDHDS0	-	Space management
DLZDXMT0	-	Index maintenance
DLZDLOC0	-	Open/Close

These modules are discussed in detail in Chapter 3 of this manual.

INITIALIZE PSBS

Upon completion of the loading of the action modules, initialization moves the specified PSBs using information stored in the PSB directory entries. After each PSB is moved, it is initialized and its corresponding PSB directory entry filled in.

ATTACH LOGGER

If data base logging has been specified by the user, the logger I/O module is initialized and attached. If the log module fails to attach, the data base log is closed and no logging takes place.

OPEN DATA BASES

The final step of initialization is the opening of the data bases. The DMB directory is scanned for DMB's that failed during initialization and the open initial attribute is reset for any found. Next the data bases are opened via an 'open all' call to the DL/I Open/Close module. All modules indicating open initial in the DDIR are opened by Open/Close at this time.

Upon completion of the open processing, the IWAIT routine address is restored and control is returned to CICS initialization.

DL/I USER TASK SCHEDULING - DLZODP00

DL/I user scheduling is initiated when a task receives control on a Transfer Control (XCTL). The CICS/VS Program Control Program (PCP) examines the DL/I user bit in the CICS/VS PPT entry. If the bit is set, CICS/VS branches to DL/I user task scheduling routine, DLZODP00. An indicator is set in the CICS/VS task control area (TCA) and control is returned to the CICS/VS PCP.

DL/I user task scheduling is comprised of the following subroutines:

- Task scheduling
- PST initialization
- PSB intent scheduling
- PSB initialization
- Deferred scheduling

The caller provides the name of the PSB to be scheduled or optionally if the caller omits the PSB name in the call list, the first PSB name encountered in this program's ACT entry is provided as default. This subroutine determines whether DL/I can support another task and creates an entry in the PST prefix area for this task.

The SCD maximum task indicator is tested. If it is on, the task cannot be scheduled, the SCD suspended task counter is incremented by one, and an indicator is turned on in the SCD. A CICS/VS SUSPEND macro is issued to suspend this task.

If the SCD maximum task indicator is off, an available PST prefix entry is located and initialized for this task. The DL/I task accumulator is incremented by one and a test is made to determine whether the number of DL/I tasks now equals the maximum allowed. If yes, the SCD maximum task indicator is set. Next the SCD current maximum task indicator is tested. If on, the task cannot be scheduled immediately, and the subroutine issues a CICS/VS WAIT against the event control block (ECB) in the assigned PST prefix. The SCD current maximum task indicator is set if the scheduling of the task causes the current maximum task value to be reached. Control is passed to the PST initialization subroutine.

PST INITIALIZATION

PST storage is acquired from CICS/VS Storage Management and the storage address is saved in the assigned PST prefix. PST initialization consists of formatting the save area chains and storing the address of the assigned PST prefix. Control is passed to the PSB intent scheduling subroutine.

PSB INTENT SCHEDULING

This subroutine determines the segment intent of the PSB being scheduled and ensures that no more than one task is scheduled to update the same segment type(s) in the same data base. For retrieve sensitive only PSBs, a duplicate PSB is created if a prior task was scheduled with the same PSB. The PSB directory entry for this PSB is located and a test is made to determine if the PSB is in use and if the PSB is retrieve sensitive only. If in use and retrieve sensitive only, a duplicate copy is made. If in use and not retrieve sensitive only, an indicator is set in the assigned PST prefix and a CICS/VS WAIT is issued on the ECB in the prefix. If not in use, but retrieve sensitive only, the in-use indicator is set and control is passed to PSB initialization. If neither of the above is true, the PSB segment intent list entry must be scanned.

The segment intent list for this PSB is located from the PSB directory entry. This list defines all segments in the data base(s) used by this PSB and also defines the PSB's sensitivity to them. The segment intent list entry is compared to the segment intent list entries of all scheduled PSBs. If no intent conflict is detected, the PSB initialization subroutine is called. Otherwise an indicator is set in the task's PST prefix and a CICS/VS WAIT is issued for the task. Upon completion of a successful segment intent scan, the PSB initialization subroutine is called.

If it is necessary to provide duplicate copy(s) of PSBs, this routine acquires storage for the copy and moves the original copy to it. Addresses in the duplicate are adjusted correspondingly and a duplicate PSB directory entry is created. The level table(s) are then reset and control passed to the PSB initialization subroutine.

PSB INITIALIZATION

PSB initialization consists of inserting the SDBs in the PSB into the SDB chain. The PSB is located from its PSB directory entry, and the address of the PCB address list is stored in the CICS TCA. Each PCB is located and the JCB pointer is used to obtain the address of the start of the SDBs for that PCB (JCBSDB1). Each JCB is accessed and the SDB chain pointers in the SDB and the PSDB in the DMB are updated. This process continues for all SDBs defined in the PSB.

The address of the assigned PST is obtained from the PST prefix and stored in the PSB. Using this address, the PSB directory entry address is stored in the PST. The "DL/I is scheduled" indicator in the PST prefix is set. If the PSB indicates the user is update sensitive, a call is made to the DL/I data base logger module (DLZRDBL0) to write an application program scheduling record (X'08'). Control is then returned to the calling routine.

SCHEDULING

A DL/I call allows scheduling. The function code is 'PCB' and the call contains the name of the PSB to be executed. The call is passed to the online program request handler via the language interface module and a scheduling validity check is made. If the call is valid, the task scheduling subroutine is called to schedule the PSB. Upon completion, control is returned to the application program through the program request handler and the language interface. If the call is invalid, a one byte error return code is stored in the CICS/VS TCA and control is returned directly to the application program.

If the 'PCB' call is made to schedule the system interface, the password is tested against the user generated one in the nucleus and the interface is tested for availability. A PST and dummy DSG are aquired for the caller, the task is marked as a system task, and control is returned to the user.

PROCESSING DL/I CALLS (ONLINE PROGRAM REQUEST HANDLER) - DLZODP00

DL/I online calls are made in the same format as batch calls. The user issues a call instruction, passing parameters in the call list, and provides a register save area address in register 13. Communication back to the user as to the results of the call is also identical to the batch system. It should be noted that although the format of the call instruction for online is the same as in batch, storage used by DL/I to process the call (i.e., register save area; all data items in the call list; I/O area) must be acquired from CICS/VS dynamic storage due to the re-enterability requirements of application programs which run under CICS/VS.

LANGUAGE INTERFACE MODULE

Although the language interface is not part of module DLZODP00, it is involved in call processing. The language interface module is link-edited to each application program via the call instruction. The module has two entry points; one for Assembler and COBOL, and the other for PL/I. The first function performed at either entry point is to save the user's registers. Then a language indicator is set, the entry point to the program request handler is acquired from the DOS/VS COMREG, and a branch is taken to the program request handler.

PROGRAM REQUEST HANDLER

This routine is responsible for communication to and from the DL/I action modules and the user. It establishes the necessary table addressability for the action modules, and formats and validity checks the call list. It also moves the requested data to the user's I/O area and returns control to the application program.

Upon entry, the determination is first made as to whether or not this is a scheduling call. If it is, the scheduling subroutine is entered. If not, addressability to the PST is established and the language indicator is set in the LIPARMS section of the PST. Next the user's call list is inspected to determine whether it is in the proper format. If not, the list is converted to the implicit direct format in an area provided in the PST. The address of the list is stored in the PST. Then the call list is checked to ensure that all addresses are valid. If valid, the call is passed to the call analyzer; otherwise, a return code is inserted in the TCA and control is returned to the user.

The DL/I action modules process the call and, when complete, control is returned to the program request handler through the call analyzer. A test is made to determine whether a pseudo-ABEND condition exists. If it does, a CICS/VS task ABEND macro is issued with an ABEND code indicating the reason. If an ABEND is not required, a test is made to determine whether the call requires data to be moved back to the user. The data is moved to the user's I/O area if required. The user's registers saved by the language interface are restored and control passed back to the calling application program.

Processing of the system calls 'CMXT', 'STRT', and 'STOP' is accomplished in the program request handler code. If these functions are identified in the call list a direct branch is taken to the appropriate routine.

IWAIT ROUTINE

The IWAIT routine is entered from the DL/I buffer handler (DLZDBH00) or from other modules whenever an I/O wait or resource enqueue wait must be issued. The following processing occurs:

- Registers 14 through 12 and 13 are saved.
- Registers 12 and 13 are initialized with the CICS/VS CSA and currently dispatched TCA.
- A CICS/VS WAIT to CICS/VS Task Control Management is issued.
- Upon return, registers 14 through 12 and 13 are restored.
- Return is to the calling module via register 14.

DL/I USER TASK COMPLETION - DLZODP00

DL/I user task completion is entered by the CICS/VS PCP when a user's task scheduled by DL/I returns through CICS/VS Program Management or when the DL/I 'TERM' call is issued from the application program. This routine is responsible for purging any buffers altered by this task, calling the data base logger to write the application program termination record (X'07'), releasing any system resources owned by this task, and posting or resuming tasks which were marked as not scheduled.

TASK TERMINATION

Task termination first determines whether this task was assigned a PST prefix. If not, this task must have been stall-purged by CICS/VS and was originally suspended by the task scheduling module. In this case the suspended count accumulator is decremented and the task's TCA removed from the DL/I suspended task chain. Control is then returned to CICS/VS Program Management. If the task terminates abnormally, its DL/I control blocks are dumped by DFHDC.

If this task was assigned a PST prefix, a test is made to determine whether the task was scheduled. If not, the task was stall-purged by CICS/VS. This means this task was suspended by a CICS/VS Storage Management attempt to acquire either PST or PSB storage. If it was due to PST storage acquisition, the assigned PST prefix is cleared and put back on the free chain and the system resource allocation routine is entered. If it was due to PSB storage acquisition, the PSB directory entry is cleared, PST storage is freed, and the PST prefix is inserted in the free chain. Control is then passed to the system resource allocation routine.

If the task was scheduled and active, normal task termination proceeds. First a DL/I internal 'TERM' call is issued to the call analyzer (DLZDLA00). This call causes the analyzer to reset the level table(s) in the PSB. If update sensitive, the buffer handler (DLZDBH00) is called to write out all buffers altered by this task. Next the PSB directory entry is tested for update sensitivity. If indicated, the data base logger (DLZRDBL0) is called to write the application program termination record (X'07'). If the task had update sensitivity, the PST prefixes are scanned and any waiting for scheduling because of segment intent conflict are 'POSTED'.

Next the PSB directory entry is released. For update sensitivity PSBs, this involves resetting the "user scheduled" indicator. For retrieve only, a test is made to determine whether this was a duplicate PSB. If so, the storage acquired for the PSB is freed and the duplicate PSB directory entry is cleared. Control passes to the system resource allocation routine.

If the system call interface is active the DDIR entries for the terminating PSB are checked for the waiting for close indicator. If the indicator is on and the use count of the DMB is now zero, the system task is resumed.

SYSTEM RESOURCE ALLOCATION

This routine is responsible for determining whether any tasks are waiting to be scheduled and, if so, for taking the proper action to cause them to be scheduled. First the DL/I suspended task counter is tested. If nonzero, the first task on the DL/I suspend chain is located and a CICS/VS RESUME macro is issued. The suspend chain is then updated by removing the task's TCA from it, the suspended task counter is decremented, and, if zero, the maximum task indicator is reset. Next the DL/I task counter is decremented. If the task count is less than the current maximum task value, the current maximum task indicator is reset and PST prefixes which were 'WAITING' due to this condition are 'POSTED' complete. Control is then returned to the CICS/VS PCP.

DL/I NORMAL SYSTEM TERMINATION - DLZODP00

The following processing occurs prior to CICS/VS termination.

- DL/I system termination (DLZOPD02) is entered from the DL/I linkage module DLZSTP00, as specified in the CICS/VS pre-termination processing list (PPL).
- The DL/I log DTF is located and a DOS/VS CLOSE is issued for the DL/I log.
- DL/I system termination is re-entered by CICS/VS System Termination Program.
- A DL/I CLOSE call is issued to the DL/I Open/Close module (DLZDLOC0) to close all data sets for all DMBs in the system.
- Return is made to the CICS/VS via the DL/I linkage module.

DL/I ABNORMAL SYSTEM TERMINATION - DLZODP00

The DL/I abnormal system termination routine is entered from CICS/VS when the DL/I partition is to be terminated abnormally. The following processing occurs:

- The DL/I log DTF is located and a DOS/VS CLOSE is issued for the DL/I log.
- The DL/I control blocks are dumped.
- Return is made to the calling CICS/VS program.

DL/I ONLINE MESSAGE WRITER - DLZODP00

The following processing occurs:

- The DL/I error code is extracted from the active PST.
- CICS/VS storage is acquired.
- The appropriate DL/I message is created and logged to the destination CSMT via CICS/VS Transient Data Management.
- Return is made to the calling routine.

If CICS/VS storage cannot be acquired or an error occurs while writing to transient data, an indicator is placed in the TCA and return is made to the calling routine.

VSAM EXCP EXIT PROCESSOR - DLZOVSEX

The EXCP exit processor receives control directly from VSAM after each SVC 0 resulting from a GET or PUT call from the buffer handler. DL/I checks the ECB for completion of the I/O request. If the request is incomplete the CICS/VS environment is re-established and a CICS/VS task control wait is issued in behalf of the current task. If the ECB was previously posted or the event completion has caused the task to be removed from the wait condition, control is returned directly to VSAM via register 14.

CHAPTER 6. DBD GENERATION

DESCRIPTION OF DBD GENERATION

DBD generation is composed of a set of DL/I macro instructions, the execution of which creates the user-specified data base description (DBD) and places it in the DOS/VS source statement library. The following macro instructions represent DBD generation:

Macro Instruction Name	Purpose
DBD	Allows the DL/I user to define the name of the DBD and the data base organization
DATASET	Allows the DL/I user to define names for data sets representing a data base, the device type used for storage of the data base, the logical record length, and the blocking factor for the physical records in the data sets representing the data base
SEGM	Allows the user to specify a DL/I segment, its parent segment, the segment length, the segment name, and segment prefix information
LCHILD	Allows the user to define an index relationship or a logical relationship in which a segment will participate.
XDFLD	Allows the user to define secondary indexing relationships.
FIELD	Allows the DL/I user to specify a data field or key field for a segment. The field definition includes the related segment field name, field start position in segment, field length, and field type.
DBDGEN	Causes the segments, fields, and data sets defined in the SEGM, FIELD, and DATASET macro instructions to be generated into an object module.
FINISH	Checks whether a DBDGEN statement was present.

The DBD generation macros utilize a universal set of globals. These globals are in the DOS/VS Source Statement Library and are named DLZDBGLB.

DBDGEN MACRO CALLING SEQUENCE

External Macro	Inner 1	Inner 2
DBD		
DATASET	DLZALPHA	
	DLZCKDDN	
	DLZDEVSI	
SEGM	DLZALPHA	
	DLZSOURS	DLZXPARM
		DLZALPHA
		DLZXTDBD
	DLZXPARM	
	DLZSEGPT	
	DLZHIERS	
	DLZXTDBD	
	DLZSETFL	DLZSEGPT
XDFLD	DLZALPHA	
	DLZCONVE	
LCHILD	DLZALPHA	
	DLZXTDBD	
	DLZSEGPT	
FIELD		
DBDGEN	DLZSEGPT	
	DLZLRECL	
	DLZCONVE	
	DLZSOURS	DLZXPARM
		DLZALPHA
		DLZXTDBD
	DLZXTDBD	
	DLZCAP	
FINISH		

SYMBOL NAME	SYMBOL TYPE	MATRIX SIZE	SYMBOL DESCRIPTION
ACC	CHAR	1	Database organization
ALIAS	BINARY	1	Current segment is logical
BLK	ALGEB	10	Blocking factor table
CDNBR	ALGEB	1	Current DMAN number
CYL	ALGEB	10	Cylinder cap. and ISAM variables
DBD	BINARY	1	DBD statement spec switch
DBDERR	BINARY	1	DBD error switch
DBN	CHAR	1	Database name
DBNAME	CHAR	1	Ext database ref table
DFL	ALGEB	10	DMAN field lengths (LLSS)
DLEV	ALGEB	10	Lowest level seg in dataset
DMN	CHAR	10	DMAN name table
DNBR	ALGEB	1	Number of OD DMAN statements
DSC	ALGEB	10	DMAN segment count
DSL	ALGEB	10	DMAN segment lengths (LLSS)
DS1	CHAR	10	DD1 name table
DS2	CHAR	10	DD2 name table
EBCDIC	CHAR	1	Inter-macro symbol
ERROR	BINARY	1	Inter-macro error switch
EXTDB	ALGEB	1	EXT database table next entry
EXTDBN	ALGEB	1	EXT database table entry
EXTDBR	ALGEB	256	Number of entries in ext dbtable
F@	ALGEB	1	Next field in segment entry
FB1	ALGEB	255	Field start & size
FB2	ALGEB	255	
FB3	ALGEB	255	
FB4	ALGEB	255	
FK1	BINARY	255	Segment field flag
FK2	BINARY	255	
FK3	BINARY	255	
FK4	BINARY	255	
FLDNAM	CHAR	255	Field in segment check table
FN1	CHAR	255	Field name
FN2	CHAR	255	
FN3	CHAR	255	
FN4	CHAR	255	
FSF1	CHAR	255	Source field name
FSF2	CHAR	255	
FSF3	CHAR	255	
FSF4	CHAR	255	
FT1	ALGEB	255	Field type & segment no.
FT2	ALGEB	255	
FT3	ALGEB	255	
FT4	ALGEB	255	
FF	ALGEB	1	Number of fields
HDB	ALGEB	1	Max. bytes (HD)
HDRBN	ALGEB	1	Max. RBN (HD)
HEX	ALGEB	1	Inter-macro symbol
IB	ALGEB	255	Non-index char value
LEV	ALGEB	1	Number of levels in database
LP	BINARY	1	Current segment has an LPARENT
MAXCHLD	ALGEB	1	Max. number of LCHILD specs
MAXDMAN	ALGEB	1	Max. number of DMAN specs
MAXFLDS	ALGEB	1	Max. number of field specs
MAXSEGS	ALGEB	1	Max. number of SEGM specs
NSEQ	BINARY	1	Seq. fields not allowed
OBLK	ALGEB	10	QSAM or output BLKFACT table
OREC	ALGEB	10	QSAM or output LRECL table
ORG	ALGEB	1	Database organization code
PARNT	BINARY	255	Segment parent pointer required
PLIST	CHAR	100	Inter-macro symbol
PNBR	ALGEB	1	Inter-macro symbol

PRD	BINARY	255	Segment paired indicator
QUITB	BINARY	1	Inter-macro error switch
RAPS	ALGEB	1	Number of root anchor points (HD)
REC	ALGEB	10	LRECL table
RTKEY	ALGEB	1	Root seq. field length
RMN	CHAR	1	Randomizing module name
S	ALGEB	1	Current segment number
SB	ALGEB	255	Segment size
SCN	ALGEB	10	Cyl. to scan for free space
SD	ALGEB	255	Source segment flag table
SEQCK	BINARY	1	Seq. field check flag
SFLD	ALGEB	255	Number of fields in segment
SF1	ALGEB	255	Segment flags (bytes 1 & 2)
SF2	ALGEB	255	Segment flags (bytes 3 & 4)
SI	CHAR	255	Segment index field name
SK	BINARY	1	Sequence field check flag
SL	ALGEB	1	Source segment entry
SLC	ALGEB	1	Number of LCHILD entries
SLCF	ALGEB	255	LCHILD flags (db, seq, flags)
SLCN	CHAR	255	LCHILD segment name
SLCP	CHAR	255	LCHILD paired segment name
SLCS	ALGEB	255	Number of LCHILD statements for segment
SLDP	ALGEB	255	Segment lev, db number & pointer number
SN	CHAR	255	Segment name
SS	CHAR	255	Source segment table
THRD	ALGEB	10	Third & quarter track capacity
TRK	ALGEB	10	Track & half track capacity

Macro	Symbol
DATASET	BLK,CDNBR,DBD,DBDERR,DEV,DLEV,DMN,DNBR,DS1, DS2,MAXDMAN,OBLK,OREC,ORG,QUITB,REC,SCN,TRK
DBD	ACC,DBD,DBDERR,DBN,HDB,HDBBN,MAXCHILD, MAXDMAN,MAXFLDS,MAXSEGS,QUITB,RAPS,RMN
DBDGEN	See the complete symbol table
DLZALPHA	QUITB
DLZCKDDN	DBD,DBDERR,DD,DDNS,IMS,NOGO
DLZCONVE	EBCDIC,HEX,QUITB
DLZDEVSI	CYL,DAE,THRD,TRK
DLZHIERS	DBDERR,HSEQ
DLZIRECL	BLK,CYL,DAE,DBDERR,DEV,DLEV,DSL,DSZ,OBLK, OREC,ORG,OVF,RAPS,REC,RKEY,SB,SFQ,SLDP, THRD,TRK
DLZSEGPT	BLK,CDNBR,DBDERR,DSL,OBLK,OREC,RKEY,S,SB, SLDP,SN
DLZSETFL	CDNBR,DBDERR,ERROR,HIER,LP,NSEQ,ORG,PARNT, PLIST,PNBR,PRD,S,SB,SF1,SF2,SLCF,SLCN,SLCP, SLCS,SLDP,SN,VV
DLZSOURS	S,SL,PNBR,EXTDBN,SLDP,SD,FT1,QUITB,ERROR, DBDERR,ACC,DBN,PLIST,SS,SN,FN1,FSF1
DLZXPARM	ERROR,PLIST,PNBR
DLZXTDBD	DBDERR,DBN,DBNAME,EXTDB,S,SN,EXTDBN
FIELD	ALIAS,CDNBK,DBD,DBDERR,DFL,F@,FB1,FLDNAM,FN1, FT1,FF,MAXFLDS,NSEQ,ORI,PRD,RKEY,S,SB,SEQCK, SFLD
ICHILD	ALIAS,CDNBK,DBD,DBDERR,IB,LP,MAXCHILD,ORG, PARNT,QUITB,S,SB,SF1,SI,SIC,SLCF,SLCN,SLCP, SLCS,SLDP
SEGM	ALIAS,BLK,CDNBR,DBD,DBDERR,DLEV,DSC,DSL, ERROR,EXTDBN,F@,FF,HIER,IB,IEV,LP,MAXSEGS, NSEQ,ORG,PLIST,PNBR,PRD,QUITB,REC,RKEY,S,SB, SEQCK,SF1,SF2,SFZ,SI,SK,SIC,SLCF,SLCN,SLCP, SLCS,SLDP,SN,VV
XDFLD	S,SL,SLC,FF,HEX,ORG,FY1,SFLD,SD,QUITB,FLDNAM, SN,SS,FN1,FSF1,DBDERR

DBDGEN MACRO DESCRIPTIONS

DATASET MACRO

This is an external macro through which data set/data set group information is specified by the user.

DBD MACRO

This is an external macro through which DBD control information is specified by the user.

DBDGEN MACRO

This macro terminates the DBD specification process. If the error switch, DBDERR, is not set, the control block generation phase is entered to create the required block entries.

DIZAIPHA MACRO

```
|        |           |  AN                       |
|        | DLZALPHA  |  AN1   ,FIELD=,CHAR=      |
|        |           |  ALL                      |
```

This macro tests a specific character position (represented by the CHAR= operand) or all character positions in a specific field (represented by the FIELD= operand) to determine if the chracter is one of the 39 aplhameric characters (A through X, #, $, @, and 0 through 9). The value range of CHAR is 1 to 255. The default value is 1. The global symbol QUITB is set in the following cases:

- If the positional parameter is not AN, AN1, or ALL and the character is not alphabetic (A through Z, #, $, @).
- If the positional parameter is AN and any chracter is not alphameric (A through Z, #, $, @, or 0 through 9).
- If the positional parameter is AN1 and the first character tested is not alphameric (A through Z, #, $, @, or 0 through 9).
- If the positional parameter is ALL and the first character tested is not alphabetic (A through Z, #, $, @).

DLZCAP MACRO

```
|        | DLZCAP    |  DEVICE, BICCKSIZ         |
```

This macro is called by DBDGEN to calculate the block capacity per track and cylinder provided the blocks do not have keys. These numbers are required to generate some entries within the DTFSD (HSAM) and ACB-extension. The capacities are returned using global arithmetic variables (GBLA). Input values are:

```
DEVICE:      2314, 3330, 3333, 3340
BLOCKSIZ:    in bytes (key length = 0)

Output (GBLA) and MNOTE:
$CAPTRK:     number of blocks per track (GBLA)
$CAPCYL:     number of blocks per cylinder (GBLA)
MNOTE:       DMAN150 if invalid device
MNOTE:       Comment containing $CAPTRK and $CAPCYL if calculation
             was successful
```

DLZCKDDN MACRO

	DLZCKDDN	FILENAME

This macro checks the validity of filenames specified by the user and verifies that the specified filenames are not duplicated.

The operand is:

FILENAME

is the one- to seven-character filename to be checked.

DLZCONVE MACRO

	DLZCONVE	VALUE= DIGITS= TO=

When used this macro converts the value of an algebraic set symbol to its printable hexadecimal equivalent; it also converts the value of a character set symbol to an algebraic value.

If the value to be converted is algebraic and is to be converted to printable hexadecimal, the converted value is returned in the character global set symbol EBCDIC. If the value to be converted is a character set symbol, the converted value is returned in the algebraic set symbol HEX.

The operands are:

VALUE=

is the character or algebraic set symbol to be converted

DIGITS=

is the number of digits to be converted. Range is 1 to 8.

TO=EBCDIC

The specified value is algebraic and is to be converted to printable hexadecimal.

TO=HEX

The specified value is character and is to be converted to algebraic.

If the input value is not convertible, or if the TO= operand is invalid, switch QUITB is set to 1 and the macro exits.

DLZDEVSI MACRO

	DLZDEVSI	DEVICE

This macro is called by the DATASET macro to set device capacity values for the specified device type. The device value specified in the DEVICE operand of the DATASET statement is passed to this macro.

DLZHIERS MACRO

	DLZHIERS	PC,PPC,PLEV

This macro is called by the SEGM macro to validate the hierarchical sequence of segment specifications. The macro maintains a 16-entry table (HSEQ) containing the lowest allowable PC at every level.

The operands are:

PC

specifies segment physical (or sequence) code

PPC

specifies parent physical code

PLEV

specifies parent level

An error message is produced if any of the following conditions exists:

- PC $\neq$ 1 and PLEV = 0
- PLEV > 14 or PPC > PC
- value of PPC $\neq$ value of HSEQ table entry represented by PLEV

DLZLRECL MACRO

	DLZLRECL	NUMBER

where NUMBER = 1

This macro is called by DBDGEN to calculate LRECL and BLKSIZE values for the file number specified in the operand field of the macro call.

DLZSEGPT MACRO

	DLZSEGPT	NUMBER

where NUMBER = 1

This macro is called by SEGM, LCHILD, and DBDGEN to maintain the symbol DSL, which contains the sizes of the largest and smallest segments in a data set. This macro produces error messages SEGM330, SEGM340, and SEGM350 if the segment referenced by the operand value violates those rules.

DLZSETFL MACRO

	DLZSETFL	FN,RULES=

This macro processes the POINTER or PTR operand of the SEGM macro and sets the &SF1(&S) and &SF2(&S) globals to reflect the entered value. The &SF1(&S) and &SF2(&S) globals set by this macro comprise the 4-byte flags field of the SEGTAB entry for this segment.

This macro is not entered if the DLZXPARM macro encountered an error while generating the &PLIST matrix, or if the SEGM macro detected an error in the POINTER or PTR parameter list.

Messages:

An error message is produced and processing is terminated if:

- An invalid keyword is encountered in the parameter list, or
- The RULES operand is omitted or invalid

Flag Byte 1 (&SF1(&S) Byte 2) is set as follows:

```
Bit 1 - CTR         If TWINBWD and/or LTWINBWD is specified,
    2 - TWIN        Bit 2 and/or Bit 5 is set on, in
    3 - TWINBWD     addition to Bit 3 and/or Bit 6,
    4 - PARNT       respectively.
    5 - LTWIN
    6 - LTWINBWD
    7 - LPARNT
    8 - HIER
```

Flag Byte 2 (&SF1(&S) Byte 3) is set as follows:

Bits 1 & 2 Indicate segment insert rule, where:

10 - Physical
01 - Virtual
11 - Logical (Default)

Bits 3 & 4 Indicate delete rule and set same as insert. (Default value is LOGICAL).

Bits 5 & 6 Indicate replace rule and set same as insert. (Default value is VIRTUAL).

Bits 7 & 8 Indicate physical location of inserts for nonsequenced segments, where:

10 - First
01 - Last (Default value)
11 - Here

Flag Byte 3 (&SF2(&S) Byte 2) is set as follows:

```
Bit 1           Segment PAIRED (set by SEGM)
    2-4         Reserved
    5           Physical parent PTR 1=DBLE (set by SEGM)
    6-8         Reserved
```

The operands are:

PN

specifies the parent segment number

RULES=

specifies the RULES= operand as specified on the SEGM statement

DLZXPARM MACRO

	DLZXPARM	PARM=,NBR=

When used this macro extracts parameters from a sublist and stores them in a global matrix (PLIST). Null values in the parameter list are stored as null values in the PLIST matrix.

The operands are:

PARM=

specifies the input parameter list values

NBR=

specifies the maximum number of operand values to be allowed in each subparameter

DLZXTDBD MACRO

	DLZXTDBD	DB,CODE

This macro builds an external data base reference table. It is called by SEGM, LCHILD, and DBDGEN.

The operands are:

DB

specifies a data base name or segment name

CODE

specifies the value SEGM or is omitted.

If the value SEGM is specified in the CODE operand, the segment name (SN) is searched to locate the value specified in the DB operand; when found, the symbol EXTDBN is set to contain an 01 in byte 0, and bytes 1, 2, and 3 contain an offset into SEGTAB. If the segment is not found, an MNOTE error message is produced.

If the CODE operand is omitted, the external data base reference table (DBNAME) is searched for the DB entry, and, if found, the symbol EXTDBN is set to contain the position of the found entry. If the DB value is not found, the value is added to the table and EXTDBN is set to that entry.

FIELD MACRO

This is an external macro used to define fields within a segment.

FINISH MACRO

This is one external macro, it checks whether a DBDGEN statement is supplied.

ICHILD MACBO

This is a external macro used to define index relationship for HIDAM and HDAM.

SEGM MACRO

This is an external macro used to define data base segments.

XDFLD MACRO

This is an external macro used to define in connection with the LCHILD statement secondary index relationships for HIDAM and HDAM.

CHAPTER 7. PSB GENERATION

DESCRIPTION OF PSB GENERATION

PSB generation is composed of a set of DL/I macro instructions, the execution of which creates the user-specified program specification block (PSB). The following macro instructions represent PSB generation:

Macro Instruction Name	Purpose
PCB	Allows the DL/I user to define a program communication block (PCB), one or more of which exist within a single PSB. A PCB must exist for each data base with which the associated application program PSB intends to interact. The PCB macro saves the type of PCB, associated data base name, the intended processing options on that data base, and the maximum key length within the data base. One or more PCB macros can be used in a single PSB generation. The limit is 20 PCB macros per PSB generation.
SENSEG	The SENSEG macro instruction allows the DL/I user to specify a segment within a data base to which the application program associated with this PSB is sensitive. Up to 255 SENSEG macros may follow a PCB macro.
PSBGEN	The PSBGEN macro allows the user to specify the associated application program language and the name of the PSB control block to be generated. The PSBGEN macro is the generating macro for the entire PSB control block and its internal PCB control blocks.

PSBGEN MACRO CALLING SEQUENCE

External Macro	Inner 1	Inner 2
PCB	DLZCKOPT DLZALPHA DLZXTDBD	
SENSEG	DLZCKOPT DLZXPARM DLZXTDBD	
PSBGEN	DLZPCBPD	

PSBGEN MACRO - GLOBAL SYMBOL CROSS REFERENCE

Macro	Symbol
DLZALPHA	QUITB
DLZCKOPT	E,P,PO,S,SPO
DLZPCBPD	DVSIZE
DLZXPARM	ERROR,PNBR,PLIST
DLZXTDBD	DBDERR,DBNAME,EXTDB,EXTDBN,DBN,S,SN
PCB	DB,E,P,PFB,PK,PN,PO,PS,PSS,QUITB,S,SEG
PSBGEN	DB,DBNAME,DVSIZE,E,EXTDB,P,PFB,PK,PN, PO,PS,PSS,S,SG,SN,SP,SPC,SPO,SS,SSE,SSG
SENSEG	E,ERROR,EXTDBN,P,PLIST,PN,PNBR,PO,PSS,QUITB, S,SEG,SG,SN,SP,SPC,SPO,SS,SSE,SSG

PSBGEN MACRO DESCRIPTIONS

DLZALPHA MACRO

A description of the DLZALPHA macro appears in Chapter 5.

DLZCKOPT MACRO

	DLZCKOPT	OPT,M

This macro is called by the PCB macro or SENSEG macro to validate the PROCOPT operand. The macro generates either the PCB or the SENSEG 'PROCOPT OPERAND IS INVALID' error message. Global symbol PO or SPO is set to contain the processing option.

The operands are:

OPT

specifies the PROCOPT operand as entered on the PCB or SENSEG statement

M

is PCB or SENSEG message number

DLZPCBPD MACRO

This is an inner macro called by the PSBGEN macro. It generates the PL/I dope vector table if LANG=PL/I is specified in the PSBGEN statement.

DLZXPARM MACRO

A description of the DLZXPARM macro appears in Chapter 5.

DLZXTDBD MACRO

A description of the DLZXTDBD macro appears in Chapter 5.

PCB MACRO

This is an external macro used to define a DB PCB.

PSBGEN MACRO

This is an external macro used to terminate PSB specifications, and, if no errors have been encountered, to cause the generation of the PSB control blocks.

SENSEG MACRO

This is an external macro used to specify sensitive segments in a data base PCB.

CHAPTER 8. APPLICATION CONTROL BLOCKS CREATION AND MAINTENANCE UTILITY

CREATE APPLICATION CONTROL BLOCKS - DLZUACB0

The application control blocks creation and maintenance utility creates the internal control blocks required by the DL/I application program. Using the PSB and DBDs as input, this utility creates DL/I internal format control blocks as output. These output control blocks must be link edited into the DOS/VS Core Image Library, either private or system, as specified by the user. These blocks contain information about the data bases and the programs which use them. They describe some device and media characteristics, the stored data structures, and the logical data structures as seen by both the system and application programs. The program accepts control card input to determine what functions are required.

The logic flow is as follows: The control card input stream is processed and each card is syntax-checked. A sorted list of requested blocks is built in main storage. Each PSB name specified on the control card is inserted into the list.

Each name on the constructed build list is then passed to the application control blocks builder module DLZDLBL0 to have blocks constructed. Each name, in turn, is passed to the ACB relocater and writer module DLZUAMB0; addresses are relocated relative to zero and the completed blocks are written to a SYSPCH or SYSLNK data set.

BLOCKS AND TABLES - DLZUACB0

Program control parameter block
PST
SCD
PDIR

SIZE OF MODULE - DLZUACB0

This module contains approximately 6,000 bytes of code.

INTERFACES - DLZUACB0

This module interfaces with the following modules:

DLZUSCH0 - Called to create and search sorted PSB lists
DLZDLBL0 - Loaded and called to build blocks
DLZUMSG0 - Called to format prebuilt messages

Register Contents

```
 R0-R1   = PARM registers
 R2-R8   = Work registers
 R9      = Pointer to PST
R10-R11 = Work registers
R13     = Pointer to save area and primary base register
R14-R15 = Operating system linkage registers
```

ACB MAINTENANCE BINARY SEARCH/INSERT - DLZUSCHO

The function of module DLZUSCHO is to create and search sorted lists in dynamic (GETVIS) storage using the binary search technique. Any number of lists may be created simultaneously (subject only to the limit of available storage). A list entry may be any length from 1 to 256 bytes. The key or sequence field may also be from 1 to 256 bytes in length and may be located anywhere in the list entry. The only restriction on keys is that they must consist of a single contiguous string of bytes within the list entry.

The number of entries in any list is limited only by available storage. However, since this routine physically moves data in storage to make room for new entries, it becomes less efficient as the number of entries increases. For large numbers of items, it might be best to consider sorting the entries in the conventional fashion.

This module is called by DLZUACBO to build and maintain the list of PSBs to be processed.

Operation

I. The following interface is used to initiate a new list:

```
L 15,=V(DLZUSCHO)
LA 1,PARMS
BALR 14,15
```

where PARMS is a 3-word list whose contents are as follows:

```
Word 1 = length of the list entry
Word 2 = offset from the beginning of the list
         entry to the key/sequence field
Word 3 = length of the key/sequence field
```

On return, register 1 contains the location of the new list control block. (This location must be submitted to the search routine on all subsequent search or insert calls for this list.)

II. The following interface is used to insert an entry into a list:

```
L 15,=V(INSRCH)
LA 1,INPARMS
BALR 14,15
```

where INPARMS is the location of a two-word list whose contents are:

```
Word 1 = address of the list control block
Word 2 = address of the list entry to be
         inserted
```

On return from INSRCH, register 15 contains zero if the entry was successfully inserted, and register 1 contains the location at which the insert was made.

If the entry was not inserted (because a duplicate was found), register 15 contains 8, and register 1 contains the location of the duplicate entry.

III. The following interface is used to locate an entry in a list created by INSRCH:

```
L 15,=V(LOCSRCH)
LA 1,LOCPARMS
BALR 14,15
```

where LOCPARMS is the location of a two-word list whose contents are:

Word 1 = address of the list control block
Word 2 = address of the search argument (key)

On return from LOCSRCH, register 15 contains zero if an entry containing the search argument in its key field was found, and register 1 contains the location of this entry.

If no entry was found, Register 15 contains 4 and register 1 remains as it was on entry to LOCSRCH.

IV. The following interface is used to delete all storage obtained by OPENSRCH and INSRCH for a given list:

```
L 15,=V(CLOSESCH)
L 1,LOCPARMS
BALR 14,15
```

where LOCPARMS contains the location of the list control block for the list to be deleted.

SIZE OF MODULE - DLZUSCH0

This module contains approximately 800 bytes of code.

CONTROL BLOCKS - DLZUSCH0

- List control block
- Sorted list block.

Programming Note

If some number of entries have been placed in a list through repeated calls to INSRCH, they can be retrieved in sorted order by locating the first block by way of CHAINLOC and all subsequent blocks by way of their CHAIN fields. The entries are in order (low to high logical sequence) with the lowest entry in block 1 entry 1, next in block 1 entry 2, etc., with the highest entry located in the last-used slot in the last block.

ERROR MESSAGE FORMATTING - DLZUMSG0 AND DLZLBLM0

Given a message number, DLZUMSG0 selects a message from a message list, formats it, and calls the print function. In addition, depending upon the input parameters, the routine accepts one or more text strings to be inserted into the standard text.

The messages are stored in a separate CSECT, DIZUMGTO. Each message is flagged to indicate if the insertion function is allowed. If the insertion function is allowed, the starting position of the insert and the insert length are also stored with the standard text.

Two macros are used in support of these functions: DLZMSG, which generates and encodes the standard message and its insert parameters for the message list CSECT, and DLZER, which is used by the programmer to invoke DLZUMSGO.

DIZLBLMO is called only by DLZDLBLO to execute the DLZER macro for an error message. The message number is supplied in register 1.

SIZE OF MODULES - DLZUMSGO AND DLZLBLMO

These modules contain approximately 2000 bytes of code.

INTERFACES - DIZUMSGO

Register Contents on Entry

R1 - Pointer to input parameter list
R13 - Save area
R14 - Return address
R15 - Entry point

Parameter List Format

```
DS 1H message number in binary
DS 1H 0=standard message,
      1=inserts
DS A(address of first insert string)
    .
    .
DS A(address of last insert string)
```

The list of addresses is only valid when the second halfword contains -1. The length and the number of inserts allowed are encoded with the standard text in the message CSECT DLZUMGTO. The message number is used to find the desired message in the CSECT.

INTERFACE - DLZLBLMO

Register Contents on Entry

R1 - Message number
R13 - Save area
R14 - Return address
R15 - Entry point

Additionally, any registers which have been defined to contain specified addresses or information pertaining to the message must be valid. These are normally registers 5, 6 and 7.

EXTERNAL ROUTINES CALLED - DLZUMSGO

PRTMSG - Entry point to the print routine in the application control blocks creation and maintenance utility (DLZUACBO).

EXTERNAL ROUTINE CALLED - DLZLBLMO

DLZUMSGO - via DLZER macro

APPLICATION CONTROL BLOCKS BUILDER - DLZDLBLO

The application control blocks builder module is responsible for loading PSBs and DBDs for use of these control blocks by DL/I. It is given control by module DLZUACBO when DL/I blocks must be created from a PSB.

This module (DLZDLBLO) builds the required control blocks for DL/I data base execution. Specifically, the module loads the specified PSB and builds a JCB for each data base PCB within the PSB. Each DBD defined by the PCBs for the PSB is checked to determine whether a DMB exists in a core image library for that DBD. If no DMB exists, one is created.

In order to build the JCBs and DMBs, the DBDs for the data bases to be used must be obtained.

All of the data base organization and structure information provided by the PSBs and DBDs is consolidated into the appropriate DL/I internal control blocks by this module for use in servicing subsequent data base I/O requests.

For all newly created DMBs, a special utility PSB is created and provided as output by DLZUAMBO. DLZDPSBO is called to build the PSB (in PSBGEN format) from a given DMB.

SIZE OF MODULE - DLZDLBLO

This module contains approximately 12,500 bytes of code.

INTERFACES - DLZDLBLO

This module interfaces with the following modules:

DLZUAMBO	-	Called to write the blocks to SYSLNK or SYSPCH
DLZDPSBO	-	Called to build a utility PSB
DLZLBLMO	-	Called to format and write error message

Register Contents on Entry

R1 - PST address
R13 - Save area address
R14 - Return address
R15 - Entry point address

Register Contents on Exit

All registers are restored. The return code appears in PSTERCOD of the PST.

PSTERCOD = 0	Valid return
PSTERCOD ≠ 0	Errors encountered

ACB RELOCATER AND WRITER - DLZUAMB0

This module (DLZUAMB0) is called by the application control blocks builder module (DLZDLBL0). It changes addresses in the DMBs and PSBs to offsets. DMB addresses are relocated and the created DMB is written to SYSPCH or SYSLNK as defined in the control card. Next, the PSB fields are converted to offsets and the PSB is written to either SYSPCH or SYSLNK. The beginning of the PSB contains a PSB intent list entry. During DL/I initialization, this information is used to create the executable PSB intent list. Control is returned to the application control blocks builder.

Should errors be encountered, appropriate error messages are written via DLZUMSG0 and control is returned to DLZUACB0 with a nonzero value in PSTERCOD.

SIZE OF MODULE - DLZUAMB0

This module contains approximately 5000 bytes of code.

INTERFACES - DLZUAMB0

This module interfaces with the following module:

DLZUMSG0 - called to write an error message.

Register Contents on Entry

R1 - Address of the parameter list
R13 - Save area address
R14 - Application control blocks builder return address
R15 - Entry point

Parameter List Format - DLZUAMB0

Hex	Dec	Name	Ln	Description
0	0	MOVELIST	4	Constant 'MOVE'
4	4	PSBSAVED	4	Address of PSB
8	8	DUMDDIRB	4	Address of temporary DDIR
C	12	ENQLSTA	4	Address of ENqueue list
10	16	PCBST	4	Address of PST
14	20	ACUMOPT	4	Address of cumulative option flags.

Register Content on Exit

All registers are restored. PSTERCOD in the PST contains the return code as follows:

PSTERCOD = 0 - Valid return
PSTERCOD ≠ 0 - Errors encounterd

UTILITY PSB BUILDER - DLZDPSBO

This module is called by the application control blocks builder module (DLZDLBLO) to dynamically construct a special utility PSB from a specific DBD. The created PSB is in PSBGEN format. A GETVIS is issued to obtain storage necessary to create the PSB. The created PSB is sensitive to all segments for the data base.

SIZE OF MODULE - DLZDPSBO

This module contains approximately 800 bytes of code.

INTERFACES - DIZDPSBO

Register Content on Entry

R1 -	Address of parameter list
R13 -	Save area address
R14 -	Return address of DLZDLBLO
R15 -	Entry point

The parameter list consists of a DBD address and a PSB address.

Registers on Exit

All registers are restored except R15 which contains a return code passed to DLZDLBLO.

R15 = 0	Valid return
R15 ≠ 0	Errors encountered

CHAPTER 9. DATA BASE RECOVERY

DATA BASE BACKOUT UTILITY

BATCH BACKOUT INTERFACE - DLZBACKO

The batch backout interface module reads the DL/I log tape and passes the data base log records to the data base backout module (DLZRDBCO) for processing.

By reading the log tapes in a backward mode, this module is able to process the data base records in reverse sequence without using an intermediate work data set. In order to read the tape backward, the RECFORM parameter of the DTFMT macro is specified as undefined (UNDEF). When a block is read in, it is searched and the sequence field located at the end of each logical record is replaced by the length of that logical record. With the length thus in the back of a record as well as in the front, it is deblocked and spanned.

The interface process includes the following record types:

X'07' - Application program termination record
X'08' - Application program scheduling record
X'50' - Data base log record
X'51' - Data base log record

The batch backout utility is executed under DL/I control as an application program. Processing of module DLZBACKO is as follows:

1. Control is received from DL/I initialization and the PSB name is obtained from the parameter data.

2. The log tape is opened to be read backward.

3. The log tape is read backward and records bypassed until the first data base log record for the PSB is obtained.

4. An application program termination record (X'07') for the PSB indicates no backout necessary, the message "BACKOUT COMPLETE" is issued at SYSLOG, the log is closed, and the job is terminated.

5. Data base log records (X'50' and X'51') are passed to module DLZRDBCO to be processed against the appropriate data base. Processing terminates when an application program scheduling record is read, the message "BACKOUT COMPLETE" is issued at SYSLOG, the log is closed, and the job is terminated.

If end of file is reached on the log (i.e., the header record is read), it is closed, a "BACKOUT COMPLETE" message is issued, and the job step is terminated. The job is terminated by returning control to DL/I which purges all buffers, closes all DMBs, and closes the output log file. Because of DOS/VS restrictions on reading backwards, a multi-volume log file cannot be processed.

SIZE OF MODULE - DLZBACKO

This module contains approximately 4000 bytes of code.

INTERFACES - DLZBACKO

Register Contents on Entry

```
R1  =   PSB list address
R13 =   Save area
R14 =   Return
R15 =   Entry point
```

CONTROL BLOCKS - DLZBACKO

Application program scheduling record
Application program termination record
Data base log record
PDIR
PST
SCD

EXTERNAL MODULES CALLED

DLZRDBCO - Called to interface with DL/I and perform backout.

RECORD AND MESSAGE FORMATS - DLZBACKO

All messages are sent to the SYSLOG and SYSLST devices. The messages are contained in module DLZBACMO.

DATA BASE BACKOUT - DLZRDBCO

This module receives control from DLZBACKO with a log record to process. It calls open/close (DLZDLOCO) to open the DMB specified in the record unless the data base is already open. The buffer handler (DLZDBHOO) is called to retrieve the KSDS or ESDS block as indicated by the key or the ESDS relative block number or relative byte address.

The data in the buffer is replaced with the 'old' information in the log, thereby nullifying the offending programs update. In the case of HD, when a physical delete or insert record is processed, space management (DLZDHDSO) is called to update the free space elements and bit map, if necessary and to build the input data for the data base logger. DLZRDBLO is called to record the changes made to the data base.

The buffer handler is then called again to mark that buffer altered and control is returned to DLZBACKO.

SIZE OF MODULE - DLZRDBCO

This module contains approximately 1500 bytes of code.

INTERFACES - DLZRDBCO

Register Contents and Control Blocks on Entry

```
     R1   = PST address
    R13   = Save area
    R14   = Return
    R15   = Entry point
PSTSCDAD = SCD address
ADDRLOG  = Address of data base log record within DLZBACKO
PSTDGU & PSTDGN must be zero on initial entry
```

CONTROL BLOCKS - DLZRDBCO

Data base log record
DDIR
DMB
DSG
PST
SCD

EXTERNAL MODULES CALLED

DLZDBHOO	-	Called to read a data base record and to mark the buffer altered
DLZDHDSO	-	Called to free or reserve space in an HDAM or HIDAM record
DLZDIOCO	-	Called to open data base
DLZRDBLO	-	Called to log backout modifications to data base

Interface with external modules

All modules expect R14 + R15 to contain return address + module entry print address.

DLZDIOCO

```
R1 = address of PST
R2 = address of DDIR entry for DMB to be opened

PSTDSGA  = address of DSG to open
PSTFNCTN = PSTOCDMB + PSTOCOPN
SCDCWRK  = address of normal log record work area
```

DLZDBHOO

```
R1 = address of PST

PSTBLKNM = RBN if HD ESDS
PSTACBNO = 1
PSTDMBNO = 1
PSTBYTNM = RBA if HISAM ESDS or address of key if KSDS
PSTFNCTN = desired function
```

DLZDHDS0

R1 = address of PST
R5 = address of PSDB of segment

PSTOFFST = offset to segment from beginning of block
PSTCODE1 = indicates backout in control (for logger)
PSTFNCTN = PSTFRSPC + X'80' (to show backout in control)

DLZRDBL0

R0 = SCD address
R1 = PST address

PSTCODE1 = PSTININT + PSTSCHED to indicate backout calling
PSTDATA = address of data in buffer
SCDCWRK = address of backout log work area containing the control information for this log record

Register Contents on Exit

All registers are restored with the exception of register 15 which contains a return code. If this code is non-zero, DLZBACK0 will print and type the appropriate error message.

ERROR CODES AND HANDLING = DLZRDBC0

All error codes are passed to DLZBACK0 in register 15.

DATA BASE DATA SET RECOVERY UTILITY - DLZURDB0

The data base data set recovery utility module DLZURDB0 is executed under DL/I control as an application program. Control is passed to DLZURDB0 from DL/I initialization. This module is comprised of two independent but logically related functions. The first consists of an image dump and a change accumulation processor. The PCB address is saved, and a GSCD call is issued to obtain the PST address. A control card which defines the data base/data set to be recovered is read. From this information, a DMB is loaded from the Core Image Library to obtain the physical characteristics of the data set to be recovered. Once the control card processing is complete, the DL/I open/close routine (DLZDLOC0) is called to open the output ACB and the input file is opened. Then the program enters a dump/cum data merge routine. This routine selects a dump record, merges any accumulated changes from the cum data set, and a call is made to the buffer handler (DLZDBH00) to write the new record to the output data set. Upon completion, a partial or completely recovered data set may exist. If no additional changes are to be applied through log tapes, the program calls the DL/I open/close routine (DLZDLOC0) to close the output ACB and terminates.

If additional changes are to be applied from log tapes, the program enters the second function. This routine opens the logs, scans the log to find a record that applies to this data set, and merges the data from the log to the data set record. Upon completion, the routine does post-processing and a recovered data set then exists.

The operation of this routine depends on certain DL/I functions to process the logs. The log is scanned for a matching data base/data set name record. When one is encountered, the record ID, either a key of a KSDS record or a relative block number of an ESDS record is saved, and a call is made to the buffer handler (DLZDBH00) requesting that the record be retrieved. Upcn successful return, the log record data is merged with the returned record, and a call is made to the buffer handler requesting that the record be marked as altered to cause rewriting. The records from the log are thus processed until an end of file is encountered on the log input. At this time, a call is made to the buffer handler requesting that all altered buffers be purged, that is, that all records that have been altered be rewritten. The program then calls the DL/I open/close routine (DLZDLOC0) to close the output ACB, and the prcgram terminates.

BLOCKS AND TABLES - DLZURDB0

This module utilizes certain DL/I blocks, including the PST, DSG, DMB, DMB directory, SDB, PCB, JCB, and SCD. Additionally, several record formats are used as follows:

1. HISAM reorganization header and data records. See Chapter 9 on HISAM reorganization unload (module DLZURUL0) for details.

2. Data base image dump header and data records. See discussion below of the data base data set image copy module (DLZUDMP0) for details.

3. Accumulated change CUM header and data records. Refer to discussion of the change accumulation module (DLZUCUM0) in this chapter for details.

4. Data base change log records.

SIZE OF MODULE - DLZURDB0

This module contains approximately 35,000 bytes, including I/O areas.

INTERFACES - DIZURDB0

Normal Entry Points

The cnly entry point to this module is DLZURDB0.

Register On Entry

R1 = pointer to fullword containing address of PCB

Registers On Exit

All registers are restored to entry conditions.

MODULES CALLED BY DLZURDB0

The DL/I open routine (DLZDLOC0) is called to open a specific ACB.

R1 = pointer to PST

The DL/I buffer handler (DLZDBH00) is called to retrieve and write a specific record, mark a buffer altered, and purge (rewrite) all altered buffers.

R1 = pointer to PST

The DL/I close routine (DLZDLOC0) is called to close a specific VSAM ACB.

R1 = pointer to PST

ERROR CODES AND HANDLING - DLZURDB0

All codes are in the form of messages. The module DLZRDBM0 contains all error messages issued by the Data Base Data Set Recovery Utility.

DATA BASE DATA SET IMAGE COPY UTILITY - DLZUDMP0

The data base data set image copy utility module DLZUDMP0 is executed as a standard DOS/VS application program and creates a backup copy of a specific data base data set. Input may be either a KSDS (HISAM, Simple HISAM, or HIDAM INDEX) or an ESDS (HISAM, HIDAM, or HDAM). The output is used as input to the data base data set recovery utility. Processing is as follows:

1. A control card is read from SYSIPT and preliminary validity checking is performed on various fields. The input card defines the data base/file to be dumped, the dump output symbolic filenames, and the number of output copies to be created.

2. The device type is determined for each output file specified and the file(s) are opened.

3. The DMB is loaded from a core image library to obtain the physical characteristics of the data base file to be dumped.

4. A header record is written to the output file. This record contains information necessary to allow the use of the image dump file by the data base data set recovery utility.

5. The appropriate VSAM control blocks are generated for the file.

6. The input file is opened.

7. Input segments are read sequentially, an 8-byte prefix is added to identify the segment, and the logical record (prefix + segment) is blocked and written to the output file.

8. After all segments have been copied (EOF), the input and output files are closed.

9. Output statistics for the file are written to SYSLST.

10. Processing continues from step 1 until there are no more input cards, at which time the program terminates.

DESCRIPTION OF MODULE - DLZDMPMO

Module DIZDMPMO is a read-only CSECT containing all the messages used by the Data Base Data Set Image Copy Utility.

CONTROL BLOCKS - DLZUDMPO

- Dump record prefix
- Dump header record.

SIZE OF MODULE - DLZUDMPO

This module contains approximately 14,500 bytes, including I/O areas.

ERROR CODES AND HANDLING - DLZUDMPO

All error codes are in the form of messages.

DATA BASE CHANGE ACCUMULATION UTILITY - DLZUCUMO

The data base change accumulation utility module DLZUCUMO is executed as a standard DOS/VS application program. DLZUCUMO controls the overall operation of the Data Base Change Accumulation Utility by opening the log input DTF. If this is successful, the control card processor module (DLZUCCTO) is called to read the input stream. Upon its return, the PROCFLAG switch is tested. If records are to be passed to sort, the sort parameter list is formatted, including a sort Exit 15 (DLZUC150) and the sort Exit 35 (DLZUC350). The sort program is then loaded, and this module (DIZUCUMO) waits for it to terminate. Upon termination, a completion code is tested and appropriate messages are provided as output. If records are not to be sorted, that is, no DBO type control cards were read, the module calls the Exit 15 module (DLZUC150) to create the new log tape.

DESCRIPTION OF MODULE - DLZUCERO

This module is the common error routine. Control may be passed to it from any of the four processing modules. It addresses a message from the message module (DLZCUMMO), depending on parameters passed to it, and prints a message to the SYSLST device. If the passed parameters indicate a multi-part message, it does not write the message on the first entry. Instead, it passes the last-used position in the output buffer back to the caller to allow the caller to insert special data in the messages. On the second entry to this routine, the message is written.

DESCRIPTION OF MODULE - DLZUCCT0

This module is the control card processor. It reads the control card input stream, checks the cards for validity, and constructs the data base name table and the date/time table if data base names are supplied.

DESCRIPTION OF MODULE - DLZUC150

This module is the sort Exit 15 routine. It reads the log input records, checks the purge date if applicable, and determines the disposition of the record. If the record matches an entry in the data base name table, the date/time table is searched and the appropriate purge date and time are compared. If the record is before the purge date, the program returns to read another record. If the record is not purged, the routing is determined from the table and written either to sort or to the new log. A table of DMB names and purge dates is prepared for Exit 35.

DESCRIPTION OF MODULE - DLZUC350

This module is the sort Exit 35 routine. It receives all records from sort. If an old accumulated data set is supplied, a record is read from the data set and a record is retrieved from sort. The data base name and file identification of the records are compared. All input cum records are purge-checked according to the date/time, if any, specified on DBO card(s). If the old cum input is low, it is written to the new cum data set. If the records are equal, the data from the sort record is merged to the old cum record, unless purged, and another record is obtained from sort. This sequence continues until an unequal condition is detected, at which point the record is written to the new cum data set. If the old cum is high, records from sort are combined and written to the new cum data set until the compare condition changes. This process continues until both the sort and the old cum records are exhausted.

DESCRIPTION OF MODULE - DLZCUMM0

This module contains all messages issued by the Data Base Change Accumulation Utility. It is read only.

CONTROL BLOCKS - DLZUCUM0

- Data base name table, containing the data base names and the address of the date/time table for this entry.
- Data/time table
- Accumulation header record
- Accumulation record

SIZE OF MODULE - DLZUCUM0

This module contains approximately 13,000 bytes, including I/O areas.

INTERFACES - DLZUCUM0

Normal_Entry_Point

The main entry point to this module is DLZUCUM0. DLZERRTN is an entry point used by DLZUC150 on any error condition.

Entry_Conditions

This is the main module which controls the overall operation of the Data Base Change Accumulation Utility program.

Control information is passed from module to module by means of an externally referenced table contained in DLZUCUM0.

SIZE OF MODULE - DLZUCER0

This module contains approximately 300 bytes.

INTERFACES - DLZUCER0

Normal_Entry_Point

The only entry to this module is DLZUCER0.

Entry_Conditions

This module is entered to output all error messages.

Register_Contents_on_Entry

R1 contains a message number in true form if a SYSLST message, and in complement form if a write to SYSLOG. R2 is negative if this is a multi-part message.

Register_Contents_on_Exit

All registers are restored to entry conditions.

SIZE OF MODULE - DLZUCCT0

This module contains approximately 2500 bytes.

INTERFACES - DLZUCCT0

Normal Entry Point

The only entry to this module is DLZUCCT0.

Entry Conditions

This module is entered to process the control card input stream.

Register Contents on Exit

All registers are restored to entry conditions.

SIZE OF MODULE - DLZUC150

This module contains approximately 6200 bytes.

INTERFACES - DLZUC150

Normal Entry Point

This module is entered at DLZUEX15 if no records are to be accumulated, and at DLZUC150 by sort.

Entry Conditions

This module is entered to read input logs and disperse records to new log or sort. R1 contains the address of the parameter list from sort or a dummy list if control was received from DLZUCUM0.

Register Contents on Exit

All registers are restored.

SIZE OF MODULE - DLZUC350

This module contains approximately 29,000 bytes.

INTERFACES - DLZUC350

Normal Entry Point

This module is entered at DLZUEX35 by sort.

Register Contents on Entry

Register 1 contains the address of the sort Exit 35 parameter list.

Entry Conditions

This module is entered by sort to dispose of all sorted records.

Register Contents on Exit

All registers are restored to entry conditions, with the sort parameter list updated as needed.

ERROR CODES AND HANDLING - DLZUCUM0, DLZUCTT0, DLZUC150, DLZUC350

All error codes are in the form of messages and are issued via DLZUCER0.

HISAM REORGANIZATION UNLOAD UTILITY - DLZURULO

The HISAM reorganization unload module DLZURULO is executed as a standard DOS/VS application program. A control card specifying the data base name, data set name, and output symbolic unit name is read. The DBD specified is loaded, and a short segment table is constructed. This table consists of the first eight bytes of each segment table entry in the DBD. This includes, among other things, the segment physical code and the segment length. The size of the prefix, as described for each segment type, is added to the segment length and entered in the table. This length is later used to move the segment from the input area to the output area.

Next, the input and output data sets are opened. A header record containing information about the data base data sets is constructed, and a statistics record is written. The first KSDS record is then read and the root segment is checked to determine whether the deleted flag is on (no prefix if Simple HISAM). If it is on, the total segment chain for that root is ignored, and the next root is processed. If the root is not deleted, it is moved to the output area, and the first dependent segment, if present, is processed. If the dependent segment is not deleted, it is moved to the output area, and the next segment is processed. This continues until the complete dependent segment chain for this root, including any overflow dependent segments on the ESDS, have been processed. If the segment is deleted, each succeeding segment that is a child of the deleted segment is also deleted. The first segment that is not a child of the deleted segment causes the normal segment processing to be resumed. The last record written is a statistics record which includes information needed for audit trail. The output data set now contains the reorganized KSDS and ESDS logical records in physical sequential format (only KSDS if Simple HISAM). An image of the KSDS record containing a root segment and dependent segment is followed by images of the ESDS records containing overflow dependent segments for the root segment. A chain pointer in the KSDS contains the correct relative byte address of the next ESDS record containing overflow dependent segments. If more than one ESDS record is needed to contain overflow dependent segments, they follow in sequence and chain pointers are maintained in the records.

Error message handling is accomplished in the following manner: When a routine within module DLZURULO requires an error message to be generated, a number is loaded into R1. This number corresponds to a message in the message CSECT (DLZRULMO). The routine then branches to a common routine which outputs the message. The number passed in R1 is multiplied by 4 and added to the start of the message CSECT (DLZRULMO). At that offset, a fullword containing the length of the message and the offset to the start of message text is obtained. These values are used to move the message to an output buffer.

DESCRIPTION OF DLZRULMO

DLZRULMO is a read-only module containing all error messages issued by module DLZURULC.

CONTROL BLOCKS - DLZURULO

- Short segment table
- Output data record
- Output header record
- Statistics record.

SIZE OF MODULE - DLZURULO

The size of this module is approximately 20,500 bytes, including I/O areas.

ERROR CODES AND HANDLING - DLZURULO

All error codes are in the form of error messages.

SAMPLE DESCRIPTION OF HISAM REORGANIZED FORMAT

Assume a HISAM data base which consists of a single root segment and dependent segments in the hierarchical format shown in Figure 10.1.

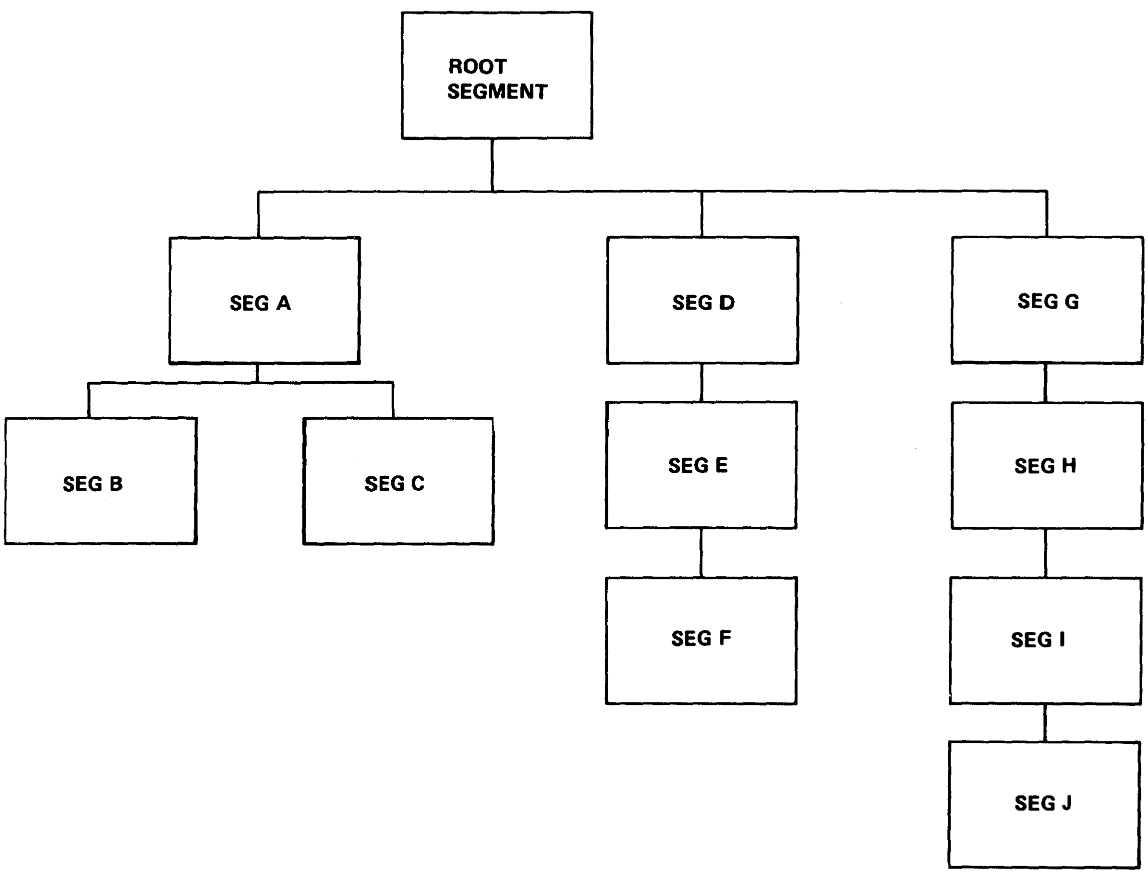


Figure 10.1. HISAM Data Base with One Root Segment

The input for the HISAM Reorganization Unload Utility appears as shown in Figure 10.2.

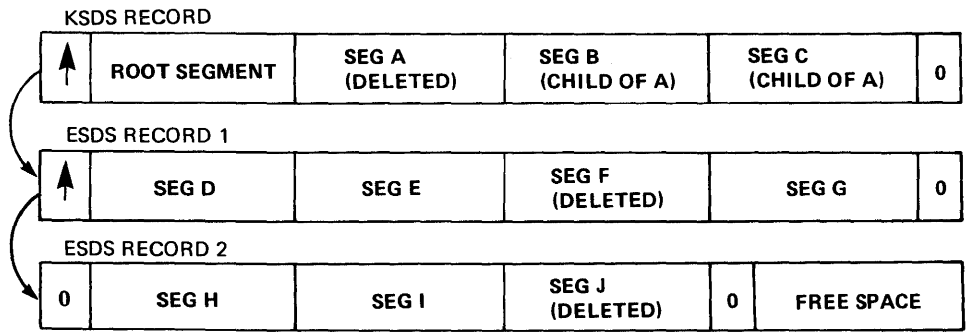


Figure 10.2. Input for HISAM Reorganization Unload Utility

Given this input, the HISAM Reorganization Unload Utility provides the output shown in Figure 10.3.

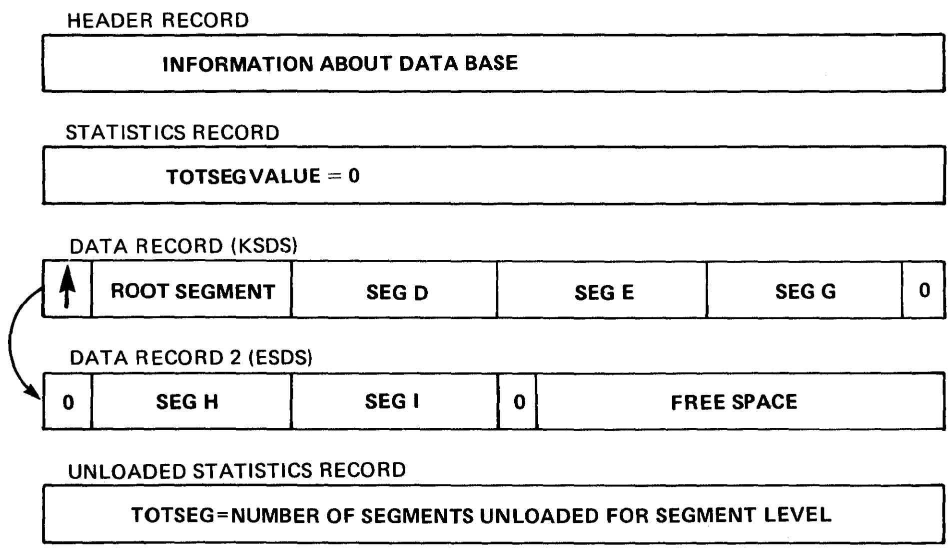


Figure 10.3. HISAM Reorganization Unload Utility Output

Note: A second ESDS record is unnecessary because space occupied by deleted segments is reclaimed.

HISAM REORGANIZATION RELOAD UTILITY - DLZURRLO

The HISAM reorganization reload module DLZURRLO is executed as a standard DOS/VS application program and is used to reload a reorganized HISAM data base data set group. The input to the program consists of a reorganized dump of the key sequenced data set (KSDS) and entry sequenced data set (ESDS) created by the HISAM Reorganization Unload Utility program. Processing is as follows:

1. A control card, which contains the filename of the input file containing the HISAM data base to be reloaded, is read. The input file is opened and the header record is read.

2. The output KSDS and ESDS ACBs are generated using the information contained in the header record and the KSDS and ESDS are opened (only KSDS if Simple HISAM).

3. The statistics record is read and the statistics table initialized.

4. Records are read sequentially from the input file. These records are images of KSDS and ESDS records.

5. KSDS records are written to the output KSDS using VSAM keyed sequential (mass) insert.

6. ESDS logical records are written to the output ESDS using VSAM addressed sequential insert.

7. After all data records have been processed, the last input statistics record is read, and a statistics report is printed, comparing segments unloaded/reloaded.

8. The files are closed.

DESCRIPTION OF DLZRRLM0

DLZRRLM0 is a read-only module containing all error messages issued by module DLZURRL0.

CONTROL BLOCKS - DLZURRL0

- Header record
- Input data record

SIZE OF MODULE - DLZURRL0

This module contains approximately 15,700 bytes, including I/O areas.

HD REORGANIZATION UNLOAD UTILITY - DLZURGU0

The HD reorganization unload module DLZURGU0 is executed under control of the DL/I system as an application program and is used to unload a data base by issuing DL/I calls. One or two files may be created and output may be to tape or DASD. The module contains two processing modes - "normal" and "restart".

Normal processing, after module DLZURGU0 receives control from DL/I, is as follows:

1. The PCB address is saved and a GSCD call is issued to obtain the PST address. The PST allows the program to access the DL/I control blocks needed to construct the prefix portion of the output record. This prefix, as described below, is used by the HD Reorganization Reload Utility.

2. The number of outputs (one or two) and output device type (tape or DASD) are determined.

3. Storage is obtained for the statistics table.

4. Each output file is opened.

5. The statistics tables, which have been initialized for all data base segment types, are written to the output file(s).

6. A Get Next (GN) call is issued for the first (or succeeding) segment.

7. The statistics table for the segment type is updated.

8. The segment is combined with the segment prefix to form an output logical record. The output logical records are blocked and written.

9. Whenever a checkpoint interval is reached (first root segment after 5000 segments have been processed), a checkpoint record is written to the output file. The current statistics are part of the checkpoint record. To insure the checkpoint record is physically written, a dummy checkpoint is also written to output. Additionally a message containing the ID of the checkpoint record is written to SYSLOG.

10. Processing continues at step 6 until end of file is encountered.

11. At end of file, the statistics table totals are written, the output file(s) is closed, and the program returns control to DL/I.

Restart processing, after module DLZURGU0 receives control from DL/I, is as follows:

1. Steps 1 - 4 of "normal processing" are performed.

2. The restart (RESTART) input file is opened. This is either the output1 (HDUNLD1) or output2 (HDUNLD2) file from the previously terminated job execution.

3. A message is issued to SYSLOG requesting the checkpoint record number (ID) at which to restart. The number is validated.

4. All records, including the requested checkpoint record, of the RESTART file are copied to the output file(s).

5. A Get Unique (GU) call is issued for the checkpointed root segment to establish positioning. If the RBA is available for the root segment, it is placed in the SSA with an internal "*T" command code; otherwise the segment's key is placed in the SSA and an internal "*C" (key retrieve) command code call is issued. The statistics table is initialized with the checkpointed statistics record.

6. Steps 6 - 11 of "normal processing" are performed.

DESCRIPTION OF MODULE - DLZRGUM0

DLZRGUM0 is a read-only module containing all messages issued by the module DLZURGU0.

SIZE OF MODULE - DLZURGU0

This module contains approximately 32,500 bytes, including I/O areas.

CONTROL BLOCKS - DLZURGU0

- Output record containing segment prefix
- SSA for GU call by RBA
- SSA for GU call by key
- Output table record
- Checkpoint record.

INTERFACES - DLZURGU0

This module interfaces with DL/I through the DL/I language interface module DLZLI000 at entry point CBLTDLI.

ERROR CODES AND HANDLING - DLZURGU0

All errors are indicated by error messages.

HD REORGANIZATION RELOAD UTILITY - DLZURGL0

The HD reorganization reload utility (DLZURGL0) is loaded under DL/I control as an application program. It reloads a data base under control of DL/I. Input to the module consists of a sequential dump data set of logical records created by the HD reorganization unload utility (DLZURGU0). A logical record consists of a segment prefix and a segment.

During the reload, a message is issued each time a checkpoint record is encountered (approximately every 5000 segments). This message is the same in content and format as that issued during unload when the checkpoint record was created, and identifies the checkpoint by number. If the reload facility fails, a restart capability called 'Reload Restart' allows restarting from a checkpoint record.

After module DLZURGL0 receives control from DL/I initialization, processing is as follows:

1. The PCB address is saved, and a GSCD call is issued to obtain the PST address.
2. The input device type is determined and the data set is opened.
3. If restarting, obtain checkpoint restart number from operator and locate checkpoint record. The data base is then positioned (GU call) and the end of data is found (GN calls).

4. An input record is read (segment), and a DL/I call list is constructed.

5. A DL/I Insert (ISRT) call is issued for the segment.

6. After all segments have been processed, the last statistics table record is read and a comparative statistics report is written.

7. The input data set is closed, and the program returns control to DL/I.

DESCRIPTION OF MODULE - DLZRGLMO

DIZRGLMO is a read-only module containing all messages issued by the module DLZURGLC.

BLOCKS AND TABLES

Input record

SIZE OF MODULE - DLZURGLO

This module contains approximately 16,800 bytes, including I/O areas.

INTERFACES - DLZURGLO

This module interfaces with the DL/I routines through the DL/I language interface module DLZLI000 at entry point CBLTDLI.

ERROR CODES AND HANDLING - DLZURGLO

All error conditions are indicated by error messages.

LOGICAL RELATIONSHIP RESOLUTION

DATA BASE PREREORGANIZATION - DLZURPRO

The purpose of this module is to examine input control cards provided by the user, and, based upon the information contained in DL/I control blocks, to generate a control data set for use by other programs concerned with the resolution of logical and index relationships.

The input control cards for this program indicate the names of data bases that a user wishes to initially load or to reorganize. The control blocks for each segment of each data base listed on an input control card are examined. For each logical relationship in which a segment participates, a prefix resolution check is performed. This check consists of generating a bit map reflecting the prefix fields involved in the logical relationship, and then checking the bit map against a table that indicates the fields which must be resolved for the types of data bases in which the logical parent and the logical child reside. For purposes of the prefix resolution check, the type of data base is considered to mean an initially loaded data base, a reorganized data base, or another data base (not reorganized or loaded, but logically related to a data base that is reorganized or loaded). If the bit map and the table entry match yields a nonzero value, prefix fields must be resolved in either or both the logical parent and logical child.

If prefix fields must be resolved, a control list entry is built for the logical parent and/or the logical child. This control list entry indicates the fields to be resolved, the work data set record format options to use, etc.

After generating the control list, the data bases to be scanned, loaded, or reorganized are listed. The scan list is punched if requested. The control list is then written to the control data set.

CONTROL BLOCKS - DLZURPRO

- Control file consisting of one or more records, each with a pointer to the next block of control file and an area containing one or more control list entries.
- List entry.
- Secondary list entry.

Interfaces - DFSURPRO

The interface with the reorganization message module (DLZURGMO) is through the tables provided in that module. See the description of that module for table format.

The interface with batch initialization to load the required blocks dynamically is accomplished with the DLZBLKLD macro.

Error Codes and Handling - DLZURPRO

This program audits all input control cards and verifies the consistency of DL/I control blocks. Any errors encountered cause one or more messages to be generated. Refer to the IMS/VS Messages and Codes Reference Manual for details.

ABENDs - DLZURPRO

None

DATA BASE SCAN - DLZURGSO

This module searches one or more data bases for all segments that are involved in logical relationships. For each such segment, DLZURGSO generates one or more output records, depending upon the relationships in which that segment is involved. The output work data set of this program serves as one of the inputs to the prefix resolution utility.

This program scans data bases as indicated either by scan control cards or by the control data set generated by the prereorganization program. If scan control cards are present, they are checked for consistency with the DL/I control blocks. Data base scanning is done by segment type for HDAM and HIDAM data bases. If scan control cards are provided for segments in an HDAM or a HIDAM data base, work data set records are generated only for those segments listed on scan control cards.

After the segments are read into core, control is passed to the work data set generator module (DLZDSEHO). DLZDSEHO generates any necessary output work data set records based upon information contained in the control data set. It then returns control to this program (DLZURGSO).

Interfaces - DLZURGSO

Module DLZURGSO interfaces with the reorganization message module (DLZURGMO) through the tables provided in that module. See the description of that module for table format.

The interface with the work data set generator module (DLZDSEHO) is as described in the documentation for that module.

The interface with the buffer handler module (DLZDBHOO) is as described in the documentation for that module. The buffer handler module is used to directly access records in a data base.

The interface with batch initialization to load the required blocks needed for processing is accomplished with the DLZBLKLD macro.

Error Codes and Handling - DLZURGSO

This program audits all input control cards and verifies the consistency of DL/I control blocks with the control data set. Any errors encountered cause one or more messages to be generated. Refer to the *DL/I DOS/VS Operator's Reference Manual and Messages and Codes*.

ABENDs - DLZURGSO

If an input card is read with "ABEND" in columns 1-5, a dump (PDUMP) will be taken if an error condition is detected. This should always be done on a rerun of this utility if an APAR is to be submitted because of an error return code.

WORK FILE GENERATOR - DLZDSEHO

This module generates the work file records that are required to resolve logical and/or index relationships after one or more data bases have been initially loaded or reorganized. This program is used by the HD reload (DLZURGLO) and scan (DLZURGSO) utility programs provided by DL/I DOS/VS. It is also called automatically by internal DL/I modules (DLZDDLEO and DLZDXMTO) when a data base is initially loaded by a user-written program.

The general operation of this program consists of creating one or more work file records for each segment that is initially loaded, reloaded, or scanned, if that segment is involved in at least one logical or index relationship. The work file records reflect the new location of each segment and, if the data base is being reloaded, its old location. Each work file record also contains related information that indicates the data bases and segments involved in the logical or index relationship described by the record, their old pointer values, etc.

This program generates all work file records that are used as input by the data base prefix resolution module (DLZURG10). The format of each output record generated by this program (DLZDSEHO) is as described for input of the data base prefix resolution module (DLZURG10).

This module contains a CSECT which is also used by scan (DLZURGSO) and index maintenance (DLZDXMTO) to open the work file DTF. Within this routine is a subroutine (FINDDTF) which is also used by scan to determine the correct DTF (disk or tape) to use for a given file depending on the assignment for it.

DLZDSEHO is loaded by batch initialization when the PROCOPT is 'load' or when HD reload or scan are to be executed. The primary entry point address is found in SCDDSEHO. The DL/I termination routine will close the work data set.

Interfaces - DLZDSEHO

The first seven fullwords of the CSECT contain information to be used by the modules which interface with DLZDSEHO. These words concern the work data set and entry points or addresses needed by scan (DLZURGSO).

Displ. from Entry Point DLZDSEHO	Contents
-28	Base address of this module
-24	Address of LPLCSV - information needed by scan
-20	Address of TEST - entry point when called by scan
-16	Address of FINDDTF - a subroutine used by scan
-12	Address of OPENWORK - entry point of routine to open WORKFIL file
-8	Address of work area available to build output record
-4	Address of opened work file DTF. If this field is zero, the file is not open.

When invoked during initial data base load or during data base reorganization, the following interface is used:

Entry Point

DLZBEGIN (Address found in SCDDSEHO)

Register Contents

R1 - PST
R13 - Save area
R14 - Return address
R15 - Entry point address

Control Blocks

JCBPRESF - Operation type (FUNCASRT or FUNCISRT)
PSTWRK1 - SDB address

Exit

Return to calling program with a return code in register 15. The values are:

0 (X'0')	=	Successful completion
4 (X'4')	=	WORKFIL could not be opened (IGN was specified). This is not an error condition if the user does not wish to create a work file.
8 (X'8')	=	Sort field size exceeded
12 (X'C')	=	GETVIS error occurred
16 (X'10')	=	Invalid DL/I control blocks
20 (X'14')	=	Length of PCB key feedback area is zero
24 (X'18')	=	I/O error occurred on WORKFIL or CONTROL data set.
28 (X'1C')	=	CONTROL or WORKFIL data set could not be opened (invalid or unassigned device)

When the OPENWORK routine is called by scan (DLZURGS0) or index maintenance (DLZDXMT0), the following interface is used:

Entry Point

OPENWORK

Register Contents

R13 - Caller's save area address
R14 - Return address
R15 - Entry point address.

Exit

All registers are restored to entry condition. Return is made to the address in R14 plus the displacement 0 if an unknown or invalid device is specified or 4 if WORKFIL is successfully opened.

When invoked during a data base scan, the following interface is used:

Entry Point

TEST

Register Contents

R3	-	Location for prefix parameter list area for segment just read
R5	-	Secondary list entry
R6	-	PSDB
R7	-	SDB
R9	-	PCB
R10	-	PST
R11	-	Location of DTF for work data set (must be open)
R12	-	Base address for DLZDSEHO
R13	-	Save area for use by DLZDSEHO
R15	-	Entry point TEST

Control Blocks

PSTWRK1	Byte 0	-	Operation type (FUNCIHPS)
	Byte 1-3		SDB address

Exit

Return to calling program with return code in register 15 as for entry point DLZBEGIN.

When the FINDDTF routine is invoked by scan, the following interface is used:

Entry Point

FINDDTF

Register Contents

R0	-	System logical unit number in hex
R2	-	Address of disk DTF
R3	-	Address of tape DTF (or 0, if not an option)
R13	-	Caller's save area address
R14	-	Return address
R15	-	Entry point of FINDDTF

Exit

Register 15 - address of chosen DTF
All other registers are restored to entry conditions.
Return is made to the address in R14 plus the displacement
0 if an unknown or invalid device specified or
4 if successful completion. When error return to R14+0 is
made, R15 is zero if IGN was specified, or nonzero otherwise.

DATA BASE REORGANIZATION MESSAGE - DLZURGM0

This module contains messages used by the following utilities: preorganization (DLZURPR0), scan (DLZURGS0), prefix resolution (DLZURG10), and prefix update (DLZURGP0).
The module consists of the two tables defined below.

Control Blocks - DLZURGM0

- MESSAGE LENGTH AND OFFSET TABLE

  One 4-byte table entry exists for each message. Each 4-byte entry contains the message length and offset.

- MESSAGE TABLE

  One variable-length entry is present for each message. Each entry contains the text of the message. The length is found in the message length and offset table.

Interfaces - DLZURGM0

This module contains messages that are used by the following modules:

DLZURPR0	(prereorganization)
DLZURGS0	(scan)
DLZURG10	(prefix resolution)
DLZURGP0	(prefix update)

ABENDs - DLZURGM0

Not applicable

DATA BASE PREFIX RESOLUTION - DLZURG10

This module accumulates the information generated on work data sets during the load and/or reorganization of one or more data bases. It produces an output data set that contains the prefix information needed to complete the logical and/or index relationships defined for the data base(s).

Operation of this program centers around at least one and possibly two, phases of the DOS Sort/Merge program execution. In the first phase, the Sort/Merge program is attached by this program. All work data set records generated during data base initial load, reorganization, or scan are input to the sort program. All input records are sorted such that all work data set records associated with a given occurrence of a logical parent follow the work data set record describing that logical parent. On exit from the first phase sort, this program has available the information needed to resolve the logical parent pointers that reside in logical children, the counter field and logical child pointers in the logical parent, and the logical twin pointers in the logical child (if a sequence field is carried in the work data set record). Any unnecessary records are dropped before entering the second sort phase. The second phase of this program is not executed if only index relationships need to be resolved.

In the second phase of this program, the Sort/Merge program is again attached. In this sort execution, the output records from phase one are sorted according to data base name and physical location within data base of each segment that must be updated by the prefix update program. On exit from the second phase sort, any remaining logical twin pointers are resolved, and further accumulation of logical parent counter fields is performed. Any records not actually necessary to update a data base are dropped at this time.

This program uses the control data set generated by the prereorganization program to govern its general operation. That is, the lists in the control data set indicate prefix fields to be resolved, etc.

Control Blocks - DLZURG10

- Input work file record - DLZURWF1
- Output work file record - DLZURWF3

Error Codes and Handling - DLZURG10

This program audits all input work data set records for consistency and for correspondence with the control list provided with the control data set. Any errors encountered cause one or more messages to be generated. Refer to the DL/I DOS/VS Operator's Reference Manual and Messages and Codes

DATA BASE PREFIX UPDATE - DLZURGPO

This module reads the input work data set provided by the data base prefix resolution module, reads the data base segment indicated by each record of the input work data set, and applies the prefix changes indicated by the work data set record to the segment read into main storage.

The input work data set is sorted in data base and segment physical location order by the data base prefix resolution module (DFSURG10) to afford most efficient update of each data base by this module. The format of each input record read by this program is as described for output of the data base prefix resolution module.

One or more input work data set records may be present for each segment that participates in logical or index relationships. The records are successively applied to the prefix of each segment affected, and the updated segment is written to its storage device. The prefix fields updated by this program include the logical parent, logical twin, and logical child pointer fields, and the counter fields associated with logical parents.

Interfaces - DLZURGPO

The interface with the reorganization message module (DLZURGMO) is through the tables provided in that module. See the description of that module for table format.

The interface with the language interface module (DLZLI000) is as described in the documentation for that module. The DL/I "ISRT" and "GHU" calls are issued by this program.

The interface with the buffer handler module (DLZDBH00) is as described in the documentation for that module. The buffer handler module is used to directly access records in a data base.

The interface with batch initialization to load the required blocks dynamically is accomplished with the DLZBLKLD macro.

Error Codes and Handling - DLZURGPO

This program audits all input work data set records for consistency with data base control blocks, checks all data base update operations, and checks input control card information. Any errors encountered cause one or more messages to be generated. Refer to the *DL/I DOS/VS Operator's Reference Manual and Messages and Codes*.

CHAPTER 11. THE DL/I PARTITION AND CONTROL BLOCK RELATIONSHIP

The purpose of this chapter is to describe the DL/I partition in a batch environment and to illustrate the relationship of the control blocks described in the preceding chapter.

THE DL/I BATCH PARTITION

Figure 11.1 is a map of main storage in the DL/I DOS/VS batch partition. Storage is allocated from the bottom or lowest storage address to the top or highest storage address of the partition. The eight areas in the DL/I batch partition are as follows:

- Area 1 contains the DL/I nucleus. The SCD is the first control block in the nucleus and contains the DL/I copyright information. This block also contains the entry point address for every module in the DL/I system. The PST prefix, PST, and PSB directory (PDIR) are in this area. There is one entry in the PSB directory (PDIR).

- Area 2 contains the DL/I program request handler, DLZPRHB0, which is loaded during DL/I initialization.

- Area 3 contains the PSB intent list and one DMB directory (DDIR) for each DMB referenced by the PSB. These blocks are created dynamically during DL/I initialization.

- Area 4 contains the PSB and DMBs loaded from the DOS/VS Core Image Library by the DL/I Batch Initialization module.

- Area 5 contains the DL/I buffer pool control blocks. These blocks are created dynamically and are aligned on a 2K page boundary. There are one buffer pool prefix, one subpool information table for each subpool specified, one DMB subpool directory entry for each DMB, and 32 buffer prefixes for each subpool specified.

- Area 6 contains the DL/I I/O buffers which comprise the buffer pool. There are 32 buffers for each subpool specified. The buffer pool is also aligned on a 2K page boundary.

- Area 7 contains the DL/I logic modules.

- Area 8 contains the user batch application program.

DL/I CONTROL BLOCK RELATIONSHIP

The purpose of this section is to show the relationships of the various DL/I control blocks and provide a means by which the user can quickly find his way to these control blocks. The following discussion references Figure 11.2.

The SCD is the major control block in the DL/I system. It is located at the beginning of the DL/I nucleus. The SCD contains DL/I copyright information, entry point addresses of the DL/I logic module, and address pointers to the major DL/I control blocks.

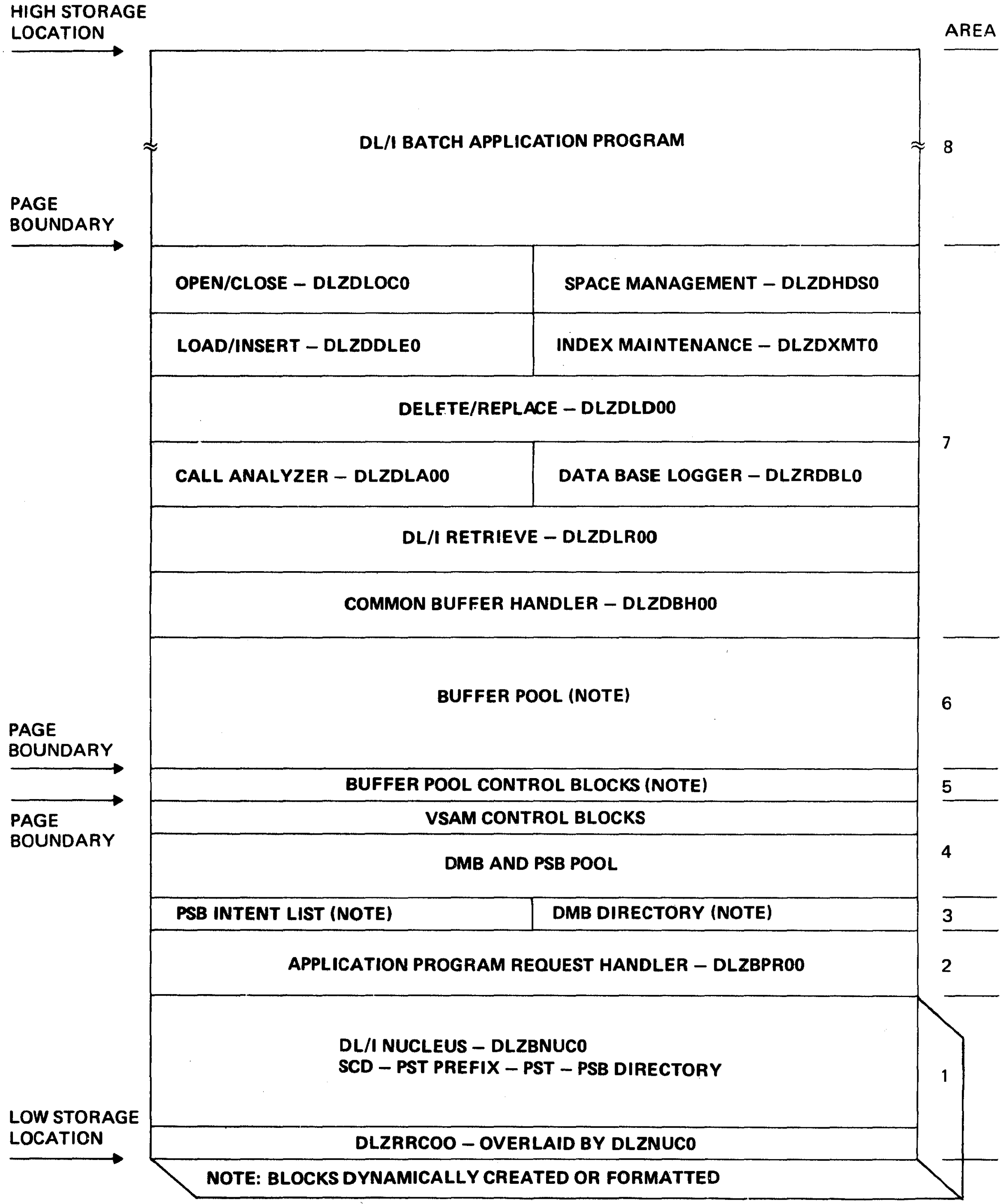


Figure 11.1. Map of Main Storage in the DL/I Batch Partition

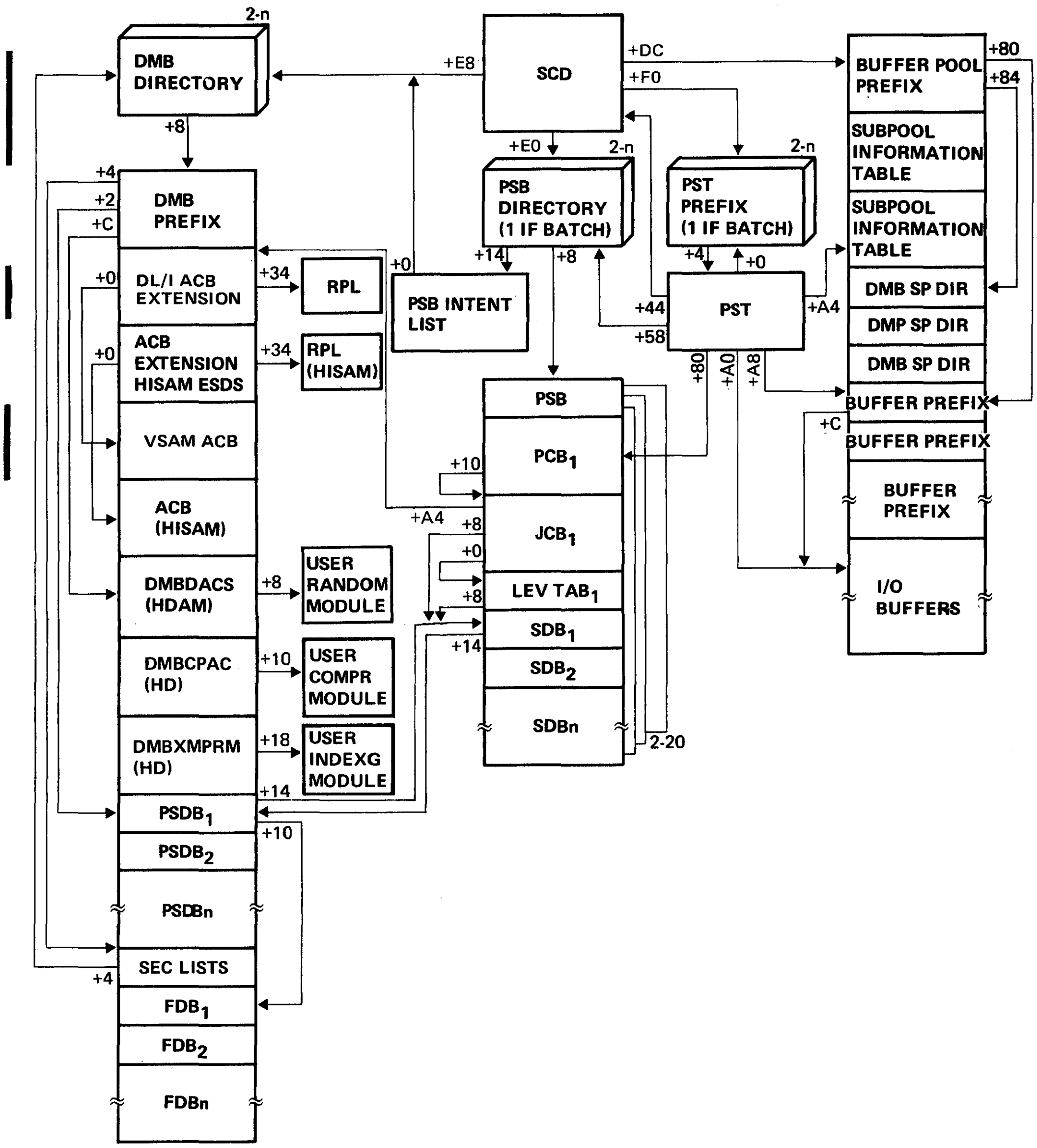


Figure 11.2 DL/I Control Block Relationships

The following address pointers, shown below in parentheses, can be obtained from the SCD:

- The buffer pool prefix (X'DC'), which is the first block of the buffer pool control blocks
- The first PSB directory (X'E0') from which the first PSB and PSB intent list may be obtained. In a batch system, there is only one PSB directory.
- The first DMB directory (X'E8'). There is one DMB directory for each DMB referenced by the PCBs.
- The first PST prefix (X'F0') from which the first PST may be obtained. There is only one PST prefix in a batch system.

The PST, including the PST prefix, functionally relates the control blocks for DL/I and represents the batch or CICS/DOS/VS - DL/I online task being served by DL/I. The PST is the dispatching block and is the only parameter passed when calling another module. The address of the PST is contained in the PST prefix (X'4'). The following address pointers are available in the PST:

- Caller's (user program) parameter list (X'48')
- SCD (X'44')
- PSB directory (X'58') for the task
- PCB currently being accessed (X'80')
- I/O buffer (X'A0') to be used for the data base call (used by the buffer handler)
- Subpool information table (X'A4') assigned to the data base (used by the buffer handler)
- Buffer prefix (X'A8') which points to the I/O buffer containing the segment for the call (used by the buffer handler)

There is one PSB directory entry and one PSB for each program that may be accessed by DL/I. In a CICS/DOS/VS - DL/I online environment, the maximum is 255; in batch, there can be only one. The PSB directory contains address pointers to the PSB (X'8') and the PSB intent list (X'14').

The PSB intent list is a variable-length control block and contains an entry for each DMB referenced by the PSB. Each entry contains the address of the DMB (X'0').

The PSB contains prefix information and one or more PCBs. Each PCB contains one JCB, one level table, and one or more SDBs. The PCB points to the JCB (X'10'). The JCB contains working storage for the program's use of that data base and points to the level table (X'0'). The JCB also points to the SDB (X'8') for the root segment and the VSAM ACB (X'A4') for the data base (KSDS ACB if HISAM). The level table contains working storage for DL/I to store its positioning data for each level of the data base. The level table points to the current level SDB (X'8').

The SDB describes the user's logical use of the sensitive segment. There is one SDB for each segment to which the user is sensitive. Each SDB points to the corresponding PSDB (X'14') in the DMB.

The DMB directory contains the address of the DMB (X'8'). Each DMB contains a prefix, one ACB extension for each data set in the DMB (two if HISAM), one PSDB for each physical segment type, and one FDB for each field defined for a segment. In addition, there is one direct algorithm communication table (DMBDACS) if HDAM is used, and secondary list entries if HIDAM or HDAM with index or original relationships is used.

The DMB prefix contains:

- A two-byte relative offset to the first PSDB (X'2')
- A two-byte relative offset to the end of the last PSDB+1 (X'4'), which is either the first secondary list entry (HIDAM) or the first FDB
- A four-byte pointer to DMBDACS if HDAM (X'C')

The ACB extension contains information about the data set as well as an address pointer to the VSAM ACB and RPL for the data set.

Each PSDB contains:

- A pointer to the first FDB (X'10') for the segment
- A pointer to the SDB (X'14') for the active PCB which is sensitive to this segment type. If more than one PCB is sensitive to this segment type, the address of the SDB for the next PCB is contained in the active PSDB (X'10').

The DMBDACS contains the address of the user's randomizing routine (X'08'); most of the secondary list entries point to the DMB directory (X'04') for the described index or logically related data base.

The following items may be obtained from the buffer pool prefix:

- The first subpool information table (immediately following the buffer pool prefix (X'88'))
- An address pointer to the first buffer prefix (X'80')
- An address pointer to the first DMB subpool directory entry (X'84')

The buffer prefix contains an address pointer to the I/O buffer (X'C') which it references.

DATA MANAGEMENT BLOCK - DMB

A skeleton DMB is created at DBDGEN time as part of the DBD. The DMB consists primarily of a description of each segment contained in the data base and information concerning the physical data base description. This is contained in ACB extensions or, in the case of HSAM, in DTFs. The DBD is loaded into storage by the DL/I application control blocks creation and maintenance utility, which builds the DMB from the DBD created by DBDGEN. The DMB is then cataloged and link edited into a core image library. The DMB is moved to its execution-time location in the DMB pool by the application control blocks load and relocate module (DLZDBLM0).

The DMB consists of a prefix section containing primarily offsets to subsections of the DMB: a prefix section (ACB extension) for each data set (DMBACBXT), a direct algorithm communication table (DMBDACS) if HDAM, the description of each segment (DMBPSDB), a secondary list to describe indexed fields or logical relationships (DMBSEC), field description blocks (FDBs) describing each field in each segment, a compression CSECT (DMBCPAC) for each compressable segment, and an index parameters CSECT (DMBXMPRM) for each secondary index exit routine.

For a HISAM organization, there is a pair of ACB extensions for each data base, for a KSDS ACB and an ESDS ACB. If the data base contains only root segments, only the KSDS ACB extension is created.

The ACBs are generated when the blocks are loaded for execution by DLZDBLM0 from the information in the ACB extensions.

• DMB LAYOUT

GENERAL STRUCTURE:

PREFIX SECTION OF DMB
DMBACBXT SECTION FOR EACH DATA SET (1 OR 2) (DTF IF SHSAM OR HSAM FOR INPUT AND OUTPUT FILE)
DMBLACS SECTION IF HDAM
DMBCPAC - COMPRESSION CSECT
DMBXMPRM - INDEX MAINTENANCE PARAMETERS
DMBPSDB SECTION FOR EACH SEGMENT
DMBSEC SECTION IF INDEXING OR LOGICAL RELATIONSHIPS
FDB SECTION FOR EACH SEGMENT FIELD
TAPE OR DASD I/O MODULE IF SHSAM OR HSAM

TAPE OR DASD I/O MODULE IF SHSAM OR HSAM

This module is included by the application control blocks creation and maintenance utility.

DL/I BUFFER POOL CONTROL BLOCKS

The DL/I buffer pool control blocks provide the control information to manage the entire buffer pool for the DL/I task. The buffer pool control blocks are as follows:

- Buffer Pool Control Block Prefix - DLZBFPL - This control block contains the statistics and other control information for the entire buffer pool.

- Subpool Information Table - DLZSBIF - This control block contains information for a specific subpool, including the size of the buffers in the subpool. There is one subpool information table for each subpool allocated.

- DMB Subpool Directory - This control block contains a one-byte subpool number relative to zero for each HDAM or HIDAM data base allocated. The DMB sequence number is used as an offset into the DMB directory and allows a DMB to be identified with a specific subpool.

- Buffer Prefix Control Block - DLZBFFR - This control block contains key information about the contents of a specific buffer in a subpool. There is one buffer prefix control block for each buffer. Each subpool contains 32 buffers.

GENERAL STRUCTURE:

```
┌──────────────────────────────────────────┐
|                                          |
|          BUFFER POOL CONTROL             |
|                                          |
|        BLOCK PREFIX - DLZBFPL            |
|                                          |
|------------------------------------------|
|                                          |
|       SUBPOOL INFORMATION TABLE          |
|                                          |
|       DLZSBIF - ONE PER SUBPOOL          |
|                                          |
|------------------------------------------|
|                                          |
.                                          .
.                                          .
.                                          .
|                                          |
|------------------------------------------|
|                                          |
|         DMB SUBPOOL DIRECTORY            |
|                                          |
|           ONE PER DATA SET               |
|                                          |
|------------------------------------------|
|                                          |
.                                          .
.                                          .
.                                          .
|                                          |
|------------------------------------------|
|        BUFFER PREFIX - DLZBFFR           |
|                                          |
|    ONE PER BUFFER (32 PER SUBPOOL)       |
|                                          |
|------------------------------------------|
|                                          |
.                                          .
.                                          .
.                                          .
|                                          |
|------------------------------------------|
|                                          |
|              I/O BUFFERS                 |
|                                          |
|            (32 PER SUBPOOL)              |
|                                          |
.                                          .
.                                          .
.                                          .
|                                          |
└──────────────────────────────────────────┘
```

PROGRAM SPECIFICATION BLOCK - PSB

A PSB must be created for every user program which will run under DL/I control. The PSB is created in "skeleton" format (principally PCBs only) by PSBGEN. The PSB must be cataloged and link edited into the Core Image Library. The PSB is loaded into main storage by the DL/I Application Control Blocks Creation and Maintenance Utility program and expanded and completed by this utility. The expansion is performed by segment definition in the DBD representing the associated data base. The expanded PSB is link edited into the Core Image Library. The PSB is moved to its execution-time location in the PSB pool by the application control blocks load and relocate module (DLZDBLM0). In expanded final format, the PSB consists of the following parts in the order specified:

1. PSB prefix - of which the most important part is the variable-length PSB list: the address list of the PCBs in the PSB.

2. A variable number of data base PCBs. For each data base PCB there is a JCB (job control block) consisting of the following parts:

   a. JCB prefix

   b. DSG (data set group) table. This table contains entries describing the data bases specifically used for this PCB. There are entries for all logically connected data bases, all primary HIDAM indexes, and a secondary index if used as the processing sequence.

   c. Level table. This table provides memory of the last DL/I CALL.

   d. SDB (segment description block). This block contains an entry for each segment to which the user has declared himself sensitive in the PCB. The SDB entry describes the sensitive segment.

   e. Work area for index maintenance, variable-length segment support, or miscellaneous function. These are allocated only when required.

• PSB LAYOUT

GENERAL STRUCTURE:

	PSB PREFIX (INCL PCB POINTERS)
DB-PCB1 GROUP	DBPCB DOPE VECTORS (IF PL/I)
	DBPCB
	JCB PREFIX
	JCB DSG ENTRIES
	JCB LEVEL TABLE
	JCB SDB ENTRIES
ADDITIONAL USER PCB GROUPS	AS SHOWN ABOVE ADDITIONAL USER PCB GROUPS
	INDEX MAINTENANCE PCB (IF REQUIRED)
	PSB WORK AREA

INDEX MAINTENANCE PCB

Required if any user PCB directly or indirectly refers to an index data base.

PSB WORK AREAS

These areas are of variable length depending on the requirements of the PCBs.

This section provides field descriptions for each control block used in the DL/I system. The control blocks are documented in alphabetical order as listed under Chapter 12 in the Contents of this publication.

ACB EXTENSION - ACBXT

The ACB extension is described in Chapter 11 as part of the general structure and description of the DMB. The information in ACBXT is repeated for each data set in the DMB.

Hex	Dec	Name	Ln	Description
0	0	DMBACBAD	4	Address of ACB for data set
4	4	DMBCINV	2	Control interval size
6	6	DMBACBDL	1	Delta cylinders to scan
7	7	DMBACBAP	1	Number of root anchor points in each control interval (HDAM)
8	8	DMBACBMX	2	Length of largest segment stored in data set
A	10	DMBACBMN	2	Length of smallest segment stored in data set
C	12	DMBECB	4	VSAM ACB event control block (ECB) used by buffer handler (DLZDBH00)
10	16	DMBHIBLK	4	Highest control interval RBA
14	20	DMBRBASN	4	RBA of last logical record assigned (HISAM) or relative block number of last control interval assigned (HD)
18	24	DMBRLBLK	4	Relative block number of last control interval written (HD); unused for HISAM
1C	28	DMBCICYL	2	Number of control intervals per cylinder
1E	30	DMBCITRK	1	Number of control intervals per track
1F	31	DMBKEYLE	1	Key length of KSDS
20	32	DMBRKP	2	Relative key position (offset)
22	34	DMBOFLGS	1	Open flags

Name	EQU	Meaning
DMBIGNOR	X'40'	IGN was specified for WORKFIL on load
DMBNUSE	X'20'	ACB does not have resolved secondary index entries, WORKFIL must be used
DMBOPEN	X'10'	The corresponding ACB is open

				DMBPUTKY X'08' Simulate not load mode to VSAM (used by XMT)
23	35	DMBVSFLG	1	Flags

Name	EQU	Meaning
DMBCISPL	X'80'	Control interval split has occurred
DMBPSEQ	X'10'	Sequence processing is possible for this KSDS

24	36		4	Reserved
28	40	DMBVSBFR	2	Number of buffers to be used
2A	42	DMBLRECL	2	Data set record length
2C	44	DMBBFACT	2	Data set blocking factor
2E	46	DMBINDO	1	Permanent indicators

Name	EQU	Meaning
DMBKEY	X'80'	File contains keys (simple HISAM and HISAM)
DMBWCHK	X'08'	Write check option

30	48	DMBSPLCT	4	Control interval split count
34	52	DMBACBRP	4	Address of RPL for this ACB
38	56	DMBACBLC	2	Log count (HISAM only)
3A	58	Reserved	2	Reserved for future use
3C	60	DMBACBNM	8	Data set name as in ACB
44	68	DMBACBEX	4	Address of exit list for corresponding ACB
48	72		8	Reserved

Note: For HSAM DMBs, the ACB extension is eight bytes in length as follows:

0	0	DMBDTFIN	4	Address of HSAM input DTF
4	4	DMBDTFOT	4	Address of HSAM output DTF

APPLICATION CONTROL TABLE - ACT

See Chapter 5 for the layout and field descriptions of the ACT.

BPC INFORMATION TABLE - DLZTWAB

This information is in the BPC task transaction work area.

Hex	Dec	Name	Ln	Description
0	0	TWAMPSFG	1	BPC flag byte

Name	EQU	Meaning
TWABPCOK	X'80'	BPC abnormal termination

Hex	Dec	Name	Ln	Description
1	1	TWAMPCPT	3	Address of MPC partition table
4	4	TWABPCID	1	Batch partition identifier (F1, F2,...)

BUFFER POOL CONTROL BLOCK PREFIX - BFPL

The BFPL is described in Chapter 11 as part of the general structure and description of DL/I buffer pool control blocks.

Hex	Dec	Name	Ln	Description
0	0	BFPLID	4	Pool ID - BFPL
4	4		12	Reserved
10	16	BFPLRQCT	4	Number of block requests received
14	20	BFPLINPL	4	Number of requests satisfied from pool
18	24	BFPLRDCT	4	Number of read requests issued
1C	28	BFPLALTR	4	Number of alterations received
20	32	BFPLOSWT	4	Number of writes issued
24	36	BFPLBKWT	4	Number of blocks written
28	40	BFPLNWBK	4	New blocks created in pool
2C	44	BFPLCHWT	4	Number of chained writes issued
30	48	BFPLCHBK	4	Number of blocks written on write chain
34	52	BFPLISTL	4	Number of retrieves by key calls
38	56	BFPLIGET	4	Number of GN calls received
3C	60	BFPLWERR	1	Number of permanent write error buffers in pool
3D	61	BFPLWERT	1	Largest number of write error buffers ever in pool
3E	62	BFPLCOUT	1	Number of rows/cols. in matrix currently in use

3F	63	BFPLROCO	1	Mask showing available rows/cols. in matrix
40	64	BFPLNQW1	4	ENQ/DEQ work area 1 - pointer to DLZSBIF

Name	EQU	Meaning
BFPLEXCI	X'00'	Switch
BFPLPECI	X'04'	Switch
BFPLSUPO	X'08'	Switch

44	68	BFPLNQW2	4	ENQ/DEQ work area 2

Name	EQU	Meaning
BFPLSW00	X'00'	Switch
BFPLSW80	X'80'	Switch

48	72	BFPLINMA	16	Interlock detection matrix
58	88	BFPLINW1	16	Interlock detection work area 1
68	104	BFPLINW2	16	Interlock detection work area 2
78	120	BFPIPSI1	4	Field 1 for pseudo interlock
7C	124	BFPLPSIF	2	PST prefix number of first waiting for matrix
7E	126	BFPIPSIL	2	PST prefix number of last waiting for matrix
80	128	BFPLPRAD	4	Address of beginning of buffer prefix area
84	132	BFPLSUBD	4	Address of beginning of DMB subpool directory
88	136	BFPLSUIN	0	Start of subpool information tables

BUFFER PREFIX - BFFR

The BFFR is described in Chapter 11 as part of the general structure and description of the DL/I buffer pool control blocks.

Hex	Dec	Name	Ln	Description
0	0	BFFRCIID	0	Control interval identifier (CI ID)
0	0	BFFRCIRB	4	Control interval RBA
4	4	BFFRDMB	2	DMB number
6	6	BFFRDCB	1	ACB number
7	7	BFFRSW	1	Switches

Name	Bit	Meaning
BFFRWCH	0	Buffer on write chain
BFFRWRT	1	Buffer being written
BFFRREAD	2	Buffer being read
BFFRMT	3	Buffer empty
BFFRWERR	5	Buffer has permanent write error
BFFREXNQ	6	Existing CI ID enqueued
BFFRPNNQ	7	Pending CI ID enqueued

Hex	Dec	Name	Ln	Description
8	8	BFFRPSTF	1	PST pointer of controlling task
9	9	BFFRPSTL	1	PST pointer of last task in chain of waiting tasks
A	10	BFFRLOGU	2	Log count
C	12	BFFRUSCT	1	Use count
D	13	BFFRADDR	3	Address of buffer
10	16	BFFRUSID	2	ID of user who altered this buffer
12	18	BFFRWCFW	1	Next lower buffer on write chain
13	19	BFFRWCBW	1	Next higher buffer on write chain
14	20	BFFRNCID	7	New CI ID
14	20	BFFRNCII	4	New CI RBA
18	24	BFFRNDMB	2	New DMB number
1A	26	BFFRNACB	1	New ACB number
1B	27	BFFRSW1	1	Switches

Bit	Meaning
0	Buffer must not be reused
1-7	Not used

Hex	Dec	Name	Ln	Description
1C	28	BFFRNPSF	2	PST pointer of task which ENQ'd on new CI ID and is first in chain
1E	30	BFFRNPSL	2	PST pointer of task which ENQ'd on new CI ID and is last in chain

COMPRESSION CSECT - CPAC

The CPAC is described in Chapter 11 as part of the general structure and description of the DMB. There is one entry for each compressible segment in the DMB.

Hex	Dec	Name	Ln	Description
0	0	DMBCPCNM	8	Segment name
8	8	DMBCPCSG	8	Compression routine name
10	16	DMBCPEP	4	Entry pcint address of compression routine
14	20	DMBCPFLG	1	Flag

Name	EQU	Meaning
DMBCPSEQ	X'08'	Segment has a sequence field
DMBCPVLR	X'04'	Segment is variable length
DMBCPNIT	X'01'	Initialization and termination processing required

Hex	Dec	Name	Ln	Description
15	21	DMBCPSQF	1	Length of key field -1
16	22	DMBCPSQL	2	Offset tc sequence field
18	24	DMBCPSGL	2	Maximum segment length
1A	26	DMBCPLNG	2	Length cf CSECT
1C	28	DMBCPRES	4	Used by batch initialization

DBD GENERATION CONTROL BLOCK OUTPUT - DBDGEN

The data base description block (DBD) is the result
of each data base generation.

- DIAGRAM OF DBDGEN CONTROL BLOCK OUTPUT

GENERAL STRUCTURE:

```
┌──────────────────────────────────────────────┐
│                 DIRECTORY                    │
│                                              │
│──────────────────────────────────────────────│
│                 PREFIX                       │
│                                              │
│──────────────────────────────────────────────│
│                 DMANTAB                      │
│                                              │
│──────────────────────────────────────────────│
│         ACB EXTENSION (SAME AS DMB)          │
│         (If HSAM or SSAM, DTFs)              │
│                                              │
│──────────────────────────────────────────────│
│                 SEGTAB                       │
│                                              │
│──────────────────────────────────────────────│
│                 FLDTAB                       │
│                                              │
│──────────────────────────────────────────────│
│                 EXTDBD                       │
│                                              │
│──────────────────────────────────────────────│
│                 LCHILD                       │
│                                              │
│──────────────────────────────────────────────│
│                 SORTAB                       │
│                                              │
│──────────────────────────────────────────────│
│                 INDXTAB                      │
│                                              │
│──────────────────────────────────────────────│
│                  DACT                        │
│              (Same as DMB)                   │
│                                              │
│──────────────────────────────────────────────│
│         COMPRESSION EXIT CSECTS              │
│             (same as DMB)                    │
│                                              │
│──────────────────────────────────────────────│
│         INDEX EXIT CSECTS                    │
│             (same as DMB)                    │
│                                              │
└──────────────────────────────────────────────┘
```

1. DIRECTORY LAYOUT

Hex	Dec	Name	Ln	Description
0	0	AMODLEV	1	Release level (X'00'=1.0, X'11'=1.1)
1	1	APREFIX	3	Address of PREFIX
4	4	ASEGTAB	4	Address of SEGTAB
8	8	AFLDTAB	4	Address of FLDTAB
C	12	ALCHILD	4	Address of LCHILD
10	16	AEXTDBD	4	Address of EXTDBD
14	20	ASORTAB	4	Address of SORTAB
18	24	ARMVTAB	4	Address of DMBDACS
1C	28	AINDXTAB	4	Address of INDXTAB
20	32	ADSGCB	4	Address of ACB extension

2. PREFIX LAYOUT

Hex	Dec	Name	Ln	Description
0	0	PREDBDNM	8	DBD name
8	8	PRENOLEV	2	Number of levels in data base
A	10	PRENOSEG	2	Number of segments
C	12	PREACCES	1	Organization

Name	EQU	Meaning
PRESHIS	X'01'	Simple HISAM
PREISAM1	X'02'	HISAM
PRESSAM	X'04'	Simple HSAM
PREHSAM	X'05'	HSAM
PREHD	X'06'	HDAM
PREHI	X'07'	HIDAM
PRENDEX	X'08'	INDEX

Hex	Dec	Name	Ln	Description
D	13	PRENODSG	1	Number of data sets
E	14	PRENODBD	2	Number of externally referenced data bases
10	16	PRERNDM	8	Randomizing algorithm name
18	24	PRENOLCH	2	Number of logical children
1A	26	PREAP	2	Number of root anchor points
1C	28	DBDPFRBN	4	Maximum relative block number (HD)
20	32	DBDPFBYT	4	Maximum bytes in prime area (HD)

3. DMANTAB LAYOUT

Hex	Dec	Name	Ln	Description
0	0	PREDD1	8	Input or prime filename
8	8	PREDEV1	4	Device type
C	12	PREID	1	Data set group ID
D	13	PRENSGA	1	Number of segments in data set
E	14	PREDELTA	2	Delta scan cylinders (HD)
10	16	PRELSL	2	Length cf longest segment plus prefix
12	18	PRESSL	2	Length of shortest segment plus prefix
14	20	PRELKL	2	Length cf longest key
16	22	PRESKL	2	Length of shortest key
18	24	PREIRECL	2	Prime/input record length
1A	26	PREBLKSZ	2	Prime/input block size (control interval)
1C	28	PREOLREC	2	ESDS/output record length
1E	30	PRECBLKS	2	ESDS/output block size (control interval)
20	32	PREDD2	8	ESDS/output filename

4. ACB EXTENSION

See "ACB Extension - ACBXT".

5. SEGTAB LAYOUT

One of these tables exists for each segment.

Hex	Dec	Name	Ln	Description
0	0	SEGDSNO	1	Segment data set number
1	1	SEGPHYCD	1	Segment code
2	2	SEGPARPC	1	Parent segment code
3	3	SEGLEVEL	1	Segment level
4	4	SEGNCLCH	1	Number of logical children
5	5	SEGNOFLD	1	Number cf fields
6	6	SEGLENG	2	Segment data length (maximum length if variable length segment)
8	8	SEGFREQ	4	Reserved
C	12	SEGSEGNM	8	Segment name

14	20	SEGFLG1	1		Prefix pointer flag
				EQU	**Meaning**
				X'80'	Counter
				X'40'	Physical twin forward
				X'20'	Physical twin backward
				X'10'	Physical parent
				X'08'	Logical twin forward
				X'04'	Logical twin backward
				X'02'	Logical parent
				X'01'	Hierarchical
15	21	SEGFLG2	1		Segment update rules
				EQU	**Meaning**
					Insert rule
				X'C0'	Logical
				X'80'	Physical
				X'40'	Virtual
					Delete rule
				X'30'	Logical
				X'20'	Physical
				X'10'	Virtual
					Replace rule
				X'0C'	Logical
				X'08'	Physical
				X'04'	Virtual
					Physical location of inserts, when no key field
				X'03'	Here (current position)
				X'02'	First
				X'01'	Last
16	22	SEGFLG3	1		
				X'08'	Parent has backward pointers to this segment
17	23	SEGFLG4	1		Number of physical children pointed to directly by this segment
18	24	SEGLCHLD	4		Offset to first LCHILD entry
1C	28	DBDSSN	2		Number of source segments
1E	30	DBDSSOFF	2		Offset to first source segment
20	32	SEGFLDTB	4		Offset to first FLDTAB
24	36	DBDSPFSZ	2		Segment prefix size
26	38	SEGLENGV	2		Minimum segment length (0 if fixed length)
28	40	Reserved	4		Reserved

2C	44	SEGPACOP	1	VL-Compression options

Name	EQU	Meaning
SEGCPRT	X'08'	Segment has compression routine
SEGTYPVL	X'04'	Segment is variable length
SEGPACIT	X'01'	Initialization exit requested for compression routine

2D	45	SEGPACRT	3	Address of compression table

6. FLDTAB LAYOUT

Hex	Dec	Name	Ln	Description
0	0	FLDNAME	8	Field name
8	8	FLDSTART	2	Start position offset
A	10	FLDFLAG	1	

EQU	Meaning
X'80'	Last field for a SEGTAB
X'40'	Sequence field
X'20'	Multiple sequence fields
X'10'	Special FDB
	Field type
X'01'	Hexadecimal
X'02'	Packed
X'03'	Character

Hex	Dec	Name	Ln	Description
B	11	FLDLEN	1	Field length
C	12	FLDSNAME	8	Source field name
14	20	FLDSEGTB	4	Pointer to SEGTAB entry

7. EXTDBD LAYOUT

Hex	Dec	Name	Ln	Description
0	0	EXTIBNM	8	Externally referenced data base name
8	8	EXTRSVD	4	Reserved

8. LCHDTAB LAYOUT

Hex	Dec	Name	Ln	Description
0	0	LCHSEGNM	8	Segment name
8	8	LCHCODE	1	

Bit	Meaning
0=0	LCHEDBD address is a EXTDBD entry
0=1	LCHEDBD address is a SEGTAB entry
1-7	Reserved

9	9	LCHEDBD	3	Offset to EXTDBD or SEGTAB entry
C	12	LCHFLAG	1	

EQU	Meaning
X'80'	Last entry for a SEGTAB
X'40'	Reserved
X'20'	INDEX entry
X'10'	Reserved
X'08'	LP definition
X'04'	INDEX pointer
X'02'	SNGL pointer
X'01'	DBLE pointer

D	13	LCHIBYTE	1	Reserved
E	14	LCHPRDSG	2	Offset to paired segment
10	16	LCHFLDNM	8	Indexed field name

9. SORTAB LAYOUT

Hex	Dec	Name	Ln	Description
0	0	DBDSCRNM	8	Source segment name
8	8	DBDSSFLG	1	Source segment flag - reserved
9	9	DBDSSDBO	3	Offset to data base entry

10. INDXTAB

See "Secondary List - SEC (Codes 64, 44, 40, 24, 20, 04)".

11. DACT

See "Direct Algorithm Communication Table - DACT".

12. COMPRESSION EXIT CSECTS

See "Compression CSECT - CPAC".

DATA MANAGEMENT BLOCK (PREFIX) - DMB

The DMB prefix is described in Chapter 11 as part of the general structure and description of the data management block.

Hex	Dec	Name	Ln	Description
0	0	DMBSIZE	2	DMB size (Zero bit on (X'80') means version 1.1 or later)
2	2	DMBLENTB	2	Offset from DMB to first PSDB (DMBPSDB)
4	4	DMBSECTB	2	Offset from DMB to end of last PSDB+1

6	6	DMBORG	1	DMB organization

Name	EQU	Meaning
DMBSHIS	X'01'	Simple HISAM
DMBISAM1	X'02'	HISAM
DMBSSAM	X'04'	Simple HSAM
DMBHSAM	X'05'	HSAM
DMBHD	X'06'	HDAM
DMBHI	X'07'	HIDAM
DMBNDEX	X'08'	Index data base

7	7	DMBLDDCB	1	ACB number minus one of sequential data set used to write index records on data base load
8	8	DMBPDATA	2	Length of system data in this index data base (protected)
A	10		2	Reserved
C	12	DMBDALGR	4	Address of direct algorithm communication table if HDAM (DMBDACS)

DATA MANAGEMENT BLOCK DIRECTORY - DDIR

The DMB directory contains an entry for every data base known to DL/I, that is, an entry for every DMB (Data Management Block) that can be accessed under DL/I control. The DMB directory is part of the DL/I nucleus and is created at DL/I system definition time for online processing. The start address of the directory (SCDDLIDM) and entry length (SCDDLIDL) are contained in the SCD.

Hex	Dec	Name	Ln	Description
0	0	DDIRSYM	8	DMB symbolic name converted from DBD name
8	8	DDIRADDR	4	Storage address of DMB
C	12	DDIRCNT	1	Count of number of users scheduled to this DMB
D	13	DDIRDMBL	3	Total length of DMB
10	16	DDIRNUMB	2	DMB number of this DMB
12	18	DDIRCODE	1	DMB code byte 1

Name	EQU	Meaning
DDIRSECL	X'80'	Security locked
DDIROPEN	X'40'	At least one ACB is opened
DDIRINOP	X'20'	DMB to be opened during online initialization
DDIRKBRQ	X'10'	Buffer pool space required for this KSDS
DDIRWAIT	X'08'	Waiting for zero DDIRCNT
DDIRNOSC	X'04'	Do not schedule this DMB
	X'02'	Reserved
DDIRNOUP	X'01'	Do not schedule updates

13	19	DDIRCOD2	1	DMB code byte 2

DDIRNDBM	X'80'	DMB not present in core image library
DDIRNRAN	X'40'	Requested HDAM randomizing module not in core image library
DDIRHSAM	X'20'	This is HSAM DMB
DDIREXCL	X'10'	DMB being used exclusively
DDIREXSD	X'08'	Exclusive control required for scheduling
DDIR1GRP	X'04'	This DMB is first in shared index
DDIRGRP	X'02'	This DMB belongs to shared index
DDIRBAD	X'01'	DMB initialization unsuccessful

14	20	DDIRVSRT	1	R15 VSAM return code
15	21	DDIRPPST	3	PPST address if DMB used exclusively

DIRECT ALGORITHM COMMUNICATION TABLE - DACT

The DACT is described in Chapter 11 as part of the general structure and description of the data management block (DMB).

Hex	Dec	Name	Ln	Description
0	0	DMBDANME	8	Name of address conversion algorithm load module
8	8	DMBDAKL	1	Root key length less one
9	9	DMBDAEP	3	Entry point to conversion module
C	12	DMBDASZE	2	Size of the conversion module
E	14	DMBDARAP	2	Number of root anchor points per block
10	16	DMBDABLK	4	Number of highest block directly addressable
14	20	DMBDABYM	4	Maximum number of bytes per root before overflow outside of directly addressable area
18	24	DMBDABYC	4	Current number of bytes consecutively inserted or loaded under root
1C	28	DMBDACP	4	Result of last address conversion

DLZXCB1 DSECT - XCB1

This DSECT describes the fields that follow the XECB used by the batch partition in MPS.

Hex	Dec	Name	Ln	Description
0	0	XCB1ECB	4	Batch partition XECB
4	4	XCB1PSB	4	Pointer to PSB name
8	8	XCB1PROG	4	Pointer to program name
C	12	XCB1CNT	4	Address of count of number of parameters
10	16	XCB1PARM	72	Addresses of parameters on call (maximum of 18)
58	88	XCB1FLAG	1	Flag byte

Name	EQU	Meaning
XCB1EOJ	X'01'	EOJ Indicator
XCB1PLI	X'02'	On if PL/I

Hex	Dec	Name	Ln	Description
59	89	XCB1RES	3	Reserved

FIELD DESCRIPTION BLOCK - FDB

The FDB is described in Chapter 11 as part of the general structure and description of the data management block (DMB).

This describes a "normal" field.

Hex	Dec	Name	Ln	Description
0	0	FDBSYMBL	8	Symbolic name
8	8	FDBOFFST	2	Field offset from segment beginning
A	10	FDBDCENF	1	Code byte

Name	EQU	Meaning
FDBLAST	X'80'	Last FDB for this segment
FDBKEY	X'40'	This is segment's sequence field
FDBEQOK	X'20'	Duplicate sequence fields allowed
FDBSPEC	X'10'	Special FDB (XDFLD, /CK, or /SX)
FDBPACK	X'02'	Field is packed decimal
FDBHEX	X'01'	Field is hexadecimal
FDBCHAR	X'03'	Field is character

Hex	Dec	Name	Ln	Description
B	11	FDBFLENG	1	Executable field length

This describes /CK system-related field.

Hex	Dec	Name	Ln	Description
0	0	FDBSYSNM	3	'/CK'
3	3		5	Remainder of name

8	8	FDBOFFCK	2	Offset from beginning of concatenated key
A	10	FDBSYSLN	2	Bits 0-3 = 0001 (FDBSPEC) Bits 4-15 = executable length

This describes the XDFLD.

0	0	FDBXDNM	8	Field symbolic name
8	8	FDBXDSEC	2	Offset to secondary list for this index
A	10	FDBXDFLG	1	Flag

Name	EQU	Meaning
FDBXDLST	X'80'	Last FDB
FDBXDSPC	X'10'	Special FDB
FDBXDSSQ	X'04'	SUBSEQ present
FDBXDEQ	X'01'	Index target segment same as index source segment

B	11	FDBXDLEN	1	Length of search field

INDEX MAINTENANCE PARAMETERS - XMPRM

This CSECT is part of the data management block (DMB). One entry exists for each secondary index exit routine. See Chapter 11 for the general structure and description of the DMB.

Hex	Dec	Name	Ln	Description
0	0	DMBXMSGN	8	Name of indexed segment
8	8	DMBXMXDN	8	Name of XDFLD
10	16	DMBXMXNM	8	Name of user exit routine
18	24	DMBXMXEP	4	Entry point address of user exit routine
1C	28	DMBXMPLN	2	Length of this entry
1E	30		2	Reserved
20	32	DMBXMRES	4	Used by initialization

JOB CONTROL BLOCK - JCB

The JCB is described in Chapter 11 as part of the general structure and description of the program specification block (PSB).

Hex	Dec	Name	Ln	Description
0	0	JCBLEVTE	4	Address of level table
4	4	JCBLEVND	4	Address of end of level table + 1
8	8	JCBSDB1	4	Address of 1st SDB entry (root's)
C	12	JCBSDBND	4	Address of end of SDBs +1

10	16	JCBTRACE	14	Prior 7 functions followed by last byte of return code

The following calls require a PCB and will be traced in JCBTRACE. Any call not requiring a PCB is not put in the trace table. However, the function code appears in JCBPREVF or JCBPREVR.

Code	Meaning
01	'GHU' GET HOLD UNIQUE 'GU' GET UNIQUE
03	'GHN' GET HOLD NEXT 'GN' GET NEXT
04	'GHNP' GET HOLD NEXT WITHIN PARENT 'GNP' GET NEXT WITHIN PARENT
21	'REPL' REPLACE
22	'DLET' DELETE
41	'ISRT' INSERT

The following calls may or may not require a PCB

90	PURG call

The following calls do not require a PCB

A0	UNLD call
A1	GSCD call
A3	TERM call

1E	30	JCBPREVF	1	Prior function
1F	31	JCBPREVR	1	Prior return code rightmost byte
20	32	JCBLEV1C	4	Address of 1st LEVTAB entry in call
24	36	JCBSIZE	2	PCB + JCB size
26	38	JCBMKYL	2	Maximum length of key feedback area
28	40	JCBRES1	4	Call characteristics set by analyzer
2C	44	JCBRES2	4	Reserved for temporary use by action modules
30	48	JCBRES3	4	Reserved for temporary use by action modules
34	52	JCBRES4	4	Reserved for temporary use by action modules
38	56	JCBRES5	4	Reserved for temporary use by action modules
3C	60	JCBCODE	1	Reserved

3D	61	JCBORGN	1	Open bit and composite organization of all SDBs in JCB

Name	EQU	Meaning
JCBOPEN	X'80'	Open done for all data sets in JCB
JCBORGRI	X'44'	Organization is root of index
JCBORGHD	X'20'	Organization is HDAM
JCBORGHI	X'10'	Organization is HIDAM
JCBORGHSH	X'05'	Organization is simple HISAM
JCBORGH1	X'04'	Organization is HISAM
JCBORGHS	X'02'	Organization is HSAM
JCBORGSS	X'01'	Organization is simple HSAM

3E	62	JCBRWKF	1	Retrieve's working function
3F	63	JCBPRESF	1	Present coded function
40	64	JCBLVT	1	Work switch for retrieve
41	65	JCBLVC	1	Work switch for retrieve
42	66	JCBPC	1	Current segment being sought by retrieve
43	67	JCBPOP	1	Parent level for within parent calls
44	68	JCBSTOR1	4	Reserved, ISRT's use across I/O or calls
48	72	JCBSTOR2	4	Reserved, ISRT's use across I/O or calls
4C	76	JCBSTOR3	4	Reserved, ISRT's use across I/O or calls
50	80	JCBSTOR4	4	Reserved, RETR's use across I/O or calls
54	84	JCBSTOR5	4	Reserved, RETR's use across I/O or calls
58	88	JCBSTOR6	4	Reserved, RETR's use across I/O or calls
5C	92	JCBSTOR7	4	Work area for retrieve
60	96	JCBSTOR8	4	Work area for retrieve
64	100	JCBWKR0	4	Work area for action modules
68	104	JCBWKR1	4	Work area for action modules
6C	108	JCBWKR2	4	Work area for action modules
70	112	JCBWKR3	4	Work area for action modules
74	116	JCBWKR4	4	Work area for action modules
78	120	JCBWKR5	4	Work area for action modules
7C	124	JCBWKR6	4	Work area for action modules
80	128	JCBWKR7	4	Work area for action modules
84	132	JCBWKR8	4	Work area for action modules

88	136	JCBWKR9	4	Work area for action modules
8C	140	JCBWKR10	4	Work area action modules
90	144	JCBWKR11	4	work area for action modules
94	148	JCBWKR12	4	Work area for action modules
98	152	JCBWKR13	4	Work for action modules
9C	156	JCBWKR14	4	Work area for action modules
A0	160	JCBWKR15	4	Work area for action modules

JCB_DSG_ENTRIES

A4	164	JCBICBA	4	Address of ACB extension for this data set (KSDS ACB if HISAM)
A8	168	JCBDMBNO	2	DMB number for this DSG
AA	170	JCBICBNO	1	ACB number of ACB in DMB (KSDS ACB number if HISAM)
AB	171	JCBINDA	1	JCB indicators data set organization

Name	EQU	Meaning
JCBDSOLS	X'80'	This is last file in JCB
JCBDSORI	X'44'	File group is root in index
JCBDSOHD	X'20'	File group is HDAM
JCBDSOHI	X'10'	File group is HIDAM
JCBDSOH1	X'04'	File group is HISAM or simple HISAM
JCBDSOHS	X'02'	File group is HSAM or simple HSAM
JCBDSOUP	X'01'	Reserved

AC	172	JCBIRECA	4	KSDS record address relative record number
B0	174	Reserved	2	Reserved for future use
B2	176	JCBINDB	1	JCB indicators
B3	177	JCBINDC	1	JCB indicators

Name	EQU	Meaning
JCBBLDEL	X'80'	This file group belongs to DELETE/REPLACE. Note: This is an additional DSG generated if PCB has delete sensitivity. This is always the first DSG in the JCB.

B4	178	JCBINDG	1	JCB indicators for variable length

Name	EQU	Meaning
JCBPREM	X'80'	Segment prefix has been moved to work area
JCBDATX	X'40'	Segment has been completely expanded

Name	EQU	Meaning
JCBVL	X'08'	The VL routine has been entered for this segment
JCBRETD	X'04'	Data return call
JCBCOMMD	X'02'	Path call

Hex	Dec	Name	Ln	Description
B5	179	Reserved	3	Reserved
B8	180	JCBNOSAM	4	Reserved
BC	184	JCBLROOT	4	RBA of current root

LEVEL TABLE - LEVTAB

The level table is described in Chapter 11 as part of the general structure and description of the program specification block (PSB).

Hex	Dec	Name	Ln	Description
0	0	LEVLEV	1	Level number
1	1	LEVPC	1	Current segment physical code
2	2	LEVSEGOF	2	Segment's physical code offset from start of record (relative offset to segment from start of buffer)
4	4	LEVTTR	4	Relative byte address (RBA)
8	8	LEVSDB	4	SDB entry address for current physical code in this entry
C	12	LEVF1	1	Code byte

Name	EQU	Meaning
LEVDLET	X'80'	Segment at this level newly deleted
LEVEMPTY	X'40'	This level table entry empty
LEVHELD	X'20'	Segment at this level in hold status
LEVHIER	X'10'	Segment at this level in hierarchic path
LEVDATA	X'08'	Segment at this level moved to user.
LEVPLAST	X'04'	Segment is last of type for parent
LEVPFRST	X'02'	Segment is first of type for parent
LEVLAST	X'01'	This is last level table for PCB

Hex	Dec	Name	Ln	Description
D	13	LEVF2	1	Code byte used by retrieve (DLZDLR00)
E	14	LEVUSEOF	2	Offset of segment in user I/O area (PSTUSER)
10	16	LEVNUPC	1	Physical code of requested segment

11	17	LEVF3	1	Code byte

Name	EQU	Meaning
LEVPSUDO	X'08'	This is pseudo SSA filling gap
LEVDATA1	X'04'	At least 1 member qualified on data
LEVKEY1	X'02'	Every set has at least one key field

12	18	LEVF4	1	Code byte

Name	EQU	Meaning
LEVCOMMT	X'80'	T command code - retrieve by direct address
LEVCOMMX	X'20'	X command code index maintenance internal call

13	19	LEVF5	1	Code byte

Name	EQU	Meaning
LEVCOMMU	X'80'	U command code - get unique at level
LEVCOMMF	X'20'	F command code - get first of segment type
LEVCOMML	X'10'	L command code - get last of segment type
LEVCOMMA	X'08'	A command code - any seg type this level
LEVCOMMD	X'04'	D command code - transfer data this level (path call)
LEVCOMMN	X'02'	N command code - do not replace this level

14	20	LEVMEMBR	1	One-byte switch for each member

Name	EQU	Meaning
LEVMEMEQ	X'80'	Operator has = sign
LEVMEMLT X'40' Operator has < sign LEVMEMGT X'20' Operator has > sign		Both > & < set if operator is not equal (¬=)
LEVMEMAC	X'08'	This member in use (unqualified if only bit)
LEVMEMKY	X'04'	Qualification is on key field
	X'03'	Both 6 and 7 set for right parenthesis

15	21		7	Same as above for other 7 members
1C	28	LEVNUSDB	4	This SSA's SDB address
20	32	LEVSSA	4	This SSA's left-parenthesis positional address

MPC PARTITION TABLE ENTRY - MPCPT

The MPC partition table is used to pass control information when processing batch partition application programs under multiple partition support (MPS). The MPC partition table is in the CICS/VS transaction work area.

Name	Length	Description
MPCPARTB	144 bytes	One 16-byte entry (see entry layout below) for each partition defined during system generation except the MPC partition. Delimiter is a full-word of X'FF'.
MPCECBLT	11 full-words	ECB pointer list. On entry for each: • Start partition XECB (DLZXCBN0) • Stop transaction XECB (DLZXCB00) • Stop partition XECB (DLZXCB01) • ABEND XECB on BPC attach failure (DLZXCBN3) Delimiter is a fullword of X'FF'.

MPCPT ENTRY LAYOUT

Hex	Dec	Name	Ln	Description
0	0	MPCDELIM	4	MPCPT delimiter field
0	0	MPCFLAG	1	MPC activity flags

Name	EQU	Meaning
MPCPACT	X'80'	Partition active indicator
MPCERR	X'40'	Error condition encountered on DL/I scheduling call, or BPC attach failure.
MPCTSTP	X'20'	Stop transaction indicator.
MPCPSTP	X'10'	Stop partition indicator.
MPCXECB	X'08'	XECBs deleted for this partition.
MPCSDEF	X'04'	Start XECB defined.
MPCADEF	X'02'	ABEND XECB defined.

Hex	Dec	Name	Ln	Description
1	1	MPCRC1	1	Error return code from TCAFCTR
2	2	MPCRC2	1	Error return code from TCSDLTR
3	3	MPCPID	1	Partition identifier (F1, F2, ...)
4	4	MPCTCA	4	Address of TCA
8	8	MPCSXECB	4	Address of stop partition XECB (DLZXCB01)
C	12	MPCAXECB	4	Address of ABEND XECB (DLZXCBN3)
10	16	MPCADBPC	4	Address of batch partition controller entry point
14	20	MPCPARML	4	Address of MPC - BPC parameter list
18	24		4	Reserved
-	-	MPCPTLN	28	Length of partition table entry

PARTITION SPECIFICATION TABLE - PST

One partition specification table (PST) exists for each task in an online or batch processing partition. All DL/I resources allocated to the task can be located through the PST. The PST also contains pointers to the task I/O area and any segments currently associated with the task.

Hex	Dec	Name	Ln	Description
0	0	PSTPREAD	4	Pointer to PST prefix for this PST
4	4	PSTDLIW0	4	Action
8	8	PSTDLIW1	4	Action
C	12	PSTDLIW2	4	Action
10	16	PSTDLIW3	4	Action
14	20	PSTDLIW4	4	Action
18	24	PSTDLIW5	4	Module
1C	28	PSTDLIW6	4	Module
20	32	PSTDLIW7	4	Module
24	36	PSTDLIW8	4	Module
28	40	PSTDLIW9	4	Module
2C	44	PSTDLIWA	4	Work
30	48	PSTDLIWB	4	Work
34	52	PSTDLIWC	4	Work
38	56	PSTDLIWD	4	Work
3C	60	PSTDLIWE	4	Words
40	64	PSTDLIWF	4	Words

* DL/I SECTION

Hex	Dec	Name	Ln	Description
44	68	PSTCODE1	1	Process code

Name	EQU	Meaning
PSTINTNT	X'40'	Cannot schedule - intent not satisfied
PSTSCHED	X'10'	Schedule function may be completed
PSTPRVWT	X'08'	Logger private wait indicator

Hex	Dec	Name	Ln	Description
45	69	PSTSCDAD	3	Address of SCD
48	72	PSTABIND	1	Task/system ABEND indicator

Name	EQU	Meaning
PSTERMSP	X'80'	PUT error message

		indicator
PSTSABND	X'20'	System ABEND message indicator
PSTTABND	X'10'	Task ABEND indicator

49	73	PSTIQPRM	3	Pointer to caller's PARM list
4C	76	PSTMI	1	Return segment indicator
4D	77	PSTUSER	3	User's segment I/O area address
50	80	PSTSEGL	4	Retrieved segment length
54	84	PSTSEG	4	Retrieved segment address
58	88	PSTPSB	4	PSB directory entry address

* DL/I USER TASK STATISTICS

5C	92	PSTACCT	0	Beginning of user task statistics
5C	92	PSTDGU	4	'GU' data base calls issued
60	96	PSTDGN	4	'GN' data base calls issued
64	100	PSTDGNP	4	'GNP' data base calls issued
68	104	PSTDGHU	4	'GHU' data base calls issued
6C	108	PSTDGHN	4	'GHN' data base calls issued
70	112	PSTDGHNP	4	'GHNP' data base calls issued
74	116	PAIDISRT	4	'ISRT' data base calls issued
78	120	PSTDDLET	4	'DLET' data base calls issued
7C	124	PSTDREPL	4	'REPL' data base calls issued

* DL/I ACTION MODULE SECTION

80	128	PSTDBPCB	4	Address of current or last-used PCB
84	132	PSTFNCTN	1	Caller's function to buffer handler, index maintenance, space management, or open/close

* CALL FUNCTION TO BUFFER HANDLER

Name	EQU	Meaning
PSTMSPUT	X'0E'	Insert record sequentially into KSDS
PSTPUTKY	X'0D'	Insert record by key into KSDS
PSTSTLBG	X'0C'	Get first record of a KSDS
PSTGETNX	X'0B'	Get next record from KSDS
PSTERASE	X'0A'	Erase a record in a KSDS
PSTSTLEQ	X'09;	Get a record from a KSDS (key equal or high)
PSTWRITE	X'08'	Write new record into HISAM ESDS
PSTPGUSR	X'07'	Purge all buffers marked

* BUFFER HANDLER AND SPACE MANAGEMENT

Hex	Dec	Name	Ln	Description
A0	160	PSTBFUSE	4	Address of buffer to be used
A4	164	PSTSUIN	4	Address of subpool information table to be used
A8	168	PSTPREAR	4	Address of buffer prefix used during this call
AC	172	PSTSUBNM	2	Subpool number used during this call
AE	174	PSTSWI	2	Work space
B0	176	PSTPOSEL	1	Counter for position of use chain element
B1	177	PSTMROCO	1	Number of rows/columns in matrix currently used by this task
B2	178	Reserved	2	Reserved for future use
B4	180	PSTSAVRE	40	Register save areas for internal use by buffer handler
DC	220	PSTRETRE	32	Register save areas for internal use by buffer handler
FC	252	PSTNUMRC	1	Number of blocks read this call
FD	253	PSTNUMWT	1	Number of writes issued this call
FE	254	PSTCLRWT	1	Buffer handler indicator

Name	EQU	Meaning
PSTIWAIT	X'80'	IWAIT issued during this call

Hex	Dec	Name	Ln	Description
FF	255	Reserved	1	Reserved for future use
100	256	PSTTSKID	4	Hashed task identification
104	260	Reserved	16	Reserved for future use

* PST WORK AREAS

Hex	Dec	Name	Ln	Description
114	276	PSTWRK1	4	Work space
118	280	PSTWRK2	4	. . for use by
11C	284	PSTWRK3	4	. . buffer handler
120	288	PSTWRK4	4	. . and data base logger

* DATA BASE LOG USAGE OF PSTWRK1, 2, 3, AND 4

Hex	Dec	Name	Ln	Description
114	276	PSTDBLFC	1	Function codes

Name	EQU	Meaning
DBLNDXC	X'80'	Index maintenance call
DBLCMC	X'00'	Chain maintenance call

				(Bits 1-3=0)
		DBLPHYI	X'40'	Insert call
		DBLPHYD	X'20'	Delete call
		DBLPHYR	X'10'	Replace call
		DBLOOPS	X'0A'	No data - end of user's call
		DBLLASTC	X'08'	Last change for this user's call
		DBLFSE1	X'00'	One FSE (Bit 1 or 2 on)
		DBLFSE2	X'04'	Two FSEs (Bit 1 or 2 on)
		DBLPHYRO	X'02'	Old copy of delete
		DBLNEWBL	X'01'	New block log call
115	277	PSTWRK1+1	3	PSDB address, if new block log call, count in low-order two bytes
118	280	PSTWRK2	4	Chain maintenance - old copy of chain counter
118	280	PSTWRK2-4	6 or 12	Insert or delete - offsets and new FSEs
124	292	PSTWRKT1	4	Work space preserved
128	296	PSTWRKT2	4	. . across
12C	300	PSTWRKT3	4	. . calls
130	304	PSTWRKT4	4	. . to the
134	308	PSTWRKT5	4	. . buffer handler
138	312	PSTWRKD1	4	Work space
13C	316	PSTWRKD2	4	. . for use by
140	320	PSTWRKD3	4	. . delete/replace
144	324	PSTWRKD4	4	..
148	328	PSTWRKD5	4	..
14C	332	PSTWRKD6	4	..
150	336	PSTWRKD7	4	..
154	340	PSTCURWA	4	Current work area
158	344	PSTDLTWA	4	Delete work area address
15C	348	PSTDLROM	84	Retrieve save and maintenance work area

* DATA BASE LOG SECTION

1B0	432	PSTLOGWA	4	Address of work area for log output
1B4	436	PSTLOGQ	4	Reserved
1B8	440	Reserved	8	Reserved for future use

* PARTITION/TASK INFORMATION

1C0	448	PSTPCPGM	8	Application program name
1C8	456	PSTPCPSB	8	PSB name
1D0	464	PSTPCT1	1	Partition/task information

Name	EQU	Meaning
PSTBATCH	X'80'	Task is batch Task is online if bit is off
PSTLODU	X'40'	DLZURGLO utility executing
PSTLODUH	X'20'	HD data base being loaded by DLZURGLO
PSTUDR	X'04'	Task is recovery utility
PSTULU	X'02'	Task is reorganization or logical relaticnship resolution utility

1D1	465	PSTPCT2	1	Program options/information

Name	EQU	Meaning
PSTCALI	X'02'	User's call list is implicit
PSTPLI	X'01'	User's program is PL/I

1D2	466	PSTERCD1	1	Error message code byte 1
1D3	467	PSTERCD2	1	Error message code byte 2
1D4	468	PSTERDT1	7	Error message data for ACB or DTF name
1DB	475	PSTERDT2	8	Variable error message data
1E3	483	PSTERIND	1	Error routine indicator byte

PSTDUMPI	X'80'	Issue DUMP after error message
PSTCANLI	X'40'	Issue CANCEL after error message

1E4	484	PSTLIPRM	72	Area to build user parameter list
22C	556	PSTPLIPR	8	PL/I region STXIT processor
234	564	Reserved	28	Reserved

* REGISTER SAVE AREAS FOR PROCESSING DL/I CALLS

250	592	PSTSV1	72	Save area 1
298	664	PSTSV2	72	Save area 2
2E0	736	PSTSV3	72	Save area 3
328	808	PSTSV4	72	Save area 4
370	880	PSTSV5	72	Save area 5
3B8	952	PSTSV6	72	Save area 6
400	1024	PSTSV7	72	Save area 7

PST PREFIX - PPST

The PST prefix contains data required for user task scheduling in a CICS online environment as well as buffer handler I/O and ENQ WAIT request information. The PST prefix is logically part of the PST. However, in order to operate more efficiently in a virtual storage environment, all PST prefixes (one for batch) are physically located in one contiguous area. This organization is more efficient.

Hex	Dec	Name	Ln	Description
0	0	PPSTCF	1	PST prefix chain forward byte
1	1	PPSTCB	1	PST prefix chain backward byte
2	2	PPSTECB	2	PST ECB
4	4	PPSTIND	1	Task schedule and dispatch indicato

Name	EQU	Meaning
PPSTIO	X'80'	Task waiting for I/O
PPSTSI	X'40'	Cannot schedule due to segment intent
PPSTTC	X'20'	Cannot schedule, over task count
PPSTBF	X'10'	Task enqueued by buffer handler
PPSTMPS	X'08"	Task is MPS
PPSTACT	X'04"	Task is active
PPSTMSDL	X'02'	MSP task scheduled by BPC
PPSTA	X'01'	Task is scheduled

Hex	Dec	Name	Ln	Description
5	5	PPSTCA	3	Address of PST
8	8	PPSTID	1	Task ID
9	9	PPSTTCA	3	Task's TCA address
		SECTION USED FOR BUFFER HANDLER ENQ/DEQ		
C	12	PPSTEXCI	4	ENQ/DEQ PTR for existing control interval
10	16	PPSTPECI	4	ENQ/DEQ PTR for pending control interval
14	20	PPSTSUPO	4	ENQ/DEQ PTR for subpool space
18	24	PPSTMATR	4	ENQ/DEQ PTR for matrix space
1C	28	PPSTCHAI	4	Chain field for ENQ/DEQ pending control interval

PHYSICAL SEGMENT DESCRIPTION BLOCK - PSDB

The PSDB is described in Chapter 11 as part of the general structure and description of the DMB.

Hex	Dec	Name	Ln	Description
0	0	DMBSC	1	Segment code

1	1	DMBPSC	1	Parent's segment code
2	2	DMBLEV	1	Segment level
3	3	DMBXNULL	1	Reserved
4	4	DMBPPFD	1	Pointer number in parent to first occurrence of segment for parent
5	5	DMBPPBK	1	Pointer number in parent to last occurrence of segment for parent
6	6	DMBDCB	1	ACB number
7	7	DMBPTR	1	Prefix flags

Name	EQU	Meaning
DMBCTR	X'80'	Counter present
DMBPTFD	X'40'	Segment has physical twin forward pointer
DMBPTBK	X'20'	Segment has physical twin backward pointer
DMBPP	X'10'	Segment has physical parent pointer
DMBLTFD	X'08'	Segment has logical twin forward pointer
DMBLTBK	X'04'	Segment has logical twin backward pointer
DMBLP	X'02'	Segment has logical parent pointer

8	8	DMBPRSZ	2	Prefix length of segment
A	10	DMBDL	2	Data length of segment
C	12	DMBISRT	1	Insert rules

Name	EQU	Meaning
DMBXPROT	X'80'	System data in index is protected
DMBIHERE	X'30'	If no key field, insert at current position
DMBILST	X'20'	If no key field, insert after existing segments
DMBIFST	X'10'	If no key field, insert before existing segments
DMBIRL	X'03'	Insert rule is logical
DMBIRP	X'02'	Insert rule is physical
DMBIRV	X'01'	Insert rule is virtual

D	13	DMBDLT	1	Delete and replace rules

Name	EQU	Meaning
DMBRRL	X'0C'	Replace rule is logical
DMBRRP	X'08'	Replace rule is physical
DMBRRV	X'04'	Replace rule is virtual
DMBDRL	X'03'	Delete rule is logical
DMBDRP	X'02'	Delete rule is physical
DMBDRV	X'01'	Delete rule is virtual

E	14	DMBCKL	2	Concatenated key length of parent of this segment
10	16	DMBUSE	0	Code byte

Name	Bit	Meaning
DMBEX	0	This PSDB in use exclusively
DMBUP	1	This PSDB in use for update
	2-7	Contain a count of read-only users

10	16	DMBFDBA	4	Address of FDBs for this segment
14	20	DMBFSDB	4	First SDB for this segment
18	24	DMBVLDFG	1	Variable length data flag

Name	EQU	Meaning
DMBCPT	X'08'	Segment has compression routine
DMBVLS	X'04'	Segment is variable length
DMBCPTIT	X'01'	Compression routine has initialization processing

19	25	DMBSCTAB	3	Address of segment compression CSECT
1C	28	DMBSGMN	2	If variable length, minimum length of segment
1E	30	DMBSGMX	2	If variable length, maximum length. If fixed length, actual length
20	32	DMBFLAG	0	Secondary list flag

Name	EQU	Meaning
DMBLPEX	X'40'	A logical parent exists
DMBLCEX	X'20'	One or more logical children exist
DMBNXEX	X'10'	One or more indexes exist
DMBXDEX	X'04'	Indexed segment exists

20	32	DMBIST	4	Address of secondary list for this segment

PROGRAM CONTROL BLOCK - PCB

The PCB is described in Chapter 11 as part of the general structure and description of the program specification block (PSB).

Hex	Dec	Name	Ln	Description
0	0	DBPCBDBD	8	DBD name
8	8	DBPCBLEV	2	Level of segment
A	10	DBPCBSTC	2	Status codes
C	12	DBPCBPRO	4	Processing options
10	16	DBPCBJCB	4	JCB address
14	20	DBPCBSFD	8	Segment name feedback

1C	28	DBPCBLKY	4	Length of key feedback area in bytes
20	32	DBPCBNSS	4	Number of sensitive segments in PCB
24	36	DBPCBKFD	Var	Key feedback area

PCB DOPE VECTOR - DPPCB

The PCB dope vector is described in Chapter 11 as part of the general structure and description of the program specification block (PSB).

Hex	Dec	Name	Ln	Description
0	0	DPPCBDBD	4	The address of the location that contains DBPCBDBD.
4	4	Maximum Length	2	Maximum length: Halfword binary number which specifies number of storage units allocated for the string; byte count if character, bit count if bit.
6	6	Current Length	2	Current length: Halfword binary number which specifies the number of storage units, within the maximum length, currently occupied by the string.
8	8	DPPCBLEV	4	The address of the location that contains DBPCBLEV.
C	12	Maximum Length	2	Maximum length: Halfword binary number which specifies number of storage units allocated for the string; byte count if character, bit count if bit.
E	14	Current Length	2	Current length: Halfword binary number which specifies the number of storage units, within the maximum length, currently occupied by the string.
10	16	DPPCBSTC	4	The address of the location that contains DBPCBSTC.
14	20	Maximum Length	2	Maximum length: Halfword binary number which specifies number of storage units allocated for the string; byte count if character, bit count if bit.
16	22	Current Length	2	Current length: Halfword binary number which specifies the number of storage units, within the maximum length, currently occupied by the string.
18	24	DPPCBPRO	4	The address of the location that contains DBPCBPRO.
1C	28	Maximum Length	2	Maximum length: Halfword binary number which specifies number of storage units allocated for the string; byte count if

				character, bit count if bit.
1E	30	Current Length	2	Current length; Halfword binary number which specifies the number of storage units, within the maximum length, currently occupied by the string.
20	32	DPPCBJCB	4	The address of the location that contains DBPCBJCB.
24	36	DPPCBSFD	4	The address of the location that contains DBPCBSFD.
28	40	Maximum Length	2	Maximum length: Halfword binary number which specifies the number of storage units allocated for the string; byte count if character, bit count if bit.
2A	42	Current Length	2	Current length: Halfword binary number which specifies the number of storage units, within the maximum length, currently occupied by the string.
2C	44	DPPCBLKY	4	The address of the location that contains DBPCBLKY.
30	48	DPPCPNSS	4	The address of the location that contains DBPCBNSS.
34	52	DPPCBKFD	4	The address of the location that contains DBPCBKFD.
38	56	Maximum Length	2	Maximum length: Halfword binary number which specifies the number of storage units allocated for the string; byte count if character, bit count if bit.
3A	58	Current Length	2	Current length: Halfword binary number which specifies the number of storage units, within the maximum length, currently occupied by the string.

PROGRAM SPECIFICATION BLOCK (PREFIX) - PSB

The PSB prefix is described in Chapter 11 as part of the general structure and description of the PSB.

Hex	Dec	Name	Ln	Description
0	0	PSBVMID	1	DOS DL/I Version ID (present only in core image library) 01 - Version 1.1
1	1	Reserved	3	Reserved
4	4	PSBXIOWK	4	Address of work area used by DL/I for indexes
8	8	PSBSEGWK	4	Address of work area used

				by DL/I for variable-length segments
C	12	PSBPST	4	Address of PST if PSB scheduled or active
10	16	PSBXPCB	4	Address of index maintenance PCB if required
14	20	PSBNDXWK	4	Address of work area used by DL/I
18	24	PSBIOAWK	4	Address of work area used by DL/I
1C	28	PSBINDEX	1	BPD index
1D	29	PSBCODE	1	Code -- A/P reuse, source language 0 - Reserved 1 - Reserved 2 - This is an Assembler or a COBOL program 3 - This is a PL/I program 4-7 - Reserved
1E	30	PSBSIZE	2	PSB Prefix size
20	32	Reserved	2	Reserved
22	34	PSBDBOFF	2	Offset from PSBLIST to first DB PCB (always zero)
24	36	PSBLIST	Var	Beginning of PCB list (a list of full-word pointers containing PCB addresses). Last PCB address word has byte 0, bit 0=1. List may contain a maximum of 64 addresses. For PL/I programs these pointers are to the dope Vector Tables in which the first word is a pointer to the associated PCB.

PSB DIRECTORY - PDIR

The PSB directory contains an entry for every program known to DL/I (one for batch), that is, an entry for every PSB (program specification block) that may run under DL/I control. The PSB directory is part of the DL/I nucleus and is created at DL/I system definition time for online processing. The start address of the directory (SCDDLIPS) and the entry length (SCDDLIPL) are contained in the SCD (system contents directory).

Hex	Dec	Name	Ln	Description
0	0	PDIRSYM	8	PSB symbolic name converted from PSB generation name
8	8	PDIRADDR	4	PSB storage address
C	12	PDIRPSBL	4	Length of PSB
10	16	PDIRZWA	2	Length of all work areas for this PSB
12	18	PDIRCODE	1	PSB code byte

Name	EQU	Meaning
PDIRUPD	X'80'	PSB is update sensitive
PDIREXC	X'40'	PSB requires exclusive control of data base
PDIRPLI	X'20'	PSB is for PL/I program
PDIRDUPL	X'10'	PSB is duplicate
PDIRDELT	X'02'	PSB is delete sensitive
PDIRTFAL	X'01'	Online task termination unsuccessful for this PSB

Hex	Dec	Name	Ln	Description
13	19	PDIROPTC	1	PSB scheduling options

Name	EQU	Meaning
PDIRNOSC	X'80'	Do not schedule this PSB
PDIRSCHD	X'40'	PSB is scheduled
PDIRNTNT	X'10'	PSB is waiting for segment intent
PDIREAD	X'01'	PSB initialization unsuccessful

Hex	Dec	Name	Ln	Description
14	20	PDIRSILA	4	PSB segment intent list entry address
18	24	Reserved	4	Reserved for future use

PSB GENERATION CONTROL BLOCK OUTPUT - PSGEN

1. PSB - PREFIX

Hex	Dec	Ln	Description
0	0	4	Address of SEGTAB
4	4	4	Address of SORTAB
8	8	4	Address of DBREFTAB
C	12	4	Reserved
10	16	4	PST address (prefix size)
14	20	12	Reserved
20	32	1	Reserved
21	33	1	PSB code
22	34	2	PSB prefix size
24	36	2	Reserved
26	38	2	Offset to first DB PCB address
28	40	Var	Address of PCB(s) (one 4-byte address for each PCB)

2. DB PCB

Hex	Dec	Ln	Description
PL/I dope vectors precede PCB if LANG=PL/I			
0	0	8	Data base name
8	8	2	Level feedback
A	10	2	Status code
C	12	4	Processing options
10	16	4	JCB address
14	20	8	Segment name feedback
1C	28	1	Position
1D	29	3	Key feedback length
20	32	2	Number of sensitive segments
22	34	2	Offset to first SENSEG
24	36	Var	Key feedback area

3. SEGTAB ENTRY

Hex	Dec	Ln	Decription
0	0	8	Segment name
8	8	4	Processing options
C	12	1	Flag
D	13	3	PCB address
10	16	2	Offset to parent segment
12	18	2	Offset to source segment

4. SORTAB ENTRY

Hex	Dec	Ln	Description
0	0	8	Segment name
8	8	1	Flag
9	9	3	Offset to data base entry

5. DBREFTAB ENTRY

Hex	Dec	Ln	Description
0	0	12	Data base name
C	12	4	Reserved

PSB INTENT LIST - PSIL

The PSB intent list is pointed to from the PSB directory and is a list of all the DMBs which may be used by that PSB (program).

Hex	Dec	Name	Ln	Description
0	0	PSILDMBN	8	DMB name for this list entry - overlaid during initialization
0	0	PSILDIRA	4	Address of DMB directory entry - resolved during initialization
4	4	PSILDIRN	2	DMB number of this DMB
6	6	Reserved	2	Reserved for future use
8	8	PSILNTNT	1	Segment intent descriptor byte

Name	EQU	Meaning
PSILDBEX	X'80'	PSB contains a PCB which requires exclusive control for this DMB
PSILDBUP	X'40'	PSB contains a PCB which is update sensitive
PSILBFRI	X'20'	Buffer pool space required for this KSDS

9	9	PSILLNGH	1	Length of this entry in list
A	10	PSILSEGD	Var	Segment intent bits. Two bits are used for each segment in the DMB and represent the PSB's sensitivity to each PSDB. Their meanings are:

Bit	Meaning
00	PSB not sensitive to the segment
01	PSB read only sensitive
10	PSB is update sensitive
11	PSB requests exclusive control (HISAM root insert)

The bits are allocated to segments in the following manner:

	BYTE 1								BYTE 2							
BIT	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
SEGMENT	4		3		2		1		8		7		6		5	

The second part of the segment intent bits is a mask. It is constructed from the segment intent bits of the first part. Part 2 has the same length as part 1.

SECONDARY LIST - SEC

The secondary list is described in Chapter 11 as part of the general structure and description of the DMB. The labels in SEC vary with the type of secondary index entry. See the field description in the next part of this section.

Hex	Dec	Name	Ln	Description
0	0	DMBSCDE	1	Code byte

Name	Bit	Meaning
DMBSND	X'80'	Last in secondary list
DMBNXXDS	X'64'	Secondary list describes index relationship as seen from index target segment. This list is not present if ISS = target - Code 64
DMBNXISS	X'60'	Secondary list describes index relationship as seen from index source segment (ISS) - Code 60
DMBINDXD	X'44'	Secondary list describes index target segment as seen from index pointer segment - Code 44
DMBEXTRN	X'40'	Secondary list describes user index exit routine - Code 40

DMBSUBSQ	X'24'	Secondary list describes index SUBSEQ field(s) - Code 24
DMBSOURC	X'20'	Secondary list describes index DDATA field(s) - Code 20
DMBSLCF	X'08'	Secondary list describes logical twin sequence field - Code 08
DMBSRCH	X'04'	Secondary list describes index search (SRCH) field(s) - Code 04
DMBSLC	X'02'	Secondary list describes a logical child - Code 02
DMBSLP	X'01'	Secondary list describes a logical parent - Code 01

* CODE_64_-_DESCRIBES_INDEX_FROM_INDEX_TARGET

1	1	DMBSKYLN	1	Executable length of key
2	2	DMBISSOF	2	Offset to Code 60 from start of ISS secondary list
4	4	DMBXNSSC	1	Segment code of index pointer segment
5	5	DMBXNSDB	3	DDIR address of index data base
8	8	DMBISSSC	0	Segment code of index source segment
8	8	DMBIPSDB	4	PSDB address of index source segment

Remaining 4 bytes are same as code 44

* CODE_60_-_DESCRIBES_INDEX_FROM_ISS

1	1	DMBSKYLN	1	Executable length of key
2	2	DMBSOFF	2	Offset to PSDB address pointer of index target segment
4	4	DMBXNSDB	1	Segment code of index pointer segment
5	5	DMBXNSDB	3	DDIR of index

Remaining 8 bytes are same as Code 44.

* CODE_44_-_DESCRIBES_INDEX_TARGET_SEGMENT

1	1	DMBSKYLN	1	Executable length of key
2	2	DMBSOFF	2	Offset to PSDB address pointer of index target segment
4	4	DMBXDSSC	0	Segment code of index target segment
4	4	DMBXDSDB	4	DDIR address of index target segment

8	8	DMBXDSC	0	Segment code of index target segment
8	8	DMBXPSDB	4	PSDB address of index target segment
C	12	DMBXDFLG	1	Code byte (FDBDCENF) from associated FDB
D	13	DMBXDPAD	1	Padding constant
E	14	DMBSYMOF	2	Offset to symbolic pointer indexing segment

* CODE_40 - DESCRIBES_INDEX_EXIT_ROUTINE

1	1	DMBSFLG1	1	Flag byte

Name	EQU	Meaning
DMBSNULL	X'01'	Null value present
DMBEXIT	X'02'	Exit routine present
DMBEXLOD	X'04'	Exit routine has been loaded

2	2		2	Reserved
4	4	DMBNBYTE	1	Null value. If SRCH field equals this value, bypass indexing
5	5	DMBXITAD	3	Address of index maintenance parameter CSECT
8	8		8	Reserved

* CODE_24 - DESCRIBES_SUBSEQ_FIELD

This entry is the same as Code 04.

* CODE_20 - DESCRIBES_DDATA_FIELD

This entry is the same as Code 04.

* CODE_08 - DESCRIBES_LOGICAL_TWIN_SEQUENCE_FIELD

1	1		1	Reserved
2	2	DMBSFNAM	8	FDB field name
A	10	DMBSFOFF	2	Offset to field in segment
C	12	DMBSFCEN	1	Code byte - same as FDBDCENF in FDB.
D	13	DMBSFLEN	1	Executable field length
E	14		2	Reserved

* CODE_04 - DESCRIBES_INDEX_SRCH_FIELDS

1	1	DMBFDFLG	5	Five one-byte flags associated the following FIB offsets
6	6	DMBFDOFF	10	Offset to FDB from first FDB of ISS if this slot is in use. Otherwise zero.

* CODE 02 - DESCRIBES LOGICAL CHILD

1	1		1	Insert rules (for logical twin chain). See DMBISRT.
2	2	DMBSLCFL	2	Number (position) of first and last logical child pointers in logical parent prefix

Remaining 12 bytes are same as Code 01.

* CODE 01 - DESCRIBES LOGICAL PARENT

1	1	DMBSFLG	1	Flag byte C'V' - key of logical parent is virtual
2	2	DMBSFD	2	Logical parent key length
4	4	DMBSECSC	1	Segment code of referenced segments
5	5	DMBSECDB	3	DDIR address of referenced data base
8	8	DMBSECNM	8	Segment name of referenced segment

SEGMENT DESCRIPTION BLOCK - SDB

The segment description block is described in Chapter 11 as part of the general structure and description of the PSB.

Hex	Dec	Name	Ln	Description
0	0	SDBSYM	8	SDB segment symbolic name
8	8	SDBLEVEL	1	Level of this segment (logical)
9	9	SDBORGN	1	Organization of data base containing segment

Name	EQU	Meaning
SDBORGRI	X'44'	This segment is root of index
SDBORGHD	X'20'	This segment is in an HDAM organization
SDBORGHI	X'10'	This segment is in a HIDAM organization
SDBORGSH	X'05'	This segment is in a simple HISAM organization
SDBORGH1	X'04'	This segment is in a HISAM organization
SDBORGHS	X'02'	This segment is in an HSAM organization
SDBORGSS	X'01'	This segment is in a simple HSAM organization.

A	10	SDBF3	1	Call sensitivity

Name	EQU	Meaning
SDBSENG	X'80'	Sensitivity is read only
SDBSENI	X'40'	Sensitivity is insert
SDBSENR	X'20'	Sensitivity is replace
SDBSEND	X'10;	Sensitivity is delete
SDBSENP	X'04'	Sensitivity is path only
SDBSENX	X'02'	Sensitivity is exclusive
SDBSENL	X'01'	Sensitivity is load

B	11	SDBF4	1	Code byte

Name	EQU	Meaning
SDBALTSQ	X'40'	Secondary index is main processing sequence
SDBDPAR	X'10'	Field is in destination parent
SDBCISP	X'04'	CI-split occurred in HISAM KSDS
SDBPOSL	X'02'	Position lost
SDBDCHG	X'01'	Temporary switch for replace; data changed

C	12	SDBPHYCD	1	Segment code
D	13	SDBDDIR	3	Address of DMB directory
10	16	SDBNSDB	4	Next SDB for this physical segment
14	20	SDBPSDB	4	Address of segment description in DMB
18	24	SDBKEYLN	0	Executable key length of key field
18	24	SDBPARA	4	Parent SDB (address of PCB for root SDB)
1C	28	SDBDSGA	4	Address of data set group section of JCB for data set containing segment
20	32	SDBTFLG	1	Logical relationship code

Name	EQU	Meaning
SDBPPTSP	X'C0'	Segment is a physical parent of the target of SDBPARA
SDBPPSP	X'80'	Segment is a physical parent of SDBPARA
SDBPCTSP	X'40'	Segment is a physical child of the target of SDBPARA
SDBGEN	X'10'	This SDB is generated
SDBSPP	X'08'	The related segment is a physical parent
SDBSNX	X'04'	The related segment is an index segment
SDBSLC	X'02'	The related segment is a logical child segment
SDBSLB	X'01'	The related segment is a logical parent segment

21	33	SDBTARG	3	Address of the logically related segment's SDB.
24	36	SDBPTNO	1	Pointer number of first physical pointer
25	37	SDBPTDS	1	Physical pointer flag byte

Name	EQU	Meaning
SDBCTR	X'80'	This LP segment has a counter
SDBPTF	X'40'	This segment has a physical twin forward pointer
SDBPTB	X'20'	This segment has a physical twin backward pointer
SDBPP	X'10'	This segment has a physical parent pointer
SDBLTFD	X'08'	This segment has a logical twin forward pointer
SDBLTBK	X'04'	This segment has a logical twin backward pointer
SDBLP	X'02'	This segment has a logical parent pointer

26	38	SDBPCF	1	Pointer number in parent to first occurrence of this segment type
27	39	SDBPCB	1	Pointer number in parent to last occurrence of this segment type
28	40	SDBKEYFD	4	The address within DBPCBKFD for key this segment
2C	44	SDBPOSP	4	Previous position
30	48	SDBPOSC	4	Current position
34	52	SDBPOSN	4	Next position
38	56	SDBXFL	0	SDB expansion flag 01 - SDB expansion for secondary index processing sequence is present
39	57	SDBXPANS	4	Address of SDB expansion block

SDB EXPANSION BLOCK - PRESENT IF INDICATED IN SDB

Hex	Dec	Name	Ln	Description
0	0	SDBXPTYP	1	SDB expansion type 01 - expansion is for secondary index used as PROCSEQ
1	1	SDBXPFDB	3	Address of XDFLD FDB used as PROCSEQ
4	4	SDBXPMSK	4	Mask of XDFLD FDBs allowed in SSAs

8	8	SDBXWMSK	4	Work area reserved for open/close
C	12	SDBXSQCF	2	Offset from DBPCBKFD to subseq area (0 if area not present)
E	14	SDBXSQLN	2	Length of SUBSEQ field(s) -1
10	16	Reserved	4	Reserved

SUBPOOL DIRECTORY - SUBD

This control block contains a one-byte subpool number relative to zero for each HDAM or HIDAM data base allocated. See the general structure and description of DL/I buffer pool control blocks in Chapter 11.

SUBPOOL INFORMATION TABLE - SBIF

The subpool information table is described in Chapter 11 as part of the general structure and description of DL/I buffer pool control blocks. The information in SBIF is provided for each subpool.

Hex	Dec	Name	Ln	Description
0	0	SUBNQF1	1	PST prefix number of first in chain for ENQ subpool
1	1	SUBNQLA	1	PST prefix number of last in chain for ENQ subpool
2	2	SUBBFNO	1	Number of buufers in subpool
3	3	SUBBUFHD	1	HDBFR indicator
4	4	SUPUCPRE	4	Use chain prefix
8	8	SUBUCHAI	32	Use chain
28	40	SUBUCSUF	4	Use chain suffix
2C	44	SUBBFSIZ	1	Size of the buffers handled in subpool

EQU	BYTES
X'01'	512
X'02'	1024
X'03'	1536
X'04'	2048
X'05'	2560
X'06'	3072
X'07'	3584
X'08'	4096

2D	45	SUBDMBCT	1	Number of DMBs assigned
2E	46	SUBLEN	0	Length of SUBINFTA

SYSTEM CONTENTS DIRECTORY - SCD

The DL/I system contents directory (SCD) is produced at DL/I system definition time for online CICS-DL/I. The SCD is preassembled as part of the DL/I nucleus in the batch DL/I system. The SCD contains major entry pointers for all DL/I facilities. The SCD is organized as follows:

- RECORD LAYOUT OF SCD

Hex	Dec	Name	Ln	Description
0	0	CPYRITE	96	Copyright information
60	96	SCD	0	Start of addressable SCD
		SYSTEM CONFIGURATION SECTION		
60	96	SCDDLIV	1	DL/I version
61	97	SCDDLIM	1	DL/I modification level
62	98	SCDDATE	4	System date - Julian
66	102	SCDMXTSK	2	Maximum number of DL/I tasks (online only)
68	104	SCDCMXT	2	Current maximum number of DL/I tasks (online only)
6A	106	SCDATSKC	2	Active DL/I task counter (online only)
6C	108	SCDLOWER	4	Partition lower boundary address
70	112	SCDUPPER	4	Partition upper boundary address
74	116	SCDNAVID	4	Next available task ID (online only)
78	120	SCDLOWID	4	Lowest task ID (online only)
		ACTION MODULE ENTRY POINTS		
7C	124	SCDPRHED	4	Entry point of program request handler - DLZBPR00/DLZODP00
80	128	SCDDDBHO	4	Entry point to buffer handler - DLZDBH00
84	132	SCDDLIRE	4	Entry point to retrieve - DLZDLR00
88	136	SCDDLICT	4	Entry point to call analyzer - DLZDLA00
8C	140	SCDDBLNT	4	Entry point to data base logger - DLZRDBL0
90	144	SCDDLIDR	4	Entry point to delete/replace - DLZDLD00

94	148	SCDDLIIN	4	Entry point to load/insert for retrieve - DLZDDLE0
98	152	SCDDHDS0	4	Entry point to space management - DLZDHDS0
9C	156	SCDDXMT0	4	Entry point to index maintenance - DLZDXMT0
A0	160	SCDDLICL	4	Entry pcint to open/close - DLZDLOC0
A4	164	SCDDSEH0	4	Entry point of work tape writer
A8	168	SCDTRACE	4	Entry pcint of trace module
AC	172	SCDTRCUM	8	Name of trace module
B4	180	Reserved	8	Reserved for future use
BC	188	SCDREENT	4	Entry point to log writer - force write
C0	192	SCDIWAIT	4	Entry point of IWAIT routine - DLZIWAIT
C4	196	SCDERRMS	4	Entry pcint of error message routine - DLZERRMS
C8	200	SCDASE	4	Entry point of scheduler/term (online cnly) - DLZSCHTR
CC	204	SCDABEND	4	Entry point of STXIT ABEND routine - DLZABEND
D0	208	SCDTKTRM	4	Entry point of online task terminator for program request handler
D4	212	Reserved	8	Reserved for future use
		SYSTEM_CONTROL_BLOCK_SECTION		
DC	220	SCDDBFPL	0	Start of buffer control information
DC	220	SCDBFPL	1	Number of buffer subpools
DD	221	SCDDBFA	3	Address of buffer pool control block prefix - BFPL
E0	224	SCDDLIPS	4	Address of PSB directory - PDIR
E4	228	SCDDLIPL	2	Length of PSB directory entries
E6	230	SCDDLIPN	2	Number of PSB directory entries
E8	232	SCDDLIDM	4	Address of DMB directory - DDIR
EC	236	SCDDLIDL	2	Length of DMB directory entries
EE	238	SCDDLIDN	2	Number of DMB directory entries
F0	240	SCDPPSTS	4	Address of beginning of PST prefix entries - PPST

F4	244	SCDPPSTL	2	Length of PST prefix
F6	246	SCDPPSTN	2	Number of PST prefix entries
F8	248	SCDPPAF	4	Pointer to first active PST prefix entry (online only)
FC	252	SCDPPAB	4	Pointer to last active PST prefix entry (online only)
100	256	SCDPPFF	4	Pointer to first inactive PST prefix entry (online only)
104	260	SCDPPFB	4	Pointer to last inactive PST prefix entry (online only)
108	264	SCDPSTLN	2	Length of PST
10A	266	Reserved	2	Reserved for future use
10C	268	SCDACTBA	4	Address of application program control table (online only) - DLZACT
110	272	SCDCDTA	4	Address of currently active DL/I task TCA (online only)
114	276	SCDDLIS	4	Address of first task TCA suspended by DL/I
118	280	SCDDLIUP	4	Address of DL/I upper boundary (batch only)
		-or-		-or-
		SCDCSABA	4	Address of CICS CSA (online only)
11C	284	SCDTKCNT	2	Count of active DL/I tasks (online only)
11E	286	SCDSPCNT	2	Count of suspended DL/I tasks (online only)
120	288	SCDSIND	1	System indicator

Name	EQU	Meaning
SCDCMTI	X'40'	DL/I maximum task indicator (online)
SCDDELT	X'20'	PSB contains PCB with delete sensitivity (online)
SCDUPD	X'10'	PSB contains PCB with update sensitivity (online)
SCDTWFI	X'08'	Task waiting for segment intent (online)
SCDHLRE	X'08'	Highlevel language reentry (PL/I)
SCDNDMP	X'04'	No Dump at STXIT if STXIT active
SCDNLOGI	X'02'	No data base logging to be performed

			SCDNABND	X'01'	No STXIT ABEND to be processed (batch). No CICS journal (online)
121	289	SCDSIND2	1		System indicator (online)

Name	EQU	Meaning
SCDSYSAB	X'80'	DL/I system abended
SCDSYACT	X'40'	System task active
SCDSYWAT	X'20'	System task waiting
SCDRLRST	X'10'	Reload restart
SCDRELOD	X'08'	Standard reload
SCDOPLG	X'01'	Open records written on journal.

122	290	SCDNTWC	2	Count of tasks waiting for segment intent
124	292	SCDABSAV	4	Save area for PC STXIT ABEND module
128	296	SCDLSTAD	4	Address of CICS interface address list
12C	300	Reserved	4	Reserved for future use
130	304	SCDEXTBA	4	Pointer to SCD Extensicn
		DATA_BASE_LOG_SECTION		
134	308	SCDDBDTF	4	Address of DB log DTF
138	312	SCDCWRK	4	Address of DB log work area
13C	316	SCDCWRKL	2	Length of DB log work area
13E	318	SCDSEQ	2	DB log sequence number
140	320	SCDREPLN	2	Length of DB log prefix
142	322	SCDDBLOP	1	DB log option byte

Name	EQU	Meaning
SCDDBLO	X'80'	DB log DTF is open
SCDDBLOR	X'40'	DB log open is required
SCDDBASL	X'02'	Asynchronous logging specified by user

143	323	SCDDBMPS	1	XECB indicator

Name	EQU	Meaning
SCDXECB	X'80'	XECBs have been defined by MPC

144	324	SCDLOCOU	2	Current log count

146	326	SCDTRFL1	1	Trace option byte 1

Name	EQU	Meaning
SCDTUSER	X'80'	
SCDAMOD	X'40'	
SCDRETR	X'20'	
SCDCPOS	X'10'	OPTION values
SCDVSAM	X'04'	
SCDBHCL	X'02'	
SCDUBDX	X'01'	

147	327	SCDTRFL2	1	Trace option byte 2

Name	EQU	Meaning
SCDOLBH	X'80'	OPTION value

STATISTICS SECTION

148	328	SCDTSKCT	8	Number of DL/I tasks (packed)
150	336	SCDMTCNT	4	Number of times at maximum task (packed)
154	340	SCDCMTCT	4	Number of times at current maximum task (packed)
158	344	SCDPDUP	4	Number of duplicate PSBs created (packed)

SYSTEM CONTENTS DIRECTORY EXTENSION - SCDEXTDS

The system contents directory extension is generated in the same manner as the SCD and is a logical extension of it.

- RECORD LAYOUT (Batch Usage)

Hex	Dec	Name	Ln	Description
0	0	SCDEREEN	4	Utility block builder entry point
4	4	SCDEABEX	4	Address of STXIT AB processor
8	8	SCDEABSV	4	Address of STXIT AB save area
C	12	SCDEPCEX	4	Address of STXIT PC processor
10	16	SCDETRAN	4	Address of additional work area
14	20	SCDETRSV	4	Address of transient area
18	24	(unnamed)	4	Address of where batch application program's loaded
1C	28	Reserved	8	Reserved for future use
24	36	SCDETRTB	4	Current entry in trace table

28	40	SCDETRTE	4	End address + of trace table
2C	44	SCDETRTS	4	Start address of trace table
30	48	Reserved	4	Reserved for future use

• RECORD LAYOUT (Online Usage)

Hex	Dec	Name	Ln	Description
0	0	SCDELECB	4	Logger I/O ECB
4	4	SCDESECB	4	System enque ECB
8	8	SCDEFECB	4	System function call ECB
C	12	SCDEVSEX	4	Address of DL/I EXCPAP processor
10	16	SCDEPASS	4	Address of system call password
14	20	SCDEIDST	4	Address of first active PPST ID in table
18	24	SCDEIDNX	4	Address of last active PPST ID
1C	28	SCDEIDWK	4	Address of PPST ID search table
20	32	SCDEMSGT	4	Address of error message module
24	36	SCDETRTB	4	Current entry in trace table
28	40	SCDETRTE	4	End address + 1 of trace table
2C	44	SCDETRTS	4	Start address of trace table
30	48	Reserved	4	Reserved for future use

RECORD LAYOUTS

The rest of this chapter provides layouts and field descriptions for the following records:

Accumulation Header Record - DLZUCUM0
Accumulation Record - DLZUCUM0
Application Program Scheduling Record - DLZRDBL0, DLZRDBL1, and DLZBACK0
Application Program Termination Record - DLZRDBL0, DLZRDBL1, and DLZBACK0
Checkpoint Record - DLZURGU0
Control File List Entry - DLZURPR0
Data Base Log Record - DLZRDBL0, DLZRDBL1, and DLZBACK0
Date/Time Table - DLZUCUM0
Delete Work Area - DLZDLD00
Delete Work Space Prefix - DLZDLD00
DL/1 Control Record (DLZREC0) - DLZDLOC0
Dump Header Record - DLZUDMP0
Dump Record Prefix - DLZUDMP0
File Open Record - DLZRDBL0 and DLZRDBL1
Header Record - DLZURRL0
Index Work Area - DLZXMTWA
Input Data Record - DLZURRL0
Input Work File Record - DLZURWF1
List Control Block - DLZUSCH0
Output Data Record - DLZURUL0
Output Header Record - DLZURUL0
Output Record Containing Segment Prefix - DLZURGU0
Output Table Record - DLZURGU0
Output Work File Record - DLZURWF3
Secondary List Entry - DLZURPR0
Short Segment Table - DLZURUL0
Sorted List Block - DLZUSCH0
SSA for GU Call by Key - DLZURGU0
SSA for GU Call by RBA - DLZURGU0
SSA for the XMAINT Call to the Analyzer - DLZXMTWA
Statistics Record - DLZURUL0

- ACCUMULATION HEADER RECORD - DLZUCUM0

Hex	Dec	Name	Ln	Description
0	0	HLENGTH	2	Length of cum header record
2	2	HSPACE	2	Zeros
4	4	HCODE	1	Header record ID X'00'.
5	5	HFLG	1	Type of data set and organization Bit 5=0 ESDS data set =1 KSDS data set 6=0 HS data set =1 HD data set
6	6	HLRECL	2	Record length
8	8	HORG	1	Prefix organization code
9	9	HPURDATE	3	Purge date for data base data set
C	12	HPURTIME	4	Purge time for data base data set

10	16	HDDNAME	8	Data set symbolic filename
18	24	HDBNAME	8	Data base name
20	32	HDSID	1	Data set ID
21	33	HDATE	3	Run date - YYDDDF
24	36	HTIME	4	Run time - HHMMSSOF
28	40	HSEQ	2	Zeros
2A	42	HBLKSIZE	2	Block size

- ACCUMULATION RECORD - DLZUCUM0

Hex	Dec	Name	Ln	Description
0	0	CLENGTH	2	Length of cum record
2	2	CSPACE	2	Zeros
4	4	CCODE	1	X'50' record identifier
5	5	CFLG	1	Type of data set and organization Bit / Meaning 5=0 ESDS =1 KSDS 6=0 HS file =1 HD file
6	6	CIDIN	2	Length of CDATAID field
8	8	CDBNAME	8	Data base name
10	16	CDSID	1	Data set ID
11	17	CDATE	3	Date - YYDDDF
14	20	CTIME	4	Time - HHMMSSOF
18	24	CSEQ	2	Sequence number
1A	26	CCOUNT	2	Number of data elements of CDATA
1C	28	CDATAID	Var	KSDS prime key or ESDS RBN
		CDATAOL	Var	One or more 4 byte data elements: bytes 0-1 - offset into data set record bytes 2-3 - length of corresponding CDATASEG
		CDATASEG	Var	One or more segment data entries to be moved into data set record.

- APPLICATION PROGRAM SCHEDULING RECORD - DLZRDBL0, DLZRDBL1, AND DLZBACK0

Hex	Dec	Name	Ln	Description
0	0	LENGTH	2	Length of record

2	2	SPACE	2	Binary zero
4	4	LOGFLAG	1	Record type code - X'08'
5	5	SCHDCODE	1	Task ID
8	8	PSBNAME	8	PSB name
E	14	CICSID	3	Packed CICS Transaction ID (online only)

• APPLICATION PROGRAM TERMINATION RECORD - DLZRDBLO, DLZRDBL1, AND DLZBACKO

Hex	Dec	Name	Ln	Description
0	0	PLENGTH	2	Halfword binary length of logical record
2	2	PSPACE	2	Halfword reserved for system use
4	4	ALLOGFLG	1	Identifies this logical record as application program termination record; value is X'07'
5	5	ALPSBNAM	8	PSB name
D	13	ALID	1	TASK ID
E	14	TSKSTAT	36	9 fullwords of Accounting from PSTACCT (online only)
32	50	CICSID	3	Packed CICS transaction I.D. (online only)

• CHECKPOINT RECORD - DLZURGU0

Hex	Dec	Name	Ln	Description
0	0	RCHKPTID	1	Always X'00'
1	1	RCHKNAME	6	Always C'CHKPNT'
7	7	RCHKNUM	4	Checkpoint number - 1 - 9,999 (dec.)
B	11	Reserved	1	Comma, for message to SYSLOG
C	12	RCHKVOL1	6	If tape, file serial number of output volume one at checkpoint time. If DASD - XXXXXX.
12	18	Reserved	1	Comma, for message to SYSLOG
13	19	RCHKVOL2	6	If tape, file serial number of output volume two at checkpoint time. If DASD - XXXXXX.
19	25	Reserved	1	Comma, for message to SYSLOG
1A	26	RCKSEGNM	8	Segment name of root segment in process at checkpoint time
22	34	Reserved	4	Reserved for future use

26	38	RCHKRECL	2	Length of I/O area needed for the GU at restart time
28	40	RCHKPOSC	4	RBN of current record, if HD organization
2C	44	RCHKPTNR	1	Number of checkpoint records (1 or 2)
2D	45	RCHKEYLN	1	Key length of current segment, if HISAM
2E	46	RCKEYVAL	236	Segment sequence field value, if HISAM
11A	282	Reserved	12	Reserved
126	294	RCHKSEG	4	Total number of segments unloaded
12A	298	RCHKROOT	4	Total number of root segments unloaded
12E	302	RCHKREND	Var	Statistics table

Note 1: Dummy checkpoint record does not contain statistics table.

Note 2: Checkpoint message written to SYSLOG consists of message prefix DLZ381I followed by bytes 1 - 34 of the checkpoint record.

- CONTROL FILE LIST ENTRY - DLZURPRO

One or more list entries may be contained in the control list. The control list may spread over one or more control list blocks.

For data base list entry:

Hex	Dec	Name	Ln	Description
0	0	LEFPTR	4	List entry forward pointer (to next list element at same level)
4	4	LENAME	8	DMB name for data base list entry
C	12	LESLPTR	4	List entry sublist pointer (to list at next lower level)
10	16	LECBNO	2	Input control card number
12	18	LELEN	1	Length of list entry
13	19	LEFLG1	1	Flag byte 1: B(0)=1 User specified scan list B(1)=0 Use SEQ scan method B(1)=1 Use SEG scan method B(6,7)=00 data base initially loaded B(6,7)=10 data base scanned

For segment list entry:

Hex	Dec	Name	Ln	Description
0	0	LEFPTR	4	List entry forward pointer (to next list element at same level)
4	4	LENAME	8	Segment name for segment list entry

C	12	LESLPTR	4	List entry sublist pointer (to list at next lower level)
10	16	LECRNO	2	Input control card number
12	18	LELEN	1	Length of list entry
13	19	LEFLG1	1	Flag byte 1: B(0)=1 User-specified scan list B(1)=0 Use SEQ scan method B(1)=1 Use SEG scan method B(6,7)=00 data base initially loaded B(6,7)=01 data base reorganized B(6,7)=10 data base scanned
14	20	LEPSDB	4	Used as intermediate storage
18	24	LELSDB	4	Used as intermediate storage

- DATA BASE LOG RECORD - DLZRDBL0, DLZRDBL1, AND DLZBACK0

Hex	Dec	Name	Ln	Description
0	0	DLENGTH	2	Length of record
2	2	DSPACE	2	Zero
4	4	DLOGCODE	1	Log record ID X'50' = Data base log record X'51' = Old copy of a replaced segment
5	5	DLOGFLG1	1	Bits 0-3 Task ID 4-7 Count of FSE records present
6	6	DLOGFLG2	1	Bits 0=1 Index maintenance record 1-3=001 Physical replace =010 Physical delete =100 Physical insert =110 Logical delete =000 POINTER maintenance record 4=1 Last record of a change group 5=0 ESDS data set =1 KSDS data set 6=0 HS organization =1 HD organization 7=1 New block call
7	7	DLOGFLG3	1	Bits 0=1 REPL call 1=1 DLET call

				2=1	ISRT call
				3&4=00 =01 =10	Modification by control region Modification by message or batch message program Modification by batch program
				5	Reserved
				6=1	First log record of a segment
				7=1	Last log record of a segment
8	8	DIDLN	2		Length of DDATAID field
A	10	DOFFSET	2		Data offset from beginning of block
C	12	DDATALN	2		Length of DDATA field
E	14	DCCCDE	2		DL/I completion code
10	16	DPGMNAME	8		PSB name
18	24	DDBDNAME	8		Data base name from the DMB
20	32	DDSID	1		File identification within the DMB
21	33	DDATE	3		Date - YYDDDF
24	36	DTIME	4		Time - HHMMSSOF
28	40	DSEQ	2		Sequence stamp
2A	42	DDATAID	Var		KSDS - KSDS prime key ESDS - Relative block number

POINTER maintenance record (DDATALN is set to H'4')

DDATA	4	New pointer value
	4	Old pointer value

LOGICAL DELETE record (DDATALN is set to H'2')

DDATA	2	Segment code and new delete byte
	2	Segment code and old delete byte

PHYSICAL INSERT record (DDATALN is set to segment length)

DDATA	V*	New segment data
DFSEOFF	2	Offset to FSE
DFSE	4	New FSE value If more than one FSE changes, DFSEOFF and DFSE are repeated for each additional one.

PHYSICAL DELETE record (DDATALN is set to segment length)

DDATA	V*	Old segment data

DFSEOFF	2	Offset to FSE
DFSE	4	New FSE value If more than one FSE changes, DFSEOFF and DFSE are repeated for each additional one.

PHYSICAL REPLACE record (DDATALN is set to segment length)

DDATA	V*	Old segment data - DLOGCODE = X'51'
		New segment data - DLOGCODE = X'50'

V* = varies with segment length

DCOUNTER The last four bytes of every log record contain the log record sequence number. Numbers are incremented by one. The sequence number of the first record is one.

- DATE/TIME TABLE - DLZUCUM0

Hex	Dec	Name	Ln	Description
0	0	TABFLAG1	1	Blank. Used as table delimiter
1	1	TABFLAG2	1	Contains a 0 or 1 to denote routing for the data base in this table
2	2	TABFLAG3	1	Contains flags as follows: Name / Bit / Meaning TABF3N / 0 / Record to LCGOUT if 1 TAB3DT / 1 / Purge date specified
3	3	TABFLAG4	1	Reserved for future use
4	4	TABFLAG5	4	Reserved for future use
8	8	TABFLAG6	8	Contains date/time, if specified

- DELETE WORK AREA - DLZDLD00

Hex	Dec	Name	Ln	Description
0	0	DLTPWAID	8	Identification of work area (DMB/ACB/*)
8	8	DLTPSCNE	4	Address of prior scan exit
C	12	DLTWANXT	4	Address of next WKA
10	16	DLTWASW	1	Switch indicating if this work area is the first in the work area space acquired from the buffer pocl
11	17	DLTWAPRI	..3	Address of prior WKA
14	20	DLTLPKOF	2	Offset from work area start to first byte of concatenated key

16	22	DLTWASZ	..2	Size of current WKA
18	24	DLTDMB	4	Address of current DMB
1C	28	DLTSPSDB	4	Address of scan-start PSDB
20	32	DLTLPSDB	4	Address of highest PSDB to be included in this scan
24	36	DLTSLEV	2	Level of scan-start segment
26	38	DLTKOF	..2	Reserved
28	40	DLTESECL	4	Address of secondary list entry causing this scan
2C	44	DLTEDMB	4	DMB address of prior scan
30	48	DLTEPSDB	4	PSDB address of current segment in prior scan
34	52	DLTERBN	4	RBN of current segment in prior scan
38	56	DLTEOFF	2	Reserved
3A	58	DLTEFLGS	..2	Reserved
3C	60	DLTPLT	4	RBN of prior logical twin when following a logical twin chain
40	64	DLTCLT	4	RBN of current logical twin when following a logical twin chain
44	68	DLTNLT	4	RBN of next logical twin when following a logical twin chain
48	72	DLTTEMP1	4	Work area
4C	76	DLTTEMP2	4	Work area
50	80	DLTTEMP3	4	Work area
54	84	DLTTEMP4	4	Work area
58	88	DLTPSDB	4	Current PSDB this level
5C	92	DLTRBN	4	Current RBN this level
60	96	DLTROFF	2	Reserved
62	98	DLTRFLG	..2	Deletability flags
64	100	DLTRWK	4	Work area - reserved
68	104	DLTWALN	16	Current information for each level specified for the data base. Mapped with DLTPSDB - DLTRWK above.
	104 + 16n		X	Start of concatenated key. Length is the longest concatenated key for the DMB.

• DELETE WORK SPACE PREFIX - DLZDLD00

Hex	Dec	Name	Ln	Description
0	0	DLTBLKNM	4	Block number of buffer (from PSTBLKNM)
4	4	DLTBUFFA	4	Address of buffer prefix (from PSTBUFFA)
8	8	DLTNXTWS	4	Address of next work space
C	12	DLTPRIWS	4	Address of prior work space
10	16	DLTSIZWS	4	Usable size of this space
14	14		4	Reserved

• DL/I CONTROL RECORD (DLZRECO) - DLZDLOC0

Hex	Dec	Name	Ln	Description
0	0	RECDATCR	3	Creation date - YYDDDF
3	3	RECTIMCR	5	Creation time - HHMMSSTHOF
8	8	RECDATRE	3	Recovery date - YYDDDF
B	11	RECTIMRE	5	Recovery time - HHMMSSTHOF
10	16	RECDATER	3	Reserved
13	19	RECTIMER	5	Reserved
18	24	RECNXRBA	4	Not used
1C	28	RECDOS	3	DL/I component code (DLZ)
1E	31	RECVERS	3	Version and modification level (V1)
22	34	RECPTF	2	PTF number
24	36	RECLKSDS	4	KSDS record length (HISAM only)
28	40	RECLESDS	4	ESDS record length
2C	44	RECORGAN	1	Data base organization

Name	Character	Meaning
RECHDAM	D	HDAM
RECHIDAM	I	HIDAM
RECHISAM	S	HISAM

Hex	Dec	Name	Ln	Description
2D	45		Var	Reserved to end of control interval

• DUMP HEADER RECORD - DLZUDMP0

Hex	Dec	Name	Ln	Description
0	0	Reserved	1	Reserved for future use

1	1	IDOUT	1	Character D
2	2	Reserved	2	Reserved for future use
4	4	DBOUT	8	Name of the DMB devised from the Data Base Description (DBD)
C	12	IDDNOUT	8	Contains the name of the key sequenced data set if this is dump of a KSDS data set
14	20	Reserved	1	Reserved for future use
15	21	DATEOUT	3	Julian date in packed decimal - YYDDDF
18	24	TIMEOUT	4	Time in packed decimal - HHMMSSOF
1C	28	ODDNOUT	8	Contains the name of the entry sequenced data set if this is dump of an ESDS data set
24	36	IBLKSOUT	2	Contains KSDS control interval size if this is dump of KSDS data set
26	38	ILRECOUT	2	Contains KSDS record length if dump of KSDS data set
28	40	OBLKSOUT	2	Contains ESDS control interval size if this is dump of ESDS data set
2A	42	CLRECOUT	2	Contains ESDS record length if dump of ESDS
2C	44	IKEYLENG	2	Contains KSDS key length if dump of KSDS
2E	46	IKEYPOS	2	Contains KSDS relative key positive if dump of KSDS

- DUMP RECORD PREFIX - DLZUDMPO

Hex	Dec	Name	Ln	Description
0	0	COUNTOUT	4	ESDS RBA identifier; unused if KSDS
4	4	DSIDOUT	1	Character I if KSDS; O if ESDS
5	5	Reserved	1	Reserved for future use
6	6	DSRECLN	2	Record size + prefix length
8	8	DATA	Var	Physical record image

- FILE OPEN RECORD - DLZRDBL0 AND DLZRDBL1

Hex	Dec	Name	Ln	Description
0	0	DLENGTH	2	Length of record
2	2	DSPACE1	2	Binary zero
4	4	DLOGCODE	1	Record type code - X'2F'

5	5	DLOGFLG1	2	Data set organization X'00' = ESDS X'04' = KSDS
7	7	DSPACE2	9	Binary zero
10	16	DPGMNAME	8	Data set filename (ACB)
18	24	DDBDNAME	8	DMB name
20	32	DDSID	1	DSGACBNO (1 if HISAM ESDS; otherwise 0)
21	33	DDATE	3	Binary zero
24	36	DTIME	4	Binary zero
28	40	DCOUNT2F	4	Log record sequence number

- HEADER RECORD - DLZURRLO

Hex	Dec	Name	Ln	Description
0	0	Reserved	1	Reserved for future use
1	1	IDIN	1	Character R
2	2	RECLNOUT	2	Size of output record, including prefix
4	4	DBDNAME	8	Name of the DMB derived from the Data Base Description (DBD)
C	12	DDNAMEI	8	Name of key sequenced data set (KSDS)
14	20	Reserved	1	Reserved for future use
15	21	DATE	3	Julian date in packed decimal -YYDDDF
18	24	TIME	4	Time in packed decimal-HHMMSSOF
1C	28	DDNAMEO	8	Name of entry sequenced data set (ESDS)
24	36	BLKSIZEI	2	KSDS record length * number of records/control interval
26	38	LRECLI	2	KSDS record length
28	40	BLKSIZEO	2	ESDS record length * number of records/control interval
2A	42	LRECLO	2	ESDS record length
2C	44	KEYLENGI	2	KSDS key length
2E	46	KEYPOSI	2	KSDS relative key position

- INDEX WORK AREA - DLZXMTWA

Hex	Dec	Name	Ln	Description
0	0	XSAVDSGA	4	Save location for caller's DSG
4	4	XSAVPCB	4	Save location for caller's PCB

8	8	XSAVUSER	4	Save location for caller's I/O area
C	12	XSAVIQPR	4	For caller's call list address
10	16	XPHYSPP	4	Save location for phys.P.Ptr
14	20	XWORKPCB	4	Save location for XMAINTs PCB
18	24	XWORKSAA	4	Address of SSA built by DLZDXMTO
1C	28	XWORKFNC	4	XMAINTs function code for call
20	32	XDPSDBAD	4	Address of PSDB of index target segment
24	36	XDSECLST	4	Secondary list of index target segment
28	40	XDRBAPTR	4	RBA of index target segment
2C	44	XSPSDBAD	4	PSDB of index source segment
30	48	XSSECLST	4	Secondary list of index source segment
34	52	XSRBAPTR	4	RBA of index source segment
38	56	XNPSDBAD	4	Address of PSDB of index pointer segment
3C	60	XDSDBAD	4	Index target segment SDB address
40	64	XSSDBAD	4	Index source segment SDB address
44	68	XPROT	2	Length of protected data
46	70	XRPREFIX	2	Record prefix length
48	72	XSPREFIX	2	Segment prefix length
4A	74	XNSEGLEN	2	Length of index pointer segment
4C	76	XNKEYLEN	2	Sequence field length of index pointer segment
50	80	STACK1	4	Return address for 1. level subr.
54	84	STACK2	4	Return address for 2. level subr.
58	88	STACK3	4	Return address for 3. level subr.
5C	92	XSAVSTC	1	Save status code
5D	93	XSAVFUN	1	Save location for function
5E	94	XCALLFUN	1	Call attributes byte

<u>Name</u> <u>EQU</u> <u>Meaning</u>

ISLOAD X'80' Load mode
ISASRT X'40' ASRT call ISDLET X'20'
DLET call ISISRT X'10' ISRT call
ISREPL X'08' Function is replace
ISUNLD X'02' UNLD call

5F	95	XTSWIT1	1	Temporary switch

Name	EQU	Meaning
XNOSUPR	X'80'	No suppression for this index
XOLDSUPR	X'40'	Old segment was surpressed
XPTRONLY	X'20'	PTR to XDS only, no CONCAT key
XISPRIM	X'10'	We found a primary index
XNULLFLD	X'01'	Null value suppression
XEXITRT	X'02'	Exit routine for suppression
XDATACHN	X'04'	XNS changed in a replace call

60	96	XWORKPUT	2	Begin of record for load
62	98	XWORKUSR	0	XMAINTs I/O area for call
62	98	XWORKDUM	2	Reserved
64	100	XWORKSEG	0	Start of segment
64	100	XWORKCD	1	Segment code
65	101	XWORKDEL	1	Delete byte
66	102	XWORKPTR	4	PTR in index pointer segment
6A	106	XWORKKEY	VAR	Area for key in index pointer segment

- INPUT DATA RECORD - DLZURRL0

Hex	Dec	Name	Ln	Description
0	0	ESDS RBA	4	ESDS RBA identifier; unused if KSDS
4	4	DSIDIN	1	Character I if KSDS; O if ESDS
5	5	Reserved	3	Reserved for future use
8	8	DATA	Var	Physical record image. The first four bytes contain the relative byte address (RBA) of the next ESDS record containing overflow dependent segments for the root segment. The RBA is zero if no (more) ESDS records follow. The last byte of the data record contains a special physical code X'0'. If the data base contains only HISAM root segments and ACCESS=SHISAM, the physical code and RBA do not exist.

- INPUT WORK FILE RECORD - DLZURWF1

- CONSTANT PORTION OF ALL INPUT RECORDS

Hex	Dec	Name	Ln	Description
0	0	ALENGTH	2	Total length of input record (all records are variable length)
2	2	ASPACE	2	Two bytes of zeros
4	4	ALTYPE	1	Type of input record, as shown below.
5	5	ALFLAG1	1	Flag byte 1 B(0)-1 Initial load of segment -0 Reload of segment B(1)-1 LC sequence field present B(2)-1 Record produced during data base scan B(3)-1 Logical parent's concatenated key is present B(4)-1 LC sequence field is unique B(5)-1 Root sequence field is present B(6)-1 Logical child pointers are used by logical parent B(7)-1 Logical twin pointers to be resolved by type 20 and 30 records
6	6	ALFLAG2	1	Flag byte 2 - Sequence field length minus one
7	7	ALFLAG3	1	Flag byte 3 Other than Type 40 - Logical parent concatenated key length minus one Type 40 - Indexed field length minus one

Record Type	Use
00	Generated once for each use of a segment as a logical parent
10	Generated once for each use of a segment as a logical child
20	Generated when a segment used as a logical child contains logical twin forward pointers and when the logical twin chain cannot be resolved by using the logical child's sequence field
30	Generated when a segment used as a logical child contains logical twin backward pointers and when the logical twin chain cannot be resolved by using the logical child's sequence field.
40	Generated once for each time a segment is indexed

- CONSTANT PORTION OF TYPE 00, 10, 20, 30 RECORDS

Hex	Dec	Name	Ln	Description
8	8	ALEVTTR	4	Physical location of record in which segment resides
C	12	ALPDBNAM	8	Name of data base in which logical parent resides
14	20	ALPSEG	1	Logical parent's segment code
15	21	ALPCKEY	var	logical parent's concatenated key
m	n	ALPCADDR	4	Logical parent's old address or zero
m+4	n+4	ALCDBNAM	8	Name of data base in which logical child resides
m+C	n+12	ALCSEG	1	Logical child's segment code

- REMAINDER OF TYPE 00 RECORDS

Hex	Dec	Name	Ln	Description
m+10	n+16	ALCFL	4	Old value of logical child first pointer or zero
m+14	n+20	ALT0001	1	X'00'
m+15	n+21	ALPLSGOF	2	Offset of segment within record in which it resides
m+17	n+23	ALPCCTR	4	Old value of counter field
m+1B	n+27	ALPDCB	1	DCB number
m+1C	n+28	ALPSEQA	var	Logical parent's root segments sequence field.

- REMAINDER OF TYPE 10, 20, 30 RECORDS

Hex	Dec	Name	Ln	Description
m+10	n+16	ALFIL	1	X'FF'
m+11	n+17	ALCSEQ	var	Logical child's sequence field
m+s	n+t	ALCM	4	Type 10 - Logical child's old address Type 20 - Logical child's old LTF pointer Type 30 - Logical child's old LTB pointer If Type 20 and 30 records not used, ALCM contains offset of segment in logical record

m+s+4	n+t+4	ALT123	1	X'10' for Type 10 X'20' for Type 20 X'30' for Type 30
m+s+5	n+t+5	ALCDCB	1	DCB number
m+s+6	n+t+6	ALCSEQA	2	Offset to control data set entry

- REMAINDER OF TYPE 40 RECORDS

Hex	Dec	Name	Ln	Description
8	8	ALICOA	4	Logical child's old address or zero
C	12	AIDBNAM	8	Index data base name
14	20	AIFLDVAL	var	Indexed field value
m	n	AISC	1	Index segment's segment code
m+1	n+1	AISEQ	var	Index segment's sequence field
m+s	n+t	AISEGN	8	Index segment's name (second level)
m+s	n+t	AIFLDN	8	Indexed field name (1st level)
m+s+4	n+t+4	AISDEN	8	Indexed segment's data base name
m+s+C	n+t+12	AISSC	1	Indexed segment's segment code
m+s+D	n+t+13	AILCNA	4	Logical child's new address
m+s+11	n+t+17	AIDATA	var	Indexed segment source field data

• LIST CONTROL BLOCK - DLZUSCH0

Hex	Dec	Name	Ln	Description
1C	28	ENTLNGTH	2	The length, in bytes, of each entry in the list
1E	30	COMPLOC	2	The offset from the beginning of each entry to the key field
20	32	COMPLNG	2	The length of the key field
22	34	NUMENT	2	The current number of entries in the list
24	36	CHAINLOC	4	The location of the first of a chain of core blocks containing sorted list entries

28	40	CHBACK	4	The location of the last block in the chain
2C	44	ENTBLKSZ	4	The size of each core block used for list entries (includes the chaining fields). This value is calculated as follows: ENTBLKSZ = 16*ENTLNGTH+8
30	48	LASTLO, LASTHI, LASTMD, ENTLOC	12	Work areas used by INSRCH and LOCSRCH

- OUTPUT DATA RECORD - DLZURULO

Hex	Dec	Name	Ln	Description
0	0	CONTOUT	4	ESDS RBA identifier; unused if KSDS
4	4	DSIDOUT	1	Character I if KSDS; O if ESDS
5	5	Reserved	1	Reserved for future use
6	6	DSRECLN	2	Record size + prefix length
8	8	DATA	Var	KSDS or ESDS physical record image. The first four bytes contain the VSAM relative byte address (RBA) of the next ESDS record containing overflow dependent segments for the root segment. The RBA is zero if no (more) ESDS records follow. The last byte of the data record contains a special physical code X'0'. If the data base contains only HISAM root segments and ACCESS=SHISAM, the physical code and RBA do not exist.

- OUTPUT HEADER RECORD - DLZURULO

Hex	Dec	Name	Ln	Description
0	0	Reserved	1	Reserved for future use
1	1	IDOUT	1	Character R
2	2	RECLNOUT	2	Size of output record, including prefix
4	4	DBDOUT	8	Name of the DMB derived from the Data Base Description (DBD)
C	12	IDDNOUT	8	Name of key sequenced data set (KSDS)
14	20	Reserved	1	Reserved for future use
15	21	DATEOUT	3	Julian date in packed decimal-YYDDDF
18	24	TIMEOUT	4	Time in packed decimal-HHMMSSOF
1C	28	ODDNOUT	8	Name of entry sequenced data set (ESDS)

24	36	IBLKSOUT	2	KSDS record length * number of records/control interval
26	38	ILRECOUT	2	KSDS record length
28	40	OBLKSOUT	2	ESDS record length * number of records/control interval
2A	42	OLRECOUT	2	ESDS record length
2C	44	IKEYLENG	2	KSDS key length
2E	46	IKEYPOS	2	KSDS relative key position

- OUTPUT RECORD CONTAINING SEGMENT PREFIX - DLZURGUO

Hex	Dec	Name	Ln	Description
0	0	RGUSEGLV	1	Physical segment code for this record
1	1	RGUHSDF	1	HSAM delete flag; always X'80' to denote HD Reorganization Unload Utility
2	2	RGUHDRLN	2	Length of prefix portion of record
4	4	RGUSEGLN	2	Length of data portion of record
6	6	RGUSEGNM	8	Segment name for this record
E	14	RGUSEGDF	1	Delete flag of segment carried forward
F	15	RGUPFCTR	4	Counter field of segment carried forward
13	19	IOTWFOR	4	Reserved for future use
17	23	IOTWBACK	4	Reserved for future use
1B	27	IOPAR	4	Reserved for future use
19	31	IOOLD	4	Old location of record carried forward
23	35	IOSEG	Var	Variable-length segment data

- OUTPUT TABLE RECORD - DLZURGUO

Hex	Dec	Name	Ln	Description
0	0	RGUSEGLV	1	Always X'00'
1	1	RGUHSDF	1	X'80' for first table record and checkpoint table record X'90' for last table record
2	2	RGUHDRLN	2	Length
4	4	RGUSEGLN	Var	A table containing one entry for each segment type.

FIELD DESCRIPTION OF RGUSEGLN

Hex	Dec	Name	Ln	Description
0	0	SEGNAME	8	Segment name
8	8	SMIMCHLD	4	Minimum immediate twins
C	12	SAIMCHLD	4	Average immediate twins
10	16	WKIMCHLD	4	Working entry for above
14	20	SMSBCHLD	4	Maximum subordinate children
18	24	SASBCHLD	4	Average subordinate children
1C	28	WKSBCHLD	4	Working entry for above
20	32	TSEGTYPE	4	Total segments for this segment type
24	36	SEGLEVEL	1	Segment level for this segment type
25	37	SEGPHYCD	1	Segment physical code
26	38	TSEGLEN	2	Segment length including prefix length (high-order bit is on if this is last entry)

- OUTPUT WORK FILE RECORD - DLZURWF3

- RECORD TYPE 00 AND 10

Hex	Dec	Name	Ln	Description
0	0	CLENGTH	2	Total length of input record (all records are variable length)
2	2	CSPACE	2	Two bytes of zeros
4	4	CTYPE	1	Type of input record as shown below.
5	5	CFLAG1	1	Flag byte 1. All flags have same meaning as ALFLAG1 of input record, except: B(3) - Matching Type 1 record found for Type 0 record
6	6	CLCDBN0	8	Logical child's data base name (Type 00)
6	6	CLPDBN1	8	Logical parent's data base name (Type 10)
E	14	CLCSEGN0	1	Logical child's segment code (Type 00)
E	14	CLPSEGN1	1	Logical parent's segment code (Type 10)

F	15	CLPSEGN0	1	Logical parent's segment code (Type 00)
F	15	CLCSEGN1	1	Logical child's segment code (Type 10)
10	16	CLCFRST	4	Logical child first pointer or zero (Type 00)
10	16	CLTFWD	4	Logical twin forward pointer or zero (Type 10)
14	20	CLCLST	4	Logical child last pointer or zero (Type 00)
14	20	CLTBKWD	4	Logical twin backward pointer or zero (Type 10)

Record Type	Use
00	Generated for purposes of updating LCF, LCL, CTR fields in LP
10	Generated for purpose of updating LTF, LTB, LP fields in LC

- SECONDARY LIST ENTRY - DLZURPRO

Hex	Dec	Name	Ln	Description
0	0	LEFPTR	4	List entry forward pointer (to next list element at same level)
4	4	LENAME	8	Referenced data base name for secondary list entry
C	12	LEFDLP	2	Length of logical parent concatenated key
E	14	LEFLG3	1	Flag byte 3: B(0)=1 Use type 20/30 records B(1)=1 Use LC sequence field B(6)=1 Use LP CK B(7)=1 Use LP old address
F	15	LELCD	1	Amount to be subtracted from LC pointer
10	16	LEFDLC	2	Position of LC pointers in prefix
12	18	LELEN	1	Length of list entry
13	19	LEFLG1	1	Flag byte 1: B(0)=1 User specified scan list B(1)=0 Use SEQ scan method B(1)=1 Use SEG scan method B(6,7)=00 data base initially loaded B(6,7)=01 data base reorganized B(6,7)=10 data base scanned
14	20	LELCSC	1	Segment code for logical child

15	21	LEFLG2	1	Flag byte 2: B(0)-Prefix counter to be updated B(1)-LC first pointer to be updated B(2)-LC last pointer to be updated B(3)-LP pointer to be updated B(4)-LT forward pointer to be updated B(5)-LT backward pointer to be updated B(6)-Use LP concatenated key B(7)-Use LP old address
16	22	LESP	2	Spare area

- SHORT SEGMENT TABLE - DLZURULO

Hex	Dec	Name	Ln	Description
0	0	Reserved	1	Reserved for future use
1	1	SEGMCODE	1	Physical segment code
2	2	PARSEGCD	1	Physical code of this segment's parent
3	3	SEGMLEVL	1	Segment hierarchical level
4	4	Reserved	2	Reserved for future use
6	6	SEGMLENG	2	Segment length, including prefix

- SORTED LIST BLOCK - DLZUSCHO

Hex	Dec	Name	Ln	Description
0	0	ENCNT	1	The count minus one of the current number of entries in this block (currently, the maximum value for count is 16)
1	1	CHAIN	3	The location of the next sorted list block in the chain. In the last block, this field contains binary zeros.
4	4	BKCHAIN	4	The location of the preceding sorted list block in the chain. In the first block on the chain, this field contains the location of the CHAINLOC field in the list control block.
8	8	ENTRIES	Var	Up to 16 full entries in sorted order. Note: All blocks are the same size regardless of the number of entries contained. Unused space at the end of a block is not zeroed.

- SSA FOR GU CALL BY KEY - DLZURGU0

Hex	Dec	Name	Ln	Description
0	0	KEYSEGNM	8	Name of segment to be retrieved

8	8	KEYCODE	2	'*C' - command code
A	10	KLEFTPAR	1	'(' - left parenthesis
B	11	KEY	1-236	key to be retrieved
-	-	KRITEPAR	2	')' - right parenthesis

• SSA FOR GU CALL BY RBA - DLZURGU0

Hex	Dec	Name	Ln	Description
0	0	RBASEGNM	8	Name of segment to be retrieved
8	8	RBACODE	2	'*T' - command code
A	10	RLEFTPAR	1	'(' - left parenthesis
B	11	RBA	4	RBA to be retrieved
F	15	RRITEPAR	1	')' - right parenthesis

• SSA FOR THE XMAINT CALL TO THE ANALYZER - DLZXMTWA

Hex	Dec	Name	Ln	Description
0	0	XSEGNAME	8	Name of index pointer segment
8	8	XCCMMCOD	2	'*X' - command code
A	10	XLEFTPAR	1	'(' - left parenthesis
B	11	XKEYVALU	VAR	Key value followed by right parenthesis ')'

• STATISTICS RECORD - DLZURUL0

Hex	Dec	Name	Ln	Description
0	0	Reserved	1	Reserved for future use
1	1	STATID	1	Character S
2	2	STATNO	2	Number of segment types in data set group
4	4	STATDBNM	8	Name of the DMB derived from the DBD
C	12	STATISDD	8	KSDS filename
14	20	STATOSDD	8	ESDS filename
1C	28	STATAB	Var	A 16-byte table entry for each segment type in the data base

FIELD DESCRIPTION OF STATAB

Hex	Dec	Name	Ln	Description
0	0	SEGNAME	8	Segment name
8	8	TOTSEG	4	Total number of segments unloaded
C	12	SEGLEV	1	Segment level
D	13	SEGPCD	1	Segment physical code
E	14	SEGLN	2	Segment length, including prefix

This section describes the executable processing macros that standardize some processing routines and DSECTS and lists the macros that provide the DSECTs.

DLZBLDL

This macro is used to search the core image libraries to determine if a specified load module is present. Optionally, if the phase is present, the length of it is calculated for the caller. The DOS/VS LOAD macro (TXT=NO) is used to obtain the directory entry information.

OPERANDS

The descriptions and valid parameters for the two keyword operands are as follows:

- PHASE — The name of the phase in the core image library.

  =(reg) — The register specified in parenthesis must point to the 8-byte name (padded with blanks if necessary).

  ='name' — The actual phase name may be specified enclosed in single quotes.

  = label — This is the label of an 8-byte field containing the phase name with any necessary blanks.

  Register 1 is the default which must be loaded with the address of the name.

- LENGTH — Specified if the caller desires the actual length of the load module to be calculated by this macro.

  =(reg) — The register specified in parenthesis will contain the length in binary of the load module as indicated in the directory entry. Register 15 is invalid.

  = label — This is the label of a fullword in the calling program which will contain the length of the found phase on exit.

  If LENGTH is omitted, no length will be calculated.

EXIT CONDITIONS

R15 = 0 — The phase was found and the length, if requested, has been returned.

R15 = 4 — The phase was not found.

Registers C and 1 are destroyed unless specified for the length register. All other registers are unchanged.

DLZBLKLD

This macro is used by some DOS/VS DL/I utility programs to request the initialization module to load all control blocks needed to process a specified utility PSB. A utility PSB is built by the application control block creation and maintenance utility for every user DBD except a primary HIDAM index, logical, or HSAM.

The utilities which use this special function have 'ULU' in the first three bytes of the parameter card. When batch initialization determines (by utility name - either DLZURPR0, DLZURGS0, or DLZURGP0) that the DLZBLKLD macro will be used, it does not load any control blocks. The action modules and PST and SCD are loaded, however. When the utility first receives control, register 1 contains the address of the PST.

OPERAND

When the utility reaches the point where blocks are needed, the DLZBLKLD macro is executed:

```
                [(reg)]
DLZBLKLD    DMB=[label]
```

The DMB operand indicates the address of the 8-byte DMB name for which blocks are required. Either the register number (reg) or the label of the field may be specified to indicate the address. If this operand is omitted, register 1 is assumed to contain the address of the DMB name.

The expansion replaces the ending 'D' of the DMB name with a 'U'. A CALL is made to ASMTDLI with the parameter list as follows:

```
       DC  A(FUNC)      Address of function
       DS  CL8          The name of the utility PSB

FUNC   DC  C'BLDB'      Function
```

EXIT CONDITIONS

After execution of this DLZBLKLD macro, register 15 contains a return code:

R15 = C   The blocks were loaded successfully. Register 1 contains the address of the list of PCB addresses.

R15 ≠ 0   The blocks were not loaded successfully. Register 1 contains the address of the name of the block which could not be loaded.

Any previously loaded blocks have been overloaded and new buffer pools have been allocated.

When the utility program returns to the language interface at end-of-job, a return code is expected in register 15. If register 15 is 0, normal unload processing will occur. If register 15 is non-zero, no UNLD call will be made. This return is used when no blocks have been successfully loaded.

DIZDEV

This internal macro is used to determine the physical device type which has been assigned to a specified logical unit. The contents of the PUB (physical unit block) are returned to the caller.

OPERANDS

```
       [ (reg) ][  SYSxxx]
DIZDEV [label ][,(reg)  ]
               [  label ]
```

- The first positional operand indicates the address of the area to which the 8-byte PUB entry is to be returned. Either a register or the label of the area may be specified. If omitted, register 1 is assumed to point to the 8-byte area.
- The second operand indicates the logical unit (system or programmer). It may be specified as:

SYSxxx	Where xxx is the system or programmer logical unit, not to exceed SYS243.
(reg)	The register number which contains the programmer unit in binary, from 0 to 243. Any register except 1 is valid.
label	The label of a halfword containing in binary programmer unit number from 0 to 243.

If omitted, register 0 is assumed to contain the programmer logical unit number.

EXIT CONDITIONS

Register 1 = the address of the 8-byte PUB information entry.

If the specified device is assigned IGN or UA, or was an invalid unit number, byte 4 will contain X'FF'. In addition, the actual value from the IUB entry (X'FF' if UA, X'FE' if IGN) is in byte 6.

If the device is assigned, the PUB entry is returned. Byte 4 (AREA+4) contains the device type information.

DLZER

This macro is used by some of the modules in the application control blocks creation and maintenance utility. It provides an interface between the caller and DLZUMSGO which writes a specified message. A parameter list is created and a call is made to DLZUMSGO.

OPERANDS

```
DLZER   ID=nn[,INSERT=({(reg)},...)]
                       {label}
```

ID = nnn — This is the message number in decimal to be printed. This operand is required.

INSERT = (reg) — This is the register containing the address of additional information to be inserted into the message. Any register except 1 or 15 may be used.

= label — This is the label of the area containing additional information.

This operand can be specified in sublist notation combining the two possible formats. The sublist must be enclosed in parentheses. If only one register is specified, the additional set of parentheses is also required: that is, INSERT=((6)). This operand is optional.

EXIT CONDITIONS

The requested message has been written. Registers 1, 14, and 15 have been used.

DLZIPOST

This macro is used by DL/I to post ECBs in an online environment.

There are no operands. Register 2 must contain the address of the ECB to be posted. Bit 0 of byte 2 is set on.

DLZIWAIT

This macro is used by DL/I to communicate with an IWAIT routine (DLZIWAIT) to wait until an ECB is unposted.

There are no operands. The PST must be addressable and register 2 must contain the address of the ECB that is to be waited for. The caller must have provided a USING SCD,15. Registers 14 and 15 are used to branch to the DLZIWAIT routine.

DLZTRCAL

This macro is used by action modules to invoke the tracing facility. Refer to "Trace Invocation for Action Modules" in Chapter 15 for a description of this macro.

DLZTRPRM

This macro is called by the DLZTRACE macro to parse parameter lists. It is similar to the DLZXPARM macro of DBDGEN (see "DLZXPARM Macro" in Chapter 6). In addition to the interface described for DLZXPARM, the length of each parameter list member is passed to the caller in the GBLA fields $PLEN(25).

DLZMPCPT

The master partition controller (MCP) partition table is used to pass control information when processing batch partition application programs under MPS (Multiple Partition Support). The MPC partition table resides in the transaction work area. There is one entry for every partition defined during system generation, except for the partition where the MPC resides.

DLZTWAB

This macro provides the mapping for the BPC batch partition control information for the DL/I task termination routine under MPS (Multiple Partition Support). This information resides in the BPC's task transaction work area.

DLZXTAB

This macro provides the mapping for the XECBTAB macro DEFINE, DELETE, and CHECK options under MPS (Multiple Partition Support).

DLZXCB1

This macro maps the DLZXCBn1 and the data that follows it. It is used to check data under MPS (Multiple Partition Support).

MACROS USED TO CREATE DSECTS FOR DL/I SYSTEM CONTROL BLOCKS

The following macros are used to generate DSECTS for the DL/I control blocks:

DLZBFFR
DLZBFPL
DLZDDIR
DLZIDLI
DLZPDIR
DLZPPST
DLZPSIL

DIZPST
DLZSCD.

Macros used only by utilities to generate DSECTs:

DLZCKPT
DIZDTF
DLZIDBD
DLZRECO
DLZUCHDR
DLZUCOID
DLZUCREC
DLZUCUMC
DLZUDHDR
DLZURGUF
DLZURHDR
DLZUSTAT
DLZTRENT.

Miscellaneous macros:

DLZHDSO	Work area for DLZDHDSO
DLZMSG	Messages for utilities
DLZQUATE	Register equates
DLZSBIF	Work area for DLZDBHOO
DLZUMSG	Messages for utilities
DLZWA	Work area used by DIZDLD00
DLZXMTWA	Work area used by DLZDXMTO.

CHAPTER 14. LOW-LEVEL CODE/CONTINUITY CHECK IN DL/I

FLOW OF CONTROL

Low Level Code/Continuity Check (LLC/CC) in DL/I is used as a subroutine of a user-written application program that runs under DOS/VS. Control passes to and from the subroutine using standard calls.

LLC/CC in DL/I is a single control section (CSECT) which is structured into seven modules (see Figure 14.1). The entry modules 000 for update and 001 for initial generation of low-level codes have multiple entry points for call statements issued by the user-written application program, that is, a separate entry point for each source language that is supported. All modules have only a single exit point, all lower level modules 002 through 006 are only entered at one point.

All modules assemble and issue DL/I calls. The entry point for DL/I depends on the source language that is identified by the entry point into LLC/CC in DL/I. The language bits in the LLC/CC execution control block (LECB) identify the source language of the application program. If an unexpected status code of DL/I is reported in the appropriate PCB, the error bits in the LECB are turned on, and control is routed back directly to the entry modules 000 or 001.

LLC/CC in DL/I consists of the following modules:

- Module 000 is the entry module for maintenance of low level codes. It passes control to module 002 for execution.
- Module 001 is the entry module for initial generation of low level codes. It passes control to module 002 for execution.
- Module 002 is the common mainline control module. It follows down a hierarchical path of a product structure. For actual explosion, control is passed to module 003. If a particular hierarchical path is exhausted, module 004 is executed to process a parallel path on the same hierarchical level. If all parts on the same level are processed, module 005 steps up one level to identify a parallel path on the higher level. If the original starting level is reached, the complete structure is processed, and control is returned to module 000 or 001. Module 002 also detects loops and executes continuity check recovery in module 006.
- Module 003 explodes a particular part into all its components. Control is passed from and to module 002.
- Module 004 removes the part which has previously been processed from the hierarchical path thus opening a new hierarchical path via the next parent part on the same level. Control is passed from and to module 002.
- Module 005 steps up one level and removes the higher level part from the hierarchical path to open another path. Control is passed from and to module 002. If module 002 is not able to follow a new path on this level, module 005 may be executed repetitively.
- Module 006 handles restoring of old low-level codes if a continuity check is detected. Control is passed to and from module 002.

For a more detailed description, see the relevant HIPO charts.

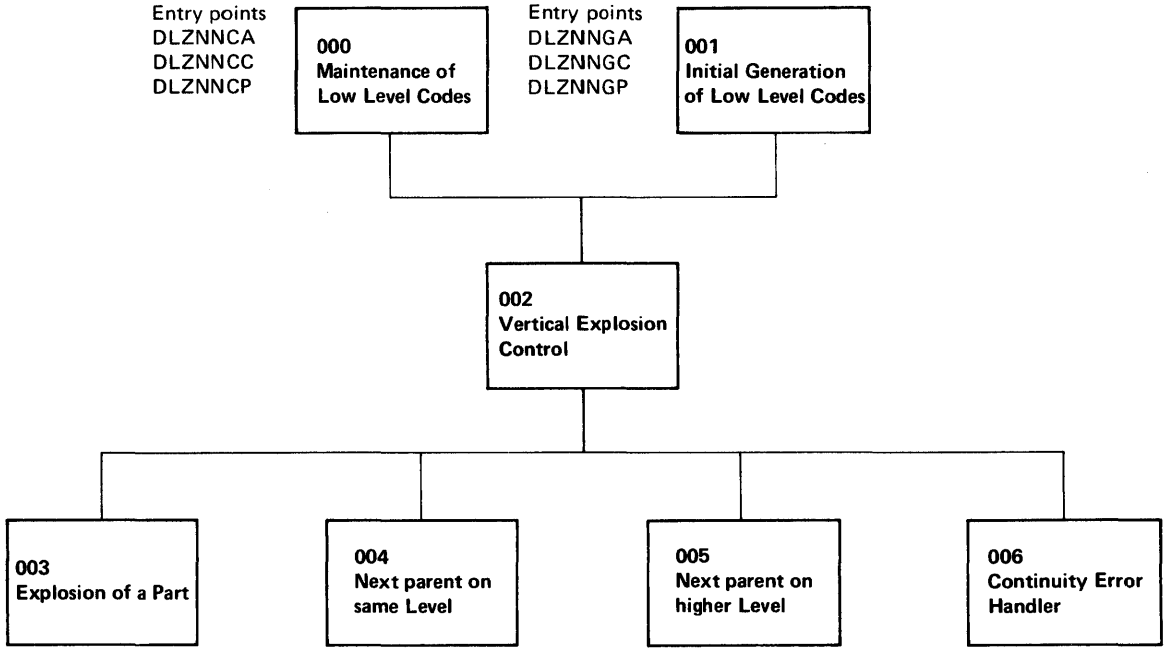


Figure 14.1 Structure of LLC/CC in DL/I

MODIFICATION AIDS

EXTERNAL NAMES

LLC/CC in DL/I uses external names in the directories and libraries of DOS/VS. The following table presents a list of all external names which are used. The user should obtain a DSERV listing to avoid duplicate names.

Type of program	SSL		RL		CIL
	A.books	E.books	Directory entries	Entry points	
Execution program	DLZNN	DLZNN	DLZNN*	DLZNNCA* DLZNNCC* DLZNNCP* DLZNNEC* DLZNNGA* DLZNNGC* DLZNNGP*	
Initialization program for the control data base	DLZNNICT	DLZNNICT			DLZNNICT

* May be modified by the user during customization.

LLC/CC EXECUTION CONTROL BLOCK (LECB)

The LECB of LLC/CC in DL/I is the focal point for all information related to actual operation of the execution program. It consists of 16 bytes which are subdivided into 4 fullwords. An entry point DLZNNEC is provided so that an application program may access the contents of the LECB.

The LECB contains the following information:

1. Identification portion (fullword 0):
   Bytes 0 through 3: C'LECB'=X'D3C5C3C2'
   This identifier facilitates location of the LECB in a main storage dump.

2. Execution control portion (fullword 1):
   Byte 4:

   - Bits 0 through 3: Run type bits
     Bit 0 and bit 1: Reserved
     Bit 2: 1 if IG run
     Bit 3: 1 if U run

   - Bits 4 through 7: Not used

   Byte 5:

   - Bits 0 through 3: Language bits
     Bit 0: Reserved
     Bit 1: 1 if Assembler
     Bit 2: 1 if COBOL
     Bit 3: 1 if PL/I

   - Bits 4 through 7: Not used

   Byte 6: Status byte

   - Bits 0 through 3: Completion bits (mutually exclusive)
     Bit 0: 1 if not completed, abnormal condition encountered
     Bit 1: 1 if component requires no change (U run only)
     Bit 2: 1 if part is already processed (IG run only)
     Bit 3: 1 if part has no components (IG run only, and only if bit 2 is off)

     Besides its function as an indicator, bit 3 also serves to transfer information whether a particular part in an explosion sequence has component parts. Bit 3 is turned off in module 002 before entering module 003. If no component parts are found during the execution of module 003, the bit is turned on. Upon return to module 002, the bit is tested to decide whether module 004 must be called.

   - Bits 4 through 7: Error bits, extending completion bit 0. A single error bit does not reflect a particular error condition, therefore, the hexadecimal representation of the total bit pattern in the status byte has to be analyzed.

X'80'	Parent part not found
X'81'	Component part not found (U run only)
X'84'	Continuity check for parent part
X'85'	Continuity check for any component part
X'87'	Input parameter in error

X'88'	Unexpected DL/I status code for parts data base
X'8A'	Unexpected DL/I status code for control data base
X'8C'	Both error conditions X'84' and X'88'
X'8D'	Both error conditions X'85' and X'88'
X'8E'	Both error conditions X'84' and X'8A'
X'8F'	Both error conditions X'85' and X'8A'

Byte 7: Not used

3. Parameter list portion (fullword 2):

   Bytes 8 through 11: Address constant pointing to the parameter list which has been previously submitted to DL/I by LLC/CC in DL/I. Contents is defined hexadecimal zeros prior to the first run through LLC/CC in DL/I. The address constant is not affected by insertion of locators if the application program is written in PL/I.

4. PCB save area portion (fullword 3):

   Bytes 12 through 15: Address constant pointing to a 64-byte save area for a PCB. This save area is initialized to blanks (X'40'), however, in case of an unexpected DL/I status code, the related PCB is saved into this save area. The PCB is stored left justified. If the length of the PCB exceeds 64 bytes, the exceeding data is truncated.

The contents of the status bytes is externally represented by the return codes of LLC/CC in DL/I.

IG stands for "initial generation of low level codes", U stands for "update of low level codes".

The LECB is located at the very end of the code of LLC/CC in DL/I. Therefore, the last byte of LLC/CC in DL/I may be addressed DLZNNEC+15.

LANGUAGE CONSIDERATIONS

During PSB generation, the source language of application programs using DL/I facilities is defined in the PSBGEN statements. While COBOL is handled like Assembler, the PCB has a different layout if PL/I is specified. Therefore, LLC/CC in DL/I has to use different entry points into DL/I depending on the source language of the invoking user-written application program.

The entry routines of the execution program of LLC/CC in DL/I offer different entry points. The x identifies initial generation mode (G) or update mode (C). Six different entry points are available for transfer of control:

- DLZNNxA and DLZNNxC are the entry points for application programs written in Assembler or COBOL, respectively. No special processing is required.
- DLZNNxP are the entry points for application programs written in the PL/I Optimizer language. Upon entry, the address constants in the parameter list pointing to the locators of the parameters transmitted are replaced by the addresses which are stored in the respective locators.

For each source language, the appropriate language bit in the LLC/CC execution control block (LECB) is set upon entry.

When a DL/I call is issued, the language bits are tested to specify the right entry point in DL/I: ASMTDLI, CBLTDLI, or PLITDLI. If the source language is PL/I, the parameter list is encoded to transfer address constants pointing to locators rather than pointing directly to the parameters.

SAVE AREAS

LLC/CC in DL/I contains a set of save areas which facilitate tracing main storage dumps. The most important save areas are:

- Standard save area, addressed by register 13. Symbolic name is SAVE.
- Return addresses for subroutines, that is, contents of register 14. Symbolic names are CALLSV, PARMJUSV, INSRSAVE, SETUPSV, M002SV through M006SV. Save areas M002SV through M006SV are reset to hexadecimal zeros when the respective modules M002 through M006 are left again.
- Save area for the contents of register 1 when entering LLC/CC in DL/I, that is, address of the parameter list submitted from the application program. Symbolic name is R1SAVE.
- Save area for the leftmost 240 bytes of a PCB if an unexpected DL/I status code is encountered. Symbolic name is PCBSAVE. The address of PCBSAVE is also available in fullword 3 of the LECB.

REGISTER USAGE

R0:	Work register
R1:	Work register, address of parameter lists during parameter transfer
R2:	Address of parameter list when preparing parameter transfer
R5:	Work register
R6:	Address of PCB for parts data base
R7:	Address of PCB for control data base
R8:	Base register
R9:	Second base register
R12:	Reserved
R13:	Address of register save area
R14:	Standard return address
R15:	Standard linkage register

CHAPTER 15. DL/I TRACING FACILITY

DL/I offers a tracing facility as a tool to be used in problem determination.

The tracing facility consists of a trace module (DLZTRACE) with Assembler macro options to provide the type of tracing desired. After being link-edited into a core image library, the module is loaded by DL/I initialization if the user has specified tracing.

All DL/I action modules have a macro inserted into the source code at defined places (trace points) which generates the necessary instructions to communicate with DLZTRACE. On invocation, the tracing module decides if tracing should be made at the current point and, if so, records the significant information.

HOW TO USE THE TRACING FACILITY

The first step in using the DL/I tracing facility is to define what information should be traced in order to identify or solve the problem. In addition, the user must determine when tracing should occur; that is, at what trace points in the execution of the DL/I code information should be recorded and for which user calls. The following guidelines and examples should help in making these decisions.

TRACING IN A BATCH ENVIRONMENT

Which Calls to Trace in a Batch Environment

First determine which calls are to be traced. The decision will be one of the following types. Find the type and then note which trace macro operands should be coded. Refer to "Defining the Tracing Facility" for detailed syntax specification information. Note that only DL/I data base calls can be traced; PCB, TERM, UNLD, GSCD, and online system calls cannot be traced.

ALL calls: Do not code the CALLCON, STRTKEY, STOPKEY, or TRCECON operands.

SPECIFIC calls not necessarily sequentially issued to DL/I, but that have a similar characteristic. Determine if they fall into one of the classes below, and code the indicated CALLCON operand parameter:

	Similar Characteristic	Parameter List of the CALLCON Operand
1.	A similar value in the key feedback area of the PCB present at the beginning of each call.	Code the KEYFDBK parameter list.
2.	A specific PCB identified by the name of the DBD it references (the DBDNAME operand of the PCB macro in PSBGEN).	Code the DBPCBDBD parameter list.

3. A similar call function used by the calls. This option allows groups of calls to be identified, for example, all get calls, all get hold calls, or all update calls.	Code the CALLFUNC parameter list.
4. A combination of a specific PCB used in making the calls and the call function.	Code the DBPCBDBD and CALLFUNC parameter lists.

A RANGE of calls (for example, each of 10 successive calls) identified by either:

1. The relative call numbers of the first of the range and the last. For example, if the 6th through the 15th calls are to be traced, in the CALLCON operand, specify (6,LE,CALLNUM,LE,15).

2. The key feedback value in the PCB at the start of the first call of the range. Code the STRTKEY operand. In addition, the beginning of the range can be further qualified by the DBPCBDBD and/or CALLFUNC parameter lists of the CALLCON operand. To identify the last call of the range, specify the STOPKEY operand. In STOPKEY, specify either the number of calls to be traced, or the value of the PCB key feedback area at the end of the last call to be traced.

ONE SPECIFIC call: Identify this call in the same manner as the first call of a range as described above.

If the problem being traced does not occur with easily identifiable calls, but with certain internal DL/I conditions, another method is available to define when tracing should occur. The following specifications are possible:

1. Trace at certain points within DL/I execution whenever a specific physical file is being processed. For example, to perform tracing when the secondary index data base SINDEX@D is being used to service a call, specify TRCECON=(DDIRSYM,EQ,SINDEX@D).

2. Trace those calls that work with a particular segment type. For example, to perform tracing when the physical code of the current SDB (as found using JCBLEV1C and LEVSDB) is equal to X'0A' at the activated trace points, specify TRCECON=(SDBPHYCD,EQ,X'0A').

3. Trace those calls in which the RBA in PSTBYTNM is a specified value. For example, specify TRCECON=(00004120,LE,PSTBYTNM,LE,5530).

Combinations of the above operands are possible. For example, the following instructions can be given to the trace module: trace if DDIRSYM is DMBNAM@D and the segment in that data base has code X'03' and the current value in PSTBYTNM is greater than 00001280.

The parameter values are checked during the execution of the call. At some trace points, the values may not be meaningful. For example, at the very beginning of a user call the PSTBYTNM value may be zero, so no tracing would occur, unless the zero value happened to satisfy the PSTBYTNM value(s) specified in the tracing macro.

What to Trace in the Batch Environment

There are several ways to define what information should be traced at certain points of the DL/I execution. The selection of one or more of these options should be based on the type of problem that has occurred. For example:

1. If the problem has not yet been isolated as occuring within DL/I, or if the user call receives unexpected return information, the USERCALL option should be used to trace the input the application program is giving to DL/I and the resulting output. The problem could also be further investigated by specifying CURRPOS, which gives the user's current position within the data base.

2. If the problem has been isclated as occurring within the retrieve module (DLZDLR00) or concerns insert positioning, the RETRIEVE option provides information concerning the events of the call.

3. If the data base does not appear to have been updated as the user requested, the BHINTF and/or VSAMINTF options can be used to trace exactly which calls have been made to the DL/I buffer handler (DLZDBH00) and/or to VSAM. The return status in each case is given.

4. If secondary (cr primary) indexes appear to have been incorrectly updated, specify the INDEXTRC option. This provides information concerning the updates to index data bases.

5. If it is necessary tc trace the path the call takes through DL/I, the MODTRACE option lists the time of entry to each module.

If the situation does not fit any of the above and assuming that many calls need not be traced, all cf the options (except ONLINEBH) can be specified. Some options, however, such as RETRIEVE and CURRPOS, could result in large trace listings if many calls are traced. The CALLCON, STRTKEY/STOPKEY, and/or TRCECON operands may be used to omit the unnecessary or uninteresting calls. In addition a SHORT trace form is available for all cptions.

Batch Trace Output

The user should decide which storage medium shculd be used for the tracing cutput. Two possibilities are available:

1. The SYSLST parameter of the OUTPUT operand causes each trace entry to be printed cn SYSLST as it is being created. Thus, whenever a DL/I action module call to the trace facility is made, the information is printed on SYSLST before control is returned to the calling module. This option is the most useful for a debugging situation when no abnormal termination dump is expected.

2. The INCORE option of the OUTPUT operand builds a table, of user-specified size, to hold all trace entries. This table is never written onto any output device, except if a storage dump is produced. When the table is filled with entries, a wrap-around condition occurs and the oldest entry in the table is overlaid. A table large enough to hold at least one entry must be defined.

TRACING IN AN ONLINE ENVIRONMENT

The situation in an online environment is different from that in batch; the purpose of tracing may, therefore, also be different. There are two major reasons to trace online DL/I calls:

- To collect information in case a future error occurs by means of a general trace run continuously.

- To aid in debugging an cnline application program or to trace a known system failure by means of a specific trace.

General Online Trace

It is assumed that the user wants to trace <u>all</u> DL/I calls, and the CALLCON, STRTKEY/STOPKEY, and TRCECON operands should be omitted from the DLZTRACE macro generation.

The OPTION parameter selected depends on the suspected problem:

- If the timing seems to be the problem, in other words, tasks seem to be "suspended", the ONLINEBH option should be selected.

- If "bad" data base updates have occurred, the BHINTF and/or VSAMINTF options should be selected.

- The other options are available, but probably do not help solve a general problem.

To limit the amount of information traced at each trace point, the SHORT parameter of the TYPETRC operand may be specified. Performance degradation is to be expected when the tracing facility is activated.

Online Debug

If a known problem occurs on demand and can be isolated to a particular application program, the tracing facility can be used in a similar way to in the batch environment. Refer to the section "Which Calls to Trace in a Batch Environment" for instructions and examples. All of the batch information applies equally to the online environment with the following additional function.

In defining which calls are eligible to be traced in an online situation, an additional parameter list may be specified to limit the trace to only those calls made by a certain PSB. The operand CALLCON should be coded in the DLZTRACE generation with the parameters (PSBNAME,EQ,YOURPSB). In addition, the DBPCBDBD and CALLFUNC parameters may also be specified to limit the calls to be traced. This means, for example, that the following instructions can be given to the tracing facility: Trace only those calls made by PSBNAM1 and the PCB in that PSB identified by DBDNAM2 and whose call function is UPD (ISRT, DLET, or REPL).

Online Trace Output

In the normal online environment, that is when not debugging, the trace output can only be kept in storage. Be sure to indicate a table size large enough to keep all the entries needed.

Because the entries are not transferred to an output storage device, they are only available by means of a storage dump. The DL/I formatted dump program prints the latest 10 entries when a DL/I or CICS abnormal termination occurs.

If the online system is in a debug environment, the SYSLST output option may be used. This means that only one task making DL/I calls can be executing at any one time. If more than one task is executing, unpredictable results occur, because no provision is made to force single-threading of trace calls due to SYSLST I/O.

DEFINING THE TRACING FACILITY

The tracing facility is defined by specifying the desired operands in the DLZTRACE macro. The user may choose the type(s) of traces desired, when he wants tracing to occur, and the output destination. For the case when the user wants more direct control, a user exit routine may be provided.

TRACE DEFINITION MACRO

The tracing module DLZTRACE has the macro operands shown below. When coding the DLZTRACE macro, the general rule is to use an operand or operands from either group (1 or group (2 or group (3 together with an operand or operands from group (4. In group (3, code, as required, one or more of the operands CALLCON (with one or more parameters), STRTKEY, and STOPKEY. In group (4, code the OPTION operand, with at least one parameter and the remaining operands as required.

```
DLZTRACE  [CALLCON=([value1,ro1,]CALLNUM,ro2,value2),]                (1
         ------------------------------------------------------------
          [CALLCON=([key1,ro1,]KEYFDBK,ro2,key2),]                    (2
         ------------------------------------------------------------
          [CALLCON=[(PSBNAME,EQ,psbname),]                            (3
                   [(DBPCBDBD,EQ,dbdname),]
                   [(CALLFUNC,EQ,func),]]

          [STRTKEY=key,]
          [STOPKEY={key},]
                   {nn }
         ------------------------------------------------------------
             OPTION=[USERCALL]                                        (4
                   [,MODTRACE]
                   [,RETRIEVE]
                   [,CURRPOS]
                   [,VSAMINTF]
                   [,BHINTF]
                   [,INDEXTRC]
                   [,ONLINEBH]
         [,TYPETRC={FULL }]
                   {SHORT}
         [,TRCECON=[(DDIRSYM,EQ,dmbname)]
                   [,(SDBPHYCD,EQ,physcode)]
                   [,([rba1,ro1,]PSTBYTNM,ro2,rba2)]
                             {32}
          [,OUTPUT={(INCORE,{nn}))}]
                   { SYSLST       }
         [,TRACSIZ={256}]
                   {nn }
         [,USREXIT=entrypt]
```

The abbreviated parameters used above have the following general meanings:

- value1,value2, represent call numbers and must be decimal self-defining terms.
- ro1,ro2 represent relational operators, for example, GT, GE, LE, etc.

- key1,key2,key represent values in the key feedback area. The values can be expressed in either hexadecimal (X'...') or character (C'...') notation.

- physcode represents a segment code in hexadecimal notatation (X'nn').

- rba1,rba2 represent relative byte addresses. They are hexadecimal values but are coded without an X or quotes.

The following description of the macro operands is primarily syntactical. For a general discussion concerning the purpose of each operand and hints on how to use the facility, refer to "How to Use the Tracing Facility."

CALLCON Operand

This operand states the call conditions that must be satisfied before any tracing occurs during the call. There are five possible conditions that can be stated with the following parameters:

CALLNUM - the relative call occurrence, the first DL/I call being number 1. The value entries represent the call numbers and must be decimal self-defining terms. If a range of calls is desired, all five parameters must be specified. Value1 must be less than value2. Only relational operators LE or LT are valid for ro1 and ro2. If only one comparison value is required, the first two parameters must be omitted. In this case, the relational operator ro2 can be LT, LE, EQ, GE, or GT. If this operand is specified, none of the other four CALLCON parameters are permitted. In addition, any STRTKEY/STOPKEY specification is ignored.

KEYFDBK - the current value of the key feedback area in the user's PCB at the beginning of a call. The key value(s) can be expressed in either hexadecimal or character notation as indicated by the leading X or C respectively. The key itself must be enclosed in single quotes. The length of this operand value is limited by the DOS/VS Assembler to 256 bytes. If a range of keys is desired then all five parameters must be coded. Only LE or LT are permitted for ro1 and ro2. No checks are performed on the key values. If one key comparison value is required, the first two parameters must be omitted. The valid relational operators in this case are LE, LT, EQ, GT, or GE. If this parameter list is specified, then no other keyword list in the CALLCON operand is permitted. In addition, the STRTKEY/STOPKEY operands are ignored.

PSBNAME - the name of the PSB used by the task issuing the call. This is only meaningful in an online environment. The name specified must be from 1 to 7 characters in length. No check is made to determine whether the name is that of a valid PSB.

DBPCBDBD - the name of the DBD directly referenced by the PCB used for the call (the value of the DBDNAME operand in the PCB macro of PSBGEN). The name must be from 1 to 7 characters in length. No check is made to determine if it is the name of a valid DBD.

CALLFUNC - the call function specified for this call in the user's parameter list. The following functions are valid and result in tracing the named call functions:

G	-	GU, GN, GNP, GHU, GHN, GHNP
GH	-	GHU, GHN, GHNP
UPD	-	ISRT, DLET, REPL

In addition, any one of the actual DL/I call functions may be specified.

The last three parameter lists may be combined. All of the named CALLCON parameter conditions must be satisfied before the decision is made to trace the call.

STRTKEY Operand

The STRTKEY operand specifies the first call to be traced in a sequence of calls. It is ignored if either the CALLNUM or KEYFDBK parameter lists are specified in the CALLCON operand.

If any or all of the PSBNAME, DBPCBDBD, or CALLFUNC parameter lists are specified in the CALLCON operand, these conditions must be satisfied before the STRTKEY value is compared. If the key value specified is equal to the value in the PCB key feedback area at the beginning of a call, tracing is activated for the call and all succeeding calls until a STOPKEY condition is satisfied or until DL/I terminates.

The key value may be either hexadecimal or character format as indicated by the initial X or C respectively. The key value itself must be enclosed in single quotes. The length of the operand value is limited to 256 bytes by the DOS/VS Assembler.

STOPKEY Operand

This operand is used to stop call tracing that was started by the STRTKEY condition being fulfilled. It is ignored if CALLNUM or KEYFDBK parameters of the CALLCON operand are specified or if the STRTKEY operand is omitted. Every call after the one that started the sequence is checked to determine if the STOPKEY condition is met. While the STRTKEY condition is operative, CALLCON condition checks are not performed. Once the STOPKEY condition is satisfied, the STRTKEY condition is again checked.

The value of the operand may be a key value in hexadecimal or character notation, as indicated by the initial X or C respectively, enclosed in single quotes. This value is compared to the contents of the key feedback area in the PCB at the beginning of a call.

A decimal self-defining term, which indicates the number of calls to be traced from the first call that satisfies the STRTKEY conditions, may also be specified.

OPTION Operand

This operand may be specified with one or more parameters. Each option has a predefined set of fields or tables that are traced at predefined points in the DL/I code.

The options selected cause the corresponding trace points within the DL/I action modules to be eligible for activation based on any call condition parameters, as already described. When the trace module is called from one of these activated trace points, tracing occurs as defined for that option.

The following chart shows the general meaning of each option. For a detailed description of the exact fields to be traced, the format of the trace entry created, and the trace points, see section "Format of Trace Entries."

Options	What Traced	Trace Points
USERCALL interface to DL/I	DBPCB SSA I/O area	DLZDLA00 entry DLZDLA00 exit
MODTRACE action module trace	Time	Entry to all action modules
RETRIEVE for get calls	JCB (some fields)	DLZDLR00 selected points
CURRPOS current position information	LEVTAB (active) SDBs with SDBPOSC	DLZDLA00 exit
VSAMINTF interface to VSAM	Request to VSAM VSAM return information	DLZDBH00 after VSAM calls
BHINTF action module interface to buffer handler	Request to buffer handler Buffer handler return information	DLZDLR00 DLZDDLE0 DLZDLD00 DLZDXMT0 DLZDHDS0 } Before and after call to buffer handler
INDEXTRC index procedure	Request to index maintenance	DLZDXMT0 entry
ONLINEBH online trace	PST prefix BFFR prefix	DLZDBH00 selected points

TYPETRC Operand

This operand enables the user to shorten the amount of information traced. For each option, a subset is defined. For the description of the "full" and "short" traces as defined for each option, refer to the trace entry format descriptions in the section "Format of Trace Entries." The full trace definition is the default.

TRCECON Operand

The operand allows the user to eliminate tracing for any activated trace points based on internal DL/I conditions. Unless the specified conditions are satisfied at a particular trace point, tracing does not occur. The first processing step at every activated trace point is to check these conditions, which must be satisfied before tracing is performed. One or more parameters may be specified.

DDIRSYM - the current name of the data base as indicated via PSTDSGA is compared with the specified DMD name. The name must be 8 bytes in length and end with the letter 'D'.

SDBPHYCD - the current segment code as found in SDBPHYCD of the SDB corresponding to the level indicated in JCBLEV1C is compared to the user value. The physical code must be of the form X'nn' where nn is the hexadecimal value of the physical code.

PSTBYTNM - the current value of PSTBYTNM is compared as indicated in this parameter. A range of RBAs or a comparison of one value may be specified. The rba values can be from 1 to 8 hexadecimal characters without an X or single quotes. The values are right-justified if there are less than 8 characters. If a range is desired, only LE or LT may be used as relational operators ro1 and ro2. If a range is not desired, then the first two parameters must be omitted. In this case, ro2 may be LT, LE, EQ, GE, or GT.

OUTPUT Operand

The OUTPUT operand controls the destination of the tracing results. There are two possibilities:

SYSLST - each trace entry is printed on SYSLST at the time it is created. This option may not be specified in an online environment, unless the system is in a debug mode with only one DL/I task active at one time.

(INCORE,nn) - the tracing entries are kept in a table in virtual storage. If the table becomes full, a wrap-around condition occurs and the oldest entry is overlaid. 'nn' specifies the approximate number of entries to be kept in the table; it can be a decimal value between 1 and 32767. This number is multiplied by the TRACSIZ value to determine the size of the table. 32 is the default. When this default is multiplied by the default 256 of the TRACSIZ operand, a table size of 8K results, which for most options acutally provides space for 200 to 300 entries. If the calculated INCORE table size is not available in virtual storage, half of the amount is requested repeatedly until space is obtained. If no space is available, no tracing is performed.

When the INCORE option is chosen, the resulting table is built from the "bottom up." That is, the first entry is placed in the high address space. The header portion is then followed by the data portion of the entry. The necessary addresses and lengths for this table are stored in the SCD.

If the formatted dump option and the INCORE option have both been selected, the formatted dump routine invokes the trace table print routine to format and print the latest 10 entries of this table.

TRACSIZ Operand

The TRACSIZ operand specifies the maximum decimal number of bytes (nn) required to build the largest trace entry for the options selected. The size of the trace entry header need not be included. Refer to the format description of the individual trace entries in the section "Format of Trace Entries" for information on how to determine the size. The value specified must be between 8 and 4096. 256 bytes is the default and for most cases should be enough.

If the INCORE parameter of the OUTPUT operand is specified, this number is used to calculate the size of the table. Only that space which is required to build the trace entry is actually used. If the resulting table size is not large enough to hold one particular entry, that entry is omitted from the table.

If the SYSLST option is selected, the TRACSIZ value is the size of the work area used to create each trace entry. Any trace entry which results in a length larger than this size is not printed.

If the third user exit option is used to create trace entries, the value of the TRACSIZ operand is used to acquire work space for the user exit routine in which to build its trace entry.

USREXIT Operand

Because of the difficulty in creating a generalized trace module that could track all situations that may arise in any user environment, the ability is offered to the user to control and perform tracing for his specific conditions by way of user exits.

Three major decisions are made by the DLZTRACE module which the user may influence. These are:

1. Is the current call eligible for tracing?

2. Is the current trace point activated?

3. What information should be traced at this selected trace point?

The entry point name of the user exit routine must be specified in the DLZTRACE macro in the USREXIT operand. The exit routine, which should be written in Assembler language, need not be reentrant. It must not issue any supervisor or I/O macros.

The exit routine should be assembled and cataloged into the relocatable library. When the DLZTRACE module is link-edited, this module must be included via AUTOLINK or an INCLUDE statement.

The trace module performs as indicated in the macro parameters. If an optional user exit routine is specified, then there are three points at which it can be given control. These are described in detail below.

The exit routine programmer is responsible for knowing what fields or table entries are valid at the time the exit routine receives control. Therefore, a good knowledge of the internal logic of DL/I is necessary to use the trace user exit functions.

User Exit Interface

The interface to the user exit routine is the same for all three exit points.

The input registers are:

R1 = parameter list address
R13 = trace module save area address
R14 = trace module return address
R15 = user routine entry point address.

On entry, the user routine must first store the trace module registers in its save area, then set up its own register save area in register 13 and chain it off the input save area. Before the user exit routine returns control to the trace module, all registers except register 15 must be restored from the trace module save area.

The output register contents must be:

R1 = parameter list address
R13 = trace module save area address
R15 = return code.

The exact meaning of the return code and the parameter list values are described with each of the user exits. Only those values that are mentioned are available or have meaning for the particular exit.

The parameter list contains both input information for the user exit routine and can contain some output information.

User Exit Parameter List Format

Hex	Dec	Name	Ln	Description
0	0	TREXPST	4	Address of current PST.
4	4	TREXCALL	4	Address of action module trace call parameter list. The format is described in "Format of Trace Entries."
8	8	TREXADAT	4	Address of the trace entry to be constructed by user exit 3.
C	12	TREXLDAT	2	Length of above trace entry.
E	14	TREXNUM	1	The current user exit to be processed: 0 = Exit 1 4 = Exit 2 8 = Exit 3
F	15	TREXFLAG	1	Flag byte: • If TREXFDEC flag is on, the trace module decided to trace the current call. • If off, the call is not to be traced.

User Exit Description

EXIT 1: The user receives control after the trace module has decided to activate or deactivate all selected trace points for this call. The decision is based on the specifications in the CALLCON and/or STRTKEY/STOPKEY operands.

Input parameter list:

TREXPST contains PST address
TREXNUM contains 0
TREXFLAG has flag TREXFDEC on if the trace module has decided to trace the call; off, if it decided not to trace call.

Output:
No change to parameter list
R15 = 0 use trace module decision
R15 = 4 reverse trace module decision.

EXIT 2: The user receives control after the decision has been made to trace at this point. Any TRCECON parameters have already been checked. This exit is only invoked if the result of the call condition test (including any user exit 1 decision) was to trace. If the trace point has been defined to process more than one option, user exits 2 and 3 are given only one exit each. If the decision at user exit 2 is not to trace, then no tracing occurs at all. If user exit 3 performs tracing, no other tracing is performed at the trace point.

Input parameter list:

TREXPST contains PST address
TREXCALL contains trace point parameter list address
TREXNUM contains 4
TREXFLAG has flag TREXFDEC on if the trace module decided to trace at this point; off, if it decided not to trace.

Output:
No change to parameter list
R15 = 0 use trace module decision
R15 = 4 reverse trace module decision.

EXIT 3: The user exit receives control before any actual tracing is performed for the specified trace point. If the trace point has been defined to process more than one option, user exits 2 and 3 are given only one exit each. If the decision at user exit 2 is not to trace, then no tracing occurs at all. If user exit 3 performs tracing, no other tracing is performed at the trace point.

Input parameter list:

TREXPST contains PST address.
TREXCALL contains the trace point parameter list address.
TREXADAT contains the address of the area to be used by the user exit to build a trace entry.
TREXLDAT contains the length of TREXADAT as specified in the TRACSIZ operand of DLZTRACE.
TREXNUM contains 8.

Output:
R15 = 0 trace module should do tracing
R15 = 4 exit routine has done tracing
If R15 return code is 4, then TREXLDAT must contain the length of the trace entry built.

The user exit routine is responsible for using only the size of the user area that is allotted in TREXLDAT at entry to the routine.

If the user performed the trace with the SYSLST option in operation, the trace module prints the standard header followed by the unformatted trace data in both hexadecimal and character format.

INVOKING THE TRACING FACILITY

In the batch environment, tracing is invoked by coding the TRACE parameter in the DL/I parameter card.

```
DLI,pgmname,psbname,buff[,TRACE=modname]
```

The named module is loaded from the core image library by batch initialization. Tracing is performed for the program execution according to the parameters specified in the generation of DLZTRACE.

In the online environment, several trace modules may be loaded at initialization by entering their names into the CICS/VS PPT. Actual selection of the trace module to be used is done by the TSTR system call, see below.

ONLINE TRACING CONTROL

To control tracing in the online environment, two system calls are available. After the defined tracing modules have been link-edited into the core image library, a user-written application program can issue the TSTR (Trace Start) system call naming the phase name of the desired trace module. This causes the tracing module to be loaded and activated. Tracing then begins with the next call to DL/I that satisfies the user-specified conditions.

To disable the tracing facility, the TSTP (Trace Stop) system call is used. If the trace entries are being accumulated INCORE when TSTP is issued, the space for the table is released and is no longer available.

Chapter 10 of DL/I DOS/VS Utilities and Guide for the System Programmer contains details of the formats and return conditions of the TSTR and TSTP calls in the section "DL/I System Call Format and Returns." Chapter 10 also includes an example of the use of the TSTR and TSTP calls.

TRACE INVOCATION MACRO FOR ACTION MODULES

The macro DLZTRCAL is inserted into the DL/I action modules at defined points. This macro expansion first checks if tracing is enabled. This can be determined by checking a byte in the SCD which indicates if the trace module is loaded and if this trace point is activated. If tracing is not enabled, normal processing continues. If tracing is enabled, a parameter list is passed to the tracing module with the following information:

Hex	Dec	Name	Ln	Description
0	0	TRCMODNM	4	Calling module name
4	4	TRCMODID	1	Trace point ID
5	5	TRCNOPT	1	Number of options following
6	6	TRCTOPT	1	Trace option for this point. More than one is possible.

The trace module then makes the final check as to whether tracing should be performed at this point or not. This includes checking the TRCECON operand values and abiding by the results of any user exit routine. After processing, control is returned to the calling module.

A special expansion of DLZTRCAL is used to request the tracing module to evaluate any call-tracing conditions as specified in CALLCON, STRTKEY, or STOPKEY. This is not a trace point and no tracing can occur. This macro is placed before the first trace point in the call analyzer.

DLZTRCAL is also used to request the tracing function to purge any buffers and to free acquired storage. This is done by the online program request handler, when a TSTP call is received, and by system termination.

The operands of DLZTRCAL are:

TYPE=TRACE — This is the default to indicate that tracing occurs at a predefined trace point.

CKCALL — This is a special invocation to indicate the trace module should check the call conditions to determine if the call should be traced.

START — This is a special invocation of the trace module that causes it to intialize itself.

STOP — This is a special invocation to stop tracing.

CALLER=cccc — The identification of the calling module. This should be characters 4-7 of the official name and must be specified if TYPE=TRACE.

ID=nn — This is a decimal number from 1 to 255 and uniquely identifies the trace point. It is required if TYPE=TRACE.

OPTION=USERCAL1
USERCAL2
MODTRACE
RETRIEVE
CURRPOS
VSAMINTF
BHINTF1
BHINTF2
INDEXTRC
ONLINEBH — This is the trace option serviced at this trace point. One is required if TYPE=TRACE, but more than one may be specified.

PSTREG=REG — This is the number or symbolic name of the register that contains the PST address. If the entry is omitted, 1 is assumed. If TYPE=START or STOP, the register contains the SCD address.

Before DLZTRCAL is executed, addressability is required to the SCD.

- If TYPE=TRACE, no registers are changed.
- If TYPE=CKCALL or STOP, the contents of register 15 are destroyed by this macro.
- If TYPE=START, register 15 contains a return code:
  0 = no error, tracing is initialized.
  4 = GETVIS failure, SIZE parameter omitted from EXEC statement.
  8 = GETVIS failure, program is executing in real mode.
  12 = GETVIS failure, no storage available.
- If TYPE=START, the entry point address of the trace module must be in register 15 before the macro is executed.

FORMAT OF TRACE ENTRIES

The contents of a trace entry vary according to the option(s) selected. Each entry is variable length with the following header fields. No boundary alignment is guaranteed for any field. The macro DLZTRENT can be used to generate DSECTs for the trace entries.

Trace Entry Header Format

TRACEHDR DSECT

Hex	Dec	Name	Ln	Description
0	0	TRHDELEN	2	Length of trace entry (including header).
2	2	TRHDCALN	2	Call number relative to 1.
4	4	TRHDMODN	4	Name of action module that caused tracing.
8	8	TRHDTRID	1	Trace point ID in action module.
9	9	TRHDFUNC	1	Code for user's call function: 00 = GU 01 = GN 02 = GNP 03 = GHU 04 = GHN 05 = GHNP 06 = ISRT 07 = DLET 08 = REPL
A	10	TRHDCODE	1	Type of trace entry

Name	EQU	Meaning
TRHDUC1	X'01'	USERCALL-Type 1 (start of call)
TRHDUC2	X'02'	USERCALL-Type 2 (end of call)
TRHDAMOD	X'10'	MODTRACE
TRHDRETR	X'20'	RETRIEVE
TRHDCPOS	X'30'	CURRPOS
TRHDVSAM	X'50'	VSAMINTF
TRHDBH1	X'61'	BHINTF-Type 1 (before call)
TRHDBH2	X'62'	BHINTF-Type 2 (after call)
TRHDINDX	X'70'	INDEXTRC
TRHDOLBH	X'80'	ONLINEBH

B	11	TRHDFLAG	1	Flag byte

Name	EQU	Meaning
TRHDSHRT	X'80'	This trace entry is short form
TRHDUEX1	X'40'	User exit 1 reversed decision
TRHDUEX2	X'20'	User exit 2 reversed decision
TRHDUEX3	X'10'	User exit 3 did tracing

The remainder of each entry depends on which option is currently being traced. Following are the formats of all trace entries and the trace points within the DL/I action modules which cause these entries to be created.

__USERCALL-Type 1 Trace Entry Format (Code X'01')__

TRUSRCL1 DSECT (Start of call)

Hex	Dec	Name	Ln	Description
0	0	TRU1TIME	8	Time call made (from STCK instruction)
8	8	TRU1PCBL	2	Length of DBPCB including key feedback
A	10	TRU1IOLN	2	Length of I/O area (0 if not traced)
C	12	TRU1SSAL	2	Length of SSAs (0 if none)
E	14	TRU1PCB	Var	DBPCB with key feedback
		TRU1IOAR	Var	I/O area contents (not traced for GET call)
		TRU1SSA	Var	SSAs.

__Note__: No short form of this trace entry occurs. The call function code in the trace entry header serves as the short form.

__I/O AREA TRACING__ is only performed for ISRT, DLET, or REPL calls. The length of the I/O area traced is the largest of:

- Longest segment length (maximum if variable)
- Longest concatenated segment length
- Longest path call length.

__SSAs__: If more than one SSA is present, at least one blank separates them.

__Trace Point__:

- DLZDLA00 (Call analyzer): At the beginning of user call validation. (Some user errors could be detected before this trace point, in which case no tracing takes place.)

USERCALL-Type 2 Trace Entry Format (Code X'02')

TRUSRCL2 DSECT (End of call)

Hex	Dec	Name	Ln	Description
0	0	TRU2STC	2	Returned status code (DBPCBSTC)

Note: Short form of entry ends at this point.

2	2	TRU2TIME	8	Time at end of call
A	10	TRU2PCBL	2	Length of DBPCB
C	12	TRU2IOLN	2	Length of I/O area (0 if none)
E	14	TRU2PCB	Var	DBPCB including key feedback
		TRU2IOAR	Var	I/O area contents

I/O AREA TRACING is only performed when data is returned to the user, in other words, for get calls. The length of the I/O area is the length of the data returned to the user.

Trace Point:

- DLZDLA00 (Call analyzer): At the end of processing before control is given to PRH.

MODTRACE Trace Entry Format (Code X'10')

TRMODTRC DSECT

Hex	Dec	Name	Ln	Description
0	0	TRMDTIME	8	Time at trace point (from STCK instruction)

Trace Points:

- DLZDLA00 (Call analyzer): On entry after CKCALL
- DLZDLR00 (Retrieve)
- DLZDLD00 (Delete/Replace)
- DLZDDLE0 (Load/Insert)
- DLZDBH00 (Buffer handler)
- DLZDHDS0 (Space management)
- DLZDXMT0 (Index maintenance)

On entry

- In buffer handler before VSAM call (module ID is 'VSAM')

Note: The short and full trace entries are identical.

RETRIEVE Trace Entry Format (Code X'20')

TRRETRV DSECT

Hex	Dec	Name	Ln	Description
0	0	TRRTJCD	1	JCB flags (JCBCODE)
				X'80' On Log Child-Log Parent insert, Log Parent is present. X'40' Deferred delete required. X'20' Retrieve deleted segments.
1	1	TRRTJL1C	1	Level number pointed to by JCBLEV1C
2	2	TRRTLVT	4	Input for DLZSKPG (JCBLVT...)
6	6	TRRTBGBF	4	Retrieve (BEGBUF)
A	10	TRRTCTTR	4	Work (CURTTR)
E	14	TRRTPRSW	4	Fields (PROCSW)
12	18	TRRTKPIT	4	(KEEPIT)
16	22	TRRTINDG	1	VL-UDC flags (DSGINDG)
17	23		1	Reserved
18	24	TRRTREGS	60	Contents of registers 14-12

Trace Points:

- DLZTAG when completing a level
- DLZSSA when completing a level
- DLZLTW when accepting a level
- DLZEODC end of data base condition
- DLZGER not found condition
- DLZREG insert positioning

Note: The short and full trace entries are identical.

CURRPOS Trace Entry Format (Code X'30')

TRCURPOS DSECT

Hex	Dec	Name	Ln	Description
0	0	TRCPCLEV	1	Current level number (LEVLEV)
1	1	TRCPCPC	1	Current segment code (LEVPC)
2	2	TRCPLTTR	4	Current position (RBN or RBA) (LEVTTR)

Note: Short form of entry ends at this point.

Hex	Dec	Name	Ln	Description
6	6	TRCPNOLV	1	Number of level entries in TRCPLEVL

7	7	TRCFNSDB	1	Number of SDB entries in TRCPSDBS
8	8	TRCPLEVL		Level table information
8	8	TRCPSDBS		SDB entry information

The first three fields contain information for the current level only (as found via JCBLEV1C). The level table information is traced for every active level plus 1. TRCPNOLV contains the number of levels traced. The format of the level entries as found beginning in TRCPLEVL is:

TRLEVEL DSECT

Hex	Dec	Name	Ln	Description
0	0	TRLVLEV	1	Level number (LEVLEV)
1	1	TRLVSGOF	2	Offset to segment (LEVSEGOF)
3	3	TRLVTTR	4	Current RBN or RBA (LEVTTR)
7	7	TRLVF1	1	Flag byte (LEVF1) X'80' Segment at this level newly deleted. X'40' This level table entry empty. X'20' Segment at this level in hold status. X'10' Segment at this level in hierarchical path. X'08' Segment at this level moved to user. X'04' Segment is last of type for parent. X'02' Segment is first of type for parent. X'01' This is the last level table for PCB.
8	8	TRLVF2	1	Flag byte (LEVF2) X'80' Used by retrieve. X'40' Level has not found position for higher level. X'20' EOD flag. X'10' LEVTAB has been modified. X'08' Used by retrieve. X'04' Used by retrieve. X'02' Used by retrieve. X'01' Used by retrieve.
9	9	TRLVUSOF	2	Offset to segment in I/O area (LEVUSEOF)
B	11	TRLVF3	1	Flag byte (LEVF3) X'80' X'40' X'20' X'10' X'08' This is a pseudo SSA filling gap. X'04; At least one member qualified

				on data.
				X'02' Every boolean set has at least one key field.
				X'01'

For each level, except the last, information from the corresponding SDB is traced. In addition, if the corresponding SDB has a target (nonzero SDBTARG) and the target has a current position (nonzero SBDPOSC), then the target is also traced. This means that for one level, one or more SDBs may be traced. Each target SDB recurs in the list immediately after its parent SDB. A target SDB is easily identified because it has no name (TRSDBSYM).

TRSDB DSECT

Hex	Dec	Name	Ln	Description
0	0	TRSDBSYM	8	Segment name (SDBSYM)
8	8	TRSDBF3	1	Flag byte (SDBF3) Call sensitivity
				X'80' Sensitivity is read only.
				X'40' Sensitivity is insert.
				X'20' Sensitivity is replace.
				X'10' Sensitivity is delete.
				X'08' Sensitivity is key only.
				X'04' Sensitivity is path only.
				X'02' Sensitivity is exclusive.
				X'01 Sensitivity is load.
9	9	TRSDBF4	1	Flag byte (SDBF4)
				X'40' Secondary index is main processing sequence.
				X'10' Field is in destination parent.
				X'04' CI-split in HISAM KSDS.
				X'02' Position lost.
				X'01' Field is in logical child.
				X'01' Temporary SW for replace. Data changed.
A	10	TRSDBPC	1	Physical code (SDBPHYCD)
B	11	TRSDBTFG	1	Target relationship code (SDBTFLG)
				X'C0' Segment is physical parent of target of SDBPARA.
				X'80' Segment is physical parent of SDBPARA.
				X'40' Segment is physical child of target of SDBPARA.
				X'20' Segment points to logical parent with physical key.
				X'10' SDB is a generated SDB or a SDB for a physical pair.
				X'08' Segment points to physical parent.
				X'04' Segment is retrieved via index.
				X'02' Segment points to logical child.
				X'01' Segment points to logical parent.

C	12	TRSDBPRV	4	Previous position (SDBPOSP)
10	16	TRSDBCUR	4	Current position (SDBPOSC)
14	20	TRSDBNXT	4	Next position (SDBPOSN)

Trace Point:

- DIZDIA00 (Call anlayzer): At exit from call analyzer.

VSAMINTF Trace Entry Format (Code X'50')

TRVSAMIF DSECT

Hex	Dec	Name	Ln	Description
0	0	TRVSREQ	1	Request (RPLREQ) X'00' Point request. X'04' Get request. X'08' Erase request. X'0C' Put request. X'0C' Update request. X'10' Insert request. X'14' Check. X'18' ENDREQ. X'1C' FORCIO. X'20' Verify. X'24' Put locate.
1	1	TRVSFBK1	1	Return code =(RPLFDBK1) X'00' No error detected. X'04' Concurrent request on same RPL. X'08' Logical error. X'0C' Physical error.
2	2	TRVSFBK3	1	Return code =(RPLFDBK3) Returns that are not errors (RPLFDBK1 = 0). X'04' EOV called during request. Logical errors (RPLFDBK1 = 8). X'04' End of data set reached. X'08' Duplicate record. X'0C' Sequence error. X'10' No record found. X'14' Data ALR in exclusive control. X'18' Volume is not mounted. X'1C' Data set cannot be extended. X'20' Invalid RBA specified. X'24' No key range specified for record. X'28' Insufficient virtual storage. X'2C' User buffers too small. X'40' PLH in use (No string available). X'44' Access type not requested at open. X'48' Keyed request for ESDS.

				X'4C' Address or CNV insert for KSDS. X'50' Invalid erase request. X'54' Invalid specification of locate mode. X'58' Positioning error. X'5C' No get UPD issued. X'60' Key change for update. X'64' Length change for address update. X'68' Invalid or conflicting RPL option specified. X'6C' Improper record length specified. X'70' Improper generic key length specified. X'74' Invalid request during data set loading. Physical errors (RPLFDBK1 = 12). X'04' Read error in data set. X'08' Read error in index set. X'0C' Read error in sequence set. X'10' Write error in data set. X'14' Write error in index set. X'18' Write error in sequence set.

Note: Short form of entry ends at this point.

3	3	TRVSLARG	1	Length of TRVSARG
4	4	TRVSRBA	4	RBA returned (RPLRBA)
8	8	TRVSAREA	4	Pointer area (RPLAREA)
C	12	TRVSTIME	8	Clock time call completed
14	20	TRVSOPCD	2	Option codes (RPLOPTCD) First option byte equates: X'80' Keyed access. X'40' Addressed access. X'20' Sequential. X'10' Direct processing. X'08' Asynchronous. X'04' Skip sequential access. X'02' CINV access (by RBA). X'01' Update. Second option byte equates: X'80' Search key greater than or equal to. X'40' Generic key request. X'20' Note string position. X'10' No update. X'08' Locate mode. X'04' User buffers.
16	22	TRVSARG	Var	RBA or key requested (from RPLARG)

Trace Point:

- DLZDBH00 (Buffer handler): After VSAM GET and VSAM PUT.

BHINTF-Type 1 Trace Entry Format (Code X'61')

TRBHTO DSECT (Before call)

Hex	Dec	Name	Ln	Description
0	0	TRBHTFUN	1	Caller's request (PSTFNCTN) Equates for buffer handler (DLZDBH00) function code:
				X'01' Locate control interval with C-I RBA. If HD, locate a data C-I by RBA pointing to a segment. If HISAM or HIDAM INDEX, read a record by RBA from a KSDS. If HISAM, read a record by RBA from an ESDS.
				X'03' Get buffer space.
				X'04' Free buffer space.
				X'04' Mark buffers empty.
				X'05' If HD, mark a buffer containing data altered. If HISAM or HIDAM INDEX, write a record by RBA to a KSDS. If HISAM, write a record by RBA to an EDS.
				X'06' Locate a C-I with an RBA pointing to a segment and mark buffer altered.
				X'07' Purge all buffers altered by a task.
				X'08' Write new record to HISAM ESDS.
				X'09' Get record from KSDS equal or high.
				X'0A' Erase a record in a KSDS.
				X'0B' Get next record from KSDS.
				X'0C' Get first record of a KSDS.
				X'0D' Insert record into KSDS by key.
				X'0E' Insert record sequentially into KSDS.

Note: Short form of entry ends at this point.

1	1	TRBHTKYL	1	Length of key at TRBHTKEY
2	2	TRBHTDMB	2	DMB number (PSTDMBNM)
4	4	TRBHTBLK	4	Relative block number (PSTBLKNM)
8	8	TRBHTBYT	4	RBA or RBN (PSTBYTNM)
C	12	TRBHTDAT	4	Address of data in buffer (PSTDATA)
10	16	TRBHTDSG	4	Address of DSG portion of JCB (PSTDSGA)
14	20	TRBHTKEY	Var	Key if PSTFNCTN = PSTSTLEQ

Note: If PSTFNCTN=PSTSTLEQ, PSTBYTNM contains the address of the key to search for.

Trace Points:

- DLZDLR00 (Retrieve)
- DLZDLD00 (Delete/Replace)
- DLZDDLE0 (Load/Insert)
- DLZDXMT0 (Index maintenance)
- DLZDHDS0 (Space management)

Before call to buffer handler

BHINTF-Type 2 Trace Entry Format (Code X'62')

TRBHFRCM DSECT (After call)

Hex	Dec	Name	Ln	Description
0	0	TRBHFRC	1	Buffer handler return code (PSTRTCDE)

X'00'	All OK.
X'04'	RBN is beyond end of data set.
X'08'	I/O error.
X'08'	Permanent read error.
X'0C'	No space in data set for additions.
X'10'	An illegal call was made.
X'14'	No record found (Retrieve by key).
X'18'	New block was created in buffer pool.
X'1C'	Not enough space in buffer pool.
X'20'	Size of requested buffer exceeds size of buffers in any subpool.
X'24'	End of data set. No record returned.
X'28'	Key or RBA higher than highest key or RBA in data set.
X'2C'	End of data set reached on a request issued by OPEN.

Note: Short form of entry ends at this point.

Hex	Dec	Name	Ln	Description
1	1	TRBHFDMB	2	DMB number (PSTDMBNM)
3	3	TRBHFOFF	2	Offset to RBA from PSTDATA (PSTOFFST)
5	5	TRBHFBLK	4	Block number (PSTBLKNM)
9	9	TRBHFBYT	4	RBA (PSTBYTNM)
D	13	TRBHFDAT	4	Address of data in buffer (PSTDATA)
11	17	TRBHFBFA	4	Address of buffer prefix (PSTBUFFA)

Trace Points:

- DLZDLR00 (Retrieve)
- DLZDLDC0 (Delete/Replace)
- DLZDDLE0 (Load/Insert)
- DLZDHDS0 (Space management)
- DLZDXMT0 (Index maintenance)

After call to buffer handler

INDEXTRC Trace Entry Format (Code X'70')

TRINDEX DSECT

Hex	Dec	Name	Ln	Description
0	0	TRIXFNCT	1	Request to index maintenance (PSTFNCTN) Perform indicated maintenance for segment to be: X'A0' Deleted. X'A1' Replaced. X'A2' Inserted. X'A3' Unloaded.
1	1	TRIXWKT4	1	Special request (PSTWRKT4 - 1ST BYTE) X'02' Delete only primary index. X'03' Do not delete primary index. X'04' Physical delete bit is set.

Note: Short form of entry ends at this point.

Hex	Dec	Name	Ln	Description
2	2	TRIXDMB	2	DMB number (PSTDMBNM)
4	4	TRIXBYT	4	ISS RBA (PSTBYTNM)
8	8	TRIXSC	1	Segment code of ISS (DMBSC)

Trace Point:

- DLZDXMT0 (Index maintenance): At beginning.

ONLINEBH Trace Entry Format (Code X'80')

TRONLINE DSECT

Hex	Dec	Name	Ln	Description
0	0	TROLIND	1	Schedule & dispatch indicator (PPSTIND) X'80' Waiting for I/O. X'40' Cannot schedule due to segment INT.

				X'20' Cannot schedule due to task count. X'10' Task enqueued by buffer handler. X'04' This is current task. X'01' Task scheduled.
1	1	TROLID	1	Task ID (PPSTID)

Note: Short form of entry ends at this point.

2	2	TROLECB	2	ECB (PPSTECB)
4	4	TRCIEXCI	2	Enqueue/dequeue Pointers for Existing CI (PPSTEXCI+2)
6	6	TROLPECI	2	Enqueue/dequeue Pointers for Pending CI (PPSTPECI+2)
8	8	TROLSUPO	2	Enqueue/dequeue Pointers for Subpool Space (PPSTSUPO+2)
A	10	TRCLMATR	2	Enqueue/dequeue Pointers for Matrix Space (PPSTMATR+2)
C	12	TROLCHAI	2	Chain field for pending CI (PPSTCHAI+2)
E	14	TROIPST	2	PST pointers for Enqueue/dequeue (BFFRPST)
10	16	TROLNPST	2	PST pointers for Enqueue/dequeue (BFFRNPST)
12	18	TROLCIID	6	CI identfication (BLKNUM-DMBNM) (BFFRCIID)
18	24	TRCISW	1	Switch (BFFRSW) X'80' Buffer on write chain. X'40' Buffer being written. X'20' Buffer being read. X'10' Buffer empty. X'08' Buffer waiting for PRED being written. X'04' Buffer has permanent read error. X'02' Existing CI ID enqueued. X'01' Pending CI ID enqueued.
19	25	TROLFUNC	1	Caller's request (PSTFNCTN)
1A	26	TROLBLKN	4	Block number (PSTBLKNM)

Trace Points: In buffer handler:

- In PSEUDINT routine: before WAIT is issued for a task because the interlock detection matrix is full.
- In ISSWAIT routine: before WAIT is issued for a task because it requested a buffer owned by another task.
- In AFTW routine: before WAIT is issued for a task because it needed a data base already in use (ACB busy).

APPENDIX A. SYSTEM MESSAGE-MODULE CROSS REFERENCE

The following table shows in which modules each of the DL/I messages is issued. Module names are identified and defined elsewhere in this publication.

Message Number	Module
DLZ001	DLZBNUC0
DLZ002	DLZBNUC0
DLZ003	DLZDDLE0
DLZ004	DLZDBH00
DLZ005	DLZDBH00
DLZ007	DLZDSEH0, DLZDXMT0
DLZ008	DLZRRC00
DLZ009	DLZRRC00
DLZ010	DLZRRC00, DLZMPI00
DLZ011	DLZRRC00
DLZ012	DLZMPI00, DLZRRC00
DLZ014	DLZRRC00, DLZMPI00
DLZ015	DLZRRC00
DLZ017	DLZRRC00
DLZ018	DLZRRC00
DLZ019	DLZRRC00
DLZ020	DLZDLOC0
DLZ021	DLZDLOC0
DLZ022	DLZDLOC0
DLZ023	DLZDLOC0
DLZ024	DLZDLOC0
DLZ025	DLZDLOC0
DLZ026	DLZRRC00
DLZ027	DLZDLOC0
DLZ028	DLZDLOC0
DLZ029	DLZOLI00
DLZ030	DLZOLI00
DLZ040	DLZOLI00
DLZ041	DLZOLI00
DLZ042	DLZOLI00
DLZ043	DLZOLI00
DLZ044	DLZOLI00
DLZ045	DLZOLI00
DLZ046	DLZOLI00
DLZ047	DLZOLI00
DLZ048	DLZOLI00
DLZ049	DLZOLI00
DLZ050	DLZOLI00
DLZ051	DLZOLI00
DLZ052	DLZOLI00
DLZ053	DLZOLI00
DLZ054	DLZOLI00
DLZ055	DLZOLI00
DLZ056	DLZOLI00
DLZ057	DLZOLI00
DLZ058	DLZOLI00
DLZ060	DLZOLI00
DLZ061	DLZOLI00
DLZ062	DLZODP

Message Number	Module
DLZ063	DLZODP
DLZ064	DLZCLI00
DLZ066	DLZODP
DLZ067	DLZODP
DLZ068	DLZODP
DLZ069	DLZODP
DLZ070	DLZODP
DLZ071	DLZOLI00
DLZ072	DLZOLI00
DLZ073	DLZCLI00
DLZ074	DLZOLI00
DLZ080	DLZMSTP0
DLZ081	DLZMPI00
DLZ082	DLZBPC00, DLZMPC00, DLZMPI00
DLZ083	DLZMSTR0
DLZ084	DLZBPC00, DLZMPC00, DLZMPI00
DLZ085	DLZMPI00
DLZ086	DLZMPC00
DLZ087	DLZMPI00
DLZ088	DLZMPC00
DLZ089	DLZMPI00
DLZ090	DLZMPI00
DLZ091	DLZMPI00
DLZ092	DLZMPI00
DLZ093	DLZMPC00
DLZ094	DLZMPC00
DLZ095	DLZMPI00
DLZ096	DLZMPI00
DLZ097	DLZMSTR0
DLZ098	DLZMPI00
DLZ099	DLZMPI00
DLZ100	DLZMPI00
DLZ101	DLZMPC00, DLZMSTR0
DLZ102	DLZMPI00
DLZ103	DLZBPC00
DLZ104	DLZMPC00
DLZ260	DLZBNUC0, DLZODP
DLZ261	DLZBNUC0, DLZODP
DLZ262	DLZRRC00, DLZOLI00
DLZ263	DLZRRC00, DLZOLI00
DLZ264	DLZRDBL1
DLZ266	DLZRRC00
DLZ301	DLZUDMP0, DLZURDB0, DLZURGL0, DLZURGU0, DLZURRL0, DLZUC350, DLZURUL0
DLZ302	DLZUDMP0, DLZURDB0, DLZURUL0, DLZURRL0
DLZ303	DLZUDMP0, DLZURUL0
DLZ304	DLZUDMP0, DLZURDB0, DLZURUL0
DLZ305	DLZUDMP0, DLZURDB0, DLZURUL0
DLZ306	DLZURUL0, DLZURDB0, DLZUDMP0
DLZ307	DLZURUL0, DLZURDB0, DLZUDMP0, DLZURRL0
DLZ308	DLZUDMP0, DLZURUL0
DLZ309	DLZUDMP0, DLZURUL0, DLZURRL0
DLZ310	DLZUDMP0, DLZURUL0, DLZURDB0, DLZURRL0
DLZ311	DLZURRL0, DLZURGU0, DLZURGL0
DLZ312	DLZURDB0
DLZ313	DLZURDB0
DLZ314	DLZURDB0
DLZ315	DLZURGU0, DLZURGL0
DLZ316	DLZURDB0, DLZUDMP0
DLZ317	DLZURDB0

Message Number	Module
DLZ318	DLZURGU0, DLZURGL0
DLZ319	DLZURUL0, DLZURGU0, DLZUDMP0, DLZURGL0, DLZURDB0, DLZURRL0
DLZ320	DLZURUL0, DLZURGU0, DLZUDMP0
DLZ321	DLZURUL0, DLZUDMP0, DLZURRL0
DLZ322	DLZURDB0
DLZ323	DLZURDB0
DLZ324	DLZURDB0
DLZ325	DLZURDB0
DLZ326	DLZURDB0
DLZ327	DLZURDB0
DLZ328	DLZURDB0
DLZ329	DLZURDB0
DLZ330	DLZURDB0
DLZ331	DLZURDB0
DLZ332	DLZURDB0
DLZ333	DLZURDB0
DLZ334	DLZURDB0
DLZ335	DLZURDB0
DLZ336	DLZURDB0
DLZ337	DLZURDB0
DLZ338	DLZURDB0
DLZ339	DLZURDB0
DLZ340	DLZURDB0
DLZ341	DLZURDB0
DLZ342	DLZURDB0
DLZ343	DLZURDB0
DLZ345	DLZURGU0, DLZUDMP0, DLZURUL0
DLZ346	DLZURGU0
DLZ347	DLZURGU0
DLZ348	DLZURGU0, DLZURGL0
DLZ349	DLZURGU0
DLZ352	DLZURGU0
DLZ353	DLZURRL0
DLZ354	DLZURGL0
DLZ355	DLZURGL0
DLZ356	DLZURRL0
DLZ357	DLZURUL0, DLZUDMP0
DLZ358	DLZURUL0
DLZ360	DLZUCCT0
DLZ361	DLZUCCT0
DLZ362	DLZUCCT0
DLZ363	DLZUCCT0
DLZ364	DLZUCCT0
DLZ365	DLZUCCT0
DLZ366	DLZUCCT0
DLZ367	DLZUCCT0
DLZ369	DLZUCCT0, DLZUC150
DLZ370	DLZURGL0
DLZ371	DLZUC150
DLZ373	DLZUC350
DLZ374	DLZUC150, DLZUC350
DLZ375	DLZUC350
DLZ376	DLZURGL0
DLZ377	DLZURGU0
DLZ378	DLZURGU0, DLZURGL0
DLZ379	DLZURGU0, DLZURGL0
DLZ380	DLZURGU0, DLZURGL0
DLZ381	DLZURGU0, DLZURGL0
DLZ382	DLZURUL0

Message Number	Module
DLZ383	DLZURUL0
DLZ384	DLZUCMN0
DLZ385	DLZUCMN0
DLZ387	DLZURGL0
DLZ389	DLZURGL0, DLZURRL0
DLZ390	DLZUC150
DLZ391	DLZUDMP0, DLZURDB0, DLZURUL0, DLZURRL0, DLZUC150, DLZUC350, DLZURPR0, DLZURGS0, DLZURG10, DLZURGP0
DLZ392	DLZUDMP0, DLZURDB0, DLZURUL0, DLZURRL0
DLZ393	DLZURRL0
DLZ394	DLZURRL0, DLZURDB0
DLZ395	DLZBACK0
DLZ396	DLZRDBC0
DLZ397	DLZRDBC0
DLZ398	DLZRDBC0
DLZ399	DLZRDBC0
DLZ476	DLZDLA00
DLZ570	DLZUAMB0
DLZ571	DLZUACB0
DLZ583	DLZUACB0
DLZ584	DLZUACB0
DLZ585	DLZUACB0
DLZ587	DLZUACB0
DLZ588	DLZUACB0
DLZ589	DLZUACB0
DLZ772	DLZDXMT0
DLZ799	DLZDLD00
DLZ800	DLZDLR00
DLZ801	DLZDLR00
DLZ802	DLZDLD00
DLZ803	DLZDLD00
DLZ804	DLZDLD00
DLZ806	DLZDLD00
DLZ807	DLZDLD00
DLZ808	DLZDLD00
DLZ830	DLZDHD00, DLZGGSP0
DLZ841	DLZDBH00
DLZ844	DLZDBH00
DLZ845	DLZDBH00
DLZ847	DLZDBH00
DLZ848	DLZDBH00
DLZ850	DLZDDLE0
DLZ855	DLZDDLE0
DLZ860	DLZDDLE0, DLZDXMT0
DLZ861	DLZDDLE0
DLZ862	DLZDDLE0
DLZ863	DLZDDLE0
DLZ864	DLZDDLE0
DLZ868	DLZDXMT0
DLZ888	DLZBACK0
DLZ894	DLZBACK0
DLZ904	DLZRDBL0
DLZ905	DLZDLBL0
DLZ906	DLZDLBL0
DLZ907	DLZDLBL0
DLZ908	DLZDLBL0
DLZ909	DLZDLBL0
DLZ910	DLZDLBL0
DLZ911	DLZDLBL0
DLZ912	DLZDLBL0

Message Number	Module
DLZ913	DLZDLBL0
DLZ914	DLZDLBL0
DLZ915	DLZDLBL0
DLZ916	DLZDLBL0
DLZ917	DLZDLBL0
DLZ918	DLZDLBL0
DLZ919	DLZDLBL0
DLZ920	DLZDLBL0
DLZ921	DLZDLBL0
DLZ922	DLZDLBL0
DLZ923	DLZDLBL0
DLZ924	DLZDLBL0
DLZ925	DLZDLBL0
DLZ926	DLZDLBL0
DLZ927	DLZDLBL0
DLZ928	DLZDLBL0
DLZ929	DLZDLBL0
DLZ930	DLZDLBL0
DLZ931	DLZDLBL0
DLZ932	DLZDLBL0
DLZ933	DLZDLBL0
DLZ934	DLZDLBL0
DLZ935	DLZDLBL0
DLZ936	DLZDLBL0
DLZ939	DLZDLBL0
DLZ940	DLZDLBL0
DLZ945	DLZDLBL0
DLZ952	DLZURPR0, DLZURGS0
DLZ953	DLZURGP0
DLZ954	DLZURPR0, DLZURGS0, DLZURG10, DLZURGP0
DLZ955	DLZURG10, DLZURGP0
DLZ956	DLZURPR0, DLZURGS0, DLZURGP0
DLZ957	DLZURG10
DLZ958	DLZURGS0, DLZURGP0
DLZ959	DLZURGS0, DLZURGP0
DLZ960	DLZURGP0
DLZ961	DLZURPR0, DLZURGS0, DLZURG10
DLZ962	DLZURPR0
DLZ963	DLZURPR0
DLZ964	DLZURPR0
DLZ965	DLZURPR0
DLZ966	DLZURPR0, DLZURGS0, DLZURG10, DLZURGP0
DLZ967	DLZURGS0
DLZ968	DLZURGS0, DLZURPR0, DLZURG10, DLZURGP0
DLZ969	DLZURGS0
DLZ970	DLZURGS0
DLZ971	DLZURGS0
DLZ972	DLZURGS0
DLZ973	DLZURGS0
DLZ974	DLZURGS0
DLZ975	DLZURGS0
DLZ976	DLZURPR0
DLZ977	DLZURG10
DLZ978	DLZURG10
DLZ979	DLZURG10
DLZ980	DLZURG10
DLZ981	DLZURG10
DLZ982	DLZURG10, DLZURGP0
DLZ983	DLZURGP0
DLZ984	DLZURPR0, DLZURGS0, DLZURG10, DLZURPR0

Message Number	Module
DLZ985	DLZURPR0
DLZ989	DLZURG10
DLZ990	DLZURGS0, DLZURGP0, DLZURG10

APPENDIX B. DL/I STATUS CODES - MODULE CROSS REFERENCE

The following table shows which modules set the DL/I status codes.

Status Code	Module
AB	DLZDLA00
AC	DLZDLA00
AD	DLZDLA00
AH	DLZDLA00
AI	DLZDLA00, DLZDLD00
AJ	DLZDLA00
AK	DLZDLA00, DLZDLR00
AM	DLZDLA00, DLZDLD00
AO	DLZDLD00, DLZDLR00, DLZDDLE0
DA	DLZDLD00
DJ	DLZDLA00
DX	DLZDLD00
GA	DLZDLR00
GB	DLZDLR00
GE	DLZDLR00
GK	DLZDLR00
GP	DLZDLR00
II	DLZDLR00, DLZDDLE0
IX	DLZDDLE0
LB	DLZDLA00, DLZDDLE0
LC	DLZDLA00
LD	DLZDLA00
LE	DLZDLA00
NA	DLZDXMT0
NE	DLZDXMT0
NI	DLZDXMT0
NO	DLZDXMT0
RX	DLZDLD00
V1	DLZDLA00

APPENDIX C. SUMMARY OF DL/I DOS/VS MODULES

This table states all DL/I DOS/VS modules residing in the core image library. For each module, the following additional information is given:

- The CSECTs that comprise each PHASE. They are listed in the order they were linked. Any indented name under a CSECT is an entry point within that CSECT. If the indented name is preceded by '*', it designates a routine within the CSECT and may, or may not, appear on the link-edit map. Unreferenced entry points have been omitted.
- The name(s) of the module(s) in the relocatable library which are needed for linkage editing.
- The name(s) of the module(s) in the source statement library. For each module, source code listings are available on microfiche (under the module name).
- The core ID for the applicable modules. This is located near the beginning address of each module and is usually followed by the date of last source change in the format MDDY where M is the number of the month (1 to C), DD is the day, and Y is the last digit of the year.
- Supplementary information. The entry SVA means that the module concerned is eligible to be loaded into the shared virtual area (SVA). Any other entry in this column is the entry point name that must be present on the END card when assembling this module, for example, END DLZBEGIN.

CORE IMAGE LIBRARY	CSECT(S)/ ENTRY POINT(S)	RELO LIBRARY	SOURCE LIBRARY	CORE ID	SUPPL INF
** ONLINE INITIALIZATION **					
DFHSIDL	DLZOLI00	DLZOLI00	DLZOLI00	OLI	
	*DLZCPI00				
** DATA BASE BACKOUT UTILITY **					
DLZBACK0	DLZBACK0	DLZBACK0	DLZBACK0	BACK	
	IJ2M0011				
	DLZRDBC0	DLZRDBC0	DLZRDBC0	RDBC	
	DLZBACM0	DLZBACM0	DLZBACM0		
	DLZLI000	DLZLI000	DLZLI000	DLZLI000	
	ASMTDLI				
	IJFUBZZZ	IJFUBZZZ			
	IJJFCBZD	IJJFCBZD			
** DL/I BATCH NUCLEUS **					
DLZBNUC0	SCDCSECT	DLZBNUC0	DLZBNUC0	5746-XX1	
	SCDSTART				
	*DLZIWAIT			IWT	
	*DLZPRHB0			PRH	
	*DLZABEND			ABN	

CORE IMAGE LIBRARY	CSECT(S)/ ENTRY POINT(S)	RELO LIBRARY	SOURCE LIBRARY	CORE ID	SUPPL INF
** MPS BATCH PARTITION CONTROLLER **					
DLZBPC00	DLZBPC00	DLZBPC00	DLZBPC00	DLZBPC00	
** BUFFER HANDLER **					
DLZDBH00	DLZDBH00	DLZDBH00	DLZDBH00	DBH	SVA
	DLZEBH00				DLZEBH00
	*MAINROUT				
	ROULINK				
	*PREPENQ				
	*PREPDEQ				
	*ABEXIT				
	*BOTTOUSE				
	*ALLDEQ				
	*BFFERREL				
	*RETURN				
	DLZDBH02	DLZDBH02	DLZDBH02	BFH2	
	*WRITE				
	*READ				
	*HSREAD				
	*HSWRITE				
	*LOWRITE				
	*PUTKY				
	*MSPUT				
	*STLEQ				
	*STIBG				
	*GETNX				
	DETIOERR				
	*TSTPST1				
	DLZDBH03	DLZDBH03	DLZDBH03	BFH3	
	*ENQ				
	*DEQ				
	*CONVADNR				
** LOAD/INSERT **					
DLZDDLE0	DLZDDLE0	DLZDDLE0	DLZDDLE0	DLE	SVA
	HDROUTIN				
	HSROUTIN				
** SPACE MANAGEMENT **					
DLZDHDS0	DLZDHDS0	DLZDHDS0	DLZDHDS0	HDO	SVA
	DLZGGSPC	DLZGGSP0	DLZGGSP0		
	DLZRRTRN				
	DLZFRSPC	DLZFRSP0	DLZFRSP0		
	DLZLLCLC	DLZLLCL0	DLZLLCL0		
	DLZMMLCT	DLZMMLC0	DLZMMLC0		
	DLZRRHPL	DLZRCHP0	DLZRCHP0		
	DLZRCHBK	DLZRCHB0	DLZRCHB0		
	DLZRCBK2				
	DLZMMUDT	DLZMMUD0	DLZMMUD0		
	DLZMMOFF				
	DLZMMON				
	DLZRRHMP	DLZRRHM0	DLZRRHM0		
	DFSBLO30	DLZDHDS0	DLZDHD00		
	*SNAPDCB				
	*SNPSW				
	*SNPCNT				

CORE IMAGE LIBRARY	CSECT(S)/ ENTRY POINT(S)	RELO LIBRARY	SOURCE LIBRARY	CORE ID	SUPPL INF
** CALL ANALYZER **					
DLZDLA00	DLZDLA00	DLZDLA00	DLZDLA00	DLA	SVA
** DELETE/REPLACE **					
DLZDLD00	DLZDLD00	DLZDLD00	DLZDLD00	DLD	SVA
	DLZDLDS0				
	DLZDLDD0				
	DLZDLDA0				
	DLZDLDR0				
** OPEN/CLOSE **					
DLZDLOC0	DLZDLOC0	DLZDLOC0	DLZDLOC0	LOC	
** RETRIEVE **					
DLZDLR00	DLZDLR00	DLZDLRA0	DLZDLRA0	LR0	SVA
	DLZDLR10				
	DLZRETN0				
	DLZEODC0				
	DLZGERC0				
	DLZGER0				
	DLZGETS0				
	DLZWIPE0				
	DLZMOVA0				
	DLZMOVB0				
	DLZDELT0				
	DLZPSDB0				
	DLZHUNT0				
	DLZSETL0				
	DLZBH0				
	DLZSSDB0				
	DLZNOOP0				
	DLZCONC0				
	DLZSSA0	DLZDLRC0	DLZDLRC0	LR0	
	DLZTAG0				
	DLZLTW0				
	DLZNOSS0				
	DLZHIDA0	DLZDLRE0	DLZDLRE0	LR0	
	DLZHDAM0				
	DLZHISA0				
	DLZSTLA0				
	DLZSTLG0				
	DLZUPDT0				
	DLZKDTE0				
	DLZPCHK0				
	DLZISRT0	DLZDLRF0	DLZDLRF0	LR0	
	DLZVLRT0				
	DLZAREJ0				
	DLZVLCH0				
	DLZXDFT0				
	DLZHSAM0				
	DLZALTS0				
	DLZXMIN0				
	DLZXMAX0				
	DLZXDBL0				
	DLZLOGR0	DLZDLRD0	DLZDLRD0	LR0	
	DLZRETK0				
	DLZRETI0				

CORE IMAGE LIBRARY	CSECT(S)/ ENTRY POINT(S)	RELO LIBRARY	SOURCE LIBRARY	CORE ID	SUPPL INF
	DLZKDRK0				
	DLZKDTL0				
	DLZUPDC0				
	DLZUPDL0				
(DLZDLR00)	DLZAPST0				
	DLZYENT0				
	DLZYSTC0				
	DLZYEND0				
	DLZPOST0	DLZDLRG0	DLZDLRG0	LR0	
	DLZPSTN0				
	DLZPSTA0				
	DLZPOSA0				
	DLZPSTT0				
	DLZPSTQ0				
	DLZSKPG0				
	DLZSKPS0				
	DLZSKPD0				
	DLZSKPE0				
	DLZRLNKD	DLZRLNKD	DLZRLNKD	LR0	
** DL/I TEST PROGRAM - BATCH **					
DLZDLTXX	DLITCBL	DLZDLTXX	DLZDLTXX	DLT	
	DLZSNAP				
	DLZLI000	DLZLI000	DLZLI000	DLZLI000	
	CBLTDLI				
	IJGFIZZZ	IJGFIZZZ			
	IJJFCBID	IJJFCBID			
	IJJFCIID				
** DL/I TEST PROGRAM - ONLINE **					
DLZDLTXY	DLITCBL	DLZDLTXY	DLZDLTXY	DLZDLTXY	
	DLZSNAP				
	DLZLI000	DLZLI000	DLZLI000	DLZLI000	
	CBLTDLI				
	IJGFIZZZ	IJGFIZZZ			
	IJJFCBID	IJJFCBID			
	IJJFCIID				
** WORK DATA SET GENERATOR **					
DLZDSEH0	DLZDSEH0	DLZDSEH0	DLZDSEH0	DSEH	DLZBEGIN
	DLZBEGIN				
	OPENWORK				
	IJFSZZWN	IJFSZZWN			
	IJFVZZWN				
	IJGFICZZ	IJGFICZZ			
	IJGQOCZZ	IJGQOCZZ			
	IJGVOCZZ				
** INDEX MAINTENANCE **					
DLZDXMT0	DLZDXMT0	DLZDXMT0	DLZDXMT0	XMT	SVA
** BATCH FORMATTED DUMP **					
DLZFSDP0	DLZFSDP0	DLZFSDP0	DLZFSDP0	FSD	
		DLZTRPR0			

CORE IMAGE LIBRARY	CSECT(S)/ ENTRY POINT(S)	RELO LIBRARY	SOURCE LIBRARY	CORE ID	SUPPL INF
** MPS MASTER PARTITION CONTROLLER **					
DLZMPC00	DLZMPC00	DLZMPC00	DLZMPC00	DLZMPC00	
** MPS BATCH MODULE **					
DLZMPI00	DLZMPI00	DLZMPI00	DLZMPI00	DLZMPI00	DLZMINIT
	*DLZMPRH				
	DLZMINIT				
	*DLZMTERM				
	*DLZMMSG				
	*DLZMABND				
	DLZCONSL				
	DLZDIMOD				
	DLZMMSGT	DLZMMSGT	DLZMMSGT		
** MPS STOP TRANSACTION **					
DLZMSTP0	DLZMSTP0	DLZMSTP0	DLZMSTP0	DLZMSTP0	
** MPS START TRANSACTION **					
DLZMSTR0	DLZMSTR0	DLZMSTR0	DLZMSTR0	DLZMSTR0	
** ONLINE NUCLEUS **					
DLZNUCxx	DLZODP	DLZODP	DLZODP	DP0	
	DLZODP00				
	DLZSCHDL				
	DLZODP01	DLZODP	DLZODP	DP1	
	DLZTKTRM				
	DLZODP03				
	DLZODP02			DP2	
	DLZPRH00	DLZODP	DLZODP	PRH	
	DLZABND0				
	DLZCWAIT	DLZODP	DLZODP	IWT	
	DLZERMSG	DLZODP	DLZODP	MSG	
	DLZOVSEX	DLZODP	DLZODP	EXP	
	DLZNUC		DLZACT		
	SCDSTART				
	DLZFTDP0	DLZFTDP0	DLZFTDP0	FTD	
	DLZMMSGT	DLZMMSGT	DLZMMSGT		
** DATA BASE LOGGER **					
DLZRDBI0	DLZRDBL0	DLZRDBL0	DLZRDBL0	DBL	DLZIDBL0
	ADDRASL0				
	DLZIDBL0				
	IOFILA1				
	LOGOUT				
	IJFUZZZN	IJFUZZZN			
	IJFUZZZZ				
	ONLLOGWR	DLZRDBL0	DLZRDBL0		
(DLZRDBL0)	SAVE				
	PRIVECB				
** DATA BASE LOGGER WITH CICS JOURNALING **					
DLZRDBL1	DLZRDBL1	DLZRDBL1	DLZRDBL1	DBJ	DLZIDBL0
	DLZRDBL0				

CORE IMAGE LIBRARY	CSECT(S)/ ENTRY POINT(S)	RELO LIBRARY	SOURCE LIBRARY	CORE ID	SUPPL INF
** BATCH INITIALIZATION **					
DLZRRC00	DLZRRC00	DLZRRC00	DLZRRC00		DLZRRCST
	*ERRORMSG			RRC	
	DLZBMSGT	DLZBMSGT	DLZBMSGT		
	DLZRDR	DLZRRC00	DLZRRC00	DLZRDR	
	DLZCONSL				
	DLZRRC10				
	*DLZRRA00				
	*DLZPCC00				
	*DLZDBLM0				
	*LOADDMBS				
	*PCBROUT				
	*DLZCPI00				
	*DMBLOADR				
** CICS/VS SYSTEM TERMINATION INTERFACE **					
DLZSTP00	DLZSTP00	DLZSTP00	DLZSTP00		
** APPLICATION CONTROL BLOCK CREATION UTILITY **					
DLZUACB0	DLZUACB0	DLZUACB0	DLZUACB0	UACB	
	PRTMSG				
	DLZDLBL0	DLZDLBL0	DLZDLBL0	DLBL	
	DLZDLBA0				
	BLDJCB				
	DLZUAMB0	DLZUAMB0	DLZUAMB0		
	PCHDTF				
	IJSYSLN				
	DLZLBLM0	DLZLBLM0	DLZLBLM0	LBLM	
	DLZUSCH0	DLZUSCH0	DLZUSCH0		
	INSRCH				
	CICSESCH				
	DLZUMSG0	DLZUMSG0	DLZUMSG0		
	DLZUMGT0	DLZUMGT0	DLZUMGT0		
	DLZDPSB0	DLZDPSB0	DLZDPSB0	DPSB	
	IJJCPD1N	IJJCPD1N			
	IJJFCBZD	IJJFCBZD			
	IJJFCIZD				
** DATA BASE CHANGE ACCUMULATION UTILITY **					
DLZUCUM0	DLZUCUM0	DLZUCUM0	DLZUCUM0	UCUM	
	DLZERRTN				
	DLZUSPKL				
	DLZWORK#				
	DLZPRNT				
	DLZSLOG				
	DLZUCONS				
	DLZUCCT0	DLZUCCT0	DLZUCCT0	UCCT	
	DLZUC150	DLZUC150	DLZUC150	UC15	
	DLZUEX15				
(DLZUCUM0)	DLZUC350	DLZUC350	DLZUC350	UC35	
	DLZUEX35				
	DLZUCER0	DLZUCER0	DLZUCER0	UCER	
	DLZCUMM0	DLZCUMM0	DLZCUMM0		
	IJFSZZWN	IJFSZZWN			
	IJFVZZZZ				
	IJFSZZWZ				
	IJGQICZZ	IJGQICZZ			

CORE IMAGE LIBRARY	CSECT(S)/ ENTRY POINT(S)	RELO LIBRARY	SOURCE LIBRARY	CORE ID	SUPPL INF
	IJGQIZZZ				
	IJGQOCZZ	IJGQOCZZ			
	IJGQOZZZ				
	IJJFCBZD	IJJFCBZD			
	IJJFCIZD				
	IJ2L0012	DLZUCUM0	DLZUCUM0		
** IMAGE CCPY UTILITY **					
ILZUDMP0	DLZUDMP0	DLZUDMP0	DLZUDMP0	DMP	
	DLZDMPM0	DLZDMPM0	DLZDMPM0		
	IJJFCBZD	IJJFCBZD			
	IJFSZZWN	IJFSZZWN			
	IJFVZZWN				
	IJGQOCZZ	IJGQOCZZ			
	IJGVOCZZ				
** RECOVERY UTILITY **					
DLZURDB0	DLZURDB0	DLZURDB0	DLZURDB0	RDB	
	DLZLI000	DLZLI000	DLZLI000	DLZLI000	
	CBLTDLI				
	DLZRDBM0	DLZRDBM0	DLZRDBM0		
	IJJFCBZD	IJJFCBZD			
	IJFSZZWN	IJFSZZWN			
	IJFVZZWN				
	IJGQICZZ	IJGQICZZ			
	IJGVICZZ				
** HD REORGANIZATICN RELOAD UTILITY **					
ILZURGL0	DLZURGL0	DLZURGL0	DLZURGL0	RGL	
	DLZLI000	DLZLI000	DIZLI000	DLZLI000	
	CBITDLI				
	DLZRGLM0	DLZRGLM0	DLZRGLM0		
	IJJFCBZD	IJJFCBZD			
	IJGQICZZ	IJGQICZZ			
	IJGVICZZ				
	IJFSZZWN	IJFSZZWN			
	IJFVZZZN				
** PREFIX UPDATE UTILITY **					
DLZURGP0	DLZURGP0	DLZURGP0	DLZURGP0	URGP	
	DLZURGM0	DLZURGM0	DLZURGM0		
	DLZLI000	DLZLI000	DIZII000	DLZLI000	
	ASMTDLI				
	CBLTDLI				
	IJJFCBZD	IJJFCBZD			
	IJJFCIZD				
	IJFSZZWN	IJFSZZWN			
	IJFVZZZN				
(DIZURGP0)	IJGQICZZ	IJGQICZZ			
	IJGVICZZ				
** DATA BASE SCAN UTILITY **					
DLZURGS0	DLZURGS0	DLZURGS0	DLZURGS0	URGS	
	DLZCONSL				
	DLZURGM0	DLZURGM0	DLZURGM0		

CORE IMAGE LIBRARY	CSECT(S)/ ENTRY POINT(S)	RELO LIBRARY	SOURCE LIBRARY	CORE ID	SUPPL INF
	DLZII000	DLZLI000	DLZLI000	DLZLI000	
	ASMTDLI				
	IJJFCBZD	IJJFCBZD			
	IJJFCIZD				
	IJFSZZWN	IJFSZZWN			
	IJFVZZZN				
	IJGQICZZ	IJGQICZZ			
	IJGVICZZ				
	IJGFICZZ	IJGFICZZ			
** HD REORGANIZATION UNLOAD UTILITY **					
DLZURGU0	DLZURGU0	DLZURGU0	DLZURGU0	RGU	
	DLZCCNSL				
	DLZLI000	DLZLI000	DLZLI000	DLZLI000	
	CBLTDLI				
	DLZRGUM0	DLZRGUM0	DIZRGUM0		
	IJJFCBZD	IJJFCBZD			
	IJFUZZZN	IJFUZZZN			
	IJGUOCZZ	IJGUOCZZ			
	IJGUICZZ	IJGUICZZ			
** PREFIX RESCLUTION UTILITY **					
DLZURG10	DLZURG10	DLZURG10	DLZURG10	RG1	
	DLZURGM0	DLZURGM0	DLZURGM0		
	IJJFCBZD	IJJFCBZD			
	IJJFCIZD				
	IJGFICZZ	IJGFICZZ			
	IJGQICZZ	IJGQICZZ			
	IJGVICZZ				
	IJFSZZWN	IJFSZZWN			
	IJFVZZZN				
	IJFVZZWN				
	IJFFZZZN	IJFFZZZN			
	IJGQOCZZ	IJGQOCZZ			
	IJGVOCZZ				
	DLZX15S1	DLZURG10	DLZURG10		
	DLZX15S2				
	DLZX35S1				
	DLZX35S2				
** PREREORGANIZATION UTILITY **					
DLZURPR0	DLZURPR0	DLZURPR0	DLZURPR0	RPR	
	DLZLI000	DLZLI000	DLZLI000	DLZLI000	
	ASMTDLI				
	DIZURGM0	DLZURGM0	DLZURGM0		
	IJJFCBZD	IJJFCBZD			
	IJGFOCZZ	IJGFOCZZ			
** HISAM REORGANIZATION RELOAD UTILITY **					
DLZURL0	DLZURRL0	DLZURRL0	DLZURRL0	RRL	
	DLZRRLM0	DLZRRLM0	DLZRRLM0		
	IJJFCBZD	IJJFCBZD			
	IJGQICZZ	IJGQICZZ			
	IJGVICZZ				
	IJFVZZWN	IJFVZZWN			
	IJFVZZWZ				

CORE IMAGE LIBRARY	CSECT(S)/ ENTRY POINT(S)	RELO LIBRARY	SOURCE LIBRARY	CORE ID	SUPPL INF

** HISAM REORGANIZATION UNLOAD UTILITY **

DLZURULO	DLZURULO	DLZURULO	DLZURULO	RUL	
	DLZRULMO	DLZRULMO	DLZRULMO		
	IJJFCBZD	IJJFCBZD			
	IJFVZZWN	IJFVZZWN			
	IJGQOCZZ	IJGQOCZZ			
	IJGVOCZZ				

** DL/I TRACING FACILITY **

user chosen	DLZTRACE	user chosen	DLZTRACE	TRC	
	DLZTRPRO	DLZTRPRO	DLZTRPRO	TRP	
	IJJFCBIC	IJJFCBIC			

ACB	Access method control block (VSAM)
ACT	Application control table
BPC	Batch partition controller
CA	Control area (VSAM)
CI	Control interval (VSAM)
CPL	Catalog parameter list (VSAM)
CSA	Common system area (CICS)
CSMT	Control sytem master terminal (CICS)
DBD	Data base description
DDIR	Data management block directory
DMB	Data management block
DSG	Data set group
ECB	Event control block (DOS)
EXLST	Exit list (VSAM)
FDB	Field description block
FSE	Free space element
ISS	Index source segment
JCA	Journal control area (CICS)
JCB	Job control block
JCT	Journal control table (CICS)
LEVTAB	Level table
LSDB	Logical segment description block
LLC/CC	Low level code/continuity checking
MPC	Master partition controller
MPS	Multiple partition support
PCB	Program control block
PCF	Physical child first pointer
PCL	Physical child last pointer
PDIR	PSB directory
PPST	PST prefix
PPT	Processing program table (CICS)
PROCOPT	Processing option
PSB	Program specification block
PSDB	Physical segment description block
PSIL	PSB segment intent list
PST	Partition specification table
PTB	Physical twin backward pointer
PTF	Physical twin forward pointer
PUB	Physical unit block (DOS)
RAP	Root anchor point
RBA	Relative byte address (VSAM)
RBN	Relative byte number
RPL	Request parameter list (VSAM)
SCD	System contents directory
SDB	Segment description block
SSA	Segment search argument
TCA	Task communication area (CICS)

INDEX OF DL/I TABLE AND RECORD FORMATS

LY12-5016-2

DL/I DOS/VS, Logic Manual Printed in U.S.A. LY12-5016-2

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N. Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N. Y., U. S. A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N. Y., U. S. A. 10601

DL/I DOS/VS
Logic Manual
LY12-5016-2

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM shall have the nonexclusive right, in its discretion, to use and distribute all submitted information, in any form, for any and all purposes, without obligation of any kind to the submitter. Your interest is appreciated.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

	Yes	*No*
• Does the publication meet your needs?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

• What is your occupation? ______________________________

• How do you use this publication:

As an introduction to the subject?	☐	As an instructor in class?	☐
For advanced knowledge of the subject?	☐	As a student in class?	☐
To learn about operating procedures?	☐	As a reference manual?	☐

Your comments:

If you would like a reply, please supply your name and address on the reverse side of this form.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

LY12-5016-2

Reader's Comment Form

Cut or Fold Along Line

Fold and tape | Please Do Not Staple | Fold and Tape

First Class
Permit 10
Endicott
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department G60
P. O. Box 6
Endicott, New York 13760

DL/I DOS/VS, Logic Manual Printed in U.S.A. LY12-5016-2

Fold | Fold

If you would like a reply, *please print:*

Your Name ______________________________

Company Name ______________________ *Department* __________

Street Address ______________________________

City ______________________________

State ______________________ *Zip Code* __________

IBM Branch Office serving you ______________________________

®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N. Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N. Y., U. S. A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N. Y., U. S. A. 10601

DL/I DOS/VS
Logic Manual
LY12-5016-2

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM shall have the nonexclusive right, in its discretion, to use and distribute all submitted information, in any form, for any and all purposes, without obligation of any kind to the submitter. Your interest is appreciated.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

	Yes	*No*
• Does the publication meet your needs?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

• What is your occupation? ______________________________

• How do you use this publication:

As an introduction to the subject?	☐	As an instructor in class?	☐
For advanced knowledge of the subject?	☐	As a student in class?	☐
To learn about operating procedures?	☐	As a reference manual?	☐

Your comments:

If you would like a reply, please supply your name and address on the reverse side of this form.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

LY12-5016-2

Reader's Comment Form

Fold and tape　　Please Do Not Staple　　Fold and Tape

First Class
Permit 10
Endicott
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department G60
P. O. Box 6
Endicott, New York 13760

Fold　　Fold

If you would like a reply, *please print:*

Your Name ____________________
Company Name ____________________ *Department* __________
Street Address ____________________
City ____________________
State ____________________ *Zip Code* __________
IBM Branch Office serving you ____________________

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N. Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N. Y., U. S. A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N. Y., U. S. A. 10601

Cut or Fold Along Line

DL/I DOS/VS, Logic Manual Printed in U.S.A. LY12-5016-2