

V70 TOTAL
DATA BASE MANAGEMENT SYSTEM
REFERENCE MANUAL

The statements in this publication are not intended to create any warranty, express or implied. Equipment specifications and performance characteristics stated herein may be changed at any time without notice. Address comments regarding this document to Varian Data Machines, Publications Department, 2722 Michelson Drive, P.O. Box C-19504, Irvine, California, 92713.



varian data machines / a varian subsidiary
2722 michelson drive/p.o. box c-19504/irvine/california/92713

© 1975 printed in USA



varian data machines

98 A 9952 410

DECEMBER 1975



TABLE OF CONTENTS

	<u>Page</u>
SECTION 1 INTRODUCTION	1-1
1.1 Data Base Management Concepts	1-1
1.2 Key Features of VORTEX TOTAL	1-3
1.3 Hardware Configuration and Requirements	1-5
1.3.1 Minimum Configuration	1-5
1.3.2 Typical Configuration	1-6
1.3.3 Expanded Configuration	1-6
1.4 Bibliography	1-7
1.5 Definition of Terms	1-7
1.5.1 Logical Components	1-7
1.5.2 Special Names	1-10
1.5.3 Data Management Language (DML) Functions	1-11
1.5.4 VORTEX Definitions	1-12
 SECTION 2 TOTAL DATA BASE MANAGEMENT SYSTEM DESCRIPTION.	 2-1
2.1 System Description	2-1
2.2 System Operation	2-1
2.3 Structure of the Data Base	2-3
2.3.1 Data Base	2-3
2.3.2 Data Set or Data File	2-3
2.3.3 Data Record	2-7
2.3.4 Data Element	2-8
2.3.5 Data Item	2-8
2.4 Data Base Schematic	2-9
2.5 Data Relatability	2-10
2.6 TOTAL Data Sets	2-14
2.6.1 Single Entry (Master) Data Set Characteristics	2-16
2.6.2 Variable Entry Data Set Characteristics	2-18
2.6.3 Coded Record Concepts (Reformatting)	2-24
2.7 Data Independence	2-27
2.8 TOTAL's Programming Languages	2-28
2.8.1 Data Base Definition Language	2-28
2.8.2 Data Management Language	2-29
2.9 Creating, Formatting, and Operating the Data Base	2-29
2.10 Memory Requirements	2-30
2.10.1 Economic Computer Resource Utilization	2-30
2.10.2 Run-Time Memory Utilization	2-34
2.11 Privacy and Security	2-36
2.11.1 Internal Privacy and Security	2-36
2.11.2 External Privacy and Security	2-36



TABLE OF CONTENTS (continued)

SECTION 3 DATA BASE DEFINITION LANGUAGE 3-1

3.1 General Description 3-1

3.1.1 Logical Unit Convention 3-1

3.1.2 Execution of DBGEN. 3-1

3.1.3 TOTAL Record Formats. 3-2

3.1.4 Data Set (File) Organization. 3-5

3.2 SYNTAX Rules. 3-6

3.3 Summary of Data Base Definition Statements. 3-7

3.3.1 Prologue Statements 3-7

3.3.2 Master Data Set Statements. 3-7

3.3.3 Variable Entry Data Set Statements. 3-8

3.3.4 Epilogue Statement. 3-8

3.4 Data Base Definition Statements 3-8

3.4.1 Prologue Statements 3-8

3.4.2 Master Data Set Statements. 3-10

3.4.3 Variable Data Set Statements. 3-15

3.4.4 Epilogue Statements 3-19

3.5 Data Base Descriptor Module (DBMOD) 3-21

3.6 Example of the use of the Data Base Definition Language. 3-23

3.6.1 Computation of DBMOD. 3-23

SECTION 4 DATA BASE FORMATTOR 4-1

4.1 The User Formattor 4-1

4.2 Adding Files to the Data Base 4-4

4.3 Format Errors 4-4

SECTION 5 DATA MANAGEMENT LANGUAGE. 5-1

5.1 Command Parameters. 5-1

5.1.1 Functional Usage. 5-1

5.1.2 Notation Conventions. 5-3

5.1.3 Detailed Descriptions of Parameters 5-3

5.2 Description of DML Commands 5-14

5.2.1 The Add Master Function 5-19

5.2.2 The Add Variable After Function 5-21

5.2.3 The Add Variable Before Function. 5-22

5.2.4 The Add Variable Continue Function. 5-25

5.2.5 The Delete Master Function. 5-29

5.2.6 The Delete Variable Direct Function 5-29

5.2.7 The Read Next Function. 5-31

5.2.8 The Read Direct Function. 5-34

5.2.9 The Read Master Function. 5-38

5.2.10 The Read Reverse Function 5-38

5.2.11 The Read Variable Function. 5-41



TABLE OF CONTENTS (continued)

5.2.12	The Request Location Function	5-43
5.2.13	The Sign-Off Function	5-45
5.2.14	The Sign-On Function.	5-46
5.2.15	The Write Master Function	5-48
5.2.16	The Write Variable Function	5-50
SECTION 6	PROGRAMMING GUIDELINES.	6-1
6.1	Logical Unit Convention	6-1
6.2	General Programming Guidelines.	6-1
6.2.1	The CALL Statement.	6-1
6.2.2	CALL Statement Parameters	6-3
6.3	Programming Languages	6-3
6.4	Common Programming Considerations	6-3
6.4.1	Initialization, File Sharing and Termination Requirements.	6-3
6.4.2	Checking the STATUS Parameter	6-4
6.4.3	Parameter List Definitions.	6-5
6.5	Standard Single-Entry File Processing	6-6
6.5.1	The RQLOC Function.	6-6
6.5.2	Data Elements that Should not be Referenced	6-7
6.5.3	Special Note on the ADD-M Function.	6-7
6.5.4	Structural Maintenance During Serial Processing	6-8
6.6	Standard Variable-Entry File Processing	6-9
6.6.1	Basic Considerations.	6-9
6.6.2	The REFER Parameter	6-9
6.6.3	Efficiency Considerations in Variable-Entry File Processing	6-13
6.6.4	Coded Variable-Entry Records.	6-14
6.6.5	List Maintenance in the Batch Mode.	6-15
6.7	Testing Data Base Programs.	6-16
6.8	Buffering TOTAL Data Sets	6-17
6.9	Data Base Loading	6-18
6.10	Serial Processing	6-19
SECTION 7	OPERATING VORTEX TOTAL DBMS	7-1
7.1	Catalog of Application Program.	7-1
7.2	TOTAL Files	7-1
7.2.1	TOTAL Logical Files and VORTEX Files.	7-2
7.2.2	Disc Utilization.	7-2
7.3	Buffer Sharing.	7-4
7.4	Creating the Data Base.	7-4
7.4.1	Data Base Generation.	7-4
7.4.2	Parameter Card Format	7-5



TABLE OF CONTENTS (continued)

7.5 Formatting the Data Base. 7-5
 7.6 Data Base Execution 7-6
 7.7 Data Base Recovery. 7-6
 7.8 Data Set Changes. 7-8
 7.8.1 Single Entry Data Set Changes 7-8
 7.8.2 Variable Entry Data Set Changes 7-8
 7.9 Using the Utility Programs. 7-9
 7.10 Debugging TOTAL DBMS 7-10
 7.10.1 Status Codes. 7-10
 7.10.2 Diagnostics 7-11
 7.10.3 Automatic SNAPSHOT (SNAP) Dump. 7-11

APPENDIX A SAMPLE APPLICATION PROGRAM. A-1

A.1 Problem Description A-1
 A.2 Data Set Definition and Classification. A-2
 A.2.1 Determining the Required Data Sets. A-2
 A.2.2 Categorizing Data Sets as to Single Entry or
 Variable Entry. A-2
 A.3 Developing Data Base Relationship Schematics. A-3
 A.4 Data Base Sample Problem Schematic. A-4
 A.5 Determining the Data Records and Data Elements
 Required. A-4
 A.6 Data Base Generation. A-11
 A.7 RPG II Sample Program ORDPRO. A-18
 A.8 COBOL Sample Program. A-24
 A.9 FORTRAN Sample Program. A-30

APPENDIX B DIAGNOSTICS/STATUS CODES B-1

B.1 Data Base Generation. B-1
 B.2 Data Base Format Status Codes B-4
 B.3 DML Command Diagnostic Status Codes B-5
 B.3.1 Status Code Testing B-5
 B.3.2 Explanation of Terms. B-7
 B.3.3 Status Code Listing B-7

APPENDIX C 'TEXT' BLOCK C-1



LIST OF ILLUSTRATIONS

Figure	Description	Page
1-1	Advantages of a Single Data Base	1-2
1-2	Comparison of Cost and Savings	1-5
1-3	TOTAL Data Base Management System	1-8
1-4	Required and Optional Hardware	1-9
2-1	VORTEX II TOTAL Data Base Management System	2-2
2-2	TOTAL DBMS Operational Flow Chart	2-4
2-3	TOTAL Data Base Components	2-5
2-4	TOTAL Data Sets	2-6
2-5	Symbols Used to Form Schematic of the Data Base	2-9
2-6	Schematic of Data Base	2-10
2-7	Two Related Data Sets	2-12
2-8	Record Association	2-13
2-9	Two Variable Data Sets Related to One Master Data Set	2-13
2-10	One Variable Data Set Related to Two Master Data Sets	2-14
2-11	TOTAL Data Sets	2-16
2-12	Single Entry (Master) Data Set Records Retrieval	2-17
2-13	Variable Number of Records Per Key	2-19
2-14	Variable Keys Per Record, Variable Conventional Method of Access, Variable Record Format	2-20
2-15	Schematic of the Customer-Customer Order Data Base	2-21
2-16	Schematic of the Customer-Customer Order and Inventory Data Base	2-24
2-17	Format of Record Types in a Variable Entry File	2-26
2-18	TOTAL Data Base Generation	2-31
2-19	TOTAL Network Structure Example	2-32
2-20	Data Base Management System	2-33
2-21	Typical TOTAL Memory Layout	2-35
3-1	TOTAL Single-Entry Data-Set Record Format	3-2
3-2	TOTAL Variable Entry Data Set Record Format	3-3
3-3	TOTAL Variable Entry Data Set Record Format	3-4
3-4	Cylinder Load Limit for Variable Entry Data Sets	3-20
3-5	Object Module DBMOD Flow Chart	3-22
3-6	Example of Using the DBDL to Create a Data Base	3-26
4-1	Data Base Formatting Flow Chart	4-3
5-1	DATA-SET Parameter	5-5
5-2	LINKAGE-PATH Parameter	5-9
5-3	Linkage Path	5-10
5-4	CONTROL-KEY Parameter	5-11
5-5	DATA-LIST Parameter	5-13
5-6	DATA-AREA Parameter	5-15
5-7	The ADD MASTER Function	5-20
5-8	The ADD VARIABLE AFTER Function	5-23
5-9	The ADD VARIABLE AFTER Function for Two Master Data Sets	5-24



LIST OF ILLUSTRATIONS (continued)

Figure	Description	Page
5-10	The ADD VARIABLE BEFORE Function.	5-26
5-11	The ADD VARIABLE CONTINUE Function.	5-28
5-12	The DELETE MASTER Function.	5-30
5-13	The DELETE VARIABLE DIRECT Function	5-32
5-14	The READ NEXT Function.	5-35
5-15	The READ DIRECT Function.	5-37
5-16	The READ MASTER Function.	5-39
5-17	The READ VARIABLE REVERSE Function.	5-42
5-18	The READ VARIABLE Function.	5-44
5-19	The SIGN-ON Function, showing the use of the SCHEMA and REALM Parameters.	5-49
5-20	The WRITE MASTER Function	5-51
5-21	The WRITE VARIABLE Function	5-53
7-1	TOTAL Data Set Relationships.	7-3
7-2	Formatting TOTAL Data Sets.	7-7
A-1	Data Base Sample Problem.	A-5
A-2	Information Requirements.	A-6



LIST OF TABLES

Table	Description	Page
3-1	Data Base Descriptor (DBMOD) Memory Requirements.	3-21
5-1	Parameters Available for Functional Usage	5-2
5-2	Effect of Values in Reference Field Before Execution.	5-6
5-3	Content of Reference Field After Execution.	5-7
5-4	Alphabetic List of DML Commands	5-16
5-5	Calling Sequences to TOTAL for Various Languages.	5-16
5-6	DML Commands with Required Parameters	5-18
6-1	Relationship of REFER to Other Parameters	6-11
B-1	DML Diagnostic Messages	B-6



varian data machines



SECTION 1 INTRODUCTION

This manual describes the software, hardware requirements, programming, and operation of the Varian TOTAL Data Base Management System (DBMS). An understanding of basic computer concepts is assumed throughout this manual.

1.1 DATA BASE MANAGEMENT CONCEPTS

TOTAL is a software system that manages the storage and retrieval of data from a data base that resides on direct access devices. The advantage of a data base is that it can be created, maintained, and modified independently of application programs. Because the data required by the application programs resides separately on a common data base, redundancy of the same data being contained in different application programs is avoided; in addition, the need for recoding application programs when only the data changes is eliminated.

Thus a data base can be constructed as a separate program module, is available to all applications, yet remains unaffected by changes to various individual application programs. This is illustrated graphically in figure 1-1 which shows the advantages of a single data base containing data available to all application programs, compared with a system which has separate files for each application program, and in which some files may contain the same data.

The TOTAL Data Base Management system has been created to manage the data base system, and does this by providing the link between the application programs using the data base, and the data residing in the data base itself. TOTAL is responsible for providing the control of the input and output of all data to and from the data base, and for maintaining the relationships between data files and records within these files. Data is entered only once, and logical relationships are coded into the data as it is entered. The use of a network structure provides direct access to data records and eliminates the need for directories and indexing functions.

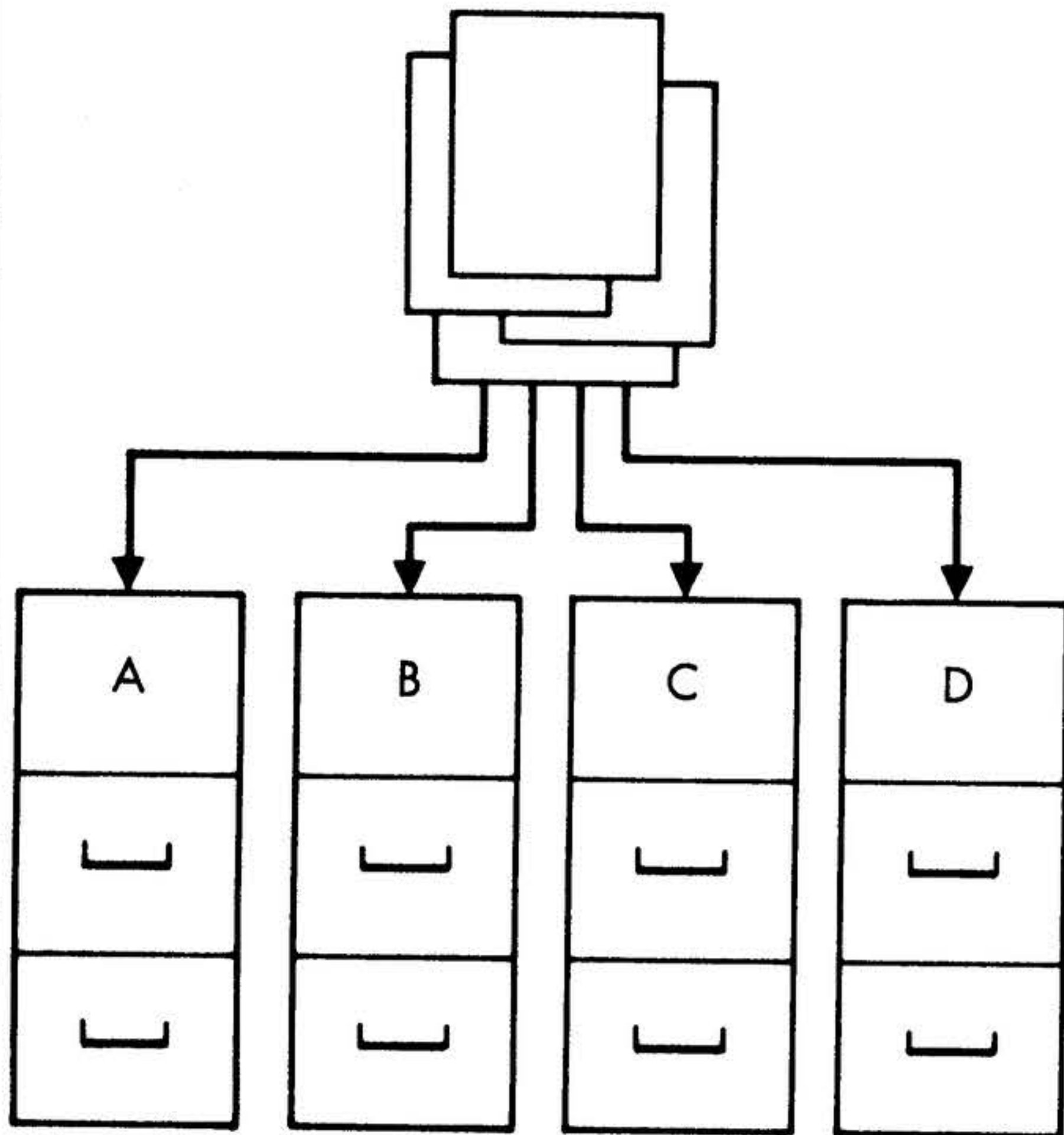
In summary, a Data Base Management System has the following advantages:

- o ADAPTABILITY - A Data Base Management System can handle all the information needed to operate an organization or business, in a manner that closely matches the organization's day-to-day activities.



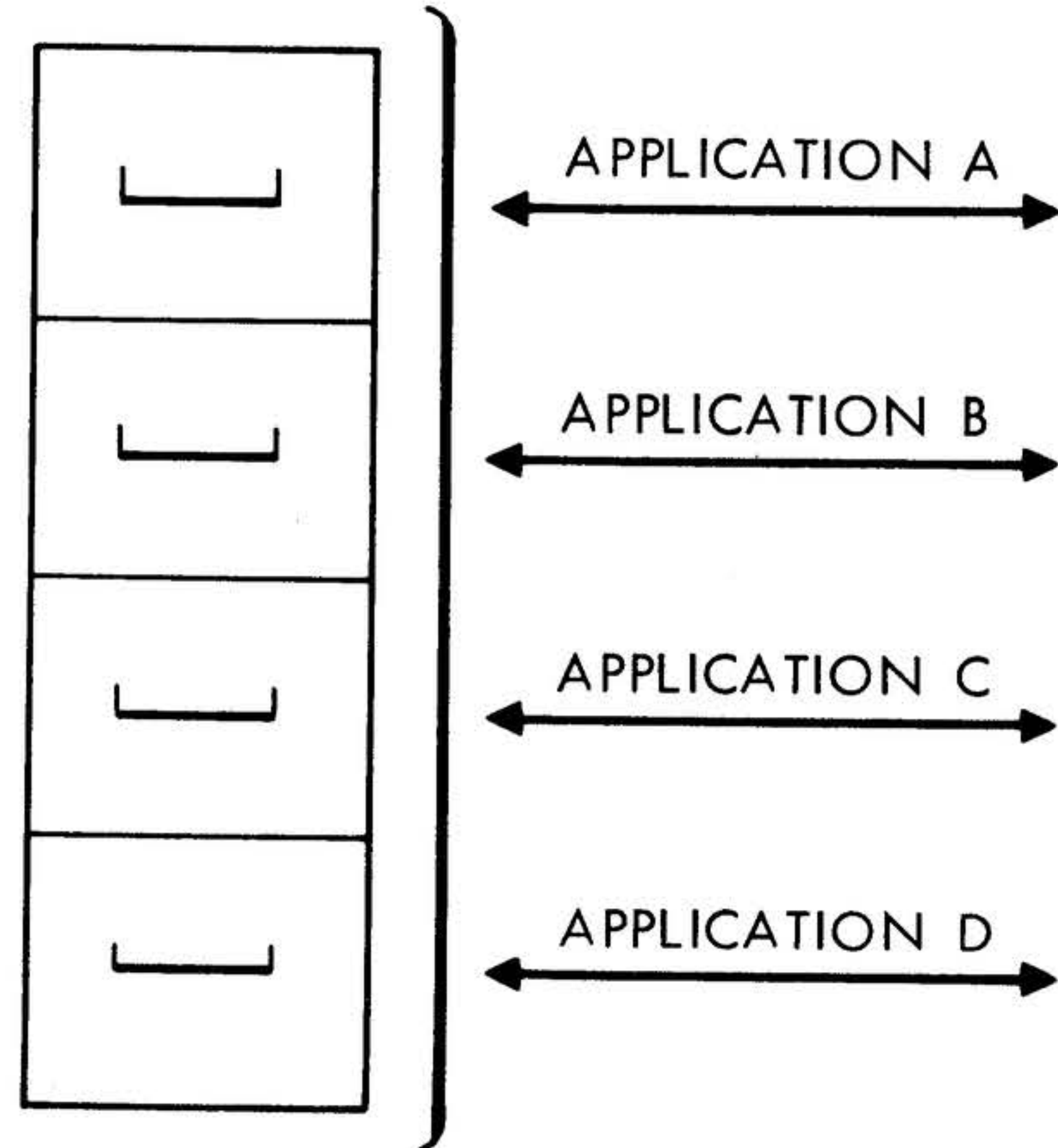
COMMON APPROACH

SEPARATE FILES FOR EACH APPLICATION PROGRAM



TOTAL APPROACH

SINGLE DATA BASE, SHARED BY ALL APPLICATION PROGRAMS



VT11-3431

Figure 1-1. Advantages of a Single Data Base



- o RELATABILITY - Every element of information is capable of being easily linked to every other related element.
- o NON-REDUNDANCY - No information details are stored more than once, and linkage codes are non-ambiguous and brief.
- o ACCESSIBILITY - All information is easily accessible, with minimum dependence on indexes or directories.
- o DATA INDEPENDENCE - Changes in one part of the data base do not jeopardize the accessibility or integrity of any other information in the system which is being used at the time of existing application programs.
- o EXPANDABILITY - New types of information and new data links are easy to add, without affecting the established data base system.
- o PROGRAM TRANSPARENCY - The physical location of the data no longer need be of concern to the application programmer; he is able to communicate with the data base system in his customary assembly or higher-level language.
- o SOFTWARE MODULARITY - Only the system modules required for the program being executed need occupy space in the computer's main memory.
- o HARDWARE UTILIZATION - Both main and secondary storage can be used in an efficient manner, without wastage due to partially filled sectors or deleted data.
- o COST SAVINGS - Each of the above advantages contribute directly to the dollar savings that result from the installation and implementation of a data base system.
- o USER ORIENTED - A Data Base Management System is user-oriented and can be learned quickly, understood easily, and used conveniently for a wide range of applications.

1.2 KEY FEATURES OF VORTEX TOTAL

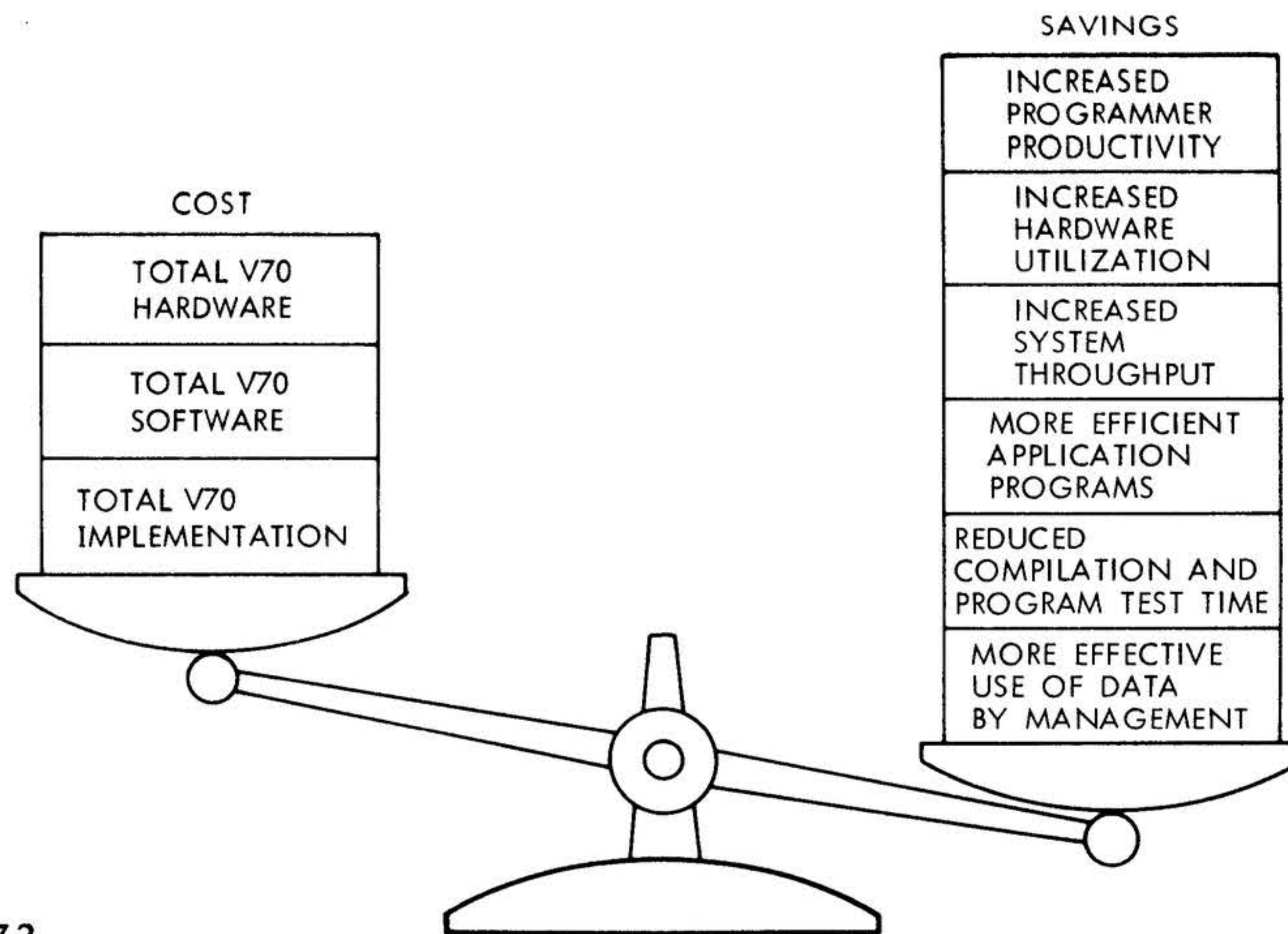
VORTEX TOTAL is a host language Data Base Management System which provides an effective method for organizing and managing diverse data to make it both efficient and convenient for application programmers to maintain and retrieve the data for processing. It performs in both a batch and on-line environment. It is physically organized, through data chaining, as a network data base.



Key features of the TOTAL Data Base Management System are:

- o An unlimited number of data files can be managed.
- o Direct linkage between logical data groups is provided.
- o Instead of using a rigid level-by-level pattern of access, only those qualifiers needed for each specific accessing operation are used.
- o Users retain control of structuring the data and the files.
- o TOTAL is designed for transaction-oriented systems.
- o Duplication and redundancy of data is eliminated, because data is stored only once, regardless of the different criteria by which the information may be accessed.
- o Changes needed in application programs can be made at minimum cost, because the data is separate from the program.
- o Changes to data files can be made without having to modify programs that do not require the new data elements.
- o TOTAL enables changes in hardware configurations to be made without affecting the data files.
- o TOTAL can be used with programs written in FORTRAN, RPG II, COBOL, and DASMR (Varian assembly language).
- o Integrity and security of data is assured by means of data set locking. Control of the data input and output by assigning a Data Base Administrator can additionally increase data security.

Figure 1-2 compares the cost of implementing a Varian Data Base Management System with the cost savings which result from its implementation.



VTI1-3432

Figure 1-2. Comparison of Cost and Savings

1.3 HARDWARE CONFIGURATION AND REQUIREMENTS

The size of the Varian computer and peripheral complement to support a TOTAL user varies considerably depending upon such factors as the size of the data base, the frequency of access and level of multiprogramming within VORTEX II. However, there is a physical minimum configuration for operation VORTEX II and TOTAL. Following is a definition of the various V70 computer systems for use with TOTAL.

1.3.1 Minimum Configuration

- o Any V70 series processor with a least 64K words (128K bytes) of main memory and the extended instruction set.
- o Memory map option - standard in V74 and V75 computers.
- o 512 words of writable control store - standard in V74 and V75 computers.



- o Priority interrupt module (PIM)
- o 33/35 ASR Teletype (or compatible CRT) connected to a PIM.
- o Rotating memory device (RMD) connected to a PIM with either a buffer interlace controller (BIC) or block transfer controller (BTC).
- o One of the following connected to a PIM:
 - o Card reader with a BIC
 - o Magnetic tape unit with a BIC

1.3.2 Typical Configuration

Data base applications typically require large amounts of disc storage and moderate to high volume printed output. Disc backup is provided by periodic dumping of the data base. Supporting this type of user would require adding the following to the minimum configuration:

- o Additional or larger capacity rotating memory devices
- o Line printer
- o Magnetic tape unit

1.3.3 Expanded Configuration

The minimum configuration for TOTAL utilizes the commercial firmware package in writable control store (WCS). In the event that a user wants to run FORTRAN IV also and utilize WCS, a second 512 word WCS for the scientific firmware (FORTRAN accelerator) must be added.

For a large scale data base application, a typical VORTEX II system would include the addition of:

- o One or more magnetic tape units
- o Data communications multiplexor with necessary line adapters
- o One or more Teletype or compatible CRT terminals



A TOTAL Data Base Management System is shown in figure 1-3. Figure 1-4 shows schematically, the required and optional hardware and their connection to the I/O bus.

1.4 BIBLIOGRAPHY

The following manuals contain information on Varian hardware and software that would be helpful to the TOTAL user (the x at the end of each document number is the revision number and can be any digit 0 through 9):

<u>Title</u>	<u>Manual Number</u>
V72 System Handbook	98 A 9906 20x
V73 System Handbook	98 A 9906 01x
V74 System Handbook	98 A 9906 21x
V75 System Supplement	98 A 9906 22x
VORTEX II Reference Manual	98 A 9952 24x

1.5 DEFINITION OF TERMS

1.5.1 Logical Components

Logical components in a TOTAL data base are entities a user sees and deals with. The basic components are:

DATA ITEM: the smallest identifiable and accessible data entry.

DATA ELEMENT: a collection of one or more data fields or a collection of one or more data items.

DATA RECORD: a collection of data elements and data items.

DATA SET: a collection of data records. Same as a data file.

DATA BASE: a collection of data sets.

LINKAGE PATH: a special associative data element reserved by TOTAL to relate associated records.

MASTER DATA SET: an independent entity which can be stand-alone or have one or more dependent (variable) data sets attached to it.

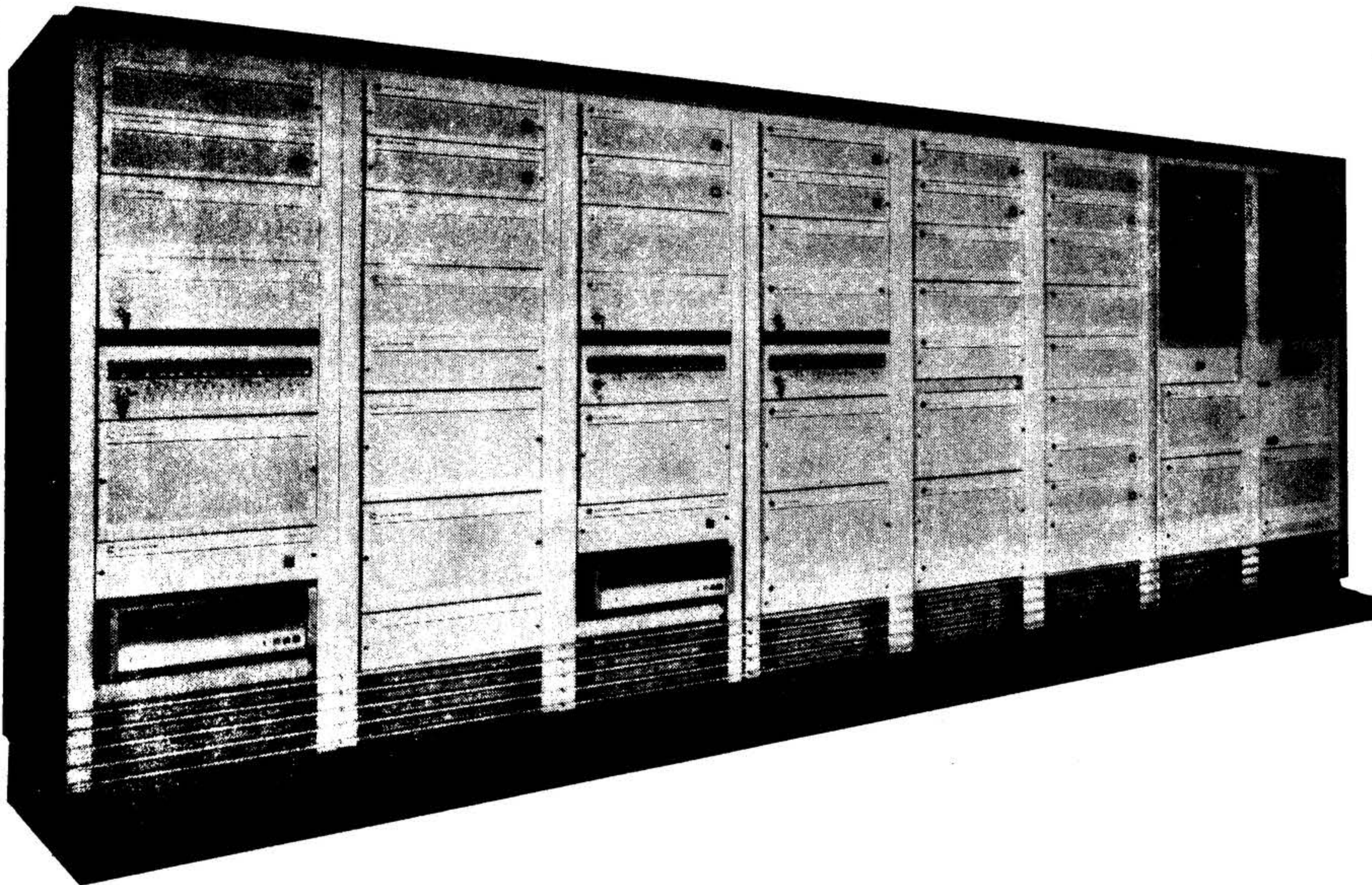
MASTER SYNONYM RECORDS: master data set records with different control keys which randomize to the same "home" address.

ROOT: a special associative data element reserved by TOTAL which relates master synonym records to their "home" record.

VARIABLE DATA SET: a data set containing records which are logically accessed through a specific master record.



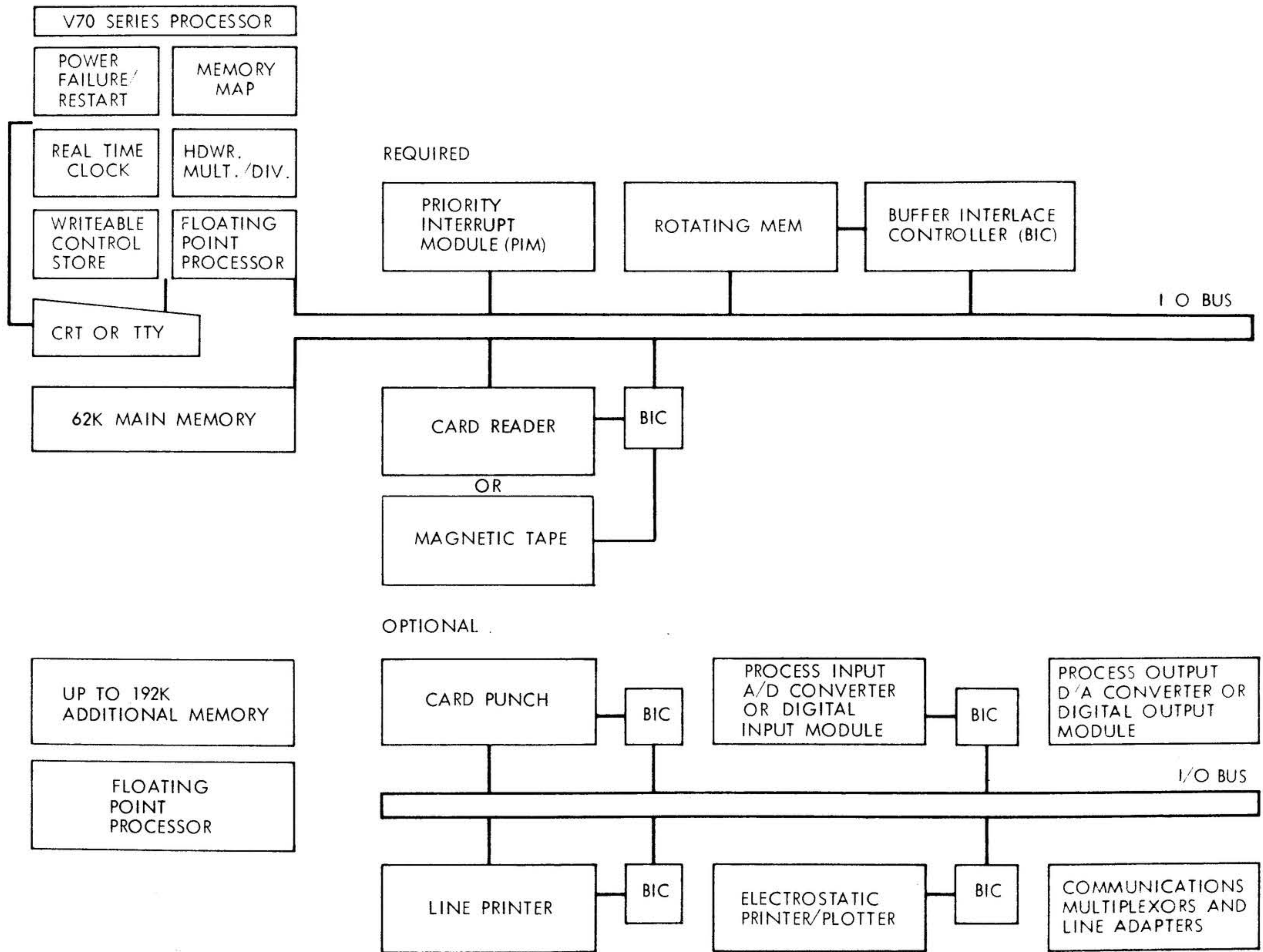
varian data machines



VW11-5455

Figure 1-3. TOTAL Data Base Management System

VT11-3434
 Figure 1-4. Required and Optional Hardware





1.5.2 Special Names

TOTAL has some terms with special names which have a specific meaning. They are:

CONTROL KEY: a unique identifier which identifies a record in a single entry data set. It is used in a variable record to identify to which master record a variable is related.

CODED RECORDS: a facility of the TOTAL system which provides the variable entry file with the ability to contain records with a variable number of data formats.

DATBAS: TOTAL entry point from user call statement. Must be defined as an external.

DBDL: Data Base Definition Language - the language by means of which the user defines the TOTAL data base files, their contents, and their associations to each other.

DBFMT: Data Base Formattor - the TOTAL object module which accepts the data set format control cards, and creates and preformats the specified RMD data set areas.

DBGEN: Data Base Generator - the TOTAL program which accepts the Data Base Definition Language (DBDL) statements and generates the Data Base Descriptor Module (DBMOD).

DBMOD: Data Base Descriptor Module - the object module defining the TOTAL data base produced by the DASMR assembler from DBDL assembler language source statements.

DML: Data Management Language - the language provided by TOTAL allowing the user to access and manipulate the data base through calls in his application program.

INTERNAL REFERENCE POINT: the relative record number (rrn) of a record placed in the Reference Field of a DML command.

LOGICAL FILE: a TOTAL file which may contain one or more VORTEX physical files.

LOGICAL RECORD: a record as defined in the DBMOD; may or may not coincide with sector size (physical record).

LOGICAL RECORD BLOCK: a group of logical records within a data set.

LOGICAL UNIT: a VORTEX I/O device or RMD partition, referenced by a number.

RANDOMIZING: a hashing procedure involving 32-bit arithmetic, computed mostly by firmware.



RANDOMIZING ALGORITHM: the algorithm which is used to achieve the hashing procedure.

REDEFINED DATA AREA: that fixed length area of a variable data set record which may be used for different data elements. Each variable data set record with a redefined area is identified by a unique record code.

RELATIVE RECORD NUMBER (rrn): the relative record position in the file e.g. the 3rd record has a 'rrn' of 3, etc.

SERIAL PROCESSING: the method of retrieving records serially.

Master serial processing: get the next available record (not entry) in the file (RDNXT).

Variable serial processing: get next record in file (RDNXT) or next record in the linkpath (READV).

1.5.3 Data Management Language (DML) Functions

TOTAL provides the user with the capability to access and manipulate the data in the data base through various DML command functions which can be invoked by a CALL statement in the application program.

Example: CALL 'DATBAS' USING READM, STAT, CUST, KEY, ELEM-LIST, USER-AREA, ENDP.

The available DML command functions are:

a. Serial Processing Functions

RDNXT: serially read a master or variable entry data set.

b. Master Data Set Functions

READM: read a master record.

WRITM: write a master record.

ADD-M: add a master record.

DEL-M: delete a master record.

c. Variable Data Set Functions

READV: read a variable record along the forward direction of a variable record chain.



READR: read a variable record along the reverse direction of a variable record chain.

READD: read a variable record directly by specifying its position.

WRITV: write the variable record retrieved by the preceding read.

ADDVC: add a variable record to the end of a variable record chain.

ADDVB: add a variable record before the one retrieved by the preceding read.

ADDVA: add a variable record after the one retrieved by the preceding read.

DELVD: delete the variable record retrieved by the preceding read.

d. Special Functions

SINON: sign on a program.

SINOF: sign off program

RQLOC: request the home location of a master record.

1.5.4 VORTEX Definitions

DAS MR: Data Assembler System Macro Relocatable - a Varian macro assembler which produces relocatable object code that can be loaded into any area of memory.

DO: VORTEX debugging output logical unit.

FCB: File Control Block.

PI: Processor Input -a VORTEX logical unit assignment.

PIM: Priority Interrupt Module.

RMD: Rotating Memory Device

SNAPSHOT Dump Program (SNAP): a VORTEX dump program which provides both register displays and the contents of specified area of memory.

SS: VORTEX system scratch logical unit.

TEXT block: a portion of the SNAPSHOT dump which is used to interpret the dump.



SECTION 2 TOTAL DATA BASE MANAGEMENT SYSTEM DESCRIPTION

2.1 SYSTEM DESCRIPTION

VORTEX TOTAL is a Data Base Management System with a network structure design which enables one data record to be linked directly with any other of the total number of data records in the data base. This network structure approach produces cost and time savings by eliminating the necessity for accessing the required data record through a hierachal process of level-by-level qualification.

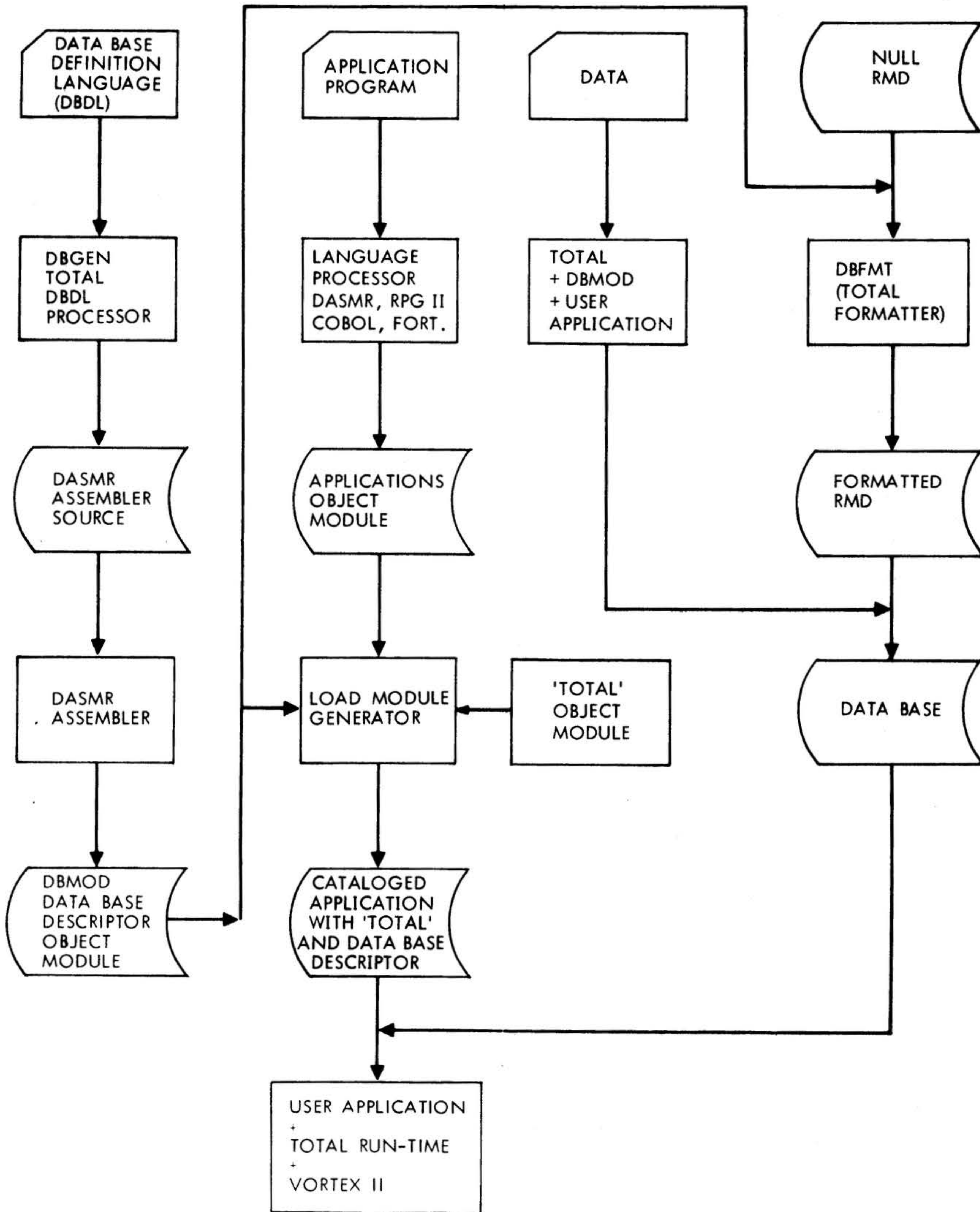
The user structures the data base to be managed by TOTAL by means of the Data Base Definition Language (DBDL). DBDL defines the data base files (data sets), their contents (data records and data elements) and their associations to each other (linkage paths).

After the DBDL statements are completed, the DBDL Processor is used to convert the DBDL statements into assembler language source statements. These, in turn, are assembled into an object Data Base Descriptor Module (DBMOD). The Descriptor module is then catalogued into the user program library. At the same time the Data Base Format program (DBMFT) is used to format the secondary storage area according to the physical requirement of the data base.

The user then uses the Data Management Language (DML) provided by TOTAL to access and manipulate the data base. DML commands are issued via CALL statements in user programs. The functions of the DML commands include the opening and closing of data sets, the accessing and manipulation of master and variable data sets, the serial processing of data sets, and special functions such as program sign-on and sign-off. A flow chart of the TOTAL Data Base Management System is shown in figure 2-1.

2.2 SYSTEM OPERATION

The run time application program consists of three modules, bound together (by LMGEM): TOTAL, the Data Base Descriptor module (DBMOD), and the user program, in that order. The entry point to TOTAL is via DATBAS. When access to the data base is required, the application program calls DATBAS which passes control to TOTAL. TOTAL analyzes the program statements and issues read/write commands. When the task has been accomplished control is returned to the user program. If the function fails to complete,



VTI1-3435

Figure 2-1. VORTEX II TOTAL Data Base Management System



the original data base condition prior to the execution of the call to TOTAL is restored, and a comprehensive diagnostic message to the user is generated.

When changes to the data base are made, the application program remains unchanged. Similarly, when changes to the application program are made, there is no need to re-define and regenerate the data base. A flow chart of the TOTAL DBMS operations is given in figure 2-2.

2.3 STRUCTURE OF THE DATA BASE

The TOTAL data base consists of five levels:

- a. Data Base
- b. Data Set or File
- c. Record
- d. Element and Subelement
- e. Item

An example of a TOTAL Data Base for customer data is shown in figure 2-3.

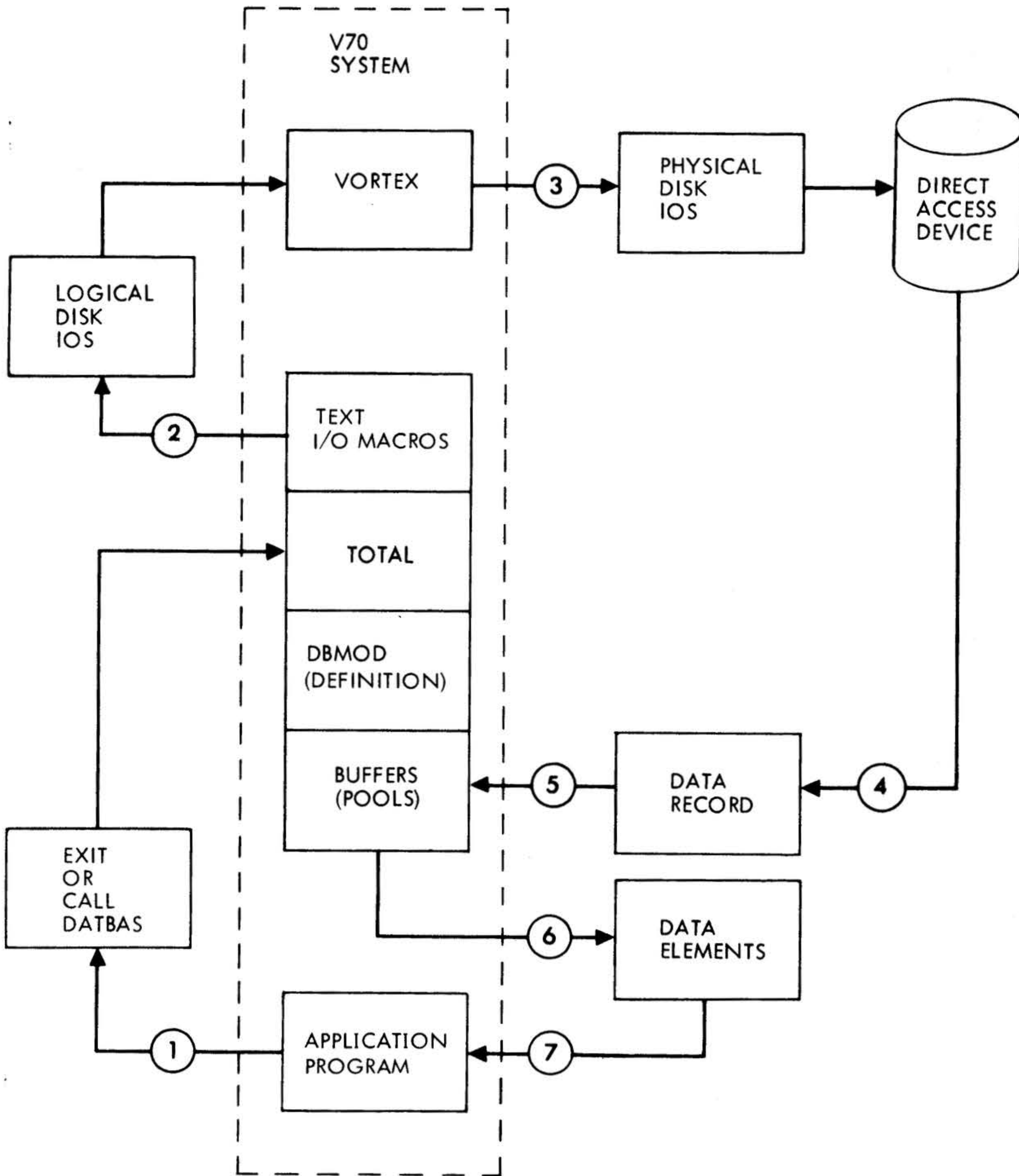
2.3.1 Data Base

A data base is a particular collection of data sets (files) usually (but not required) related and used by a family of application programs. Data bases are assigned 6-character names, such as CUSTMR.

TOTAL manages the data base by managing the relationships among records in the different data sets of the data base. For example, a customer record should be related to its open order records; a finished product record should be related to its component records; a work center record should be related to its production operation records.

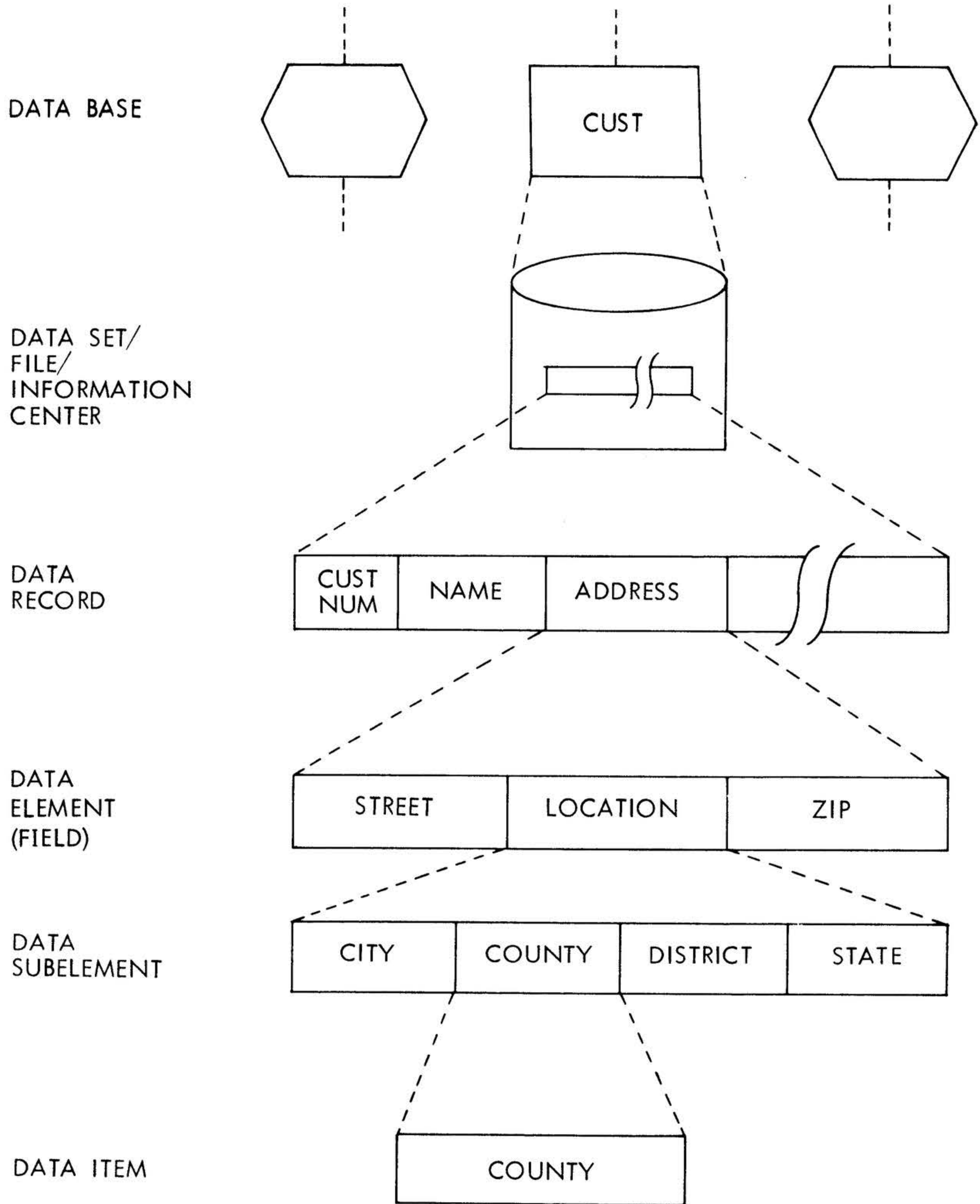
2.3.2 Data Set or Data File

A data set or data file is a collection of data records with the same structure (or collection) of data elements. There are two types of data sets: master (or a single) entry and variable entry (figure 2-4).



VTI1-3436

Figure 2-2. TOTAL DBMS Operational Flow Chart



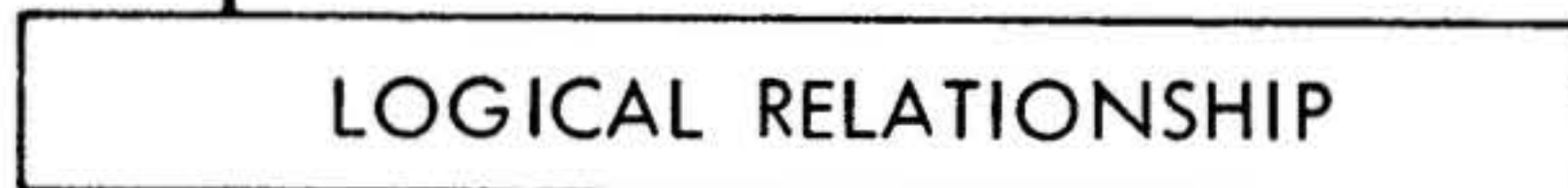
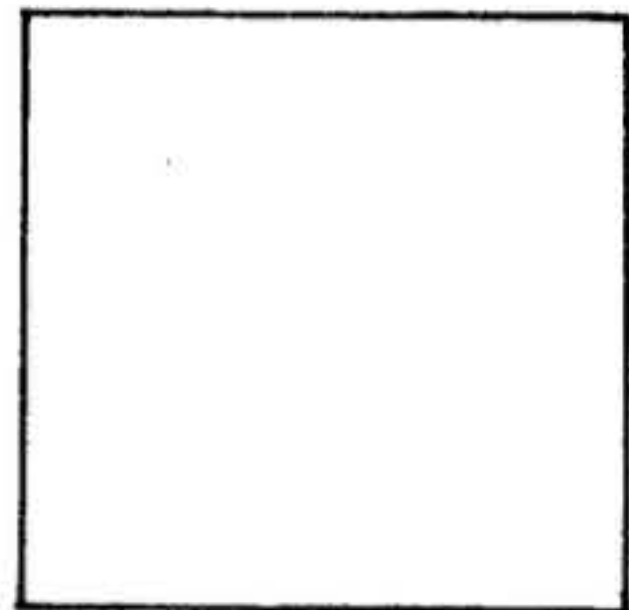
VTI1-3437

Figure 2-3. TOTAL Data Base Components



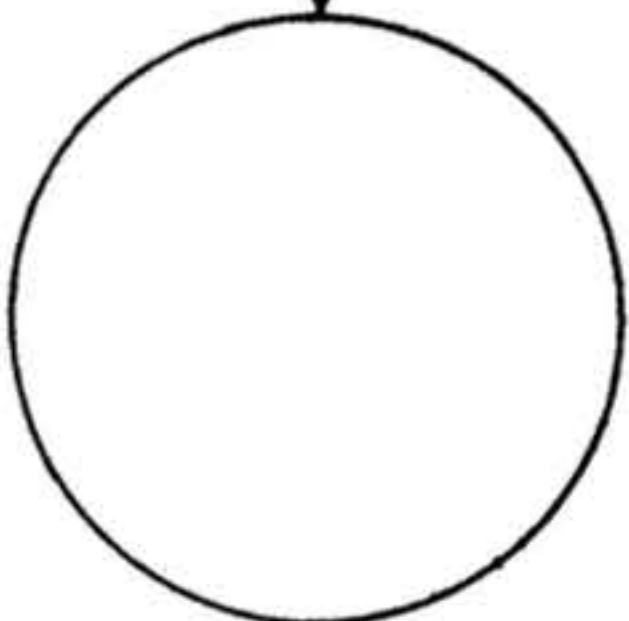
SINGLE ENTRY (MASTER)

- IDENTIFICATION THROUGH UNIQUE CONTROL KEY
- DIRECT (RANDOM) ORGANIZATION
- DIRECT ACCESS THROUGH REFERENCE TO CONTROL KEY
- FIXED-LENGTH RECORDS



VARIABLE ENTRY (DETAIL/TRANSACTION)

- DESCRIPTIVE/SUPPORTIVE OF SINGLE ENTRY RECORDS
- RELATED TO ONE OR MORE SINGLE ENTRY DATA SET RECORDS
- ACCESSIBLE THROUGH CONTROL KEY OF LOGICALLY RELATED DATA SET(S)
- FIXED LENGTH RECORDS
- VARIABLE FORMATS



VTI1-3438

Figure 2-4. TOTAL Data Sets



A master data set is an independent entity that can stand-alone or have one or more dependent (variable) data sets attached to it. Each record in a master data set has a unique control key and the record is to be accessed directly according to the value of this control key.

A variable data set must attach to at least one master data set. Records in a variable data set are logically accessed through a particular master record and then from one variable record to the next within the same variable record chain related to the master record.

In TOTAL, a 4-character name is given to the data set, such as CUST, ROUT, PART, or YARD. The general notation for a data set is ffff; occasionally it is indicated as mmmm or vvvv when a distinction between a master or variable data set is required.

The concept of data set corresponds to that of a traditional file. A data set is often referred to as an information center. However, the term data set and file is used interchangeably in this manual.

TOTAL manages data sets by managing the space allocated to each data set. TOTAL knows how much space is allocated to each data set and loads records directly into their prime location within the data set. Upon deletion of a record, TOTAL reuses the freed location to its best advantage through this optimization process.

2.3.3 Data Record

The Data record is a collection of data elements and data items. It conforms to the traditional concept of a record and is identified by a control field or logical key.

TOTAL manages the data records by transferring physical records between memory and the direct access device with the logical read and write. TOTAL supports only fixed length records, blocked or unblocked. The structural capabilities of variable entry data sets satisfy all situations for which variable length records were devised.

Within data records there are special associative data elements reserved for use by TOTAL to relate data records. They are linkage path and root.



Linkage path relates associated records. A master record can be linked to a variable record. A variable record can be linked to another variable record. It is permissible to have multiple linkage path fields in a record and thereby to either (1) relate a given record to many other records within the data base, or (2) relate two records in more than one way.

Root relates master synonym records to their "home" record. When randomly adding records to a data set, it is possible to have more than one record assigned to a specific physical location. When this occurs, TOTAL automatically finds a different unused physical location for the extra records and then uses the root element to relate these displaced records to their "home" locations.

2.3.4 Data Element

The data element is a collection of one or more data items. Data elements must be given a unique 8-character name; the name and the content of the element is determined by the analyst.

An element name is indicated by mmmmxxxx or vvvvxxxx e.g., CUSTCTRL, PARTDESC, ROUTMACH, or YARDNAME, where mmmm is the master data set name, vvvv is the variable data set name, and xxxx is the name of the data element.

TOTAL manages the data elements by selection and insertion of the elements from or to the logical record by means of the Data Management Language (DML) function specified by the application program. Under TOTAL the application program requests and offers data elements. These elements are specified to TOTAL via an element-list. This programmer-written list names the elements desired which have previously been described in the data definition statements (refer to section 3.3).

2.3.5 Data Item

Data item includes the conventional field of data; for example, customer's name, amount of sale, or department number.

In effect, data item is the smallest identifiable and accessible data entry.

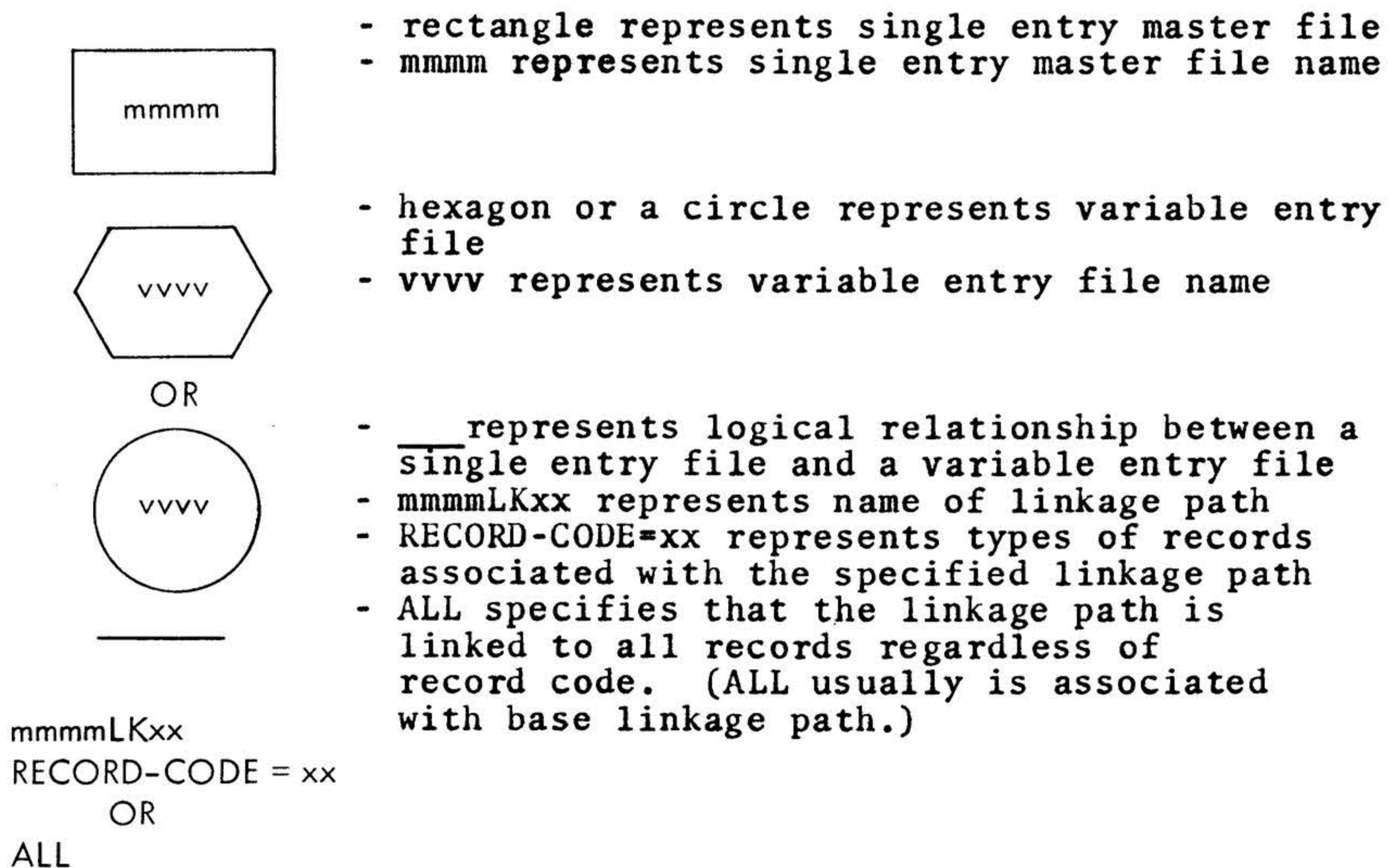


TOTAL manages the data items by documenting the structure of the data element with the Data Base Definition Language (DBDL). The flexibility of data element definition gives the system analyst the capability to define anything from the smallest item in a record to the entire record itself as an element.

2.4 DATA BASE SCHEMATIC

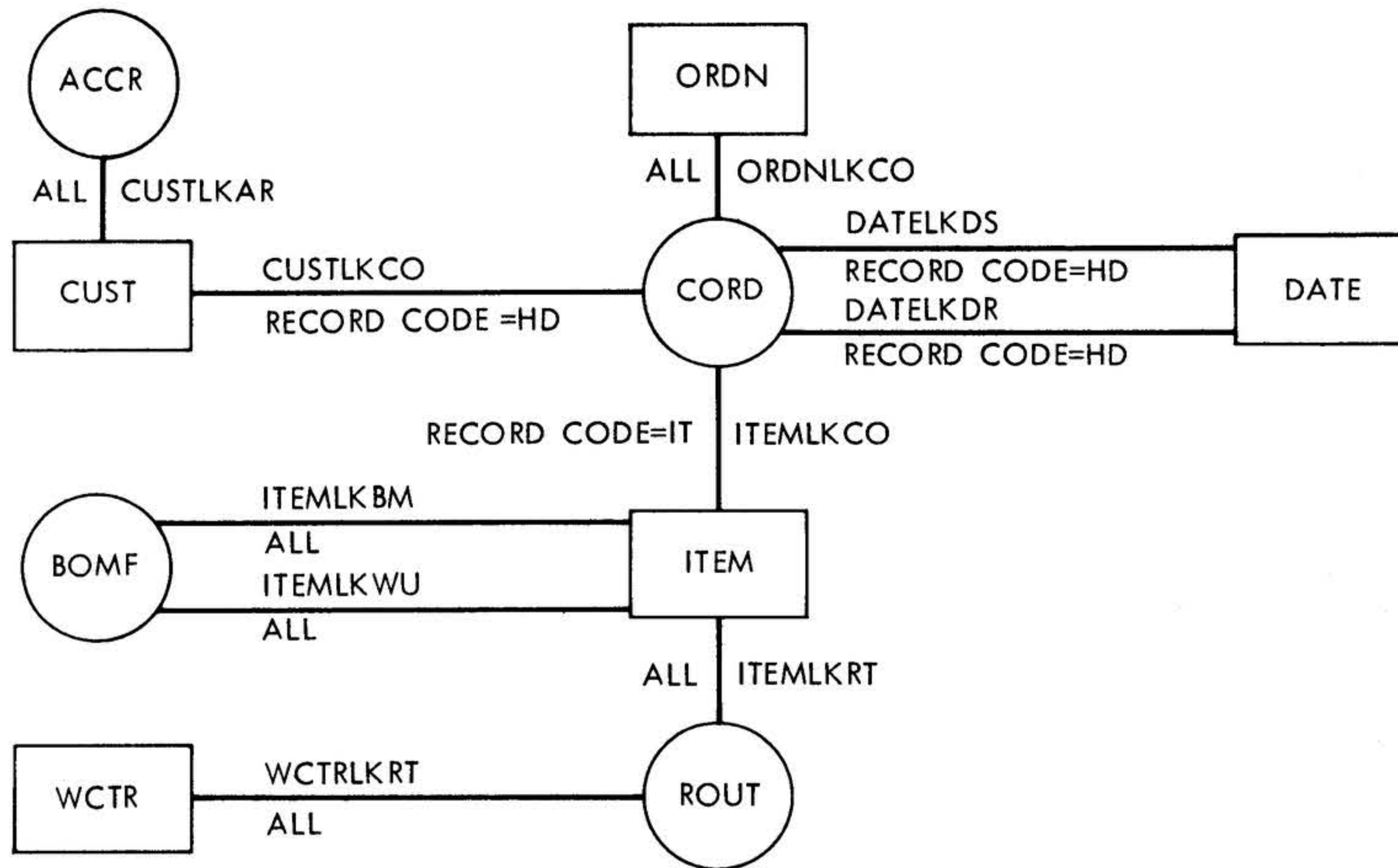
Since the TOTAL system provides the facilities and techniques for the creation of large, complex network data bases and the effective processing of these data bases, it follows that there should be an effective technique to graphically document and represent these data bases. The technique is called the Data Base Schematic.

The data base schematic uses graphic symbols to represent the data base module and submodules. These symbols are shown in figure 2-5. A schematic of a data base using these symbols is shown in figure 2-6.



VTI1-3439

Figure 2-5. Symbols Used to Form Schematic of the Data Base



VTI1-3440

Figure 2-6. Schematic of Data Base

2.5 DATA RELATABILITY

The direct relatability among different groups of data is the foundation of a TOTAL data base. If you examine various day-to-day applications, you can find many examples where one group of data is related to another group of data, which may relate to still another group, such as:

- Customers and open orders
- Open orders and inventory orders
- Finished items and components
- Production orders and labor tickets
- Labor tickets and employees
- Employees and skill classifications
- Student and courses
- Bank depositors and account balances



Insurance policyholders and policy coverages

Patients and medical history

Tax payers and tax payments

Authors and publications

In a typical business environment, you want to access information about any customer quickly. Since there are transactions involved with each customer, the efficient processing of transactions corresponding to respective customers is important. With TOTAL the logical approach to organize the two sets of information and their association is:

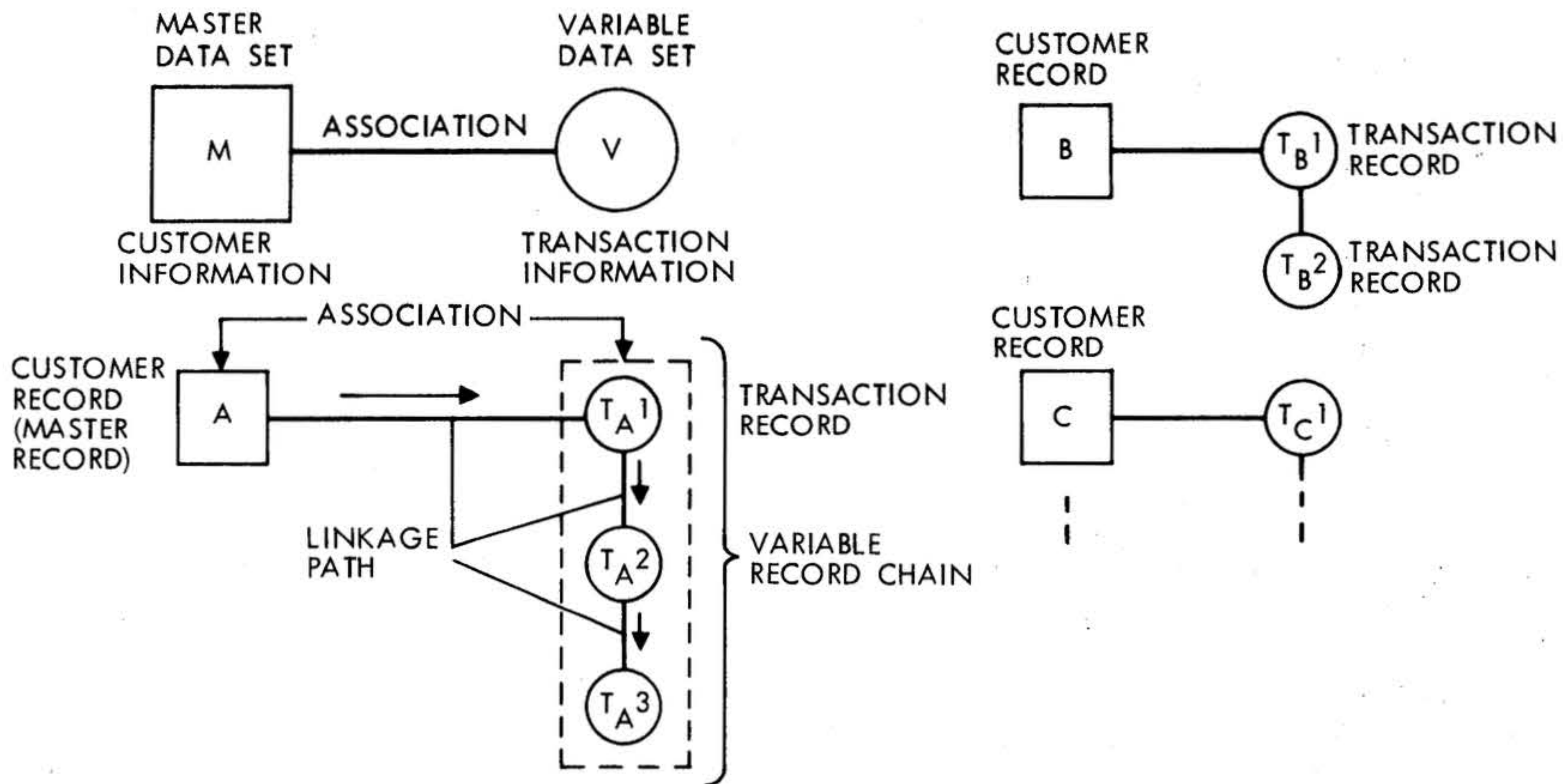
To have the information about each customer in a record directly accessible by customer number. All the customer records are grouped together to form a data set or file. Such a data set is called a master data set because of its independence. The records in it are considered master records.

Transactions corresponding to a particular customer are linked together in a chain, which, in turn, is linked to the associated customer record in the master data set. The transaction records collectively comprise a variable data set. A variable data set is dependent and must be attached to a master data set. The records in a variable data set are called variable records. Figure 2-7 illustrates two related data sets.

Figure 2-7 also illustrates that:

- o Transaction records corresponding to a particular customer are chained together because of their association with that customer.
- o A customer record is chained together with a group of transaction records because of its association with those transaction records.
- o A customer record can be accessed directly by a customer number.
- o From a customer record, the first associated transaction record in a chain can be accessed through an access path or linkage path: from the first transaction record, the second transaction record in the chain can be accessed, and so on.

NOTE: Linkage path information is directly stored within a data record.



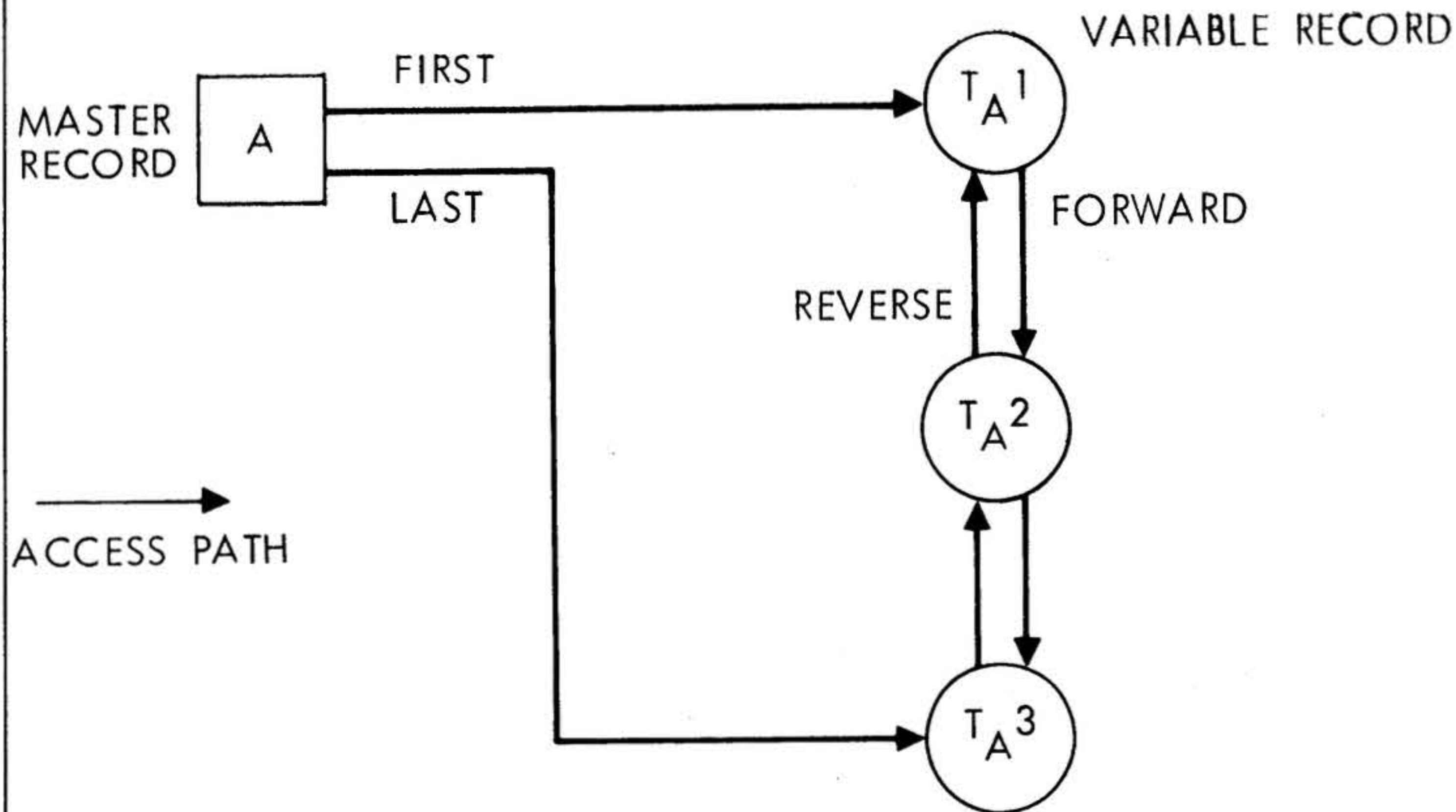
VT11-3441

Figure 2-7. Two Related Data Sets

Figure 2-8 shows that when a chain of variable records is linked to a master record, a two way closed loop is formed.

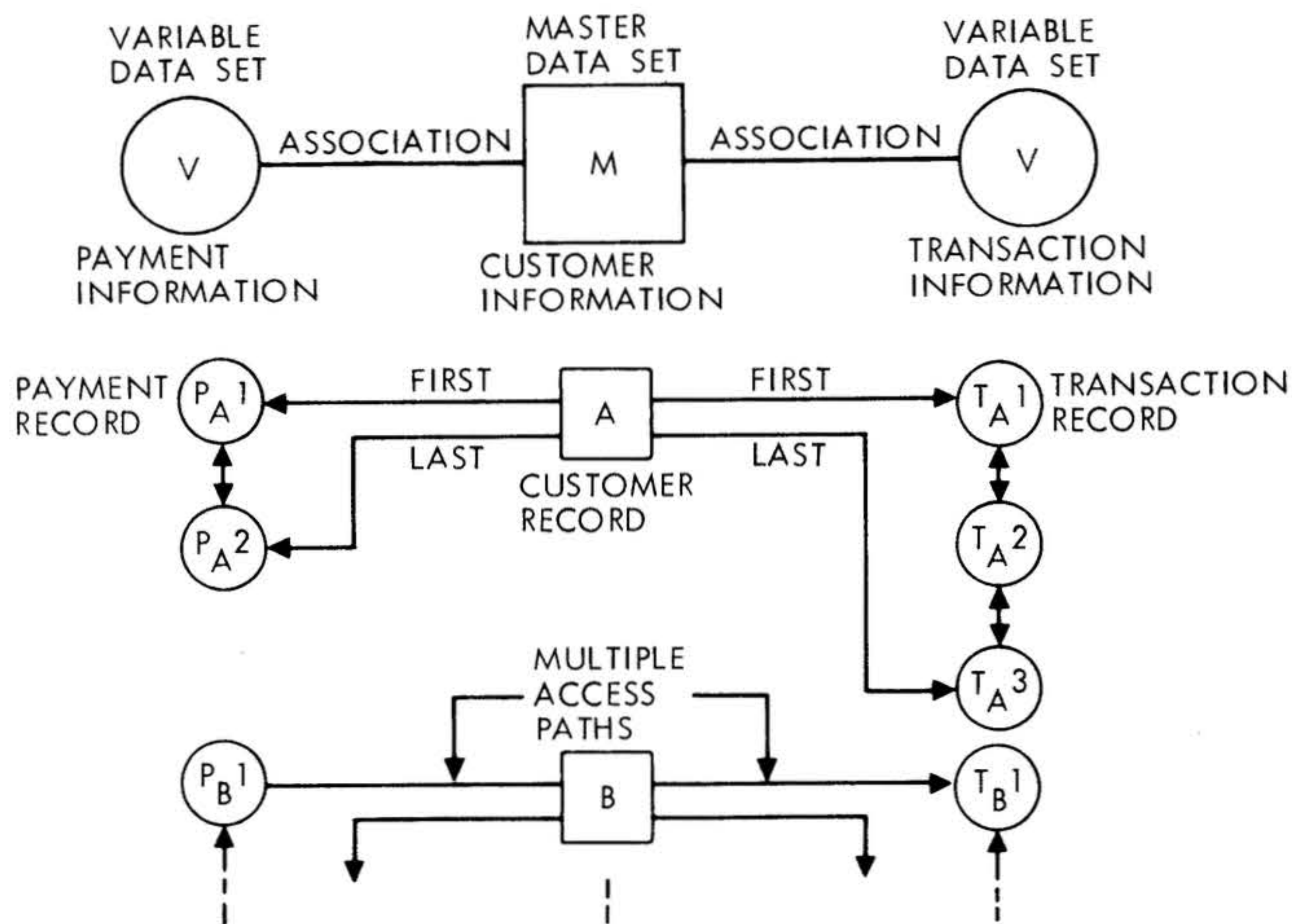
The master record stores information (or links) of two link paths: one to the first record of the variable record chain and the other to the last record of the variable record chain. Each variable record also stores information of two access (or link) paths: one to the next variable record in the chain and the other to the previous variable record in the chain (a forward link and backward link). Thus, the processing of a variable record chain can be started from one end or the other; and in forward or reverse direction, indicating the presence of multiple access paths.

The association can now be expanded beyond two data sets. Again, in a typical situation, a customer may make payments from time to time. If the information about each payment made by each customer is kept in a record, another variable data set can be established. Records of payments made by the same customer are linked to the corresponding customer record of the customer data set. Now there are two variable data sets associated with one master set as shown in figure 2-9. It is significant to note that by accessing a customer record, one can immediately reach the information of transactions and payments made by that customer. Thus, multiple access paths are again established.



VTI1-3442

Figure 2-8. Record Association



VTI1-3443

Figure 2-9. Two Variable Data Sets Related to One Master Data Set

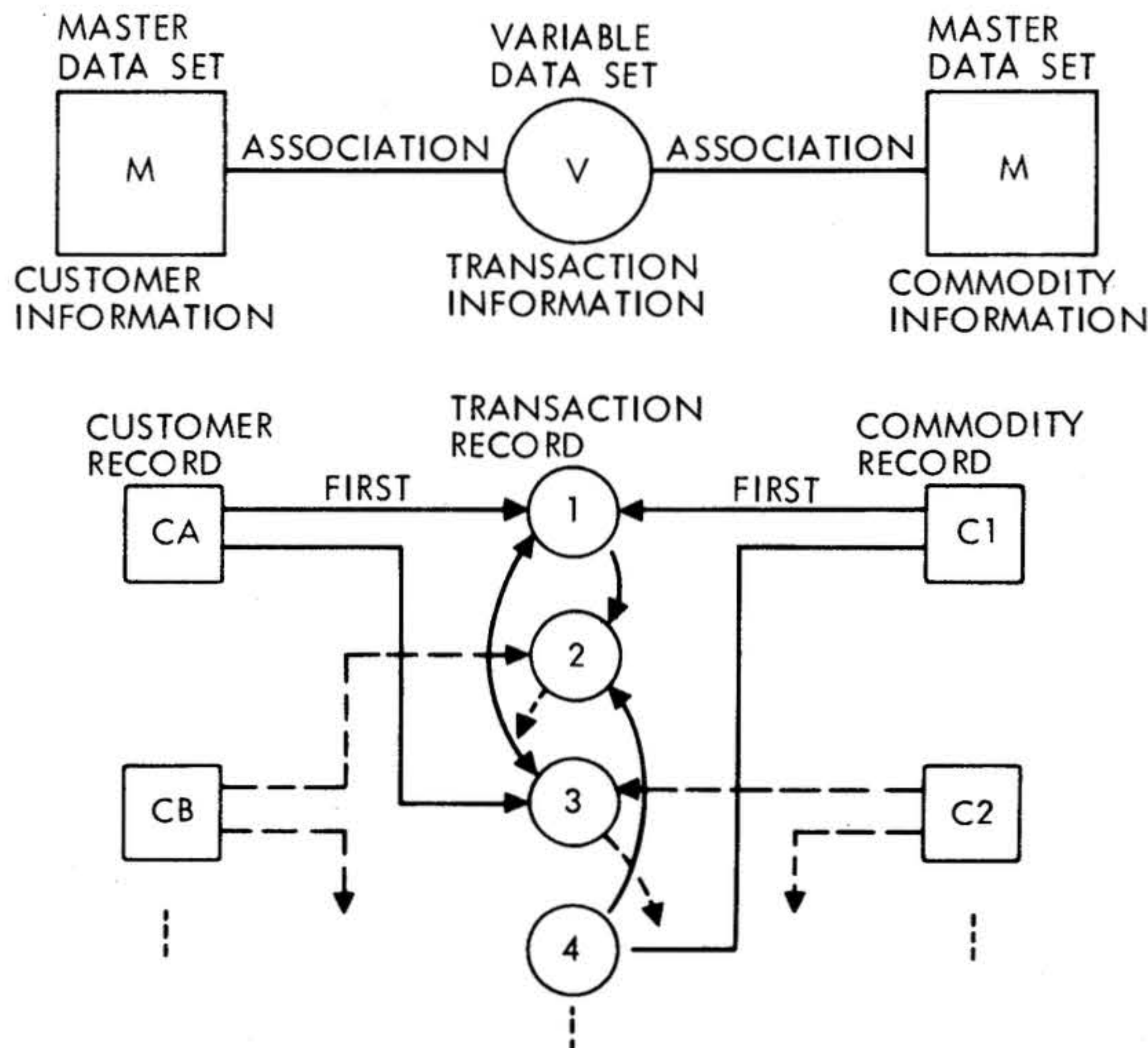


However, each customer order transaction may be involved with a particular commodity. If a record is used to describe each commodity, a master data set can be established to include all such records. The master data set may simply be stand-alone. When a commodity number is encountered during the processing of a transaction record, the commodity number can be used as a control key to directly access the record describing that commodity from the commodity master data set.

2.6 TOTAL DATA SETS

TOTAL's data sets are logical direct access data sets. The most significant direct access characteristic of TOTAL is - records are fixed length, blocked or unblocked.

However, if it is important to know which transactions have requested a certain commodity, you can establish an association between the transaction data set and the commodity data set. Thus, transaction records have requested the same commodity link together to form a chain, which, in turn, links to the record describing that commodity. In this case, one variable data set associates with two master data sets as shown in figure 2-10. Records 1,2,3,4.... belong to the variable TRANSACTION



VT11-3444

Figure 2-10. One Variable Data Set Related to Two Master Data Sets



data set. Records 1,2, and 4 are chained to Master record C1 of the Master Data Set "Commodity Information." Records 1 and 3 are chained to Master record CA of the Master Data Set "Customer Information."

Since there are no indexes to the TOTAL files, all logical records are retrieved directly; that is, with a typical maximum access operation of one seek and one read. In the case of blocked records, if the next record requested is in main storage, no physical operation is performed. TOTAL does not perform redundant operations.

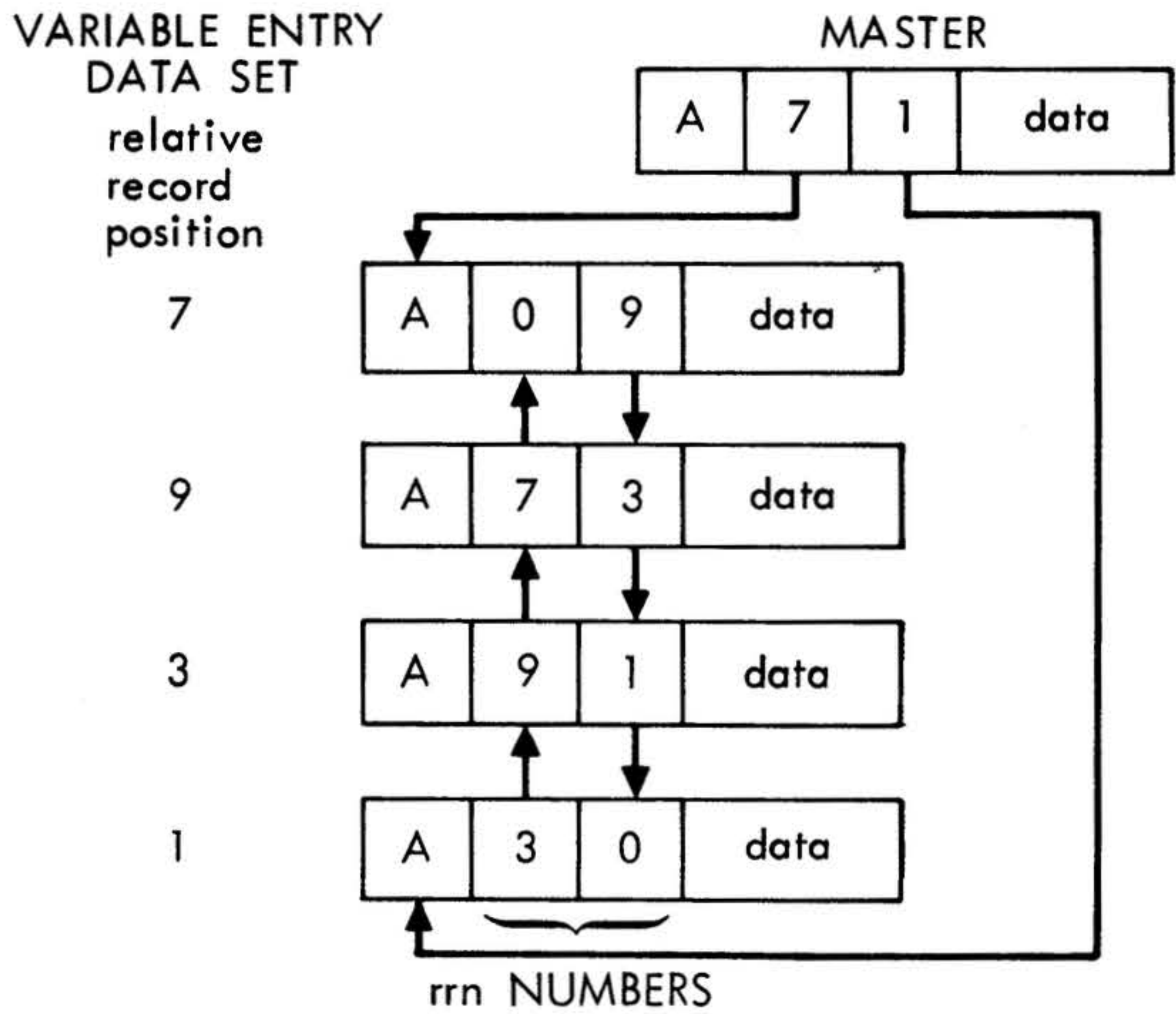
With no separate additional overflow areas available, all added records go into the prime data area's most optimum location according to TOTAL's space management techniques. Therefore, record addition to TOTAL files does not degrade performance.

All record deletions in TOTAL's files are physically removed, freeing the record location for immediate reuse by the TOTAL system. In fact, the freed location can be immediately used by the TOTAL system.

Since TOTAL files are not degraded by record additions and are optimized upon record deletions, TOTAL files do not need to be reorganized. However, a TOTAL file must be reloaded when a physical parameter is changed; for example, increased record length or increased number of records. Even in this case, neither related files nor existing application programs are affected.

TOTAL files can be processed serially but not sequentially; that is, there is no inherent logical serial/sequential nature among the logical records of a TOTAL file. TOTAL stores records randomly according to its internal space management techniques and retrieves records directly. To rearrange a TOTAL file into some desired logical sequence, the user must simply sort the file.

There are two function types of data sets in the TOTAL system: the single entry (master) data set and the variable entry data set. Figure 2-11 depicts the characteristics and relationships of these data sets. All links denote relative record number (rrn).



VTI1-3445

Figure 2-11. TOTAL Data Sets

It is permissible to have stand-alone master data sets, one variable data set associated with multiple master data sets, and one master data set associated with multiple variable data sets. However, it is not permitted to have a stand-alone variable data set, direct association between two master data sets, or between two variable data sets.

When a number of master data sets and variable data sets are meaningfully associated in a group, a data base is formed. Since data sets and their associations spread out horizontally and vertically like a net, the corresponding data base can be described as having network structure.

2.6.1 Single Entry (Master) Data Set Characteristics

Single entry data sets provide the following functions:

- a. Store information for immediate direct retrieval by an application program



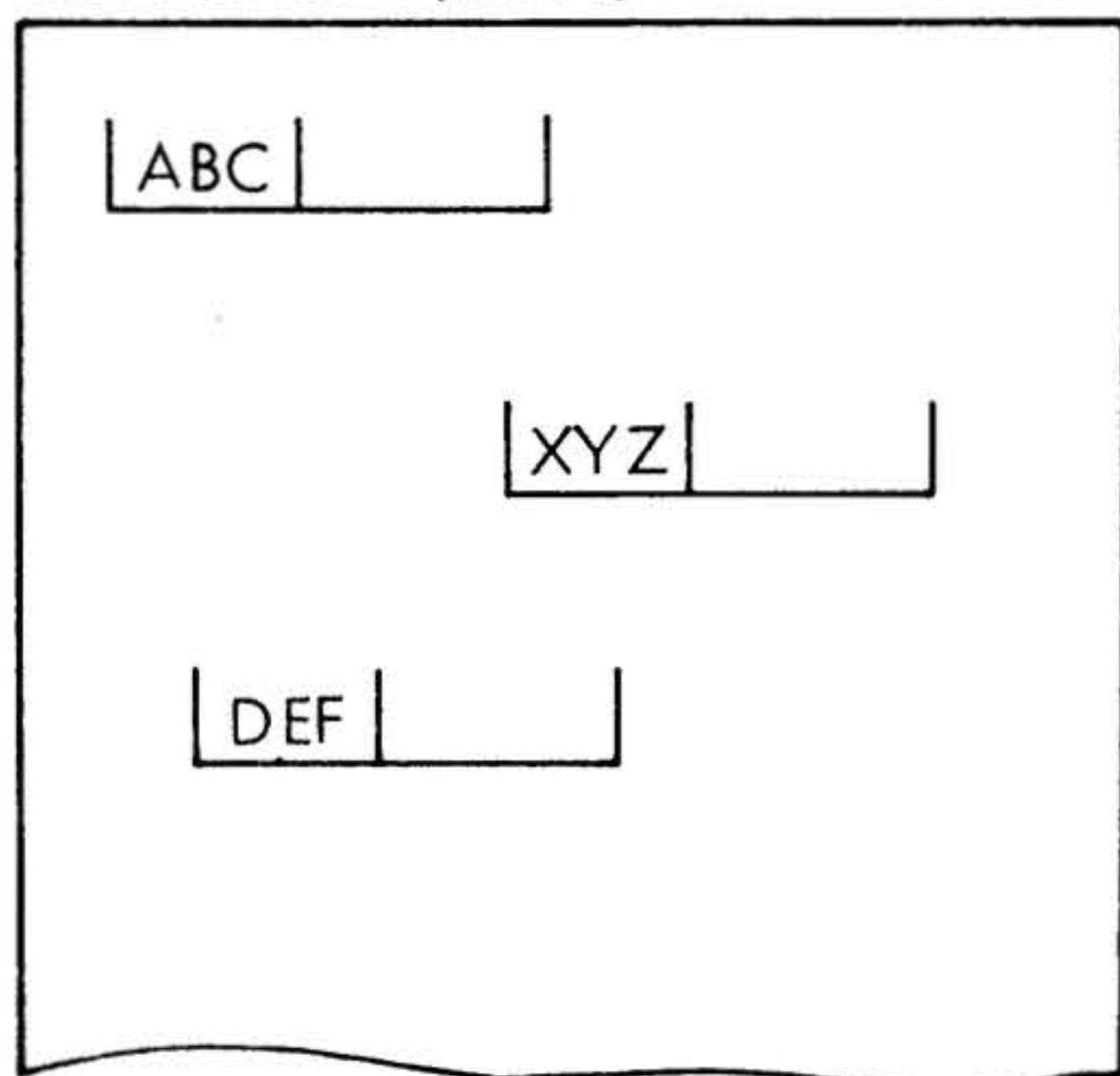
- b. Serve as entries or masters to strings of related information, and
- c. Edit the control and qualification of the data entering the data base.

Single entry (master) data set records are stored randomly and retrieved directly by the identification of the control field or logical key of each record (figure 2-12). Each single entry (master) data set record has only one key (each key has only one record) and each record is retrieved by requesting the record associated with a particular key.

Each single entry (master) data set record can "own" records of many variable entry data sets. This facility, call relatibility, is accomplished by TOTAL's linkage path techniques and is discussed in detail later in this section.

To illustrate the characteristics of the single entry (master) data set, consider the example of a customer master file, such as a patient master file for hospitals. It contains all pertinent information about the customer: number, name, address, and other information unique to the specific business environment. Assume this information is stored on disc so that customer information can be retrieved directly. This is easily and quickly accomplished

CUSTOMER (CUST)



KEY (CONTROL FIELD) PER RECORD
 RECORD PER UNIQUE KEY
 FORMAT RECORD
 CONVENTIONAL METHOD OF ACCESS

VTI1-3446

Figure 2-12. Single Entry (Master) Data Set Records Retrieval



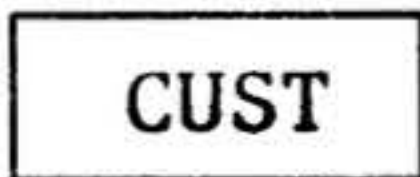
with the TOTAL system. Using the statements of the TOTAL Data Base Definition Language (DBDL), the definition of the data base contains only one data set. The DBGEN program could be coded as follows:

```

BEGIN-DATA-BASE-GENERATION:
.
.
.
BEGIN-MASTER-DATA-SET
DATA-SET-NAME=CUST
IOAREA=POOL1
MASTER-DATA
CUSTROOT=8          REQUIRED BY TOTAL
CUSTCTRL=6          CUSTOMER NUMBER (control field; key)
CUSTNAME=30         CUSTOMER NAME
CUSTADDR=30         CUSTOMER ADDRESS
CUSTCTYS=20         CUSTOMER CITY-STATE
CUSTDATA=100        CUSTOMER INFORMATION
END-DATA
.
.
.
END-MASTER-DATA-SET
END-DATA-BASE-GENERATION

```

The schematic of this data base would be:



(Rectangle represents single-entry data set)

This data base now permits direct access to any element or list of elements within any customer record. The control field (or logical key) is the 6-character customer number.

2.6.2 Variable Entry Data Set Characteristics

Variable entry data set records are also stored randomly and retrieved directly, but not only by the identification of a unique control field. Variable entry records within the data set are stored serially. The identification of a control field directs TOTAL to relate variable entry records containing that same control field to each other in what is called a chain of records, each record being a member of a logical linkage path.

Each control field of a variable entry file must be defined as the logical key of an existing single entry (master) record. The single entry record for the unique key is linked to the first and last record in the linkage path of the variable entry record controlled by that key. Thus the single entry record "owns" all variable entry records that contain that unique control field.

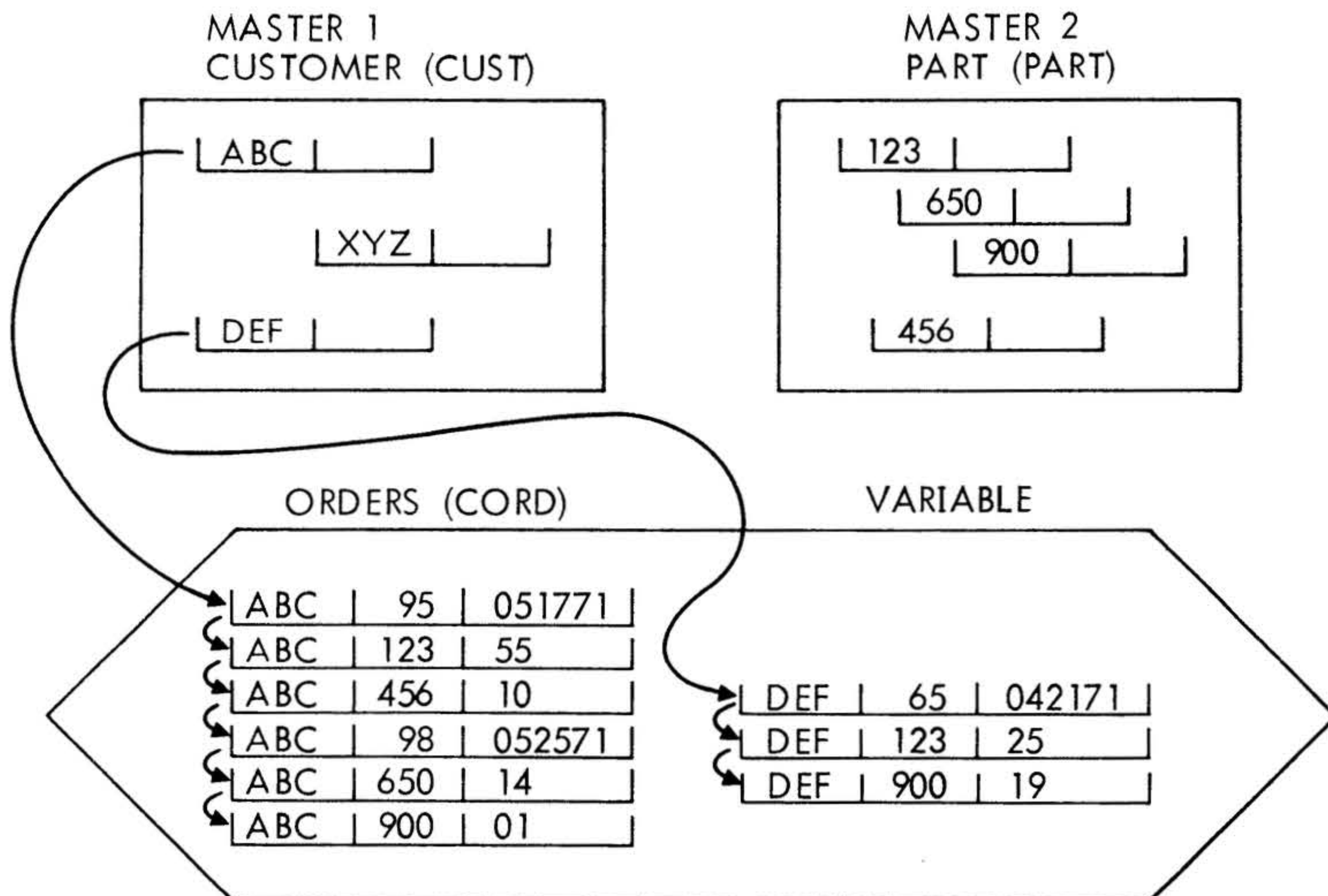


In a variable entry file each record can have a variable number of control fields; therefore, each record can be a member of a variable number of linkage paths.

Each unique control field can have a variable number of records containing the control field, each record being linked to the next record in the chain and the previous record in the chain.

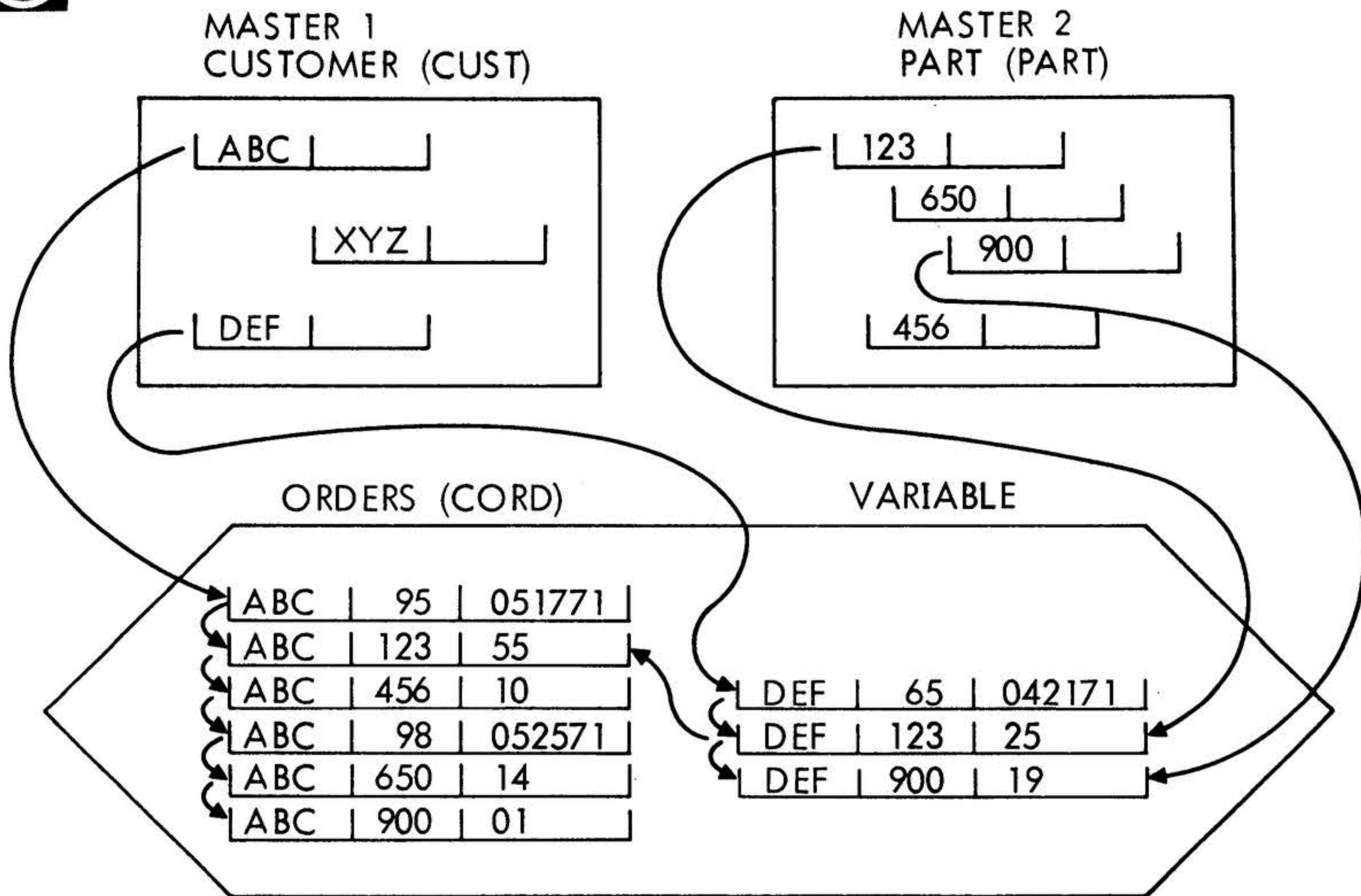
Each record can be retrieved by variable means; that is, each record may be retrieved as a member of a chain (or list) of records for each defined control field in the record as shown in figure 2-13.

The variable entry data set can have variable format records. Furthermore, these different "record types" can be linked to different control fields as shown in figure 2-14. Remember that even though the records may have different amounts of significant data, they are still fixed length.



VTI1-3447

Figure 2-13. Variable Number of Records Per Key



VTI1-3461

Figure 2-14. Variable Keys Per Record, Variable Conventional Method of Access, Variable Record Format

Each single entry (master) data set can be linked or related with up to 2,500 variable entry data sets; conversely, each variable entry data set can be linked to an unlimited number (up to 2,500) of single entry (master) data sets.

This linkage or relatability capability is completely maintained by the TOTAL system by the fact that certain fields are defined to the TOTAL system as linkage-control fields and the application program simply processes logical records that contain the control field data.

To illustrate a variable entry data set, consider the example of a customer open order file. This file essentially consists of a series of records for each customer containing order information for each order the customer has placed. A conventional technique for storing this information is to provide "buckets" within each customer master record into which the order information is placed. A severe limitation of this technique is that some customers exceed their number of buckets, thereby causing the problem of record overflow; while other customers do not use all of their buckets, thereby causing wasted space.

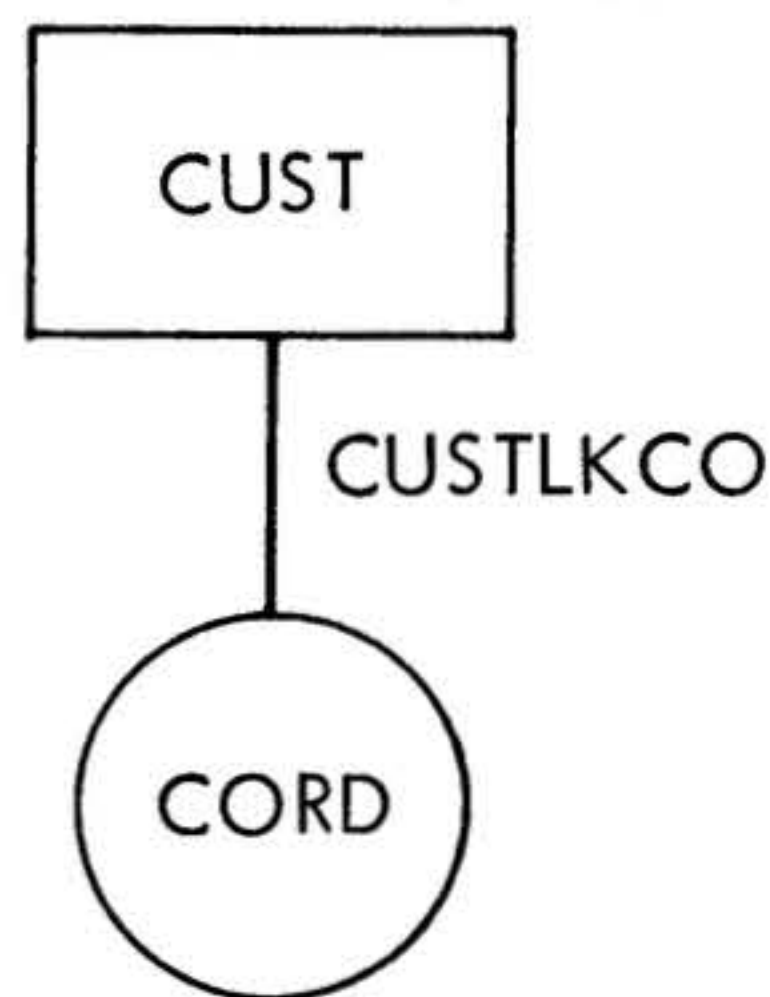


Another conventional technique is storing the customer order records in a serial file or perhaps an indexed-sequential file. In this technique the customer order file is usually volatile; therefore, file maintenance (additions and deletions) become a problem.

Another shortcoming of the conventional technique, explored later in this text, is that the customer orders may only relate to one control field. Furthermore, that relationship is influenced by the physical position of the record and by manipulation of a subfield of the control field (for example, order line number within order number within customer number).

The solution to the preceding problems is TOTAL's variable entry data set. The open order file has more than one record per control field and the open order file can have more than one control field. For example, the item number gives the ability to directly retrieve all open orders (for a given item number). The open order file can have different format records: the order header record, the order comment record, and the line item record. Figure 2-15 illustrates the schematic of the Customer-Customer Order data base.

The DBGEN program could now be written to include the variable entry data set CORD in the data base, and can be coded as follows:



(CIRCLE REPRESENTS VARIABLE ENTRY DATA SET)

VTI1-3448

Figure 2-15. Schematic of the Customer-Customer Order Data Base



The data base would be defined as:

BEGIN-DATA-BASE-GENERATION

.
. .

BEGIN-MASTER-DATA-SET

DATA-SET-NAME=CUST

MASTER-DATA

CUSTROOT=8

CUSTCTRL=6

CUSTLKCO=8

LINK TO CUSTOMER ORDERS

CUSTNAME=30

CUSTADDR=30

CUSTCTYS=20

CUSTDATA=100

END-DATA

.
. .

END-MASTER-DATA-SET

BEGIN-VARIABLE-ENTRY-DATA-SET

DATA-SET-NAME=CORD

BASE-DATA

CORDCUST=6=

CUSTOMER NUMBER

CUSTLKCO=8=CORDCUST

LINK FROM CUSTOMER MASTER

CORDORDN=2

ORDER NUMBER

CORDITEM=5

ITEM NUMBER

CORDQTYP=3

ORDER QUANTITY

END-DATA

.
. .

END-VARIABLE-ENTRY-DATA-SET

END-DATA-BASE-GENERATION

Assume now that the open order records are related to the item master (single entry) data set. This enables direct retrieval of all open order records for any particular item number.

To illustrate the use of linkage paths, assume that in addition to the customer order file being related to the customer file, the customer order file is also related to the item master file. To enable direct retrieval of all customer orders for any particular item, a linkage path is established between the customer order file and the item master file.

This is accomplished by including the master data set ITEM and the linkage path ITEMLKO in the DBGEN program coding as follows:



The data base would be defined as:

BEGIN-DATA-BASE-GENERATION

.
.
.

BEGIN-MASTER-DATA-SET

DATA-SET-NAME=CUST

IOAREA=MAS1

MASTER-DATA

CUSTROOT=8

CUSTCTRL=6

CUSTLKCO=8

CUSTNAME=30

CUSTCTYS=20

CUSTDATA=100

END-DATA

.
.
.

END-MASTER-DATA-SET

BEGIN-MASTER-DATA-SET

DATA-SET-NAME=ITEM

IOAREA=MAS2

MASTER-DATA

ITEMROOT=8

ITEMCTRL=5

ITEMLKCO=8

ITEMDESC=30

ITEMCOST=4

ITEMPRIC=4

ITEMONHD=4

ITEMONOR=4

END-DATA

.
.
.

END-MASTER-DATA-SET

BEGIN-VARIABLE-ENTRY-DATA-SET

DATA-SET-NAME=CORD

IOAREA=VAR1

BASE-DATA

CORDCUST=6

CUSTLKCO=8=CORDCUST

CORDORDN=2

CORDITEM=5

ITEMLKCO=8=CORDITEM

CORDQTYP=3

END-DATA

.
.
.

END-VARIABLE-ENTRY-DATA-SET

END-DATA-BASE-GENERATION

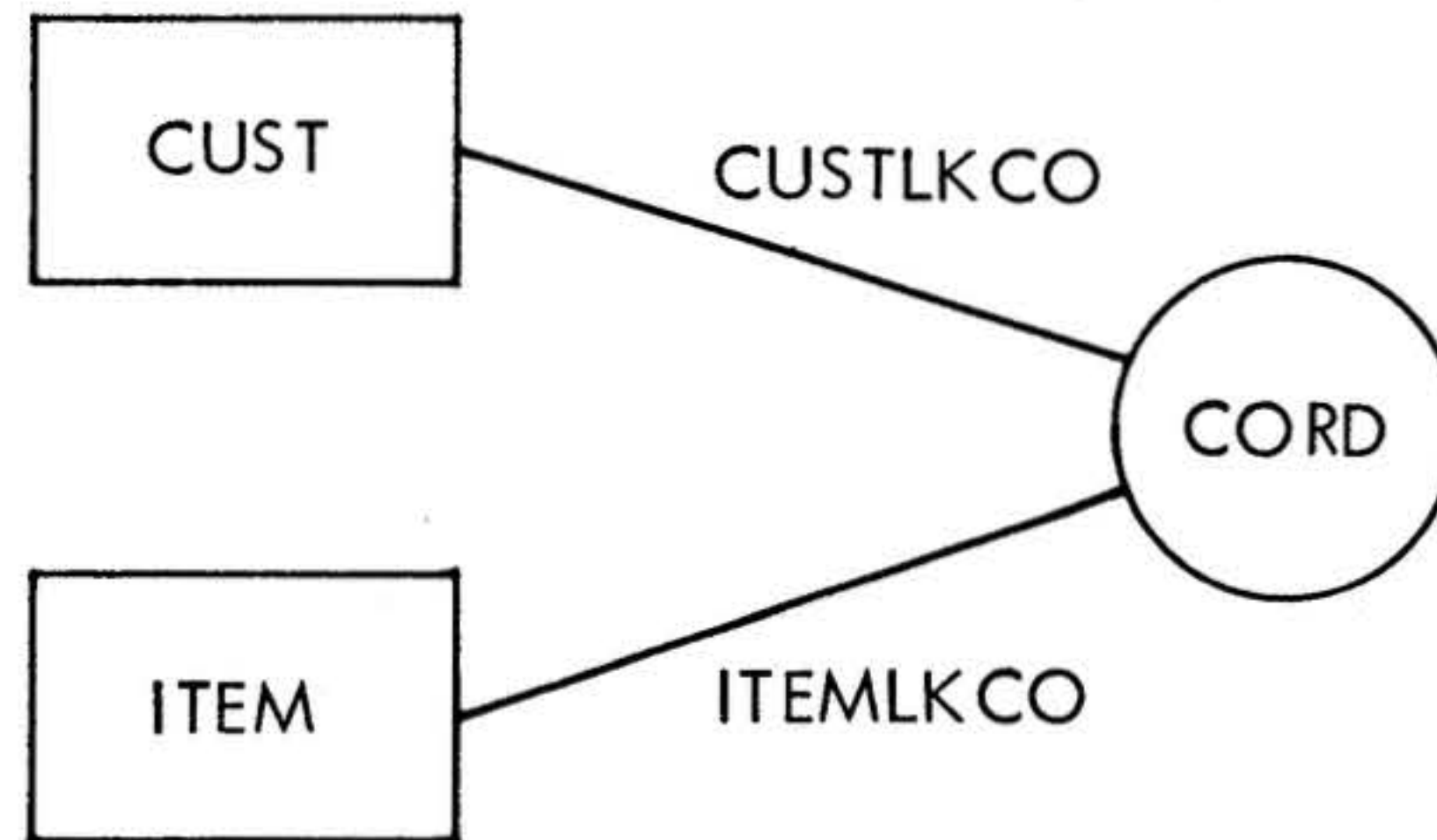
CUSTOMER NUMBER
LINK TO CUSTOMER ORDER

REQUIRED BY TOTAL
ITEM NUMBER
LINK TO CUSTOMER ORDERS
ITEM DESCRIPTION
ITEM COST
ITEM PRICE
QUANTITY ON HAND
QUANTITY ON ORDER

LINK FROM ITEM MASTER



The schematic of the customer-customer order and inventory data base is illustrated in figure 2-16.



VTI1-3449

Figure 2-16. Schematic of the Customer-Customer Order and Inventory Data Base

The user now has the ability to retrieve the same open order records as members of the two linkage paths, the customer linkage and the item linkage. Similarly, this technique can be applied to enable retrieval of the open order records by order number, date received, date shipped, or any other linkage path desired.

2.6.3 Coded Record Concepts (Reformatting)

The coded-record facility of the TOTAL system provides the variable entry data set with the capability to contain different data format records. The records are the same length, but are formatted differently. Of course, if all records are of the same format, the coded-record facility is not needed; therefore, coded records are optionally defined by the system analyst as the technique is needed.



Why would the coded-record technique be needed? Because the variable entry data set can be thought of in a new way, that of a "pool" of records. In the pool of records, it is quite conceivable that different records will have different functions. Some records might be for linkage only, that is, to link different single entry files together; while other records may be for data only - data that will support the linked single entry files as well as logical data. Since different records can have different functions, it is probable that these different records would also be linked to different single entry records according to their function.

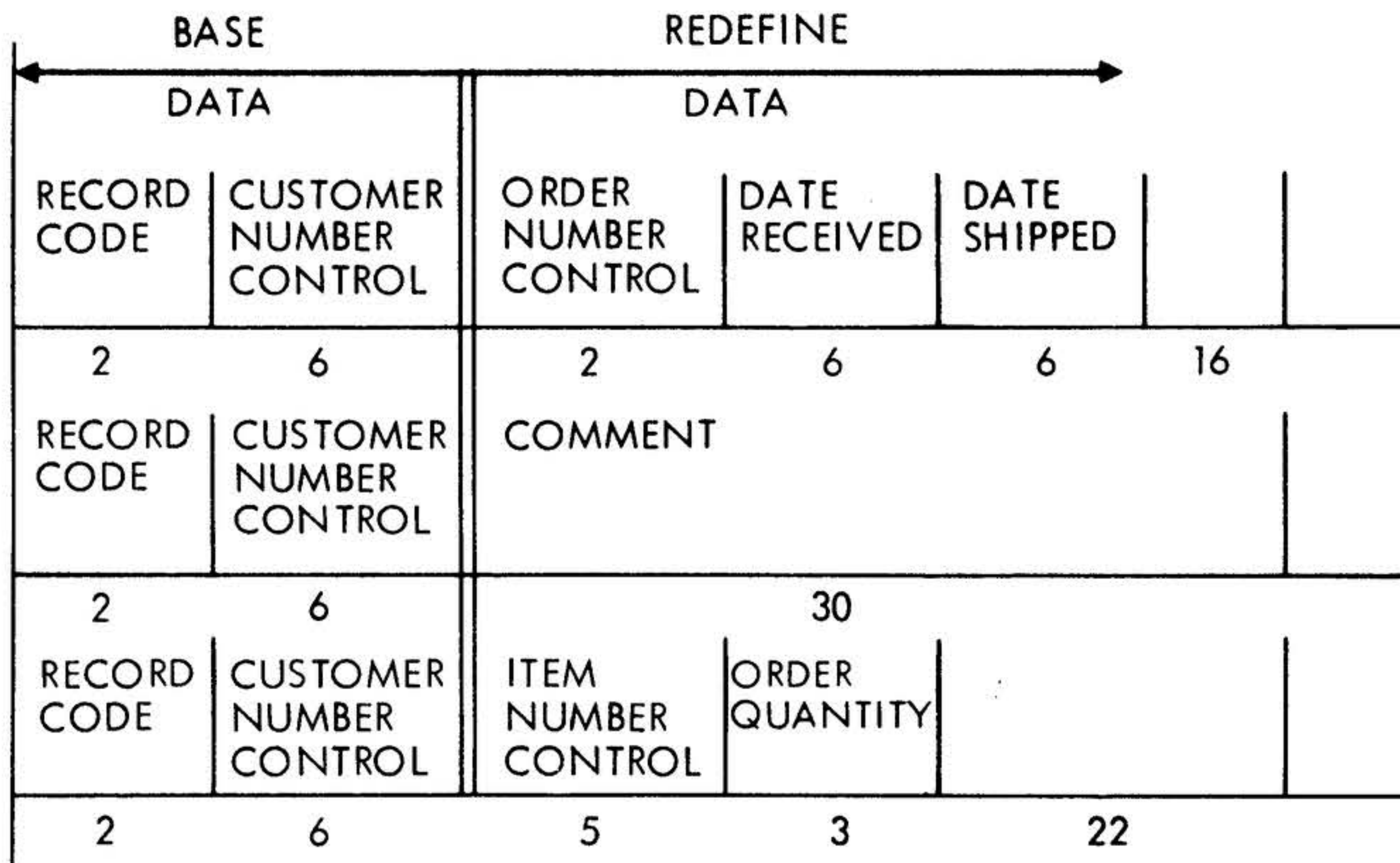
Re-examine the customer open order file. Three functional types of records have been identified: the order header record with data applying to entire order, the order comment record to appear anywhere on the order, and the line item record with data pertinent to the individual ordered item. In addition to the differing data format requirements, the records can be linked to different single entry files according to function.

Figure 2-17 illustrates the use of coded records to link information concerning a particular customer order to related master data sets. The first record is linked to both a customer master data set and an order number master data set and therefore can be accessed by either customer number or order number. Once accessed, data in both the base data portion and the redefined data portion (redefined in the Data Base Descriptor module) is available to the application program. Through further redefinition, a second record type is formatted. The second record contains a comment pertaining to the particular customer order. It is linked to only the customer master data set. The third record type has two links: one to the customer master data set and another to an item number master data set.

The logical record length would be determined by the longest of the different record definitions. It is likely that some records will be shorter than others, leaving zeroed, unused portions. If the data requirements of the different record types are widely divergent, the system analyst should then consider putting the different record types into different variable entry files.

Coded records enable the user to store data in variable entry records without linkage. Linkage at some later date can be achieved by changing the the record code into a code whose format identifies the stored fields as linkage control fields.

The redefine data record portion will be passed to the user program with data fields left-justified in the record. The linkage path fields are not passed but are defined in the user area, right-justified, for alignment purposes only.



VTI1-3450

Figure 2-17. Format of Record Types in a Variable Entry File

The variable entry data set would be defined as:

```

BEGIN-VARIABLE-ENTRY-DATA-SET
DATA-SET-NAME=CORD
BASE-DATA
CORDCODE=2                RECORD CODE
CUSTLKCO=8
CORDDATA=30              REDEFINE FORMAT ELEMENT
RECORD-CODE=HD          ORDER HEADER
ORDNLKCO=(8)
CORDDREC=(6)
CORDDSHP=(6)
RECORD-CODE=CM          ORDER COMMENT
CORDCMNT=(30)          ITEMS
RECORD-CODE=IT          ORDER ITEM
ITEMLKCO=(8)
CORDQTYP=(3)
END-DATA
.
.
.
END-VARIABLE-ENTRY-DATA-SET
    
```

NOTE: Coded records with linkage paths defined in the redefine data record portion give the ability to link the different records to different single entry files without the needless



overhead of redundant control fields carried in records for the purpose of linkage only. They also give the ability to add new record definitions to the variable entry file merely by defining the new format to the TOTAL system and inserting the records into their logical position.

2.7 DATA INDEPENDENCE

One of the key capabilities provided by the TOTAL system is data independence. This means that the application program is not affected by physical changes to the data base or any data set within the data base.

This capability is achieved by defining the data record as a collection of data elements. A data element can be defined as a single data item or the entire record. In other words, the data elements are defined by the system analyst as the logical or functional parts of the data record that are to be processed by the different application programs.

The application program requests the elements data record via the element list. This element list is defined as a literal constant in the application program. It specifies the 8-character names of all elements that are to be processed by the application program. Data independence is therefore achieved because each application program has the opportunity to specifically request only the data elements pertinent to that program, changes to other nonpertinent data elements, or addition of new data elements into the data record.

The application program does not have to read or write whole records as in non-data base applications; it receives and passes elements of the record to and from TOTAL. TOTAL reads and writes the data record; TOTAL selects the elements of the record according to the element list as defined by the application program, and passes the selected elements of data to the application program.

Since the application programs are independent of data, they do not have to be reprogrammed, recompiled, or relinked due to changes to nonpertinent data elements or additions of new data elements to data records. Since programs process only data elements required, core requirements for record areas are reduced. With data independence, programs can request data elements in any sequence without regard to actual record format. Data integrity is introduced, since programs do not have the opportunity to inadvertently destroy associated data files.



2.8 TOTAL's PROGRAMMING LANGUAGES

Conventional programming languages support various techniques for file and record definition and various input and output commands for accessing defined records and files. TOTAL's two languages, Data Base Definition and Data Base Management, do the same.

- o Data Base Definition Language (DBDL) is an English-like language that provides for initial generation of a data base module and all subsequent modifications and expansions to this data base.
- o Data Base Management Language (DML) interacts with the data base, the operating system, or supervisor and the application programmer. It allows TOTAL to function with the host language (DASMR, COBOL, RPG II, FORTRAN) for all communication with the data base.

Utilizing the facilities of the DBDL and the DML, TOTAL provides a completely integrated data base available to any application programmer using supported most language processors. TOTAL provides data elements to application programs so that new elements, new data sets, and new data relationships can be added to the data base without adversely affecting the current operational programs.

2.8.1 Data Base Definition Language

TOTAL DBDL is a high-level, independent, key-phrase language that defines the data base, i.e., data sets, data records, data elements, data set relationships, and data items within the data base.

The data base need be defined only once for all programs to have access to it. The data base does not have to be defined for and within each program that will use it as with conventional file definition techniques.

After the DBDL statements describing a data base are prepared, they are converted into Assembler Language source statements by the DBGEN program (a part of the TOTAL system).

The resulting Assembler source statements are assembled to produce an object Data Base Descriptor Module (DBMOD). When the application program requests access to the data base, the corresponding Data Base Descriptor Module is loaded and referenced by TOTAL to locate data sets and their components by name (section 3).



2.8.2 Data Management Language

After the data base is compiled and cataloged, application programming can be started using host-language processors and the TOTAL Data Management Language. DML communicates between the program and the data base. DML is not a complete language by itself; it relies on a host language at the CALL level to furnish a framework and to provide the procedural capabilities required to manipulate data in primary storage.

All calls to and from the data base to retrieve data, to add new data or data relationships, to delete data or data relationships, or to modify data or data relationships, reside in DML. Diagnostics and messages indicate the successful operation of a function, or the status in case of an unsuccessful execution. For example, DML indicates that a duplicate record already exists if you attempt to add such a duplicate record to the data base.

TOTAL functions at the element level, an element being one or more of the items that comprise a logical record. Upon the execution of a TOTAL DML command, one or more elements, as specified by an element list, are passed to or from the host program in the stated sequence of that element list. It is not required to do any further manipulation such as sequencing, positioning, including, or omitting of elements. Subsequent expansions of the record for additional elements or relationships have no adverse effect on programs that use the originally defined record. Old programs do not require recompilation when new elements are added to records. After the host-language program has received the data, the host language is used for whatever logical, arithmetic, or manipulative processing desired. The host language, then, is the language of specific data "policy" manipulation. This manipulation is application-oriented and very specific.

2.9 CREATING, FORMATTING, AND OPERATING THE DATA BASE

The TOTAL Data Base Management System consists of the programs:

- o DBGEN - the data base generator program. DBGEN accepts the Data Base Definition Language (DBDL) statements which define the data base and generates the assembly language Data Base Descriptor Module DBMOD. For full details, refer to section 3.
- o DBFMT - the data base format program. DBFMT accepts the data set format control card(s), creates and preformats the specified RMD data set areas. For full details, refer to section 4.



varian data machines

- o TOTAL - the Data Base Management program. It is activated when initiated by the CALL DATBAS statements given in the application program. These CALL instructions are written in Data Management Language (section 5). TOTAL accesses the data base, performs the necessary retrieval, add, or change operations, and issues read/write instructions as required.

For full details on operating the data base, refer to section 7.

Both DBGEN and DBFMT are non re-entrant background tasks (DBFMT can be run as a foreground task) which are run rarely. The sequence of events for TOTAL Data Base generation is shown in figure 2-18.

The TOTAL program is a general access method of getting at data files easily, and is bound with the application program. It acts upon the data sets defined in the data base object module according to the parameter lists given by the application program CALL statements.

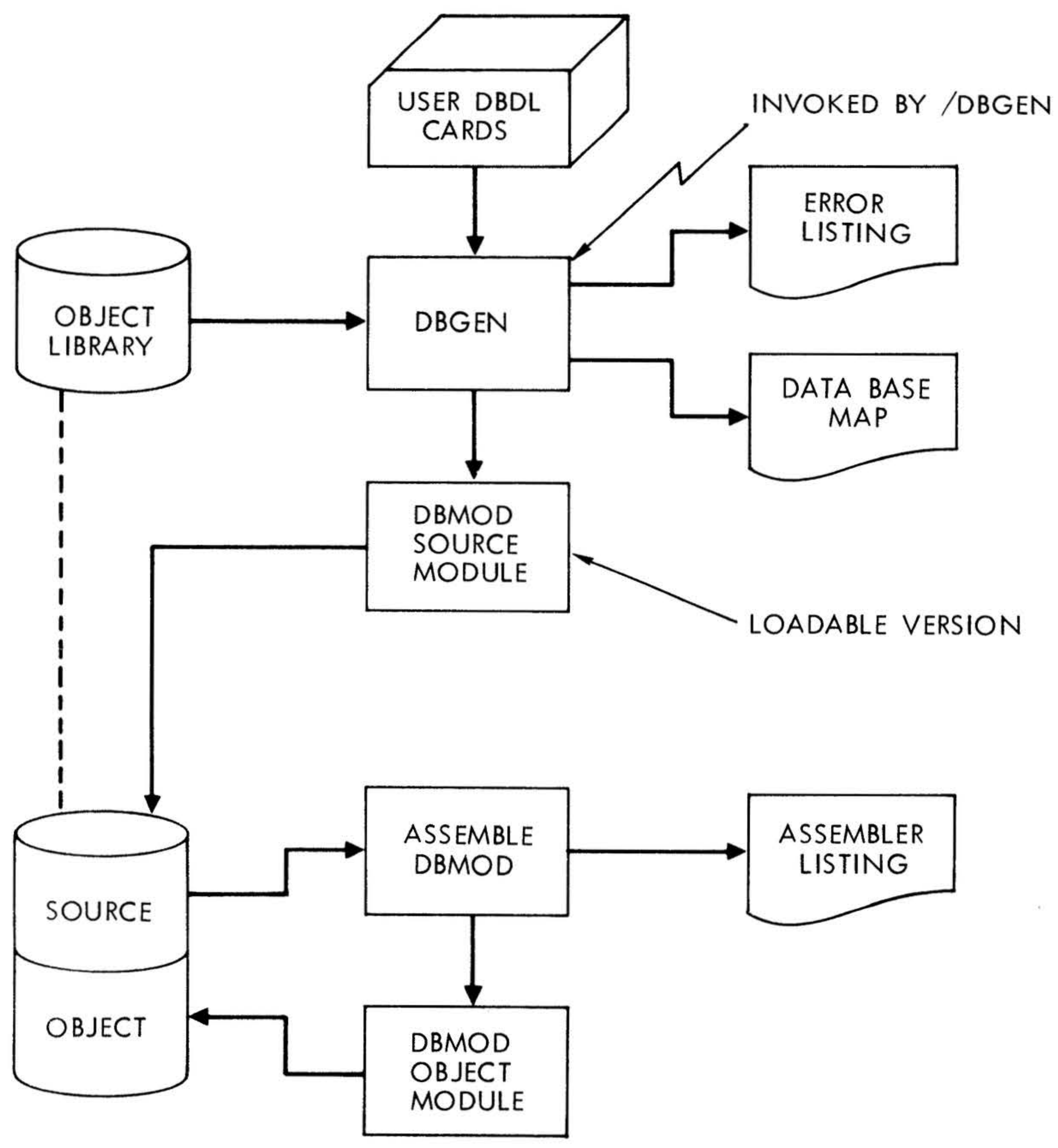
2.10 MEMORY REQUIREMENTS

TOTAL enables a considerable savings in main memory because:

- a. TOTAL extracts and passes to the application program only the data elements required, using direct access via control keys and linkage paths, resulting in a network type data base. An example of a TOTAL network structure is shown in figure 2-19.
- b. TOTAL allows many application programs to access the same data base, resulting in memory savings as each application program does not have to store data. This concept of a data base management system is shown in figure 2-20, in which four application programs are shown accessing a data base comprised of two master data sets and two variable data sets, linked together.

2.10.1 Economic Computer Resource Utilization

The computer resources under consideration are computing time, main and secondary memory, and I/O utilization. Considerable effort is taken in TOTAL to optimize their use. The structure of a TOTAL data base inherently allows the elimination of redundant data, index areas and overflow areas. Deleted records space is immediately reusable and hence there is no need to



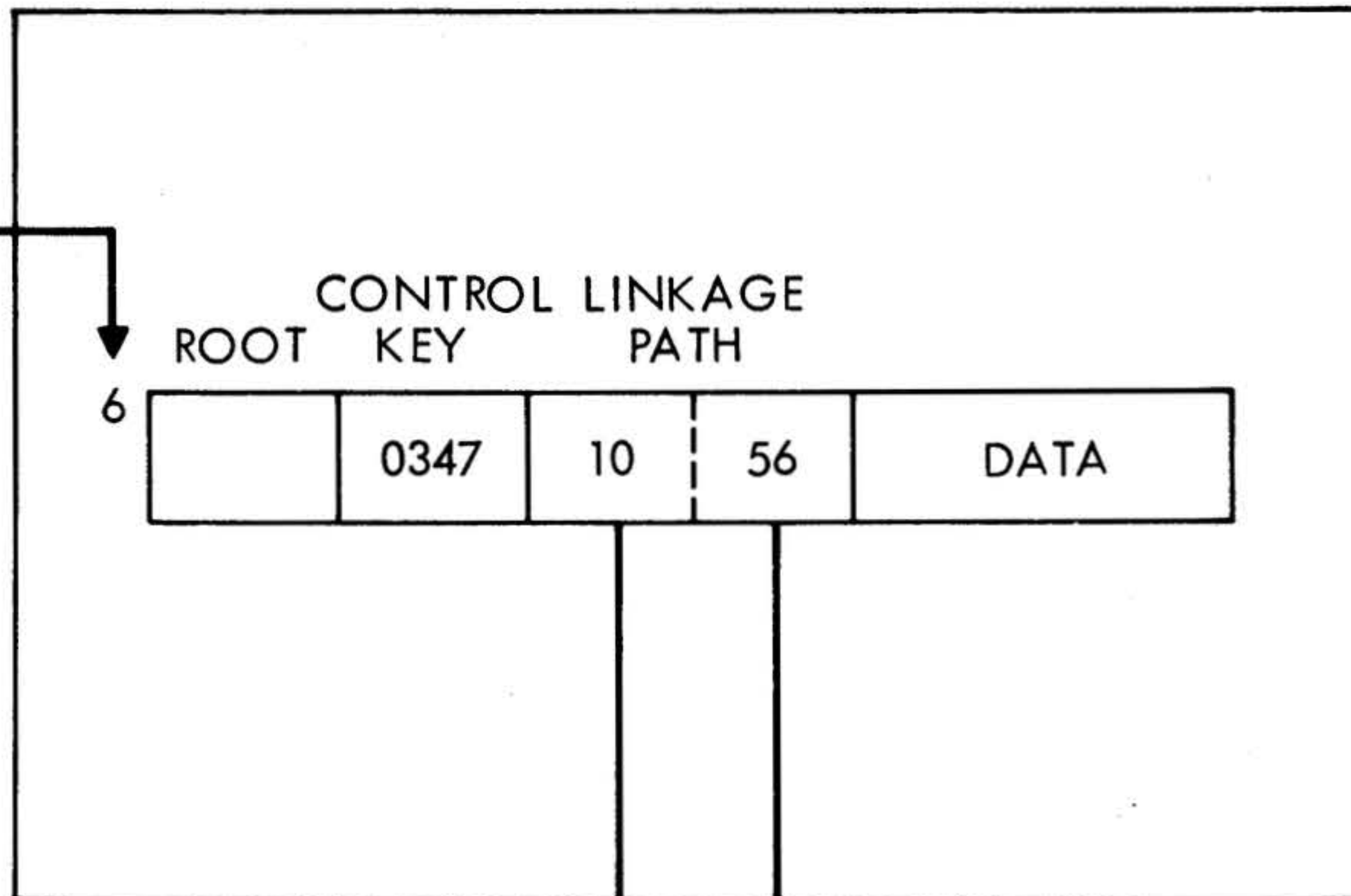
VTI1-3451

Figure 2-18. TOTAL Data Base Generation

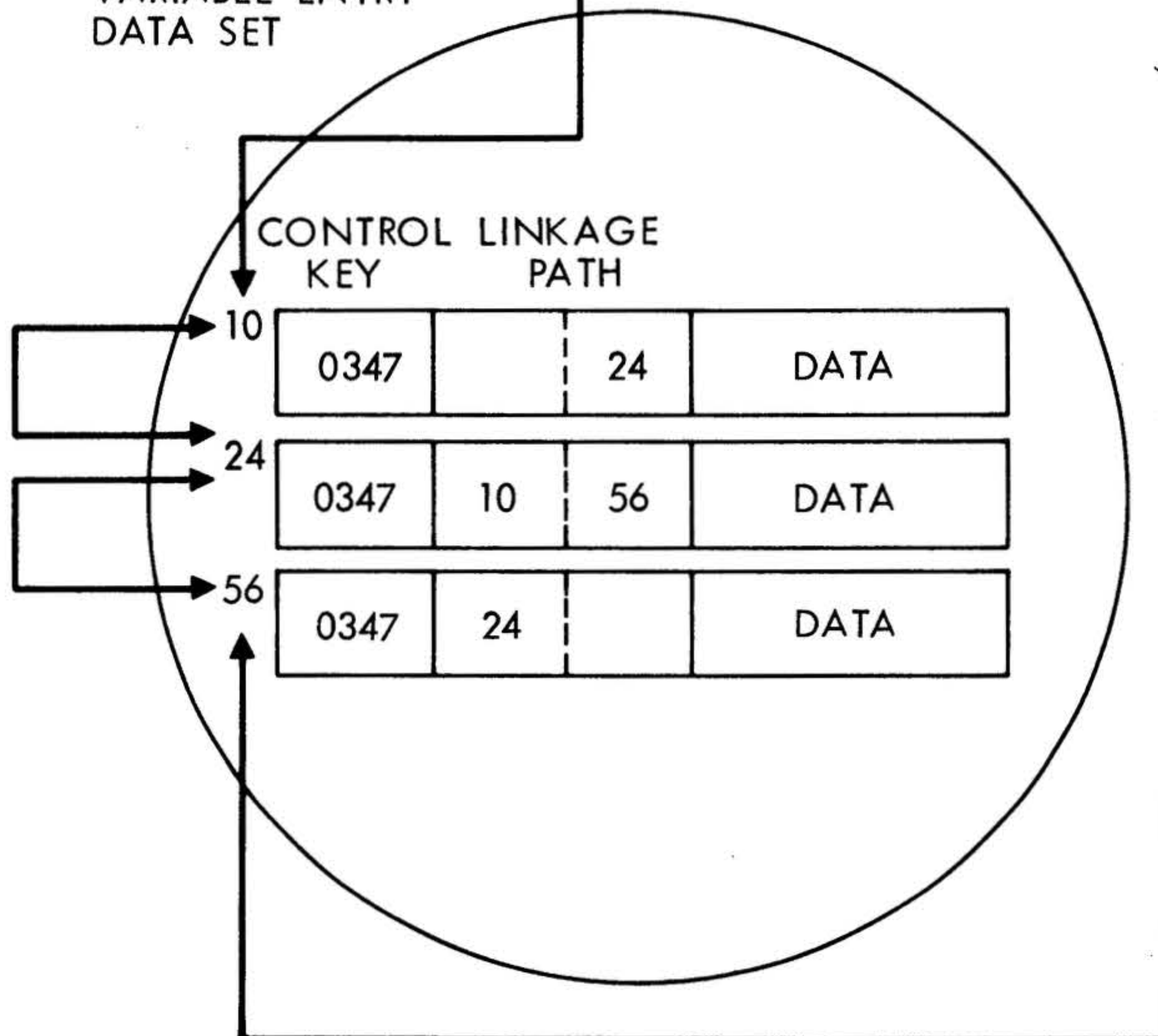


0347

SINGLE ENTRY DATA SET

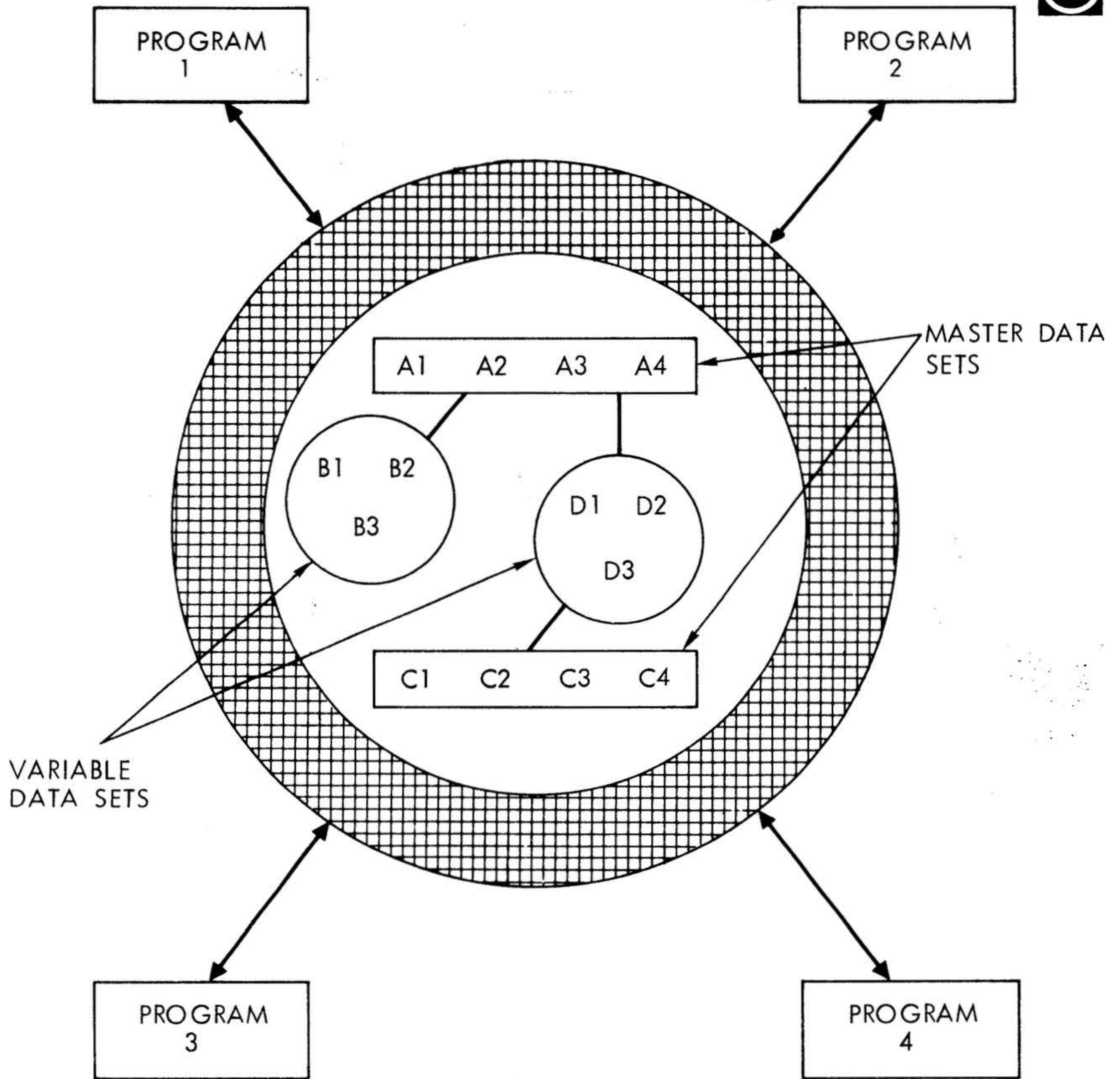


VARIABLE ENTRY DATA SET



VTI1-3452

Figure 2-19. TOTAL Network Structure Example



DBMS

- CAPABLE OF ESTABLISHING AND PROCESSING USER-DEFINED DATA RELATIONSHIPS
- CAPABLE OF EVOLVING EASILY AS THE STYLE OF MANAGEMENT CHANGES
- TRUE INDEPENDENCE
- PERFORMANCE AND EFFICIENCY

VTI1-3453

Figure 2-20. Data Base Management System

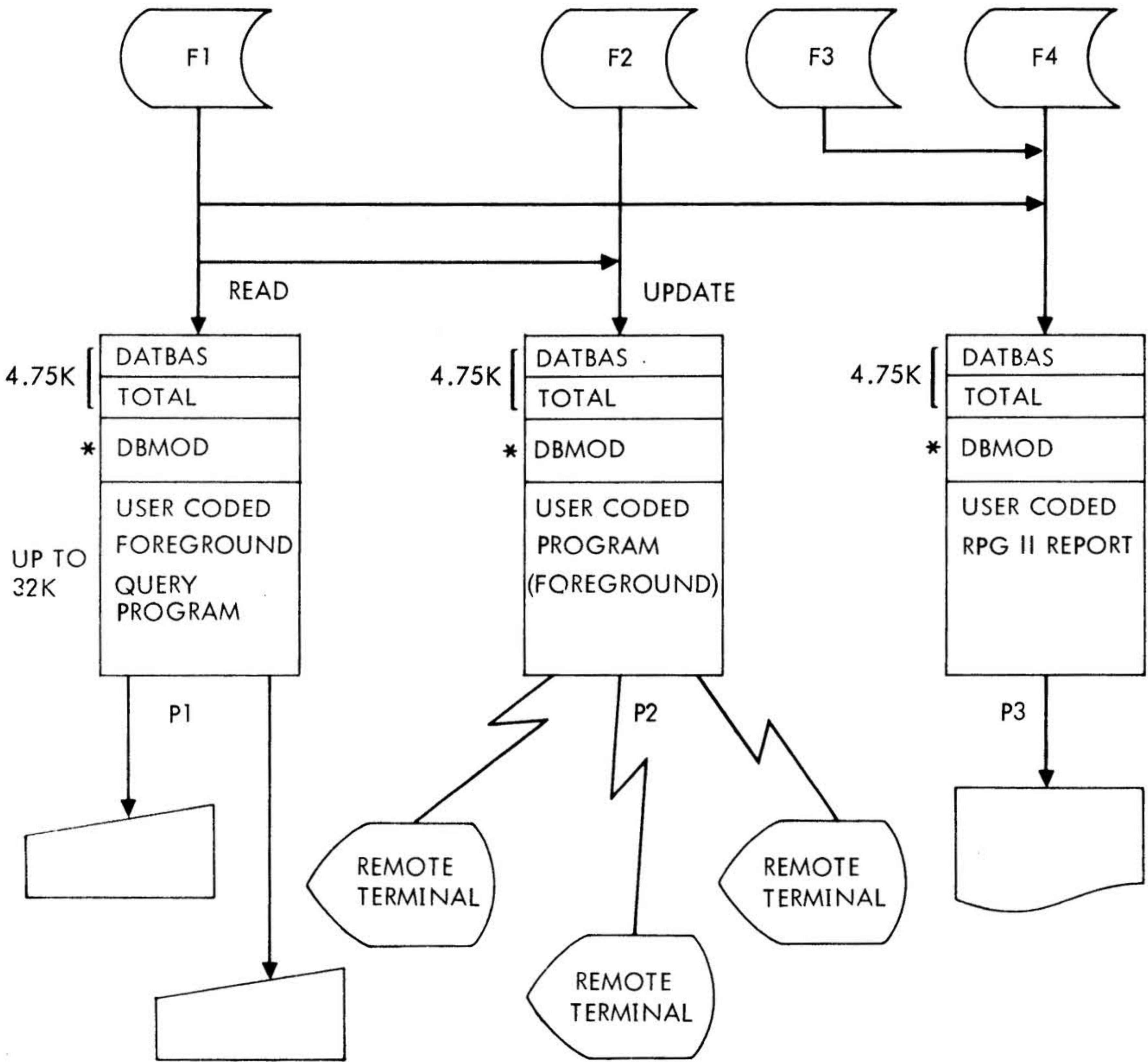


reorganize records in a data base in order to eliminate dead records. Associated records tend to be physically placed together, resulting in the reduction of data access time. An I/O buffer can be unique to a data set or shared by multiple data sets, thereby conserving main memory buffer requirements. During information retrieval, a block which may contain a multiple number of records will be brought into main memory. The block will be written back to the data base only when it has been modified and the next request record is not in it. Since the probability of processing a record next to the one just processed is very high, this scheme effectively reduces the number of I/O operations.

2.10.2 Run-Time Memory Utilization

The run-time portion of TOTAL (about 4.7K), including the Data Base interface module DATBAS (0.5K) and the Data Base definition module DBMOD, are bound together to the user program and run in the same user logical memory space.

The size of the DBMOD depends on the user specifications (see table 3-1, section 3.5 as a guide to determine the size of the DBMOD). As an example, an average program utilizing a data base composed of five master files and three variable files requires about 2600 words for the DBMOD. Figure 2-21 illustrates a typical TOTAL memory layout and the use of TOTAL data base by three user programs. P1 a user coded foreground query program which reads file F1 and F4. P2 is an on-line program that serves three remote terminals. P2 updates F2 and hence this file is locked and may be used by P2 only. P2 also reads F1 and F3. P3 is an RPG background program which reads files F1, F3, and F4. TOTAL allows multiple tasks to concurrently read from the same file. However, a file that is being updated may be accessed only by the updating task.



*DEPENDENT ON USER DATA BASE DEFINITION. A "TYPICAL" DATA BASE DEFINING FIVE MASTER FILES AND THREE VARIABLE FILES IS LESS THAN 3K WORDS.

VTI1-3454

Figure 2-21. Typical TOTAL Memory Layout



2.11 PRIVACY AND SECURITY

2.11.1 Internal Privacy and Security

A file locking facility to prevent concurrent file updating is provided by the SCHEMA parameter of the Data Management Language Sign-On function.

The SCHEMA parameter contains a SHRE or PRIV option. If SHRE is coded, the file called for by the application program may be shared among concurrent programs in the Read Only mode. If PRIV is coded, the file called for is exclusively assigned to the application program requesting it, and no other program may have access to it during any program run. The file is unlocked by the Sign-Off function.

For details of the SCHEMA parameter, refer to section 5.2.13.

In addition, when an attempt to execute a TOTAL command fails to complete, the original condition of the data base prior to the request is restored and a diagnostic message is returned to the user indicating the possible cause of failure.

2.11.2 External Privacy and Security

External privacy and security should be maintained by a Data Base Administrator, who should have the responsibility of controlling access to the data base files and logical unit codes.



```

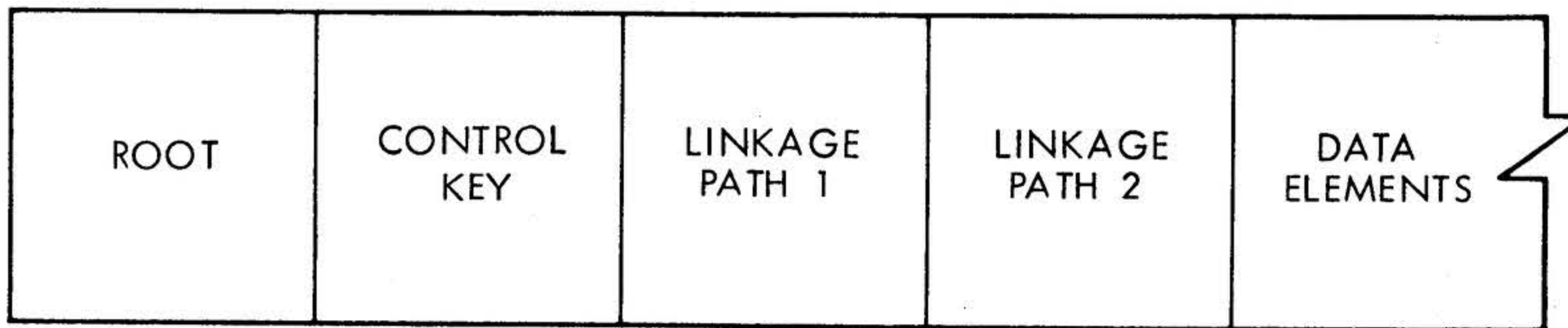
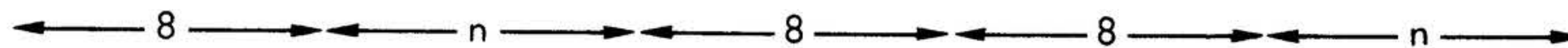
END- DATA-BASE-GENERATION
/ASSIGN,PI,,,SS
/MEM,10
/DASMR
/ENDJOB

```

3.1.3 TOTAL Record Formats

When designing the data base and writing the DBDL statements, one should be aware of some basic facts about the data records used.

- a. Single entry (master) file records. Each record must have the following (figure 3-1):
 1. A ROOT. 8-byte field used by TOTAL to link synonyms. 4 bytes - backward pointer; 4 bytes - forward pointer.
 2. A Control Key. n-byte field to contain the key by which linking to a variable file is accomplished



n=USER DEFINED

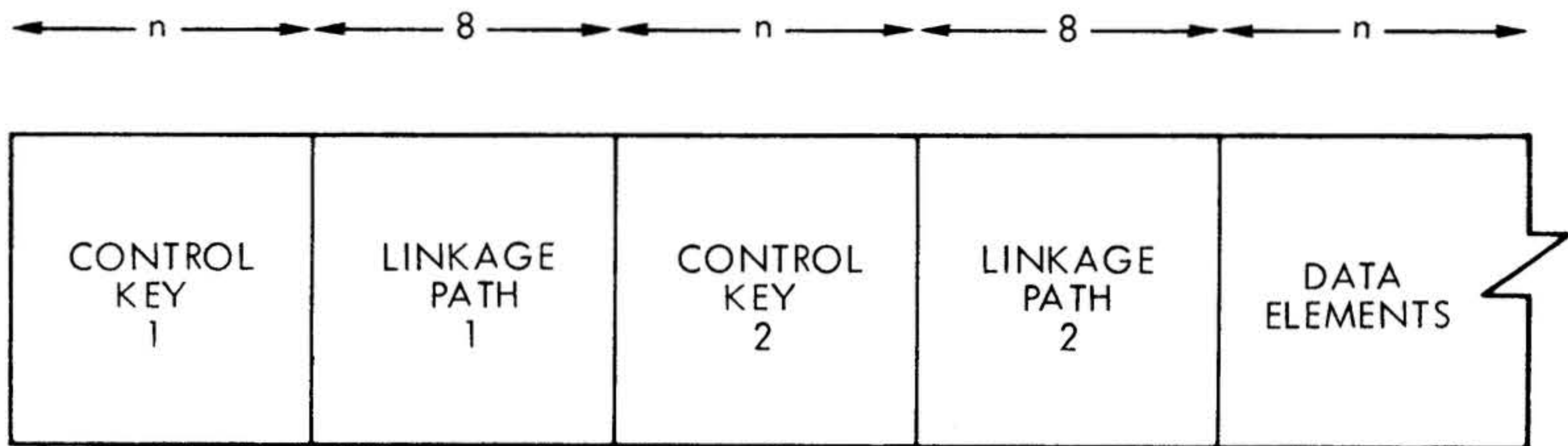
VTI1-3355

Figure 3-1. TOTAL Single-Entry Data-Set Record Format



3. A linkage path. At least one. Eight-byte field(s) maintained by TOTAL to link to a variable file record containing the control key. There can be as many of these 8-byte fields as required.
- b. Variable Entry file records. This record must have the following (figure 3-2):
1. Control key.
 2. Linkage path. As many as there are links to master records.

When coded records are used, the record is divided into two parts: the base data area and the redefined data area. The base data area must have at least a record code which is a 2-byte code containing the record code type. The rest of the record format is shown in figure 3-3.



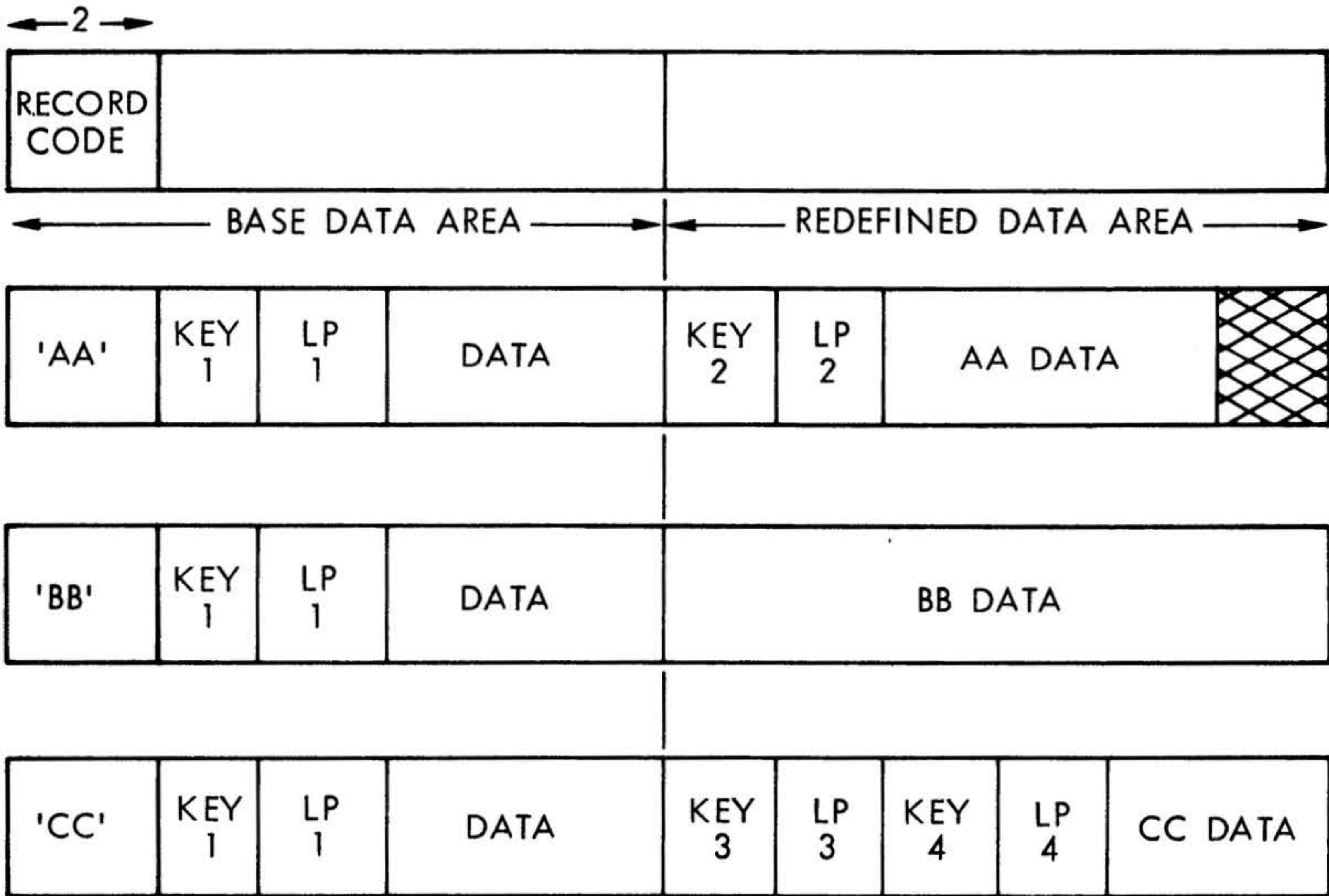
n = USER DEFINED

VTI1-3456

Figure 3-2. TOTAL Variable Entry Data Set Record Format



CODED RECORDS



- FIXED LENGTH RECORDS
- VARIABLE FORMATS IDENTIFIED BY RECORD CODE
- SELECTIVE LINKAGE

NOTE LP = LINKAGE PATH

VTI1-3457

Figure 3-3. TOTAL Variable Entry Data Set Record Format



3.1.4 Data Set (File) Organization

A TOTAL file, is considered to be a logical file and as such it may consist of one or more VORTEX (physical) files. The number of VORTEX files in a TOTAL file is determined by the number of DRIVE statements for the TOTAL file. Each VORTEX file is located on the logical unit number (LUN) specified in the DRIVE statement. This means that a TOTAL file can be stored on many different RMD partitions, some of which may be located on a different RMD device.

TOTAL names the TOTAL file as the name given in the DATA-SET-NAME directive e.g., a 4-letter word, mmmm for master or vvvv for variable. Each VORTEX file name (six characters) is derived from the TOTAL file name by (TOTAL) adding a 2-digit ordinal number e.g., mmmm01, mmmm02,... etc. (for master).

3.1.4.1 Master File

All RMD addresses are computed and maintained as relative record numbers (rrn). Each TOTAL master file has two control records one at the front and one at the rear. To allocate a master record, its key is randomized (a hashing procedure involving 32-bit arithmetic computed mostly by firmware) and an rrn is obtained. This number is called the "home address." If the home address is empty, the master record is stored there. If the home address is already occupied, the record occupying it can be either a synonym (e.g., a different control key randomized to the same place) or not a synonym. If it is a synonym, then the new record is stored closest (physically) to its synonym and linked to it via the ROOT field. If the occupying record was not a synonym, the new record is replacing the old one, and the old record is allocated elsewhere (closest to its home address).

3.1.4.2 Variable Entry Files

Variable file management requires a series of control records inserted at regular intervals throughout the TOTAL logical file. The control interval is 480 sectors long (approximately 1 cylinder, e.g., 24 sectors per track, 20 tracks per cylinder).

Each TOTAL file starting point coincides with the control interval starting point.



Example:

TOTAL file name (variable entry) is:

DATA-SET-NAME=FILE

VORTEX files are defined by:

DRIVE=031,200

DRIVE=032,100

DRIVE=033,400

There are three VORTEX files:

FILE01	on LUN 31	200 sectors
FILE02	on LUN 32	100 sectors
FILE03	on LUN 33	400 sectors

The control records are allocated at the beginning of the logical file and at the beginning of each logical interval of 480 sectors. For the above example, control record at FILE01 is record one; control record at FILE03 is record 181 (481st).

The LOAD-LIMIT (in percent) is computed and maintained for each control interval.

3.2 SYNTAX RULES

DBDL statement entries are made up of 80-character records, using the following rules:

- a. All entries must begin at character position 1.
- b. A blank terminates an entry.
- c. Capitalized entries and punctuation must be coded as shown.
- d. Square brackets ([]) enclose an optional parameter.
- e. Braces ({ }) mean a choice between parameters.
- f. "mmmm" is to be replaced by a single entry data set name.



- f. "vvvv" is to be replaced by a variable entry data set name.
- g. "xxxx" is to be replaced by any valid characters (A thru Z, 0 thru 9).
- h. "n" is to be replaced by any valid natural number, or zero.
- i. Comments are recommended as additional documentation throughout the definition language. They may begin after termination of any entry, or may be entered on separate statements simply by leaving position 1 blank.

3.3 SUMMARY OF DATA BASE DEFINITION STATEMENTS

Unless otherwise specified, all numbers are in decimal and bytes (not words).

3.3.1 Prologue Statements

```
BEGIN-DATA-BASE-GENERATION
DATA-BASE-NAME=xxxxxx
OPTIONS=OUTPUT={ Y }
                { N }
```

```
SHARE-IO
IOAREA=xxxx[=n] (n = number of buffers, default n=1)
END-IO
```

3.3.2 Master Data Set Statements

```
BEGIN-MASTER-DATA-SET
DATA-SET-NAME=mmmm
IOAREA=xxxx
MASTER-DATA
mmmm ROOT=8
mmmm CTRL=n
mmmm LKxx=8
mmmm xxxx=n
[.p.] mmmm xxxx=n
END-DATA
[ TOTAL-LOGICAL-RECORDS=n ]
[ LOGICAL-RECORD-LENGTH=n ]
[ LOGICAL-RECORDS-PER-BLOCK=n ]
DRIVE=nnn,nnnn
END-MASTER-DATA-SET
```



3.3.3 Variable Entry Data Set Statements

```
BEGIN-VARIABLE-ENTRY-DATA-SET
DATA-SET-NAME=vvvv
IOAREA=xxxx
BASE-DATA
[vvvvCODE=2]
vvvvxxxx=n
[.p.] vvvvxxxx=n
mmmmLKxx=8=vvvvxxxx
[RECORD-CODE=xx]
[.p.vvvvxxxx=n]
mmmmLKxx=8=vvvvxxxx
END-DATA
[TOTAL-LOGICAL-RECORDS=n]
[LOGICAL-RECORD-LENGTH=n]
[LOGICAL-RECORDS-PER-BLOCK=n]
DRIVE=nnn,nnnn
[LOAD-LIMIT=n] n is in percent
END-VARIABLE-ENTRY-DATA-SET
```

3.3.4 Epilogue Statement

```
END-DATA-BASE-GENERATION
```

3.4 DATA BASE DEFINITION STATEMENTS

In the following subsections, each statement is followed by a description of the statement.

3.4.1 Prologue Statements

```
BEGIN-DATA-BASE-GENERATION
```

Description: This statement must be the first statement of the data base definition, and the first statement in the deck. No comment statements are allowed before this statement.

```
DATA-BASE-NAME=xxxxxx
```



Entry: xxxxxx is a 6-character alphanumeric name.

Description: This name will be used as the data base identifier throughout the system. The use of meaningful names is recommended throughout the definition process.

OPTIONS-OUTPUT={ Y }
 { N }

Description: Y means yes, generate DBMOD and print. N means print only and suppress generation of DBMOD.

SHARE-IO

Description: This statement and subsequent "IOAREA=" entries indicate the names of I/O areas that will be specifically used in data-set definitions within this Data Base Descriptor.

IOAREA=xxxx [= n]

Entries: xxxx is a 4-character alphanumeric name
n is the number of buffers contained in the pool

Description: Each occurrence of this statement enters a name into a list of named I/O areas.

- a. No duplicate names are allowed.
- b. Each data set may have one and only one I/O assigned to it when the data set is defined.
- c. The same area can be used by several data sets.
- d. An I/O area assigned to only one data set must be listed and is regarded as a private I/O area.
- e. Each occurrence of this statement will reserve an area whether or not it is ever referenced in a data set definition.



- f. There is no limit to the number of I/O areas which can be defined.
- g. If "=n" is omitted, the default value for n is 1.

END-IO

Description: This required statement terminates the definition of "IOAREA=" entries.

3.4.2 Master Data Set Statements

BEGIN-MASTER-DATA-SET

Description: This must be the first statement to begin definition of the master data set.

DATA-SET-NAME=mmmm

Entry: mmmm is a 4-character alphanumeric name.

Description: This name will be used as the data set identifier throughout the system.

IOAREA=xxxx

Entry: xxxx is a 4-character alphanumeric name.

Description: This statement designates the I/O area to be used by the master data set. The I/O area named in this statement must have been defined in the prologue.

- a. Only one I/O area may be assigned to a data set.
- b. This I/O area may be shared with other data sets.



MASTER-DATA

Description: This statement precedes the definition of logical data elements for a master data set.

```
mmmmROOT=8
```

Entries: mmmm is the 4-character master data set name.
8 is the required length. It states the length of the ROOT field. This length is included in the length of the record.

Description: This field is for internal use to manage synonyms. This statement must be the first data element definition of a master data set.

```
mmmmCTRL=n
```

Entries: mmmm is the master data set name.
n is the control key length

Description: This statement defines the record control key. The record control key must fall immediately after the ROOT statement.

```
mmmmLKxx=8
```

Entries: mmmm is the master data set name.
xx is a 2-character linkage code.
8 is the required length of eight bytes.

Description: This statement defines a linkage path from a master data set based on the record control key. It is called the LINKAGE-PATH statement (LP) or (LK). There must be as many LK statements as links used.

- a. A master data set may have any number of linkages.
- b. A variable entry data set may be linked from multiple master data sets and may have multiple linkages from the same master data set.



- c. Linkage paths are never given a level number.
- d. Statement must be given for each link.

[.p.] mmmmxxxx=n

- Entries:
- mmmm is the master data set name.
 - xxxx is any four valid characters which with the preceding four character entry comprises a unique element identification.
 - p is a level number (1, 2, or 3).
 - n is the length of the element.

- Description:
- This statement defines a data element. Any data element may be subdefined:
- a. The values which may be used to specify level range from 1 through 3. The level number zero (0) is reserved. Any element with no prefix to specify its level number will be assigned level number zero (0).
 - b. The level number is specified as an integer preceded and followed by a single dot (e.g., ".3.").
 - c. The length entry for a "parent" element will be the sum of the lengths of the "child" elements.
 - d. A unique name must be used for every element of the data set.

END- DATA

- Description:
- This statement ends the definition of logical data elements for the data set.

[TOTAL-LOGICAL-RECORDS=n]

- Entry
- n is a numerical value.



Description: This optional statement specifies the total logical record capacity of the data set.

This entry is optional. When this statement is omitted, the TOTAL logical record capacity for the data set will be calculated. DBGEN uses TOTAL sectors available (Drive Statement) and the logical record length. The DBGEN program provides an informative message if this value is calculated.

[LOGICAL-RECORD-LENGTH=n]

Entry: n is a numerical value.

Description: This optional statement specifies the length of logical records in a data set. This includes all data fields, linkages, and the ROOT field of a master data set.

NOTE: Sector length (S) is 240 bytes. Record length (n) must be chosen in such a way that either $n = 0 \text{ mod } S$, or $S = 0 \text{ mod } n$.

There are two ways to accomplish this:

- a. If the actual record size is 110, add 10 blank bytes at the end and let TOTAL compute the length, or
- b. Force the record length to 120 bytes by writing:

LOGICAL-RECORD-LENGTH = 120

Length of 28 becomes 30
Length of 80 is okay
Length of 110 becomes 120
Length of 400 becomes 480

It is advisable to let TOTAL compute the record length.

This entry is optional. When this statement is omitted, the logical record length will be calculated as the summation of all element length entries. The DBGEN provides an informative message if this value is calculated.



[LOGICAL-RECORDS-PER-BLOCK=n]

Entry: n is a numerical value.

Description: This optional statement specifies the blocking factor or number of records within each block in this data set.

DRIVE=nnn,nnnn

Entries: nnn is a logical unit number (partition).
This is a number, not a name.

nnnn is the number of physical sectors available within the logical unit (partition).

Description: This statement specifies the RMD area required for the data set. As many drive statements as needed should be included.

- a. Multiple drive statements must be specified if the data set requires more than one partition.
- b. Each DRIVE statement may require a different logical unit number as well as a different RMD unit.

The drive statement is the only required physical specification entry. This entry to the DBGEN program will compute the TOTAL record capacity. In addition, an unused sector count will be provided to indicate the logical unit block and number of sectors not used in that block.

These calculations will be notated on the DBDL listing.

END-MASTER-DATA-SET

Description: This must be the last statement to end definition of a master data set.



3.4.3 Variable Data Set Statements

BEGIN-VARIABLE-ENTRY-DATA-SET

Description: This must be the first statement to begin definition of the variable data set.

```
DATA-SET-NAME=vvvv
```

Entry: vvvv is a 4-character alphanumeric name.

Description: This name will be used as the data set identifier throughout the system.

```
IOAREA=xxxx
```

Entry: xxxx is a 4-character alphanumeric name.

Description: This statement designates the I/O area to be used by the variable data set. This I/O area must have been defined in the prologue.

- a. Only one I/O area may be assigned to a data set.
- b. This I/O area may be shared with other data sets.

```
BASE-DATA
```

Description: This required statement precedes the definition of logical data elements for the variable data set.

```
[ vvvvCODE=2 ]
```

Entry: vvvv is the 4-character variable entry data set name.



Description: This statement is required to reserve space in a record for a record code if coded records are used. If this variable entry data set has multiple record codes, then this is a required entry. This entry indicates that two characters are to be reserved for the record code.

[.p.] vvvvxxxx=n

Entries:

- vvvv is the 4-character variable data set name.
- xxxx is any four valid characters which, with the preceding four character entry comprises a unique element identification.
- p is the level number (p can equal 1, 2, or 3)
- n is the length of the element.

Description: This statement defines a data element which may be a data item or a control field. This statement may occur anywhere within the record definition. If this is the last element in the Base-Data portion of the variable record, it is the element redefined by "RECORD-CODE=" groups.

Any data element may be subdefined:

- a. The values which may be used to specify level range from 1 through 3. The level number zero (0) is reserved. An element with no prefix to specify its level number will be assigned level number zero (0).
- b. The level number is specified:
As an integer preceded and followed by a single dot (e.g., ".3").
- c. The length entry for a "parent" element will be the sum of the lengths of the "child" elements.
- d. A unique name must be used for every element of the data set.



mmmmLKxx=8=vvvvxxxx

Entries: mmmm is the master data set name.
 xx is a 2-character linkage code.
 8 is the required length of eight.
 vvvvxxxx is the element name containing the key.

Description: This statement defines a linkage path from a master data set based on the record control key. The definition of the element containing this key (vvvvxxxx) must precede this statement.

- a. "mmmmLKxx" is the linkage path as defined in the master data set which links to this variable data set. This entry must be specified exactly as in the master data set in order to establish the required linkage.
- b. A variable entry data set may be linked from multiple master data sets and may have multiple linkages from the same master data set.
- c. Linkage paths are never given a level number.

RECORD-CODE=xx

Entry: xx is a 2-character alphanumeric name.

Description: This optional statement identifies the beginning of a set of data element definition statements which redefine the last data element specified in the base data portion of the variable record. The 2-character code identifies the specific record.

- a. The redefined portion of the record may differ from one coded record to another in the same data set, as opposed to the base portion which is identical throughout the data set.



- b. The coded or redefined portion of the record may be redefined as many times as necessary but each redefinition must be identified by a different record code.
- c. Record codes must not be given a level number.

```
[.p.] vvvvxxxx=n
```

See previous explanation of the [.p.] vvvvxxxx=n statement following the [vvvvCODE=2] statement. Since element entries under the "RECORD-CODE" specification are actually redefining the last element in the Base Data portion of the record, a level number is always required for these entries.

```
mmmmLKxx=8=vvvvxxxx
```

See previous explanation of the mmmmLKxx=8=vvvvxxxx statement following the [vvvvCODE=2] statement.

```
END-DATA
```

Description: This statement ends the definition of logical data elements for the data set.

```
[TOTAL-LOGICAL-RECORDS=n]
```

Entry: n is a numerical value.

Description: Optional entry. See previous explanation of the same statement in the master data set statements (section 3.4.2).

```
[LOGICAL-RECORD-LENGTH=n]
```

Entry: n is a numerical value.

Description: Optional entry. See previous explanation of the same statement in the master data set statements (section 3.4.2).



[LOGICAL-RECORDS-PER-BLOCK=n]

Entry: n is a numerical value.

Description: Optional entry. See previous explanation of the same statement in the master data set statements (section 3.4.2).

DRIVE=nnn,nnnn

See previous explanation of the same statement in the master data set statements (section 3.4.2).

[LOAD-LIMIT=n]

Entry: n is a percentage, expressed as an integer value.

Description: This statement specifies a percentage used to create a threshold for space management. Additions to existing chains may occupy space above the threshold. Additions which start a new chain exceeding the threshold are spread across the remaining file space (see figure 3-4). If n is omitted or its value is 0 or 100, a default value of 80 is used (e.g., 80 percent).

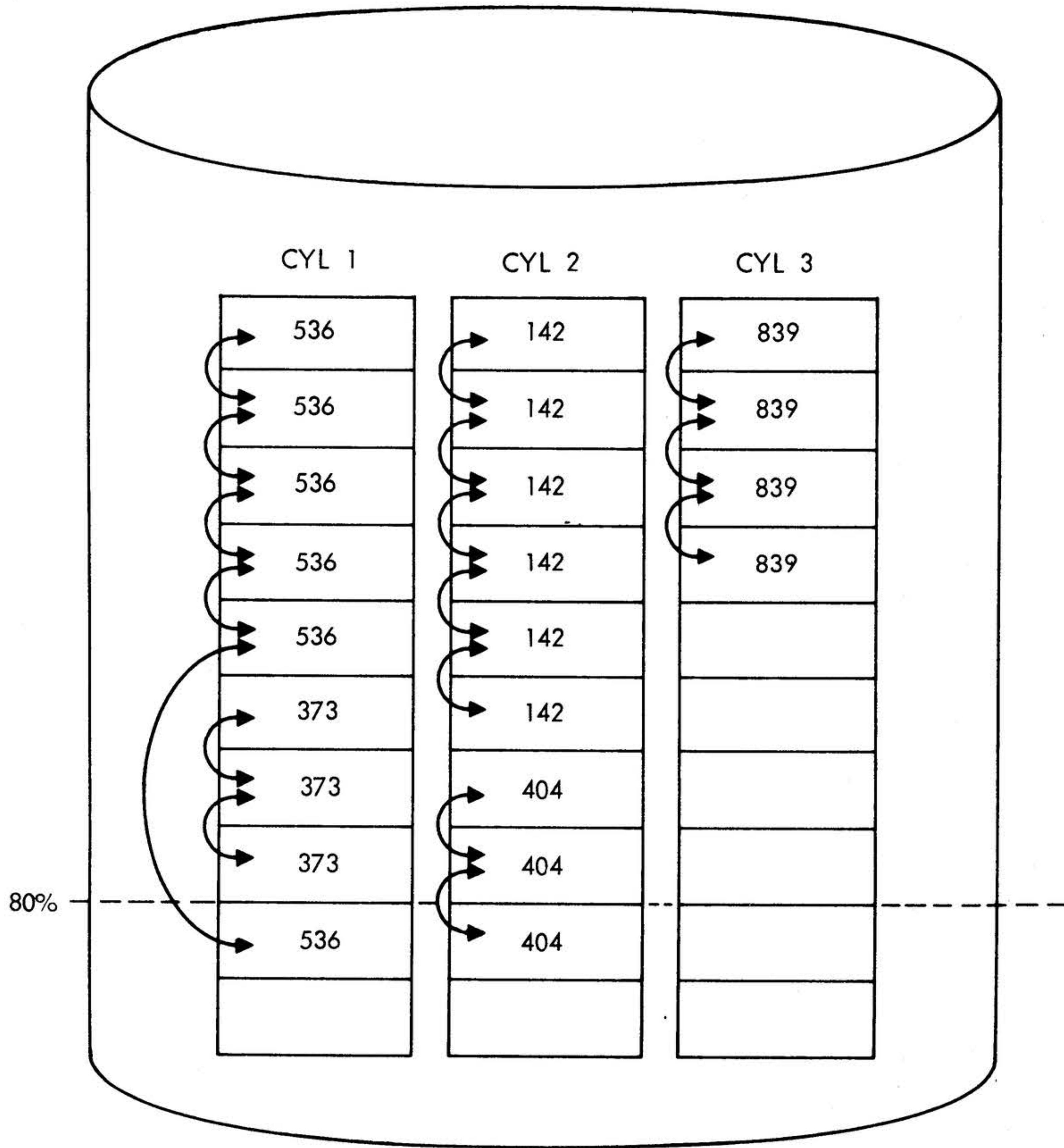
END-VARIABLE-ENTRY-DATA-SET

Description: This must be the last statement to end definition of a variable data set.

3.4.4 Epilogue Statements

END-DATA-BASE-GENERATION

Description: This statement must be the last statement of a data base definition.



VTI1-3458

Figure 3-4. Cylinder Load Limit for Variable Entry Data Sets



3.5 DATA BASE DESCRIPTOR MODULE (DBMOD)

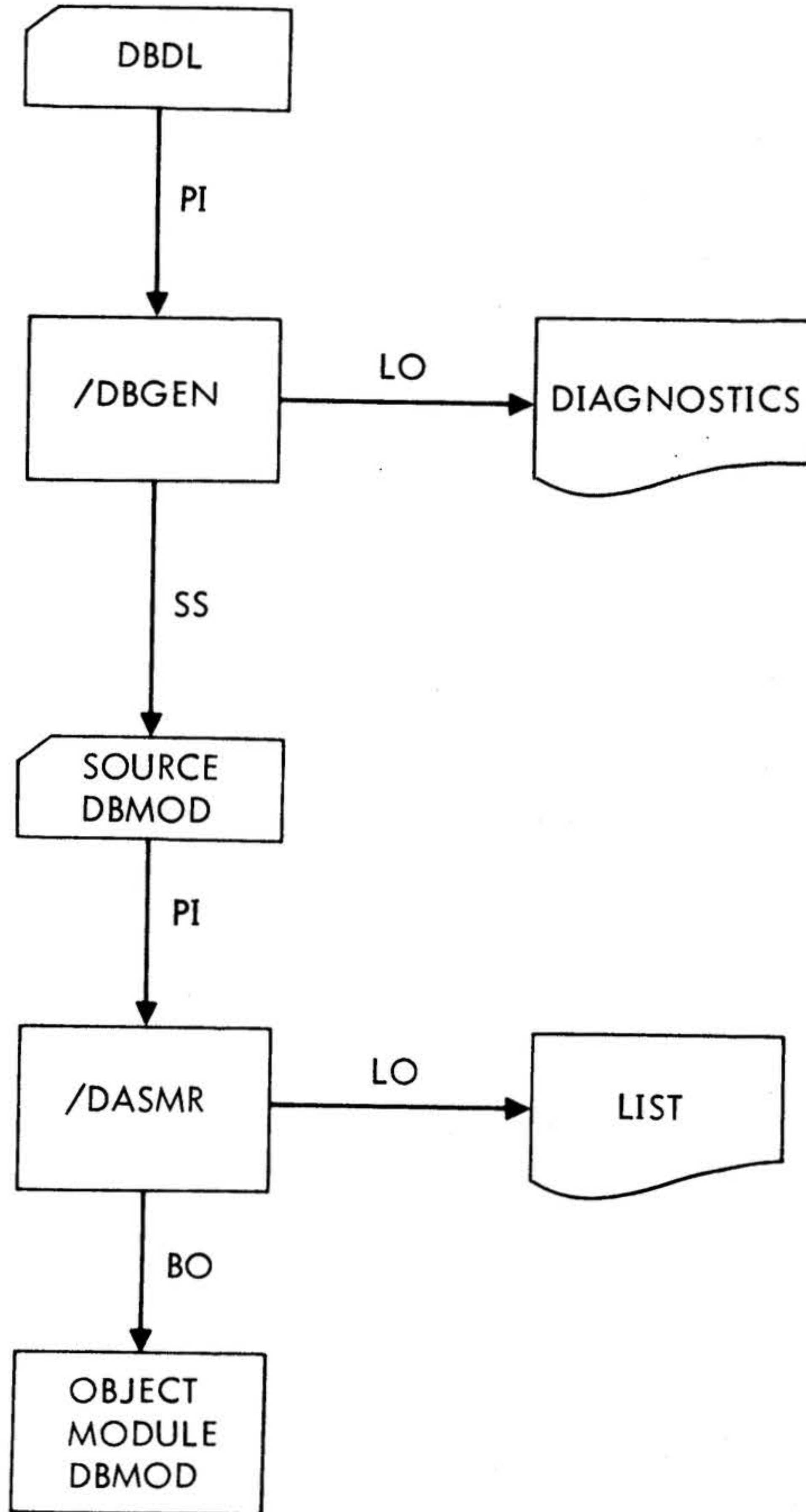
The DBDL statements describing a data base are input via the PI logical unit to DBGEN and are converted into assembler language source statements on the SS logical unit. Diagnostic messages on the LO logical unit indicate possible problems or errors.

The resulting assembler source statements are assembled by the DASMR assembler to produce an object data base descriptor module DBMOD. A flow chart showing the creation of the DBMOD module from the DBDL statements is given in figure 3-5.

The memory requirements for the Data Base Descriptor Module may be calculated from the number of words required for each element of the module, as listed in table 3-1.

Table 3-1. Data Base Descriptor (DBMOD) Memory Requirements

<u>DBMOD Element</u>	<u>Words Required</u>
General overhead	19 words
Overhead per file	6 words
Save area (once per DBMOD)	Largest logical record size or sector size, whichever is larger
Each Master File (Basic)	31 words
Each Variable File (Basic)	36 words
Each Master element (including Root)	6 words
Each Variable element	6 words
Each linkage (Master or Variable)	8 words
I/O area	4 words per pool plus buffer size
Buffer size (per buffer)	11 words plus the largest of sector size or logical record size
For each Logical Unit (LUN) i.e., for each DRIVE statement	17 words



VTI1-3459

Figure 3-5. Object Module DBMOD Flow Chart



3.6 EXAMPLE OF THE USE OF THE DATA BASE DEFINITION LANGUAGE

An example of a set of DBDL statements required to generate a DBMOD module using the DBGEN program is shown in figure 3-6.

The example shows the statements required to create a data base SUBRPE containing a personel master data set PERS linked to a skills inventory variable data set SKIV.

3.6.1 Computation of DBMOD

The computation which follows is taken from the example in appendix A.

EXAMPLE:

5 Master Files
 3 Variable Files
8 Files (total)

Master File CUST

Size (words)

Basic Master File overhead 31
 6 elements 36
 2 links 16
83

Master File DATE

Basic Master File overhead 31
 2 elements 12
 2 links 16
59

Master File INVT

Basic Master File overhead 31
 9 elements 54
 2 links 16
101

Master File ORNM

Basic Master File overhead 31
 2 elements 12
 1 link 8
51



EXAMPLE: (continued)

<u>Master File VEND</u>	<u>Size (words)</u>
Basic Master File overhead	31
7 elements	42
1 link	8
	<u>81</u>
<u>Variable File CORD</u>	
Basic Variable File overhead	36
15 elements	90
5 links	40
	<u>166</u>
<u>Variable File ACCR</u>	
Basic Variable File overhead	36
14 elements	84
1 link	8
	<u>128</u>
<u>Variable File PORD</u>	
Basic Variable File overhead	36
10 elements	60
2 links	16
	<u>112</u>

I/O AREAS

		<u>Buffer Size</u>
MAS1	2 buffers	120
VAR1	4 buffers	120
MAS2	6 buffers	120
MAS3	1 buffer	120
	<u>13 buffers</u>	

NOTE: The sizes of all logical records are less than the sector size.

$$\begin{aligned}
 & 4(\#areas) + (\#buffers)(11 + \text{sector size}) \\
 = & 4(4) + 13(11 + 120) \\
 = & 1719 \text{ words}
 \end{aligned}$$

LOGICAL UNIT BLOCKS

8 files x 17 = 136 words

Based on the preceding, the number of words required for the DBMOD is:

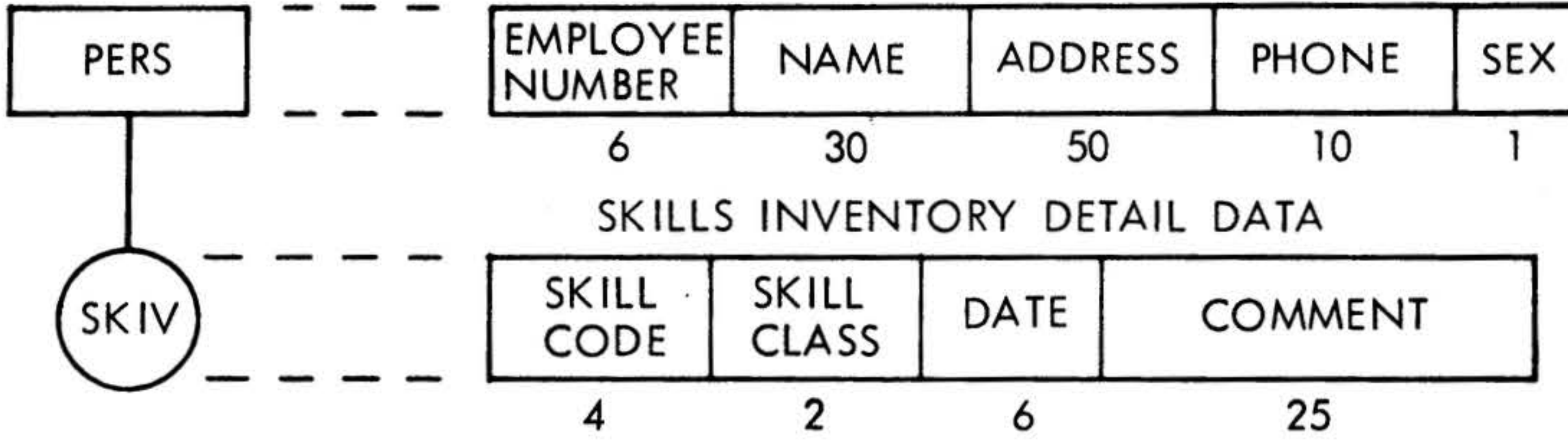
General overhead	19
File overhead 6 x 8	48
Save Area	120
Master File CUST	83
Master File DATE	59
Master File INVT	101
Master File ORNM	51
Master File VEND	81
Variable File CORD	166
Variable File ACCR	128
Variable File PORD	112
I/O Areas	1719
Logical Unit Blocks	<u>136</u>

DBMOD size = 2823 words

(approximately 3K)



SAMPLE DBGEN
PERSONNEL MASTER DATA



```

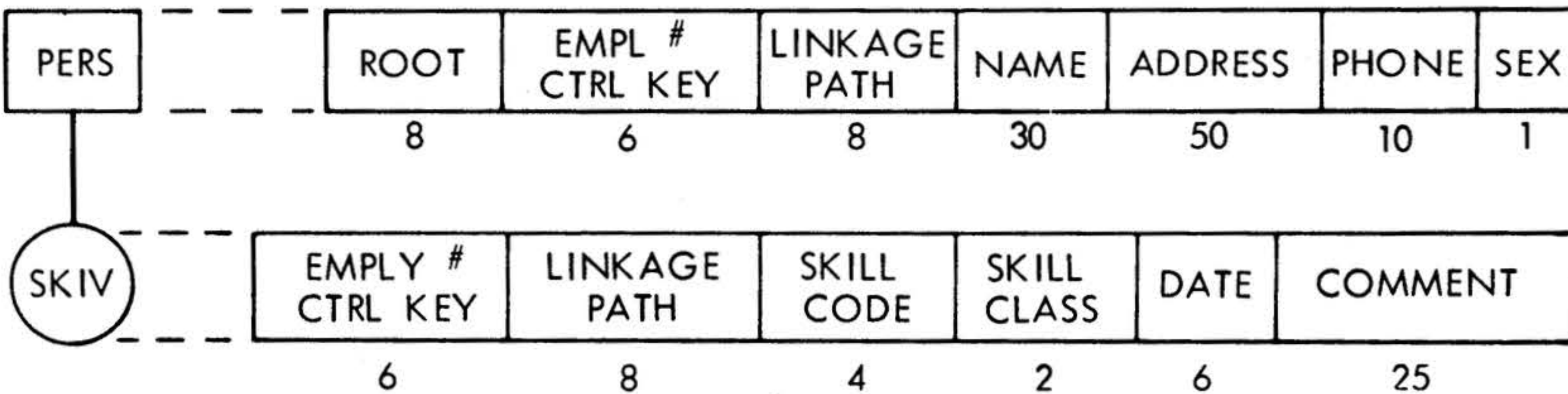
BEGIN-DATA-BASE-GENERATION
DATA-BASE-NAME = SUBRPE
OPTIONS  OUTPUT = Y
SHARE-IO
IOAREA = MASI
IOAREA = VARI
END-IO
BEGIN-MASTER-DATA-SET
DATA-SET-NAME = PERS
IOAREA = MASI
MASTER-DATA
PERSROOT = 8
PERSCTRL = 6
PERSLKSK = 8
PERSNAME = 30
PERSADDR = 50
PERSPHON = 10
PERSSEXX = 1
END-DATA
DRIVE = 26,500
END-MASTER-DATA-SET

```

```

BEGIN-VARIABLE-ENTRY-DATA-SET
DATA-SET-NAME = SKIV
IOAREA = VARI
BASE-DATA
SKIVPERS = 6
PERSLKSK = 8 = SKIVPERS
SKIVCODE = 4
SKIVCLAS = 2
SKIVDATE = 6
SKIVCOMM = 25
END-DATA
DRIVE = 30,1000
END-VARIABLE-ENTRY-DATA-SET
END-DATA-BASE-GENERATION

```



VTI1-3460

Figure 3-6. Example of Using the DBDL to Create a Data Base



SECTION 4 DATA BASE FORMATTOR

It is necessary to create the files and format the disc area before data can be written on the data base. This is achieved by means of the user FORMAT program which reads data set control cards and outputs a serial disc file according to physical parameters.

4.1 THE USER FORMATTOR

Part of the TOTAL package is a formatter program which resides on the OM library under the name DBFMT. DBFMT and the data base DBMOD object modules are LMGENed together (in that order) to form the (load module) user formatter.

Example: Assume that DBFMT is on the OM library and the user's DBMOD is on partition LV with protection code P. The following job stream will catalog the user formatter, called FORMAT, into the BL library.

```
/JOB,DBFORM
/LMGEN
TIDB,FORMAT,1,0
LD,OM,D,DBFMT
LD,LV,P,DBMOD
LIB
END,BL,E
/FINI
```

The user formatter may execute as a foreground or priority 1 background task. In the preceding example, FORMAT is catalogued as a priority 1 task on the BL library. To execute FORMAT, the JCP "/PLOAD,FORMAT" directive is used. FORMAT could have been catalogued as a foreground task and executed via a SCHED request.

FORMAT reads data base control directives from the PI logical unit, and creates and formats the requested data sets on the RMD devices. Diagnostics are printed out on the LO logical unit. Control directives are 80-character records.

The control directive format is:

```
bFORMAT xxxxxx file1, [file2,.....,] END.
```



where:

b is a required blank in character position 1.

FORMAT is the directive name and must start at character position 2.

xxxxxx is the data base name

file1...filen are data set (file) names

The last parameter must be END.

Continuation records may be used if there is not enough space on one record for all the file names that are required. However, FORMAT xxxxxx must be duplicated for each record. As each file is successfully formatted, FORMAT outputs a message on LO which includes the file name and the number of sectors formatted.

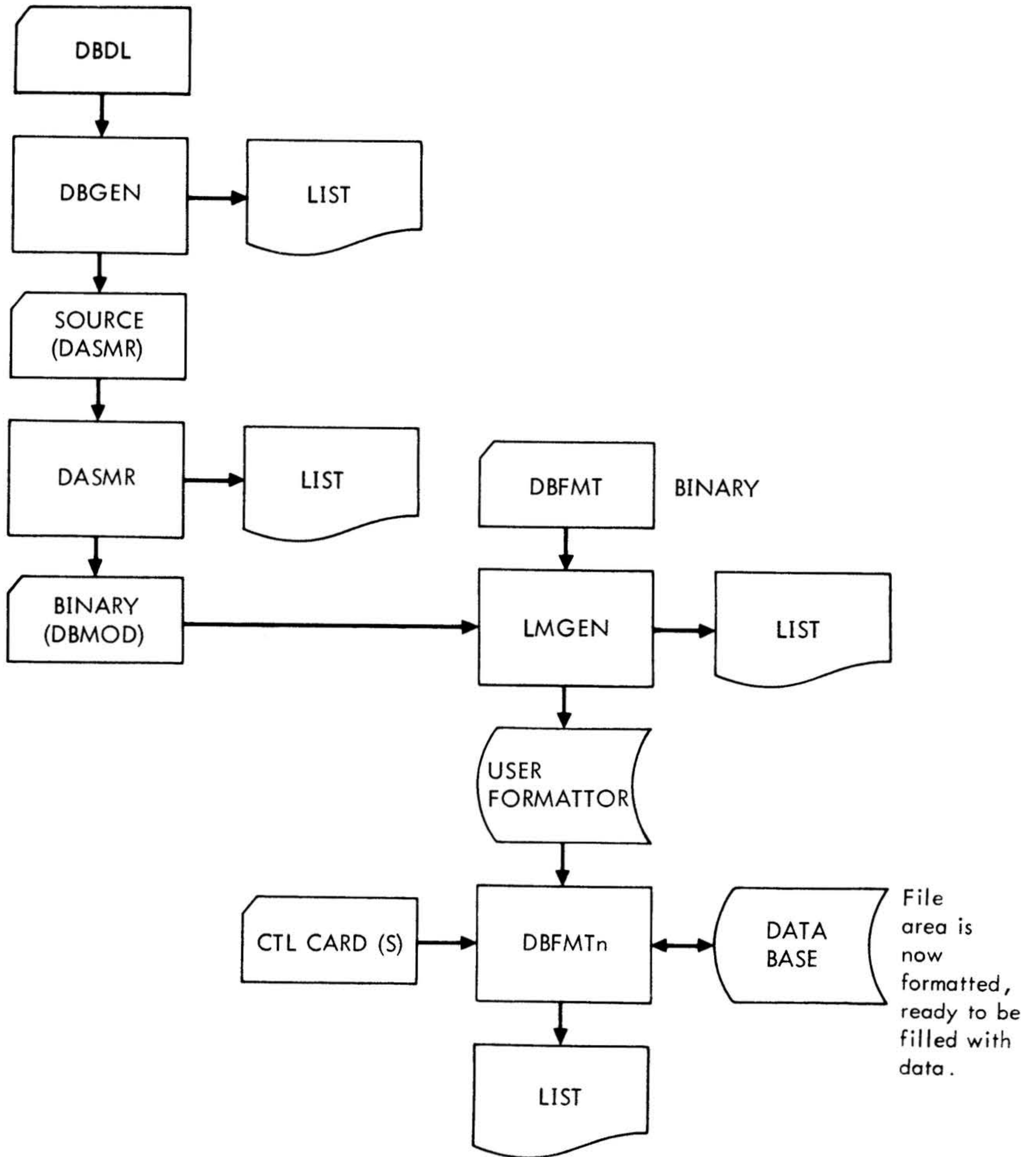
Example 1: Format data base MFGDBS with files PART,BILL,WCTR, and ROUT, all written in one line. Note that the first character in the FORMAT control directive is a blank.

```
/JOB,DBFORM
/ASSIGN,PI,SI
/PLOAD,FORMAT
  FORMAT MFGDBS PART,BILL,WCTR,ROUT,END.
/ENDJOB
```

Example 2: Format the same data base with example 1, utilizing multiple control directives.

```
/JOB,DBFORM
/ASSIGN,PI,SI
/PLOAD,FORMAT
  FORMAT MFGDBS PART
  FORMAT MFGDBS BILL
  FORMAT MFGDBS ROUT
  FORMAT MFGDBS WCTR
  FORMAT MFGDBS END.
/ENDJOB
```

A flow chart showing the Data Base formatting sequence of events is shown in figure 4-1.



VTI1-3483

Figure 4-1. Data Base Formatting Flow Chart



4.2 ADDING FILES TO THE DATA BASE

It is not necessary to format all of the data base at once. Each application can format only those files in the data base which were not previously formatted. Where a file is to be added to an existing data base, only the new file need be formatted, not the complete data base. If the new file is to be linked to an existing file and the logical record size of the existing file can accommodate a new link (four words per link), reformatting of the existing file is not necessary. Otherwise the existing file must be dumped, reformatted, and reloaded. The user can reduce the reformatting of existing files by anticipating their growth and establishing longer logical records when initially formatting the files.

4.3 FORMAT ERRORS

Refer to appendix B for a description of error messages output by DBFMT.



SECTION 5 DATA MANAGEMENT LANGUAGE

The Data Management Language (DML) is a means of accessing and manipulating a defined data base. The language operates by invoking TOTAL through the CALL facility of the host programming language. When such a CALL is encountered, control is passed to TOTAL, which analyzes a parameter list to determine the function (i.e., "command") to be performed and the data to be acted upon. Communication between the application program and TOTAL is affected through work areas referenced in the parameter list. When control returns to the application program from TOTAL, a status code is also returned to indicate the result of the operation. If the operation is completed successfully, a code of "****" is returned. If the operation was unsuccessful, the data base is restored to its condition before the operation, if necessary, and an appropriate status code is returned to indicate the cause of failure.

5.1 COMMAND PARAMETERS

The parameter list in the CALL statement is the medium of communication between TOTAL and the user's program. The parameters themselves are the names of areas defined elsewhere in the user's program. As might be expected of any called sub-program, TOTAL demands that the parameter list be in a certain order; the order shown in the examples throughout this manual must be strictly followed.

5.1.1 Functional Usage

Functions are provided to:

SIGN ON the data base

SIGN OFF the data base

ADD RECORDS to the data base

DELETE RECORDS from the data base

READ DATA ELEMENTS from the data base

WRITE DATA ELEMENTS to the data base



Of the fifteen different parameters which are available, some are used in every CALL to TOTAL, some depend on the particular type of data set being accessed (i.e., master versus variable), and a few are used only in certain specialized functions.

The following parameters are used in all but a few special functions:

OPERATION

STATUS

DATA-SET

REFERENCE

LINKAGE-PATH

CONTROL-KEY

DATA-LIST or ELEMENT-LIST

DATA-AREA

ENDP

The parameters can be used for serial functions, master data set functions, or variable data set functions, as shown in table 5-1.

Table 5-1. Parameters Available for Functional Usage

PARAMETER	SERIAL FUNCTIONS	MASTER DATA-SET FUNCTIONS	VARIABLE DATA-SET FUNCTIONS
OPERATION	X	X	X
STATUS	X	X	X
DATA-SET	X	X	X
REFERENCE	X		X
LINKAGE-PATH			X
CONTROL-KEY		X	X
DATA-LIST	X	X	X
DATA-AREA	X	X	X
END.	X	X	X



5.1.2 Notation Conventions

In the descriptions and definitions which follow, certain notation conventions are used to express the format of a statement or a parameter. These may be simply explained by the following rules:

- a. Lower case letters are to be replaced by a symbol of the user's choosing.
- b. Upper case letters are to be inserted as they appear.
- c. Square brackets ([]) enclose a choice of options of which none, one, or several may be chosen.
- d. Braces ({ }) enclose a choice of options of which one and only one must be chosen.

5.1.3 Detailed Descriptions of Parameters

The nine "standard" parameters are described in the paragraphs which follow, before the discussion of the individual commands. There they will be shown where they occur, but described only to the extent that they vary from the discussion below. The only exception is the parameter OPERATION which will be shown as the operation code of the function to be performed.

OPERATION:

This parameter is the name of (points to) a 5-character field defined by the user into which he must place the operation code of the function to be performed, e.g., READM - read a master data set randomly.

STATUS:

This parameter is the name of (points to) a 4-character field defined by the user into which TOTAL places a code indicating the result of the operation, e.g., "*****" (the function has successfully completed), "FNTF" (File Note Found and the function has not been performed). THIS FIELD SHOULD BE EXAMINED AFTER EVERY COMMAND. A complete list of status codes and their meanings may be found in the Diagnostic section.

**DATA-SET:**

This parameter is the name of (points to) a 4-character field defined by the user into which the user must place the name of the data set to be operated upon as defined in a data base generation (DBMOD). The DATA-SET parameter is shown graphically in figure 5-1.

REFERENCE:

This parameter is the name of (points to) a 4-character field defined by the user which is used to maintain the internal reference point of the current variable record or a position in either a master or variable using the RDNXT function. This field is used by both TOTAL and the user to communicate information about processing along a relationship within a variable data set or along a serial retrieval of a data set. As such, each performs a specific role by inserting appropriate values into the reference field and expecting certain values to be present under certain conditions. This may be best described by listing the acceptable contents of the reference field, qualified by the role of the participant:

a. LKxx

This is the last four characters of a linkage path name (mmmmLKxx) as defined in the Data Base Descriptor Module. The user places this value into the reference field to indicate that TOTAL is to retrieve a chain (depending on the operation code) and that processing is expected to continue along the specified linkage path.

TOTAL places this value into the reference field to indicate that the first record of a chain has been deleted.

b. rrrr

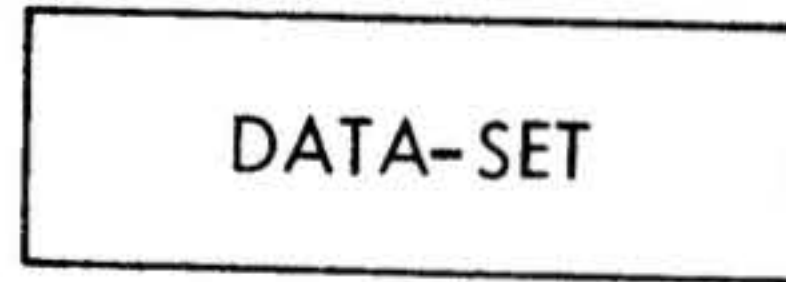
This is the Internal Reference Point (relative record number) of the record currently being processed.

The user places such a value into the reference field to directly retrieve a specific record whose Reference Point was previously known.

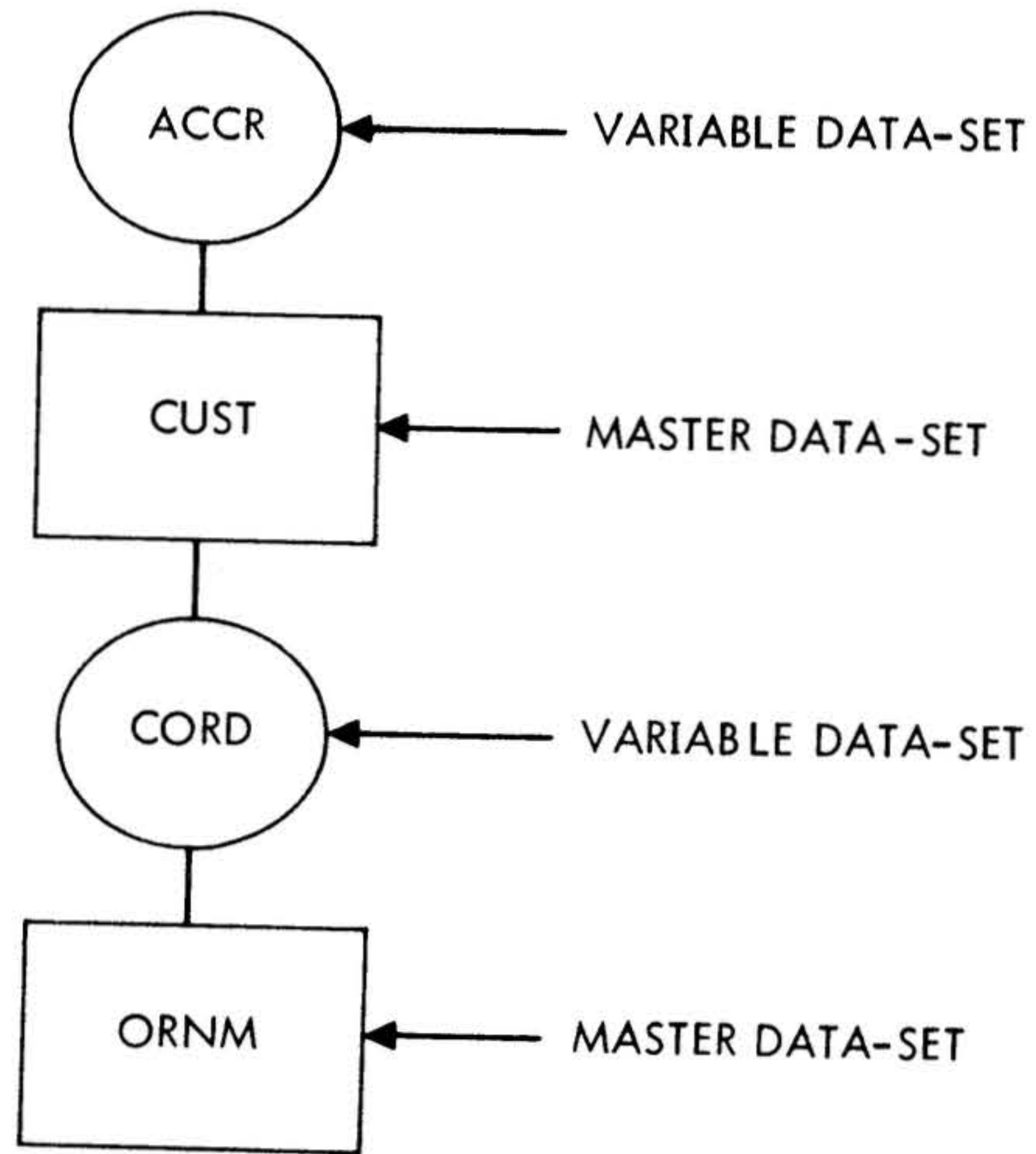
The user also may place into the reference field a value which he previously saved upon interrupting continuous processing along a chain or reset a serial retrieval to some point in a data set.



DATA MANAGEMENT LANGUAGE
PARAMETER



- 4-CHARACTER AREA
- IDENTIFIES THE DATA SET TO BE OPERATED ON



VT11-3463

Figure 5-1. DATA-SET Parameter



TOTAL places into the reference field the Internal Reference Point of the record just read, added, or written, or the "back pointer" from a deleted record (unless the deleted record was the first of a chain).

The effect of placing appropriate values into the reference field prior to execution of a TOTAL command is given in table 5-2. The contents of the reference field after execution of a TOTAL command is shown in table 5-3.

Table 5-2. Effect of Values in Reference Field Before Execution

FUNCTION	CONTENT		
	LKxx	rrrr	END.
ADDVA	The operation is not performed and a status code of IVRP is returned.	The record in the user's data area is added logically after the record addressed by reference.	IVRP
ADDVB	The operation is not performed and a status code of IVRP is returned.	The record in the user's data area is added logically before the record addressed by reference.	IVRP
ADDVC	The record in the user's data area is added logically to the end of all chains controlled by keys within the record.	The record in the user's data area is added logically to the end of all chains controlled by keys within the record.	IVRP
DELVD	The operation is not performed, and a status code of IVRP is returned.	The record addressed by reference is deleted.	IVRP
READD	The operation is not performed, and a status code of IVRP is returned.	The record addressed by reference is retrieved.	IVRP
READR	The record at the end of the chain is retrieved.	The record logically before the one addressed by reference is retrieved.	IVRP



Table 5-2. Effect of Values in Reference Field Before Execution (continued)

FUNCTION	CONTENT		
	LKxx	rrrr	END.
READV	The first record in the chain is retrieved.	The record logically after the one addressed by reference is retrieved.	IVRP
WRITV	The operation is not performed, and a status code of IVRP is returned.	The record addressed by reference is processed.	IVRP

Table 5-3. Content of Reference Field After Execution

FUNCTION	CONTENT
ADDVA ADDVB ADDVC	Internal Reference Point of the record just added.
DELVD	"Back pointer" from the record just deleted (e.g., the reference point of the record logically before the record just deleted) or LKxx if the first record in the chain was deleted.
READD	Internal Reference Point of the record just read.
READR READV	Internal Reference Point of the record just read or "END." (if the read attempted to go off the end or beginning of the chain).
WRITV	Internal Reference Point of the record just written.



c. END.

This value is placed into the reference field by TOTAL when the user, while continuously processing along a chain of records, attempts to go beyond the end of the chain if reading forward or beyond the beginning if reading reverse.

d. BEGN

This value placed into the reference field by the user and used in conjunction with the "RDNXT" function will cause the "RDNXT" to start serially reading a specific data set at the absolute beginning of that file. Upon reaching the end of a file, "END." is placed in the reference by TOTAL.

LINKAGE-PATH:

This parameter is the name of (points to) an 8-character field defined by the user into which he must place the 8-character name of the linkage path (mmmmLKxx) as defined in the Data Base Descriptor Module. This is the vehicle through which the user dynamically names a specific relationship between a chain of variable records and a master record by the record control key.

The terms "primary linkage path" and "controlling linkage path" refer to the linkage path named by the LINKAGE-PATH parameter. The term "secondary linkage-path" refers to any other linkage path defined for this record in the Data Base Descriptor. The LINKAGE-PATH is shown graphically in figures 5-2 and 5-3.

CONTROL-KEY:

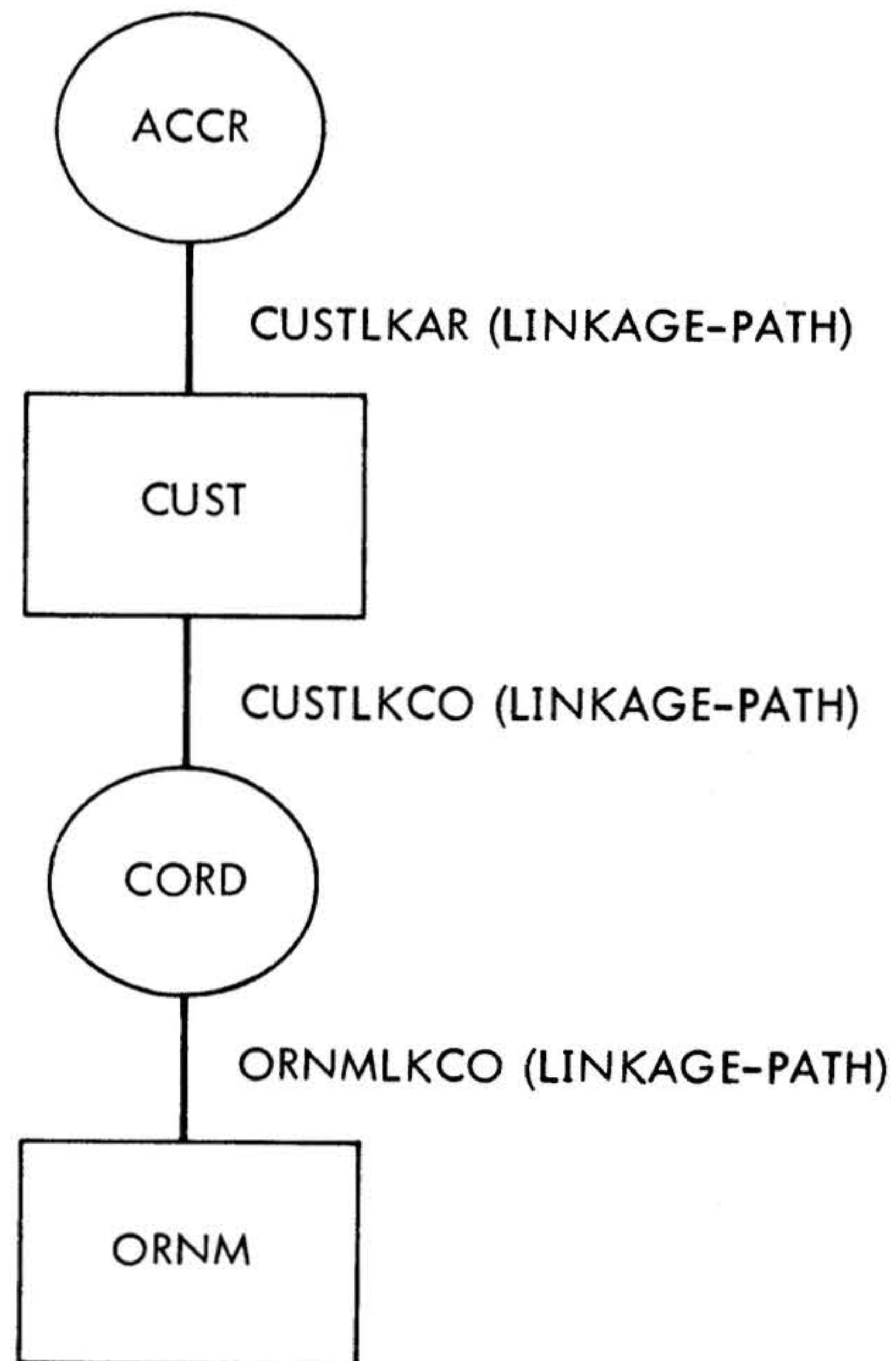
This parameter is the name of (points to) a field defined by the user into which he places the record control key. TOTAL will "randomize" on this data, whether to locate a master record or to link from a master record to a variable record. If, during further processing of this command, it is found that the CONTROL KEY does not agree with the corresponding field in the user's data area, a status code of UCTL will be returned. To avoid this, it is recommended that the user name the control key field in the data area rather than define a separate field. The length of the CONTROL KEY is taken to be that defined in the Data Base Descriptor Module. The CONTROL-KEY is shown graphically in figure 5-4.



DATA MANAGEMENT LANGUAGE
PARAMETER

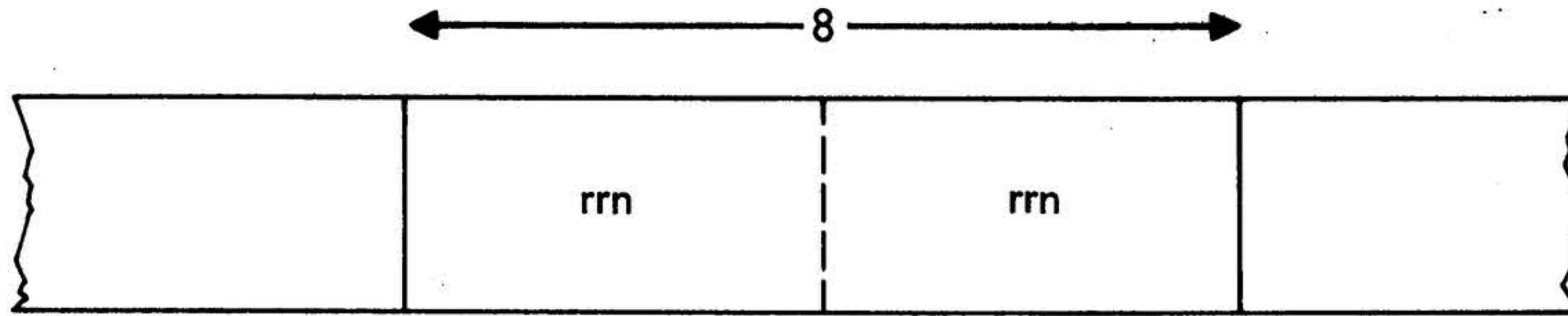
LINKAGE-PATH

- 8-CHARACTER AREA
- IDENTIFIES THE LINKAGE KPA PATH BY A 2-CHARACTER VALUE TO BE PROCESSED



VTI1-3464

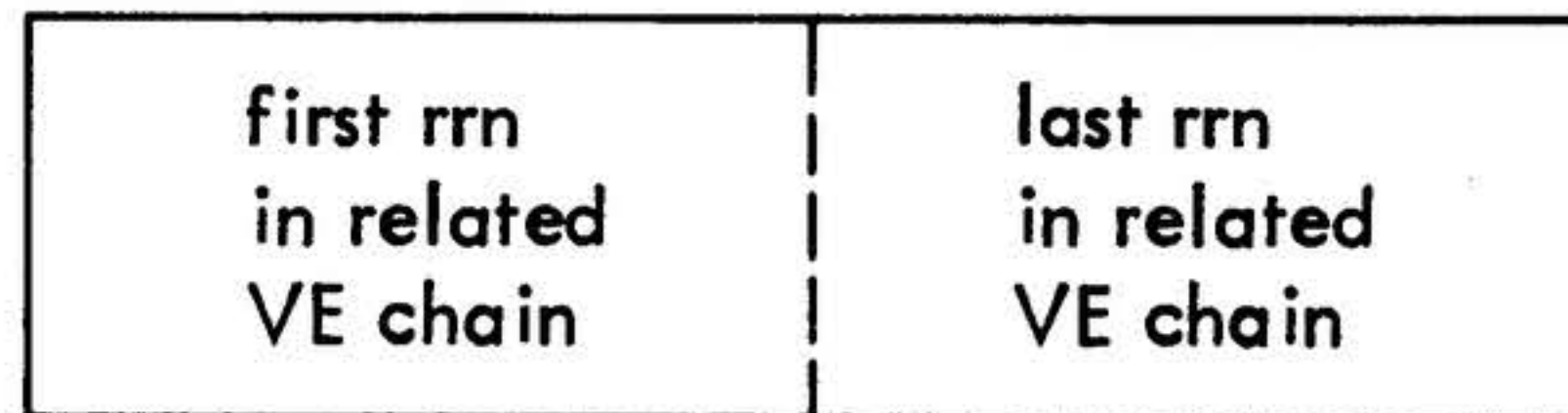
Figure 5-2. LINKAGE-PATH Parameter



A field used by TOTAL to maintain logical relationships between records

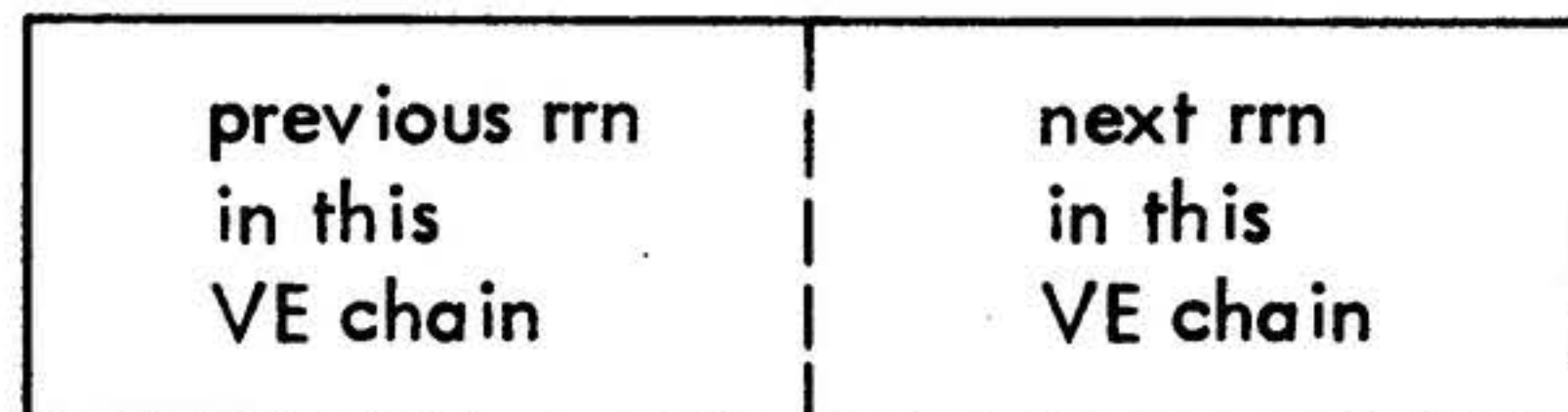
- 8 byte (character)
 - binary pointers (rrn)
- Contents automatically managed by TOTAL

SINGLE ENTRY LINKAGE PATH



- Establishes relatibility from SE record to VE record chain

VARIABLE ENTRY LINKAGE PATH



- Establishes bi-directional relatibility in a chain of VE records

VTI1-3465

Figure 5-3. Linkage Path



DATA MANAGEMENT LANGUAGE PARAMETER

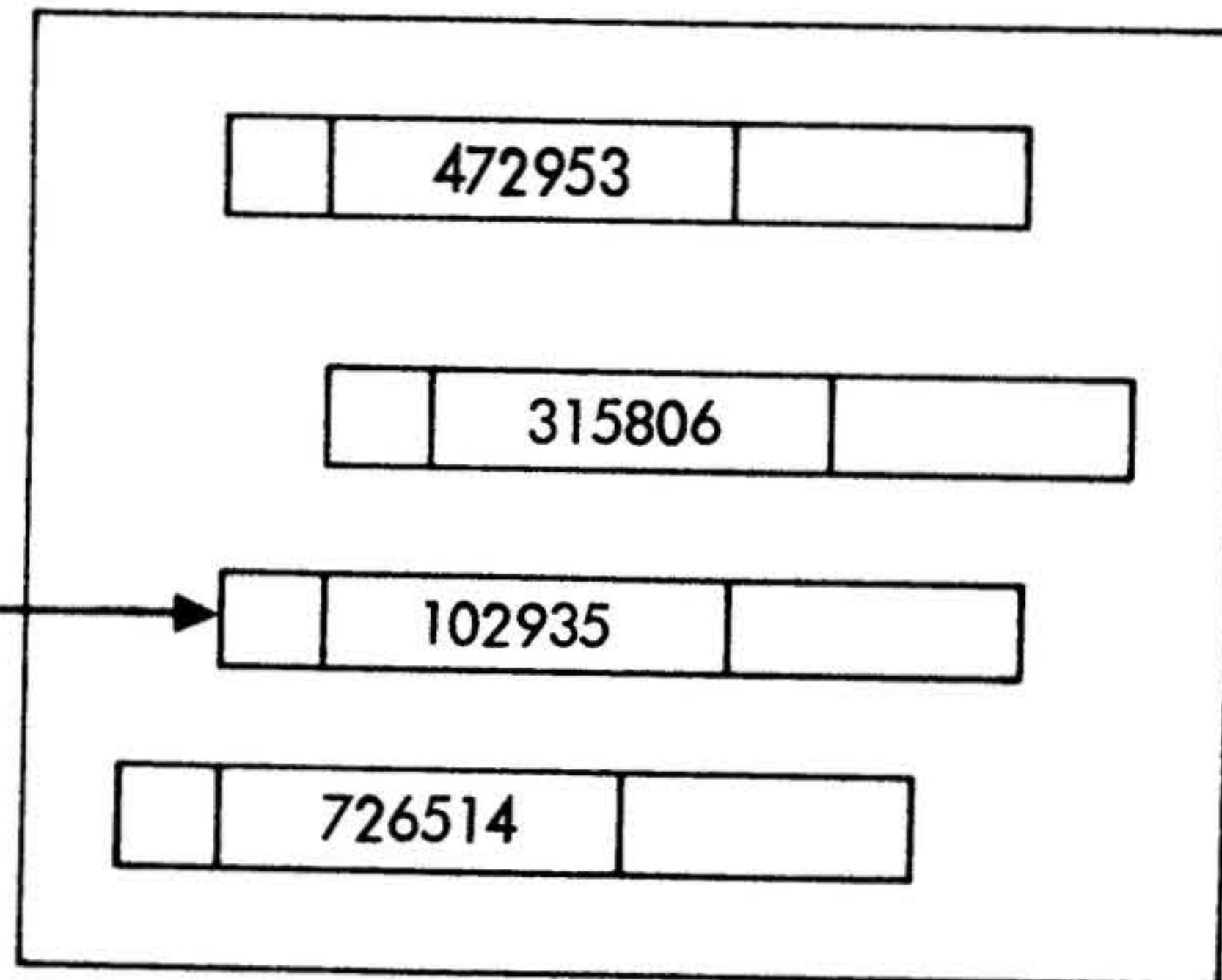
CONTROL-KEY

- AREA DEFINED TO CONTAIN CONTROL KEYS FOR SINGLE ENTRY DATA SETS
- IDENTIFIES A MASTER RECORD IN THE SINGLE ENTRY DATA SET

MASTER RECORD
CONTROL-KEY

102935

CUST



VTI1-3466

Figure 5-4. CONTROL-KEY Parameter



DATA-LIST or ELEMENT-LIST:

This parameter is the name of (points to) a list of data names. The list is a character string defined by the user which is composed of 8-character data names declared in the data base generation. This list must conform to the following format:

```
elem1elem2.....elem(n)END.
```

The data names in the list may include:

- data elements
- data items
- control keys
- record codes

The list may not include:

- the ROOT field (master records only)
- linkage fields

The data names in the list appear in any order, but the data elements they name will be processed in the order listed. Thus, the data list is ordered in the same manner as the user's data area, not necessarily as the record on the data set. Only the data elements named in the data list will be processed, i.e., transferred to or from the user's data area. It is suggested that the order of element names coincide with the generated order from DBGEN. A DATA-LIST is shown graphically in figure 5-5.

One very significant feature of TOTAL is what is referred to as a CODE DIRECTED READ. This is the ability, when processing Variable Entry Data Sets that contain coded-records, to only retrieve into your program, data elements that pertain to one or more coded-records, not every record.

In order to achieve this CODE DIRECTED read capability, a slight variation in the presentation of the data-list is required.

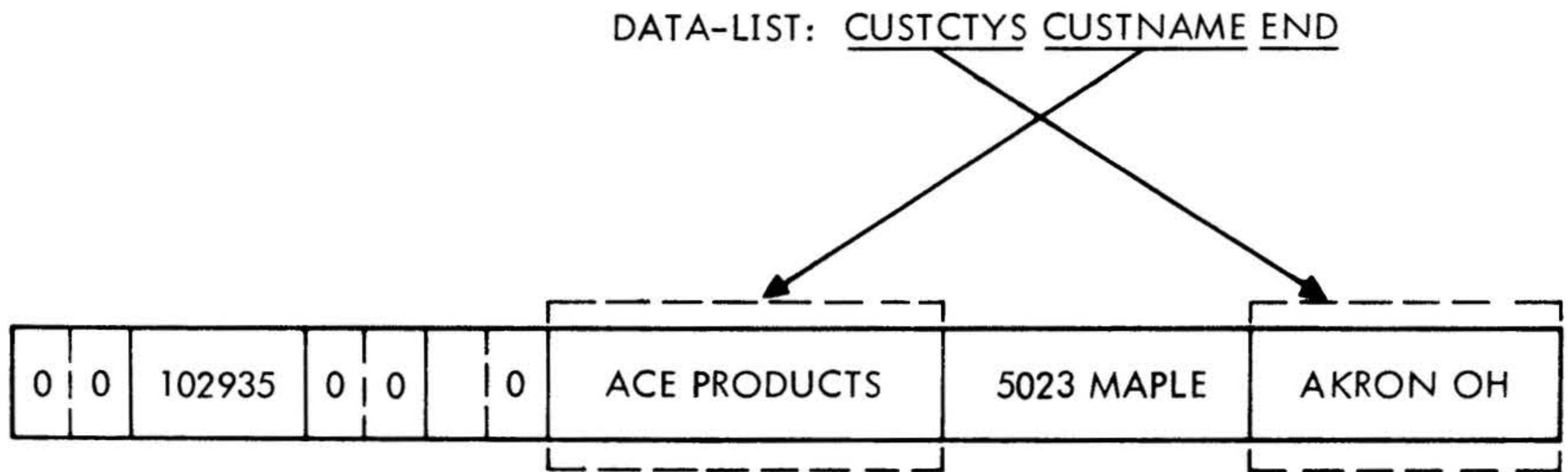
In our example above, the inclusion at the immediate beginning of the data-list of one or more entries which follow will allow this feature.



DATA MANAGEMENT LANGUAGE
PARAMETER

DATA-LIST

- AREA CONTAINING A LIST OF DATA ELEMENTS
- LIST IS TERMINATED BY 'END.'
- LIST IDENTIFIES THE DATA ELEMENTS TO BE REFERENCED



VT11-3467

Figure 5-5. DATA-LIST Parameter



Example:

Read only variable entry data set records with CODES SR and CM.

```
*CODE=SR*CODE=CM elem1elem2.....elem(n)END
```

The entry *CODE=xx at the beginning of the data list will cause TOTAL to only retrieve and pass back elements for these type of records.

NOTE: When constructing element lists for variable entry file processing when coded records are defined, the "VVVVCODE" element name must always be the first one presented.

DATA-AREA:

This parameter is the name of (points to) an area of memory defined by the user which is used as an input/output area for the data elements named in the Data List. The structure and characteristics of this area must conform exactly to the data elements as named in the Data List and in the same order. The DATA-AREA is shown graphically in figure 5-6.

ENDP:

This parameter is the name of (points to) a 4-character field defined by the user which must contain the value 'END.' This parameter serves as the delimiter to the parameter list. As noted earlier, the parameter will not be passed for any function requiring nine parameters (variable functions only) and will not appear in the descriptions of the Variable Entry commands.

5.2 DESCRIPTION OF DML COMMANDS

The following pages list in alphabetic order all of the Data Management Language commands with a detailed description of each. The commands are summarized in table 5-4, and the calling sequences for various languages are given in table 5-5.

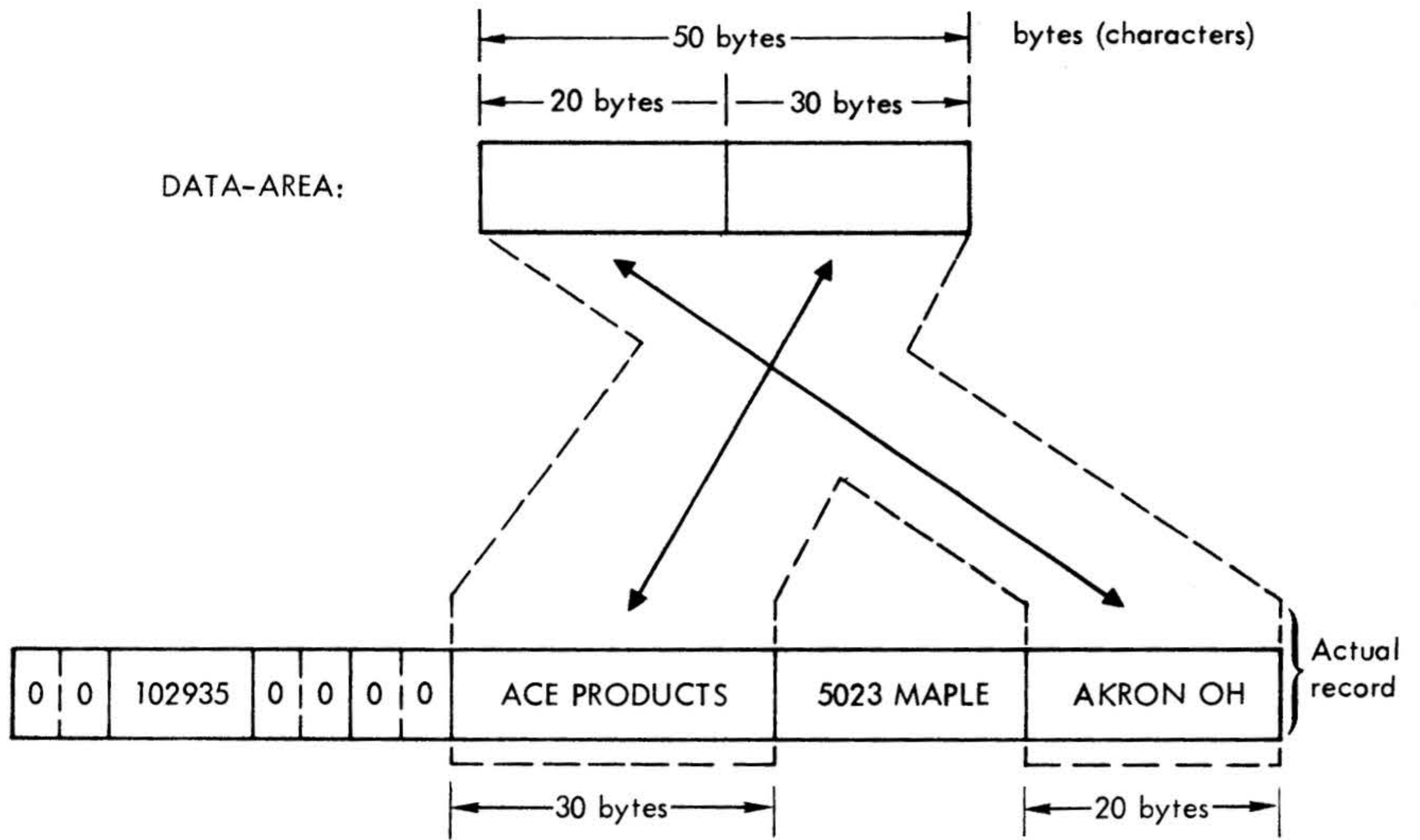
An attempt has been made to make each description as complete as possible to avoid the need to consult several sources. The assumption is made however, that the user has read section 5.1, which discussed each of the standard parameters from a general functional point of view. In the following descriptions, therefore, some of the parameters will not be described at all (e.g.,



DATA MANAGEMENT LANGUAGE
PARAMETER

DATA-AREA

- AREA DEFINED TO HOLD THE DATA OF THE ELEMENTS IDENTIFIED IDENTIFIED IN DATA-LIST
- MUST BE AS LARGE AS THE SUM OF THE LENGTHS OF THE DATA ELEMENTS INDICATED



VTI1-3468

Figure 5-6. DATA-AREA Parameter



ENDP, DATA-SET, etc.), while others may be described only where they depart from the standard or need enhancement. Also instead of using the standard parameter OPERATION, the function mnemonic itself will be shown.

The status code list in each description is not intended to be all-inclusive, but rather should serve to point out some of the more common errors made in using a particular function. See appendix B for the repertoire of the status codes.

Each command with its required parameters is shown in table 5-6.

Table 5-4. Alphabetic List of DML Commands

- 1. ADD-M
- 2. ADDVA
- 3. ADDVB
- 4. ADDVC
- 5. DEL-M
- 6. DELVD
- 7. RDNXT
- 8. READD
- 9. READM
- 10. READR
- 11. READV
- 12. RQLOC
- 13. SINOF
- 14. SINON
- 15. WRITM
- 16. WRITV

Table 5-5. Calling Sequences to TOTAL for Various Languages

1. DASM	EXT	DATBAS
	CALL	DATBAS, param1, param2, ..., paramn
	or	
	EXT	DATBAS
	JMPM	DATBAS
	DATA	param1
	DATA	param2
	.	
	.	
	DATA	paramn



Table 5-5. Calling Sequences to TOTAL for Various Languages
(continued)

2. FORTRAN

CALL DATBAS (param1, param2, ..., paramn)

3. RPG II

EXIT DATBAS

RLABL param1

RLABL param2

.

.

RLABL paramn

4. COBOL

ENTER ASSEMBLER.

CALL 'DATBAS' USING param-1, param-2, ..., param-n

ENTER COBOL

NOTE: The appropriate language manual should be referred to for the exact meaning and procedure of CALL. Sample programs in FORTRAN, RPG II, and COBOL are given in appendix A.

Table 5-6. DML Commands with Required Parameters



varian data machines

5-18

5 Bytes	4 Bytes	4 Bytes	4 Bytes	8 Bytes	User Defined	8n+4 Bytes	User Defined	4 Bytes
Function	Status	File	Reference	Linkpath	Control	Elements	Area	END.
READM WRITM ADD-M DEL-M	X	X			X	X	X	X
RDNXT	X	X	X			X	X	X
READV READR READD WRITV DELVD ADDVC ADDVA ADDVB	X	X	X	X	X	X	X	X
							4-Char. Location	
RQLOC	X	X			X		X	X
		26+12n* SCHEMA						
SINON SINOF	X	X						X

Note: X indicates function or functions on the left have the parameters listed at the top of column.

* n is the number of files in the SCHEMA.



5.2.1 The Add Master Function

This function operates by randomizing on the contents of the Control Key to locate space, selecting data elements specified by the Data List, and writing the new record into the master data set.

Required Parameters:

ADD-M, STATUS, DATA-SET, CONTROL-KEY, DATA-LIST,
DATA-AREA, ENDP

ADD-M: Add Master Function Mnemonic

The user must place this mnemonic into the Operation Field.

STATUS: Status Code

Significant status codes which may be returned are:

BCTL: The Control Key Field contains blanks or zero.

DUPM: A master record with the same Control Key already exists on the data set.

FULL: There is no space available for this record.

UCTL: The Control Key Field does not match the corresponding field in the Data Area.

Programming Considerations:

- a. The record control key must be included in the data list; TOTAL does not move it from the Control Key Field to the Data Area.
- b. Any data fields not specified in the Data List will be zeros in the new record.

An example of coding the Add Master function and a graphic illustration of the randomizing of the contents of the control key is given in figure 5-7. It should be noted that the control-key record contents ('102935' in the example) should be the same as in the control key field in the Data-Area record.



DATA MANAGEMENT LANGUAGE

ADD-M	ADD MASTER FUNCTION
-------	---------------------

EXAMPLE:*

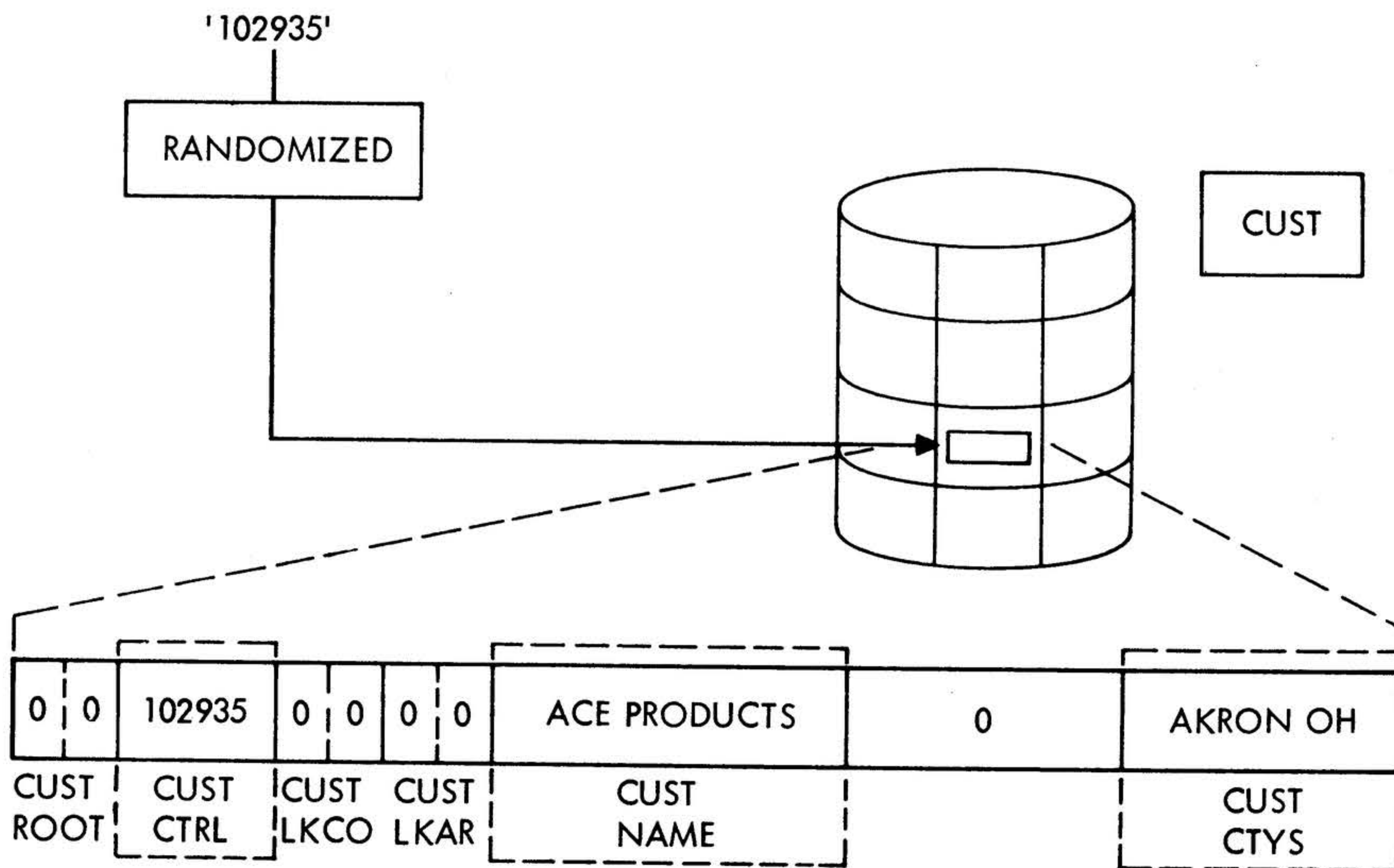
```

FUNCTION = 'ADD-M'
STATUS = blanks
DATA-SET = 'CUST'
CONTROL-KEY = '102935'
DATA-LIST = 'CUSTCTRL CUSTNAMECUSTCTYSEND.'
DATA-AREA = '102935 ACE PRODUCTS AKRON OH'
ENDP = 'END.'

```

AFTER THE FUNCTION COMPLETES

*All parameters are pointers to areas which contain the above data



VTI1-3469

Figure 5-7. The ADD MASTER Function



5.2.2 The Add Variable After Function

This function operates by logically adding the record in the Data Area after the record whose Internal Reference Point is in the Reference Field. This logical "insertion" is made only on the linkage path specified by the Linkage-Path parameter. On all other linkage paths defined for this record, the addition is made to the end of the respective chains.

Required Parameters:

ADDVA, STATUS, DATA-SET, REFERENCE, LINKAGE-PATH,
CONTROL KEY, DATA-LIST, DATA-AREA, ENDP

ADDVA: Add Variable After Function Mnemonic

The user must insert this mnemonic into the Operation Field.

STATUS: Status Code

Significant status codes which may be returned are:

BCTL: A control field contains blanks or zeros
FULL: The data set has no room for this record.
IVRC: The record code in a record pointed to on a coded linkage path has not been defined in the DBMOD.
MLNF: The linkage path name is invalid for either the file or the record code.
MRNF: A master record does not exist for a control field (primary).
UCTL: The Control-Key Field does not match the corresponding field in the Data Area.

REFERENCE: Internal Reference Point

This parameter normally specifies the Internal Reference Point (relative record number) of the record after which the user's record is to be added.

If the Reference Field contains any other value, the function will not be performed and a status code of IVRP will be returned.

When the function has successfully completed, the Reference Field contains the Internal Reference Point (relative record number) of the added record.



Programming Considerations:

- a. Addition of the record on secondary linkage paths requires retrieving a master record for each Control Key defined for this record. Therefore, these Control Keys must be present in the Data Area and must be valid, i.e., they must represent records which actually exist in the respective master data sets.
- b. Data elements not named in the Data List will be zeros in the new record. Examples of coding the Add Variable After function are given in figures 5-8 and 5-9. Note that in figure 5-8 the new record C is added after record B because the relative record number of record B is 0002, and this is the value given in the Reference field. Note similarly that in figure 5-9, the new record B is added after record A because its Internal Reference Point value is 0010, and this is the value given in the Reference field. The new record is added to the two Master Data Sets BOOK and BORW.

5.2.3 The Add Variable Before Function

This function operates by logically adding the record in the Data Area before the record whose Internal Reference Point is in the Reference Field. This logical "insertion" is made only on the linkage path specified by the Linkage-Path parameter. On all other linkage paths defined for this record, the addition is made to the end of the respective chains.

Required Parameters:

ADDVB, STATUS, DATA-SET, REFERENCE, LINKAGE-PATH,
CONTROL-KEY, DATA-LIST, DATA-AREA, ENDP

ADDVB: Add Variable Before Function Mnemonic

The user must insert this mnemonic into the Operation Field.

STATUS: Status Code

Significant status codes which may be returned are:

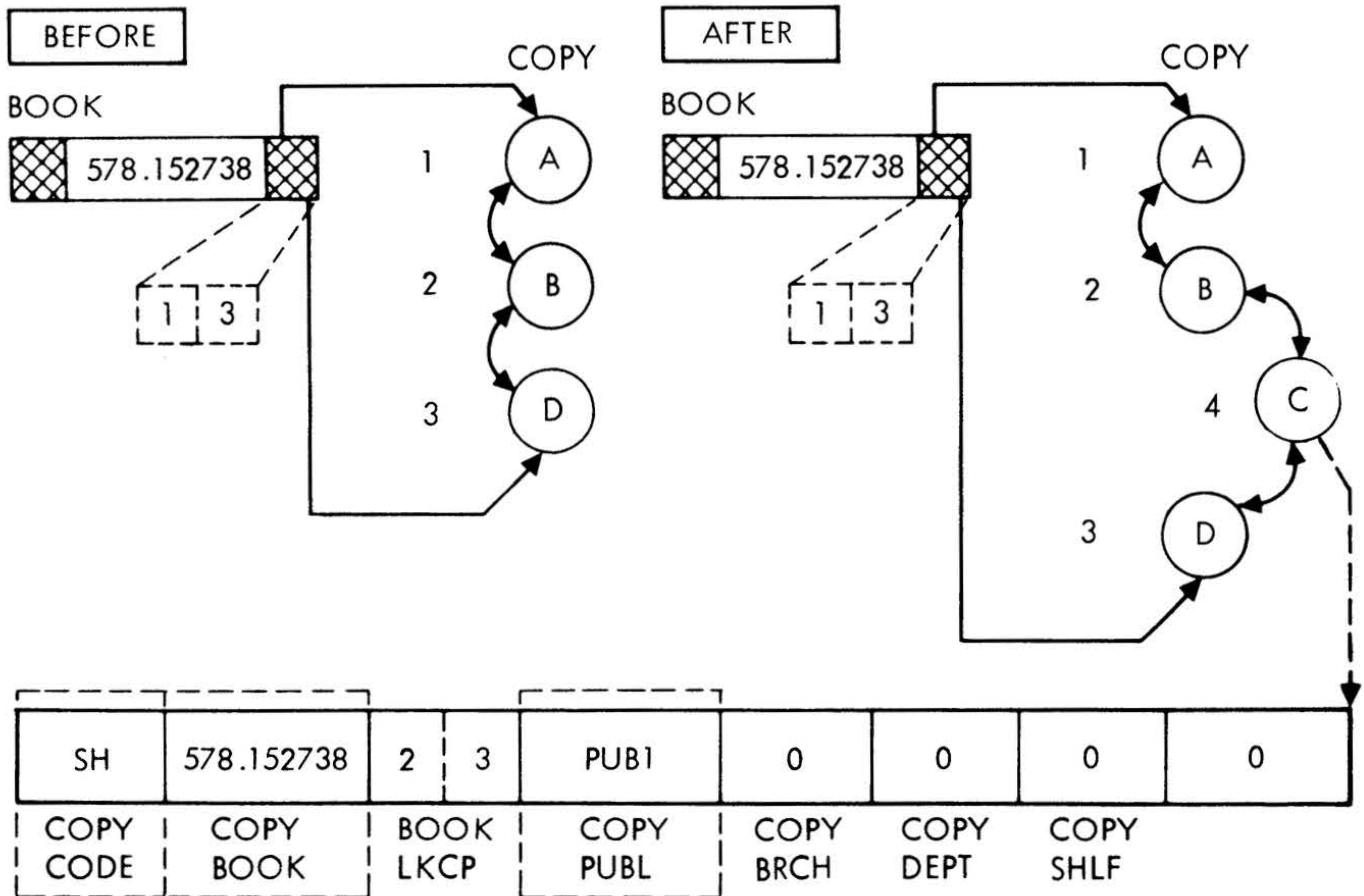


DATA MANAGEMENT LANGUAGE

ADDVA	ADD VARIABLE AFTER FUNCTION
-------	-----------------------------

EXAMPLE:

FUNCTION = 'ADDVA'
 STATUS = blanks
 DATA-SET = 'COPY'
 REFERENCE = '0002' (binary)
 LINKAGE-PATH = 'BOOKLKCP'
 CONTROL-KEY = '578.152738'
 DATA-LIST = 'COPYCODECOPYBOOKCOPYPUBLEND.'
 DATA-AREA = '578.152738PUB1 SH'
 ENDP = 'END.'



VTI1-3470

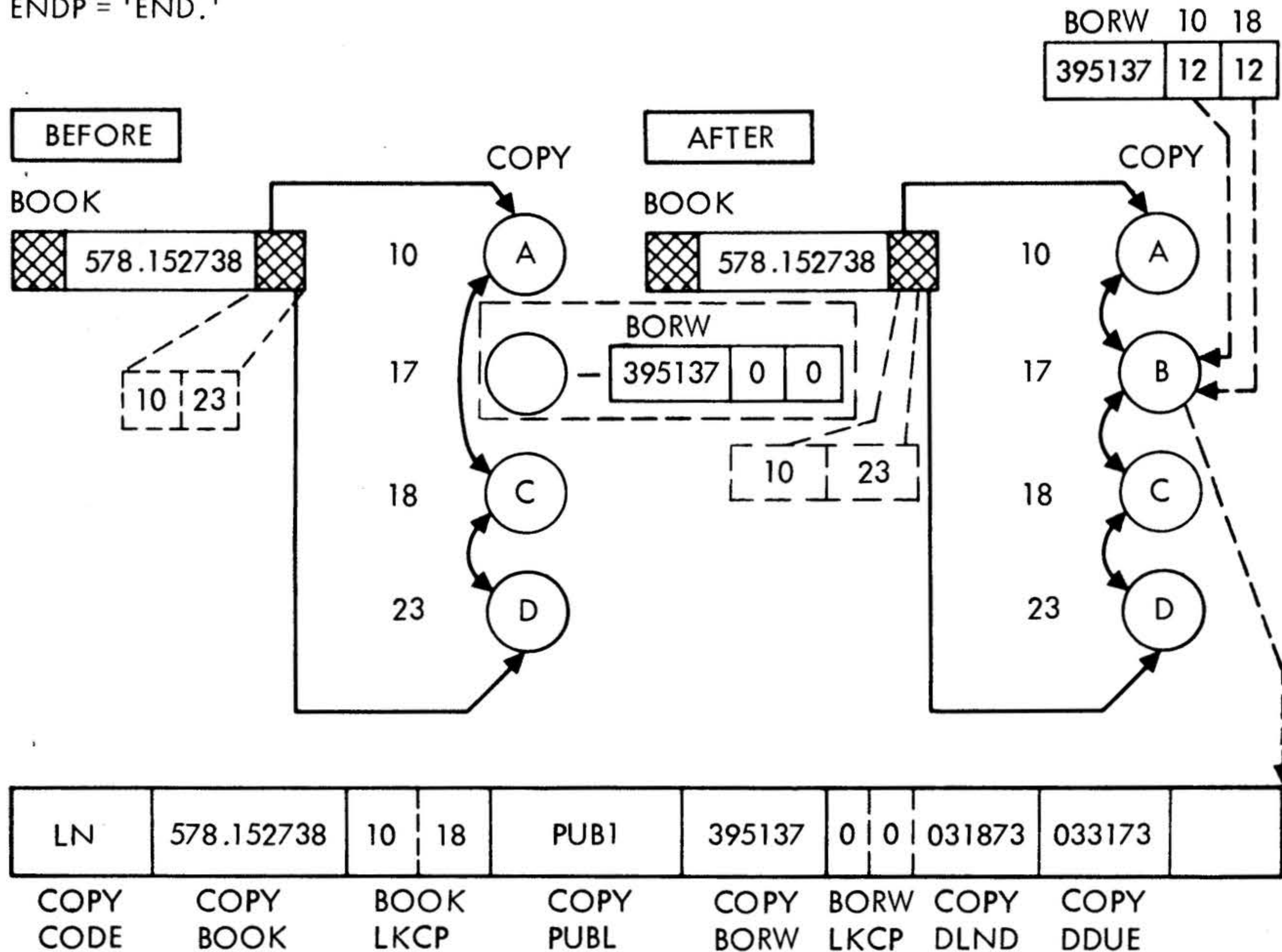
Figure 5-8. The ADD VARIABLE AFTER Function



DATA MANAGEMENT LANGUAGE

ADDVA - TWO MASTERS

FUNCTION = 'ADDVA'
 STATUS = blanks
 DATA-SET = 'COPY'
 REFERENCE = '0010' (binary)
 LINKAGE-PATH = BOO KLKCP
 CONTROL-KEY = '578.152738'
 DATA-LIST = 'COPYCODECOPYBOOKCOPYBORWCOPYDLNDCOPYDDUECOPYPUBLND.
 COPYBLEND.'
 DATA-AREA = '578.152738395137LNO31873033173PUB1'
 ENDP = 'END.'



VT11-3471

Figure 5-9. The ADD VARIABLE AFTER Function for Two Master Data Sets



BCTL: A control field contains blanks or zeros.
FULL: The data set has no room for this record.
IVRC: The record code in a record pointed to on a coded linkage path has not been defined in the DBMOD.
MLNF: The linkage path name is invalid for either the data set or the record code.
MRNF: A master record does not exist for a control field.
UCTL: The Control-Key does not match the corresponding field in the Data Area.

REFERENCE: Internal Reference Point

This parameter normally specifies the Internal Reference Point of the record before which the user's record is to be added.

If the Reference Field contains any other value, the function will not be performed and a status code of IVRP will be returned.

When the function has successfully completed, the Reference Field contains the Internal Reference Point of the added record.

Programming Considerations:

- a. Addition of the record along secondary linkage paths requires retrieving a master record for each Control Key defined for this record. Therefore, these Control Keys must be present in the Data Area and must be valid, i.e., they must represent records which actually exist in the respective master data sets.
- b. Data elements not named in the Data List will be zeros in the record.

An example of the Add Variable Before function is given in figure 5-10. Note that the new record B is added before record C, because its Internal Reference Point is 0002, and this is the value given in the Reference Field.

5.2.4 The Add Variable Continue Function

This function operates by logically adding the record in the Data Area to the end of all linkage paths defined for the record in the Data Base Descriptor.

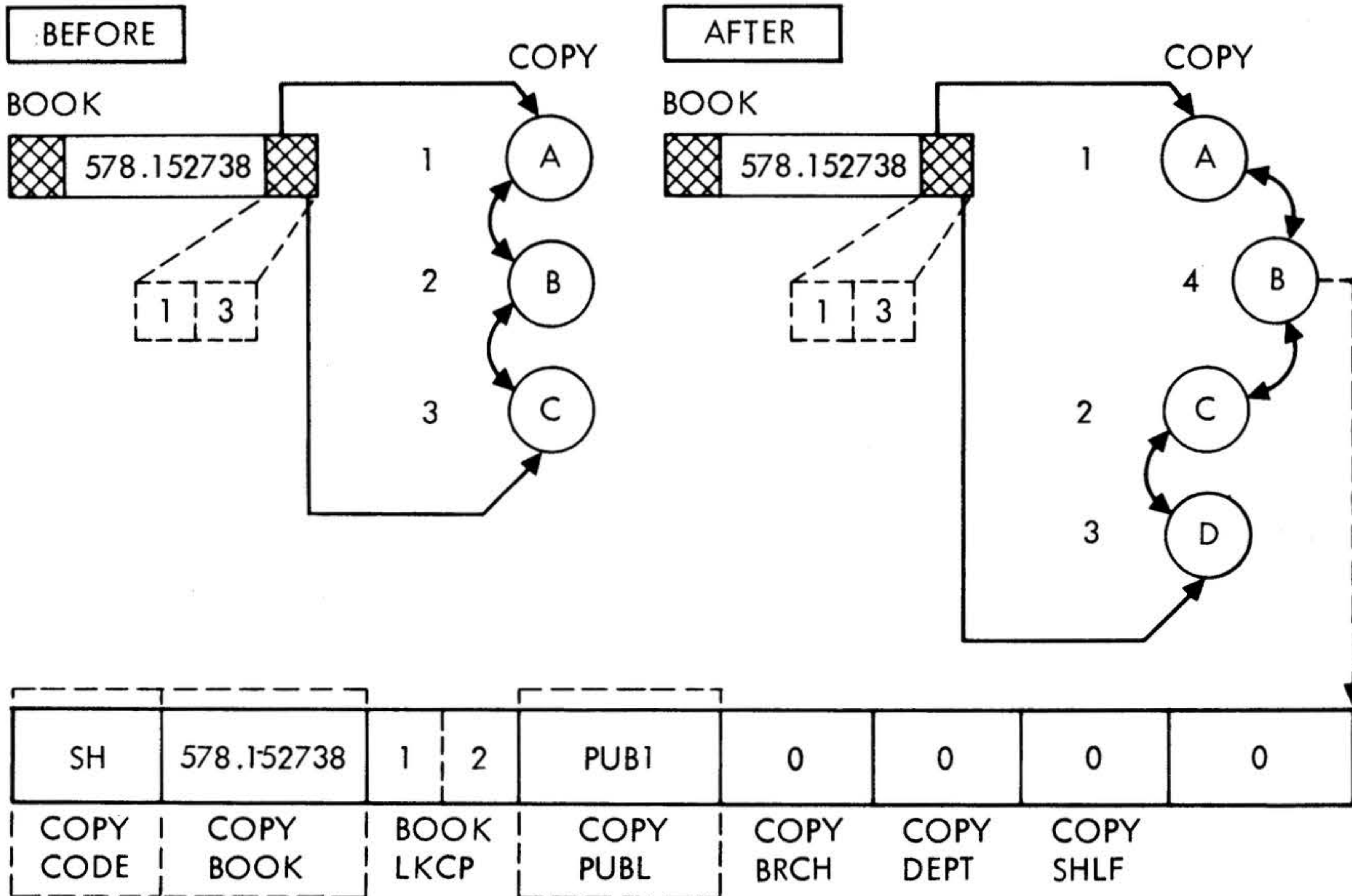


DATA MANAGEMENT LANGUAGE

ADDVB	ADD VARIABLE BEFORE FUNCTION
-------	------------------------------

EXAMPLE:

FUNCTION = 'ADDVB'
 STATUS = blanks
 DATA-SET = 'COPY'
 REFERENCE = '0002' (binary)
 LINKAGE-PATH = 'BOOKLKCP'
 CONTROL-KEY = '578.152738'
 DATA-LIST = 'COPYCODECOPYBOOKCOPYPUBLEND.'
 DATA-AREA = '578.152738PUB1 SH'
 ENDP = 'END.'



VTI1-3472

Figure 5-10. The ADD VARIABLE BEFORE Function

**Required Parameters:**

ADDVC, STATUS, DATA-SET, REFERENCE, LINKAGE-PATH,
CONTROL-KEY, DATA-LIST, DATA-AREA, ENDP

ADDVC: Add Variable Continue Function Mnemonic

The user must insert this mnemonic into the Operation Field.

STATUS: Status Codes

Significant status codes which may be returned are:

BCTL: A control field contains blanks.

FULL: The data set has no room for this record.

IVRC: The record code in a record pointed to on a coded linkage path has not been defined in the DBMOD.

MLNF: The linkage path name is invalid for either the data set or the record code.

MRNF: A master record does not exist for a control field.

UCTL: The Control-Key Field does not match the corresponding field in the Data area.

REFERENCE: Internal Reference Point

The contents of the Reference Field are not used to perform the function, but are edited for validity.

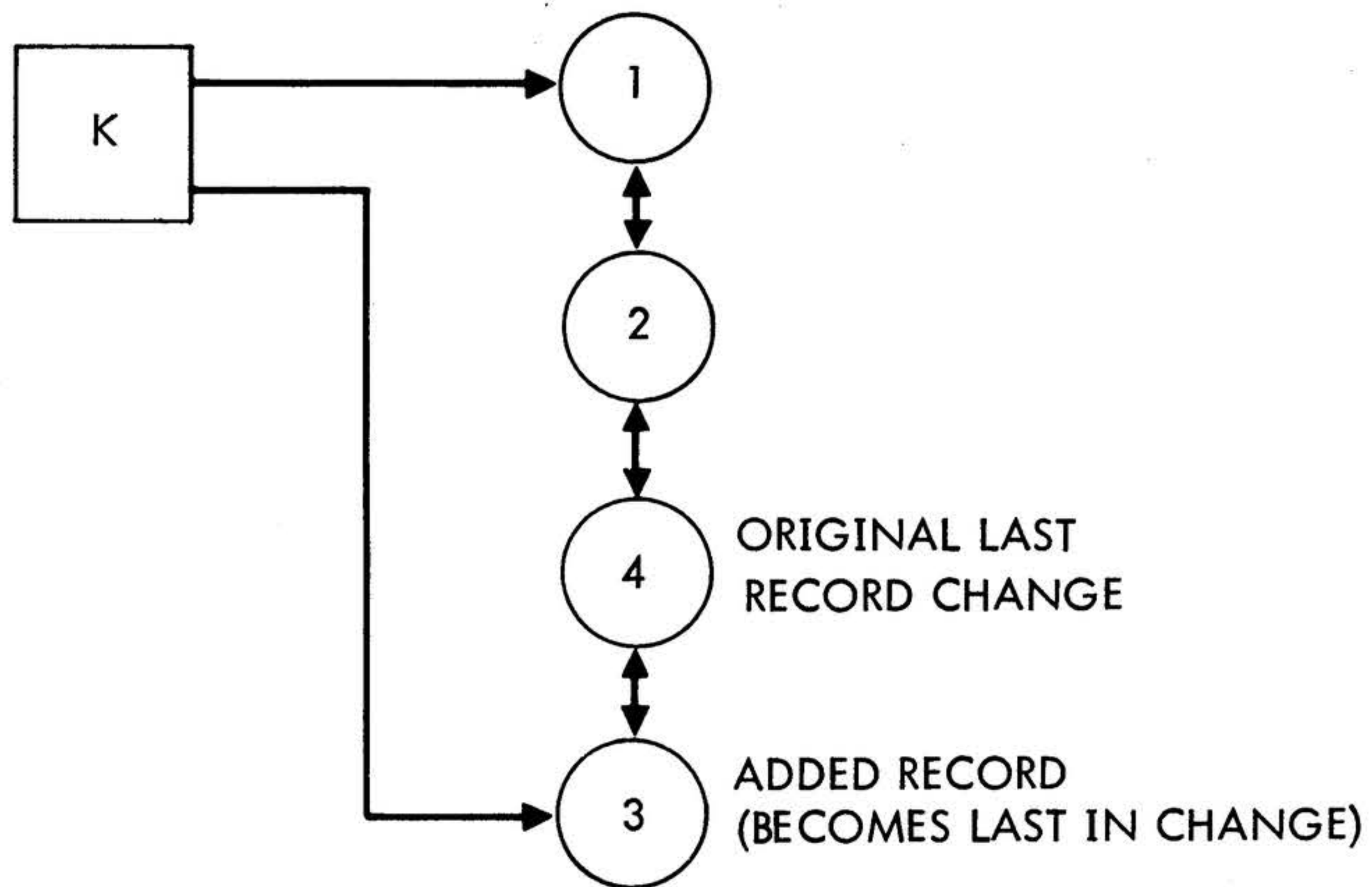
If the Reference Field contains anything but LKxx or an Internal Reference Point, the function will not be performed and a status code of IVRP will be returned.

When the function has successfully completed, the Reference Field contains the Internal Reference Point of the added record.

Programming Considerations:

- a. Addition of the record along secondary linkage paths requires retrieving a master record for each Control Key defined for this record. Therefore, these Control Keys must be present in the Data Area and must be valid, i.e., they must represent records which actually exist in the respective master data sets.
- b. Data elements not named in the Data List will be zeros in the record.

The Add Variable Continue function is shown schematically in figure 5-11.



VTI1-3489

Figure 5-11. The ADD VARIABLE CONTINUE Function



5.2.5 The Delete Master Function

This function operates by randomizing on the contents of the Control-Key Field to find the specified record. The record is deleted by setting it to zeros. The space thus freed is made immediately available to be re-used.

Required Parameters:

DEL-M, STATUS, DATA-SET, CONTROL-KEY, DATA-LIST,
DATA-AREA, ENDP

DEL-M: Delete Master Function Mnemonic

The user must place this mnemonic into the Operation Field.

STATUS: Status Code

Significant status codes which may be returned are:

BCTL: The Control-Key Field contains blanks.

IMDL: Variable records are still linked to this master record.

MRNF: The requested record is not on the data set.

Programming Considerations:

- a. Although not necessary, the record to be deleted should be read and verified against user criteria before actually deleting.
- b. TOTAL will not delete a master record if any variable records remain linked to it. The user must first physically delete each variable record.

An example of the Delete Master function and a graphic illustration of randomizing of the contents of the Control Key field to find the specified record is given in figure 5-12.

5.2.6 The Delete Variable Direct Function

This function operates by deleting the variable record whose Internal Reference Point is in the Reference Field. The Record is deleted by setting it to zeros.



DATA MANAGEMENT LANGUAGE

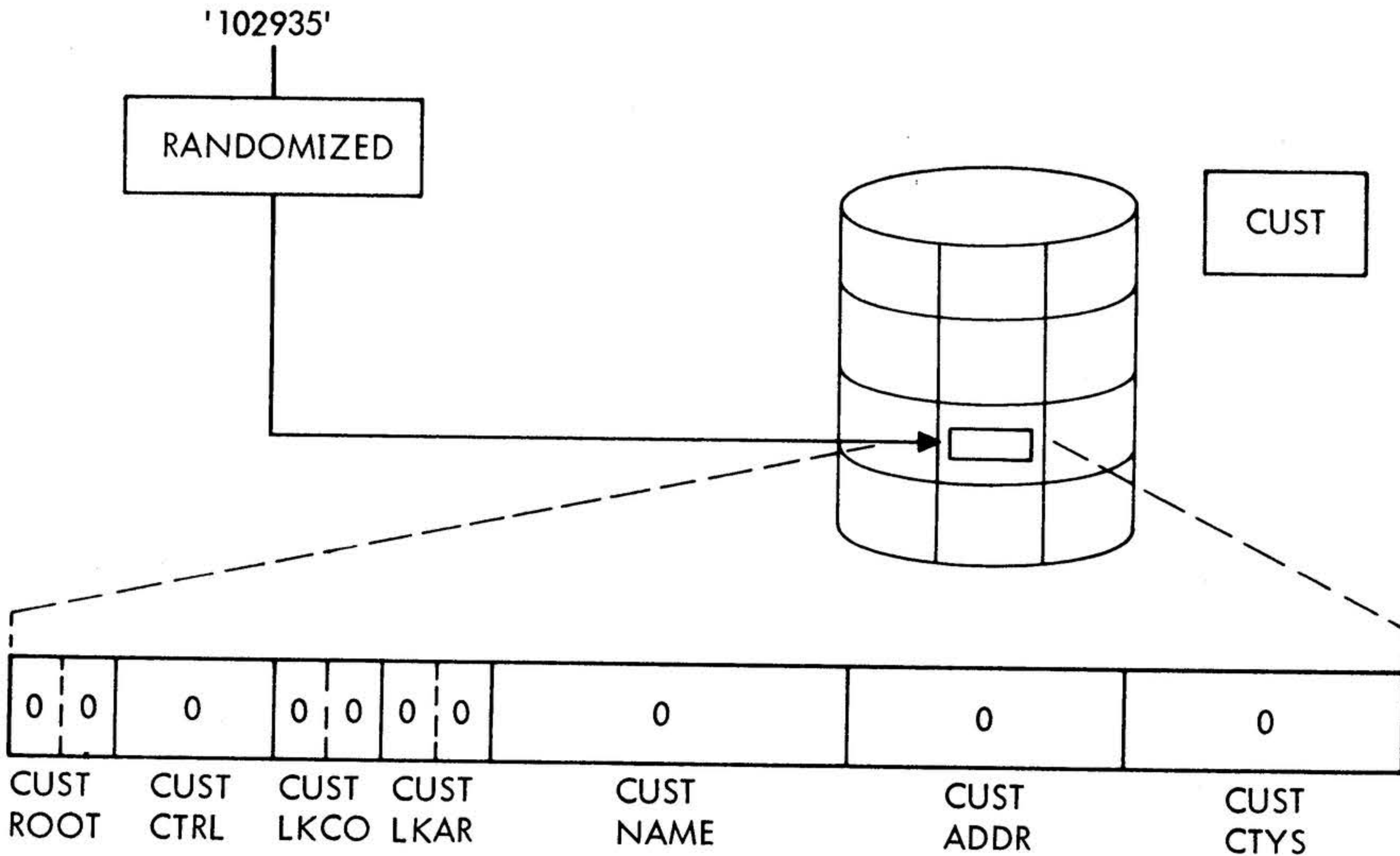
DEL-M	DELETE MASTER FUNCTION
-------	------------------------

EXAMPLE:

FUNCTION = 'DEL-M'
 STATUS = blanks
 DATA-SET = 'CUST'
 CONTROL-KEY = '102935'
 DATA-LIST = 'END.'
 DATA-AREA = zero
 ENDP = 'END.'

AFTER THE FUNCTION COMPLETES

RECORD HAS BEEN ZEROED OUT



VTI1-3473

Figure 5-12. The DELETE MASTER Function

Required Parameters:

DELVD, STATUS, DATA-SET, REFERENCE, LINKAGE-PATH,
CONTROL-KEY, DATA-LIST, DATA-AREA, ENDP

DELVD: Delete Variable Direct Function Mnemonic

The user must insert this mnemonic into the Operation Field.

STATUS: Status Code

Significant status codes which may be returned are:

IVRC: The record code in a record pointed to on a coded linkage path has not been defined in the DBMOD.

MLNF: The linkage path name is invalid for the file or record code.

REFERENCE: Internal Reference Point

The contents of the Reference Field are used to point to the specific record to be deleted. Therefore, it may contain only an Internal Reference Point.

When the function has successfully completed, the Reference Field will contain the "back pointer" from the deleted record.

Programming Considerations:

If the DELVD command is followed immediately by a READV command, the record logically after the deleted record is retrieved. If the DELVD command is followed immediately by a READR command, the record logically before the deleted record is skipped and the next preceding record is retrieved.

An example of the Delete Variable Direct function is given in figure 5-13. Note that record B is deleted and is set to zeros, because its Internal Reference Point is 0017, and this is the value given in the Reference Field.

5.2.7 The Read Next Function

This function operates as a generalized serial retrieval method. The retrieval may be directed to a specific point in the data set, namely, to the beginning or to a specific record location.

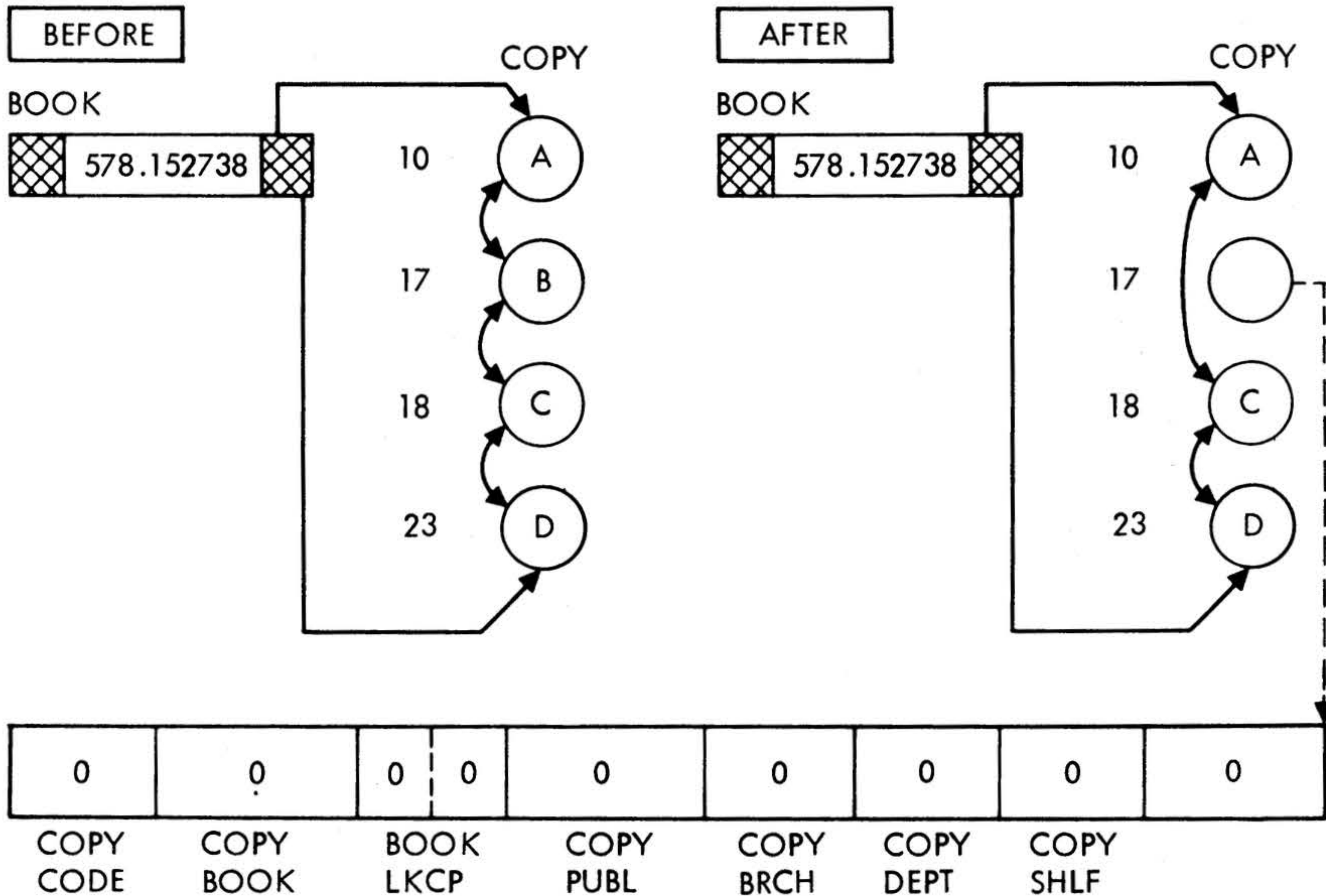


DATA MANAGEMENT LANGUAGE

DELVD	DELETE VARIABLE DIRECT FUNCTION
-------	---------------------------------

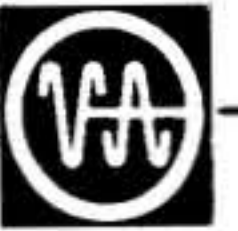
EXAMPLE:

FUNCTION = 'DELVD'
 STATUS = blanks
 DATA-SET = 'COPY'
 REFERENCE = '0017' (binary)
 LINKAGE-PATH = 'BOOKLKCP'
 CONTROL-KEY = '578.152738'
 DATA-LIST = 'END.'
 DATA-AREA = zero
 ENDP = 'END.'



VTI1-3474

Figure 5-13. The DELETE VARIABLE DIRECT Function



Each record thus retrieved is placed in the Data Area and retrieval may continue by simply re-executing the Read Next Command until the end of the data set is reached. Only data records are returned to the user; blank records and control records are bypassed. The specific method of retrieval is described in more detail within the discussion of the function.

Required Parameters:

RDNXT, STATUS, DATA-SET, QUALIFIER, DATA-LIST,
DATA-AREA, ENDP

RDNXT: Read Next Function Mnemonic

The user must place this mnemonic into the Operation Field.

QUALIFIER: Relative Record Number Field

This parameter is the name of (points to) a field defined by the user which is used to maintain the current position in the data set being processed. The Qualifier Field function is similar to the Reference field in other operations.

The size and content of the Qualifier Field is always four bytes in length.

The Qualifier Field may contain:

```
[ BEGN ]  
[ rrrr ]  
[ END. ]
```

BEGN:

If the user places "BEGN" into the Qualifier Field, the Read Next Function retrieves the record physically first in the data set and places it into the Data Area according to the Data List. The Internal Reference Point (relative record number) replaces "BEGN" in the Qualifier Field. Subsequent executions of the Read Next Command will then continue processing serially from that point.



rrrr: Binary Relative Record Number (rrn)

The Qualifier Field may contain an Internal Reference Point as the result of a prior execution of the Read Next Command, or the user may place an Internal Reference Point into the Qualifier Field to resume the retrieval from some specific location. Since it is the Internal Reference Point of the last record read, it is incremented by one to retrieve the next record.

END.: "END." is placed in the Qualifier Field at end-of-file.

Programming Considerations:

- a. The "WRITM" Command may be executed while processing master data sets with the Read Next Function.
- b. The "WRITV" and "DELVD" Commands may be executed while processing variable data sets with the Read Next Function. The Reference Numbers for these commands may be found in the Qualifier Field.
- c. At end-of-file, "END." is placed in the Qualifier Field.
- d. The code-directed read feature does not apply to this function.

An example of the Read Next function is given in figure 5-14. In this example, seven logical records with Internal Reference Point (relative record number) values of 58, 95, 147, 196, 197, 201, 213 are retrieved from the data base, and their contents are placed in the data area specified by the application program.

5.2.8 The Read Direct Function

This function operates by directly retrieving the logical record specified by the Internal Reference Point in the Reference Field. Processing may then continue along any linkage path valid for that record with whatever function is appropriate.



DATA MANAGEMENT LANGUAGE

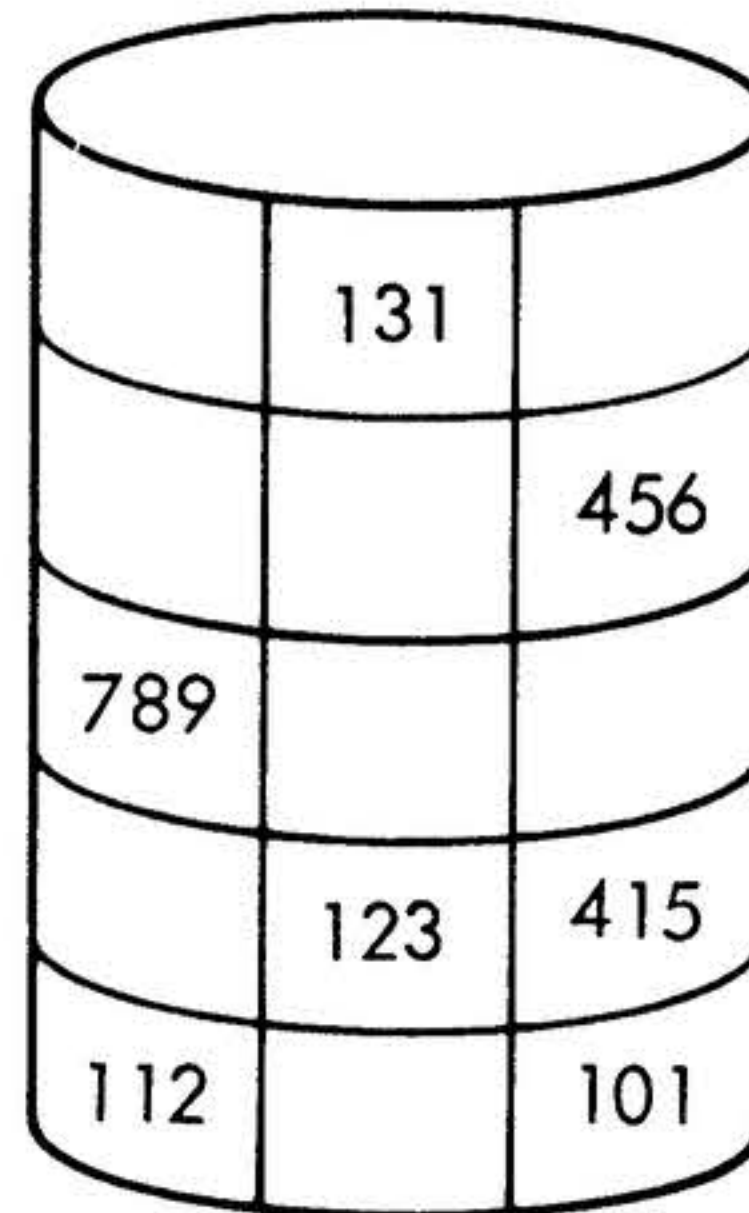
RDNXT	READ NEXT FUNCTION
-------	--------------------

EXAMPLE:

FUNCTION = 'RDNXT'
 STATUS = blanks
 DATA-SET = 'BOOK'
 QUALIFIER = '00581' (binary)
 DATA-LIST = 'BOOKCTRLBOOKDATAEND.'
 DATA-AREA = zero
 ENDP = 'END.'

EXAMPLE:

		KEY	FIELD
rrn	58	131	DATA
rrn	95	456	DATA
rrn	147	789	DATA
rrn	196	123	DATA
rrn	197	415	DATA
rrn	201	112	DATA
rrn	213	101	DATA
	END		



SINGLE ENTRY
OR
VARIABLE ENTRY

- ELEMENTS EXTRACTED ONLY FROM ACTIVE RECORDS
- RECORDS ARE EXTRACTED IN PHYSICAL SEQUENCE WITHOUT REGARD FOR KEY OR LINKAGES

VTI1-3475

Figure 5-14. The READ NEXT Function



Required Parameters:

READD, STATUS, DATA-SET, REFERENCE, LINKAGE-PATH,
CONTROL-KEY, DATA-LIST, DATA-AREA, ENDP

READD: Read Direct Function Mnemonic

The user must insert this mnemonic into the Operation Field.

STATUS: Status Code

Significant status codes which may be returned are:

IVRP: The contents of the Reference Field are invalid.

IVRC: The Data-List contains element names invalid for
the record code.

REFERENCE: Internal Reference Point

The Internal Reference Point in the Reference Field is used to retrieve a specific record from the variable data set; therefore, it may contain only an Internal Reference Point.

When the function has successfully completed, the Reference Field will still contain the Internal Reference Point of the retrieved record.

Programming Considerations:

It is not anticipated that the user will be able to programmatically compute the Internal Reference Point of a specific record. An Internal Reference Point supplied to the READD function is likely to have come from one of two sources:

- a. the currently executing program which saved the Internal Reference Point when interrupting continuous processing along a chain and is now preparing to resume processing by retrieving the last record read.
- b. another program, through some input medium.

An example of coding the Read Direct function is given in figure 5-15. In this example, the contents of record B are transferred to the Data Area specified by the application program, because the Internal Reference Point value of record B is 0017 and this is the value given in the Reference field.



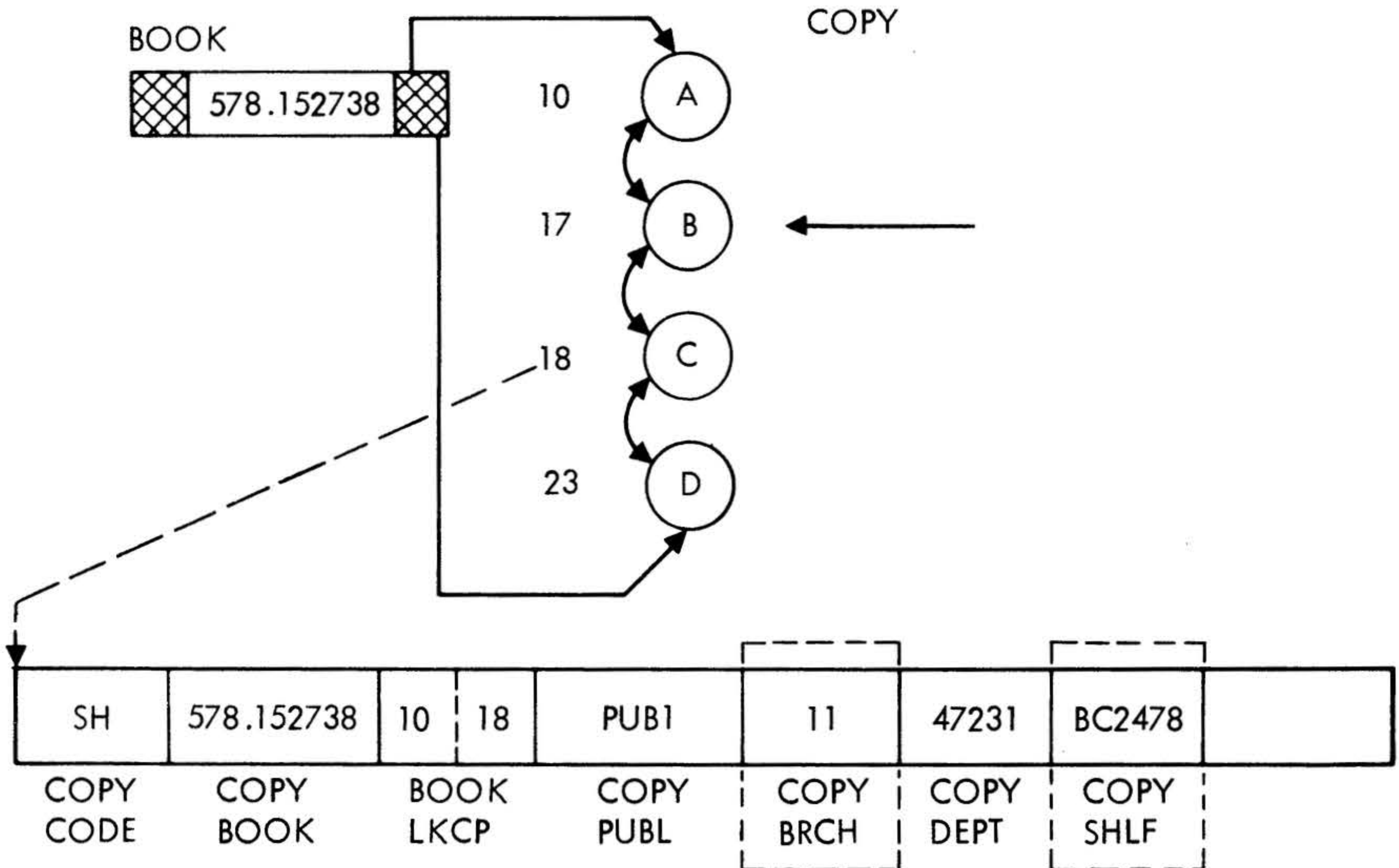
DATA MANAGEMENT LANGUAGE

READD	READ VARIABLE DIRECT FUNCTION
-------	-------------------------------

EXAMPLE:

FUNCTION = 'READD'
 STATUS = blanks
 DATA-SET = 'COPY'
 REFERENCE = '0017' (binary)
 LINKAGE-PATH = 'BOOKLKCP'
 CONTROL-KEY = '578.152738'
 DATA-LIST = 'COPYBRCHCOPYSHLFEND.'
 DATA-AREA = zero
 ENDP = 'END.'

AFTER THE CALL COMPLETES
 DATA-AREA = '11BC2478'



VT11-3476

Figure 5-15. The READ DIRECT Function



5.2.9 The Read Master Function

This function operates by randomizing on the contents of the Control Key Field to find the specific record and place it into the Data Area according to the Data List.

Required Parameters:

READM, STATUS, DATA-SET, CONTROL-KEY, DATA-LIST,
DATA-AREA, ENDP

READM: Read Master Function Mnemonic

The user must place this mnemonic into the Operation Field.

STATUS: Status Code

Significant status codes which may be returned are:

BCTL: The control Key Field contains blanks.

MRNF: The requested record is not on the data set.

An example of coding the Read Master function is given in figure 5-16. In this example, the record with the Control Key value 102935 is found. The fields CUSTNAME, CUSTADDR, and CUSTCTY are transferred to the Data Area specified by the application program.

5.2.10 The Read Reverse Function

This function operates by logically following backpointers along a specified linkage path. To read an entire chain, processing is initiated by placing LKxx into the Reference Field and issuing the READR command. TOTAL uses the Control Key to access a master record from which the pointer to the logical end of the chain is obtained. This last record of the chain is then returned to the user. Thereafter, processing continues by re-issuing the READR command; since the Reference Field contains an Internal Reference Point, the back chain is followed and records are retrieved in reverse order until the first record in the chain has been processed. When the READR command is issued for the record before the first, the code "END." is returned in the Reference Field to indicate that the chain has been completely processed.



DATA MANAGEMENT LANGUAGE

READM	READ MASTER FUNCTION
-------	----------------------

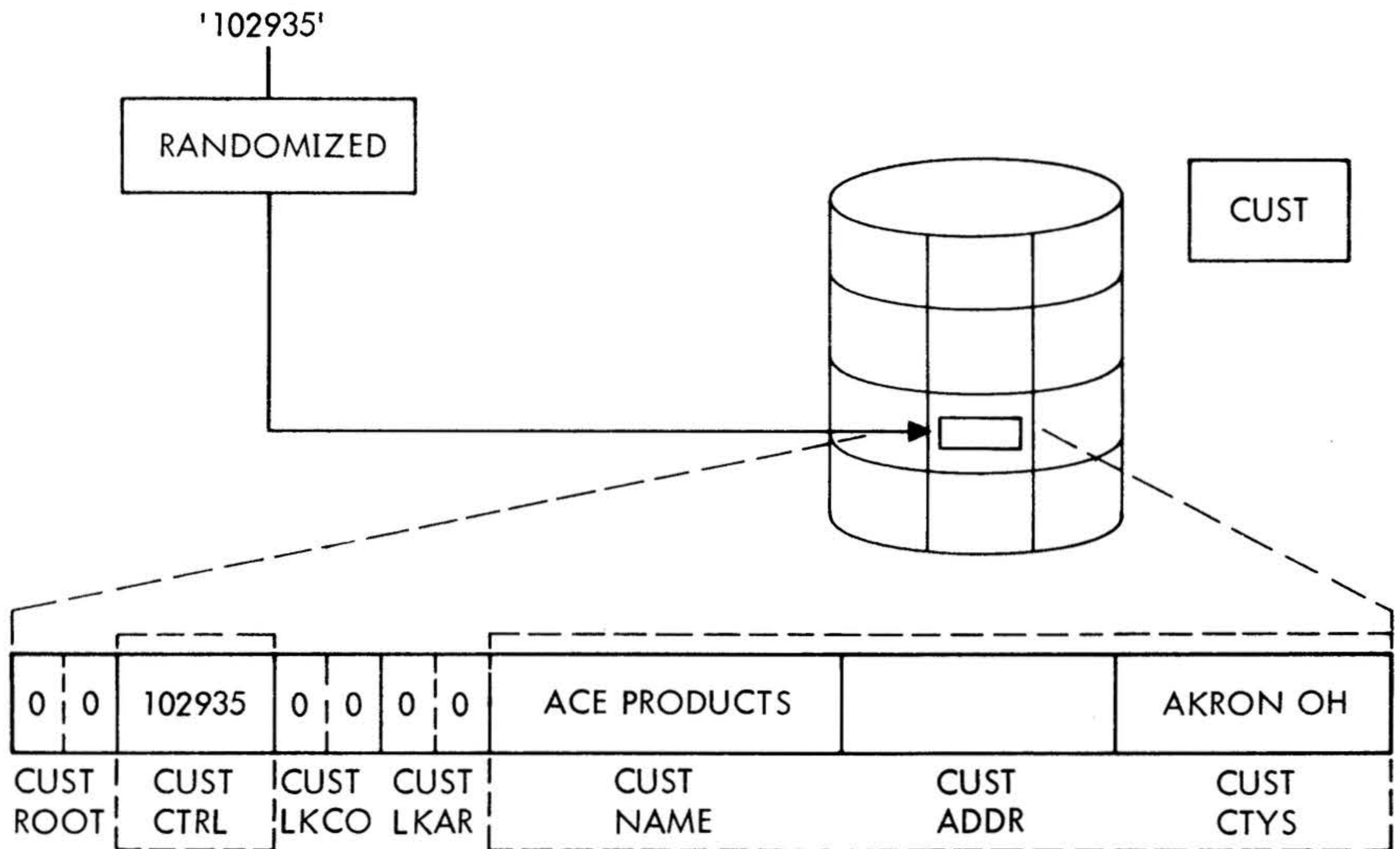
EXAMPLE:

```

FUNCTION = 'READM'
STATUS = blanks
DATA-SET = 'CUST'
CONTROL-KEY = '102935'
DATA-LIST = 'CUSTNAMECUSTADDRCUSTCTYSEND.'
DATA-AREA = zero
ENDP = 'END.'
    
```

AFTER THE FUNCTION COMPLETES

DATA-AREA = 'ACE PRODUCTS AKRON OHIO'



VTI1-3477

Figure 5-16. The READ MASTER Function



Required Parameters:

READR, STATUS, DATA-SET, REFERENCE, LINKAGE-PATH,
CONTROL-KEY, DATA-LIST, DATA-AREA, ENDP

READR: Read Reverse Function Mnemonic

The user must insert this mnemonic into the Operation Field.

STATUS: Status Code

Significant status codes which may be returned are:

BCTL: A Control Field contains blanks.

MLNF: The specified linkage path name is invalid for
the file or the record code.

MRNF: The related master record cannot be found.

REFERENCE: Internal Reference Point

If the Reference Field contains LKxx, TOTAL retrieves the record logically last in the chain. If the Reference Field contains an Internal Reference Point, TOTAL uses it to point to the record from which to obtain the back pointer to the next record.

When the function has successfully completed, the Reference Field contains the Internal Reference Point of the record just read.

When the logical beginning of the chain has been processed, "END." is contained in the Reference Field.

Programming Considerations:

- a. Care must be exercised in performing deletes when processing along a chain in reverse. Since the Reference Field contains the back pointer from the deleted record after the delete is executed, the next READR command will skip the deleted record after the delete is executed, the next READR command will skip the deleted record in the chain.
- b. When the chain has been completely processed and the Reference Field contains "END.", it should be cleared before executing another variable function to avoid getting the IRLC status code.



An example of the Read Variable Reverse function is given in figure 5-17. In this example, the value LKCP in the Reference Field enables TOTAL to use the Control Key 578.152738 to access the Master Record BOOK, from which the pointer to the logical end of the chain record D is obtained. The data in record D is then read into the Data Area specified by the application program and is followed by the data in records C, B, and A.

5.2.11 The Read Variable Function

This function operates by logically following forward pointers along a specified linkage path. To read an entire chain, processing is initiated by placing LKxx into the Reference Field and issuing the READV command. TOTAL uses the Control Key to access a master record from which the pointer to the logical beginning of the chain is obtained. This first record of the chain is then returned to the user. Thereafter, processing continues by re-issuing the READV command; since the Reference Field contains an Internal Reference Point, the forward chain is followed and records are retrieved in forward order until the last record in the chain has been processed. When the READV command is issued for the record after the last, TOTAL returns END. in the Reference Field to indicate that the chain has been completely processed.

Required Parameters:

READV, STATUS, DATA-SET, REFERENCE, LINKAGE-PATH, CONTROL-KEY, DATA-LIST, DATA-AREA, ENDP
--

READV: Read Variable Function Mnemonic

The user must insert this mnemonic into the Operation Field.

STATUS: Status Code

Significant status codes which may be returned are:

BCTL: A Control Field contains blanks.

MLNF: The specified linkage path name is invalid for the file or the record code.

IVRC: The record code in a record pointed to on a coded linkage path has not been defined in the DBMOD.

MRNF: The related master record cannot be found.

REFERENCE: Internal Reference Point



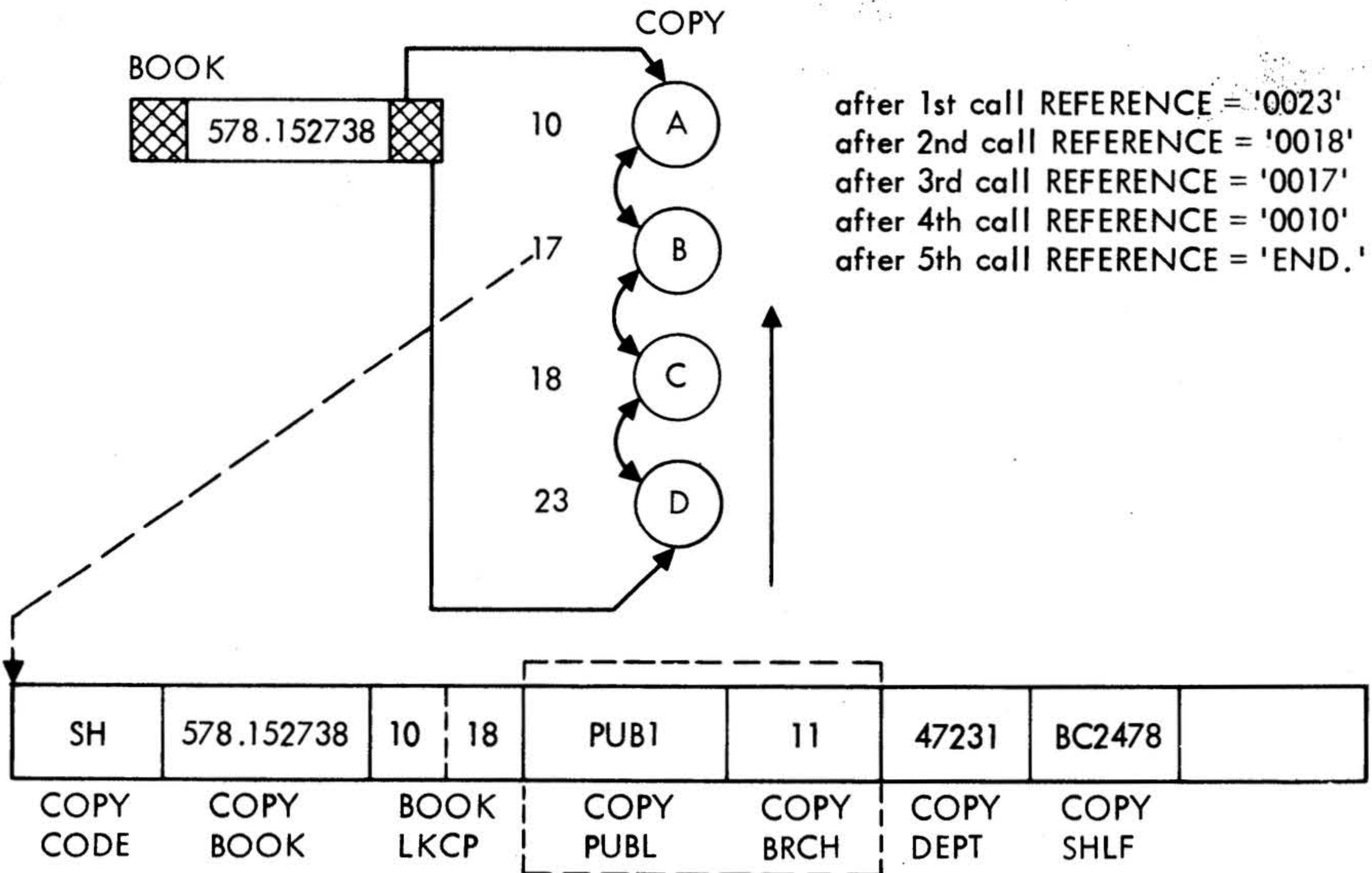
DATA MANAGEMENT LANGUAGE

READR	READ VARIABLE REVERSE FUNCTION
-------	--------------------------------

EXAMPLE:

FUNCTION = 'READR'
 STATUS = blanks
 DATA-SET = 'COPY'
 REFERENCE = 'LKCP'
 LINKAGE-PATH = 'BOOKLKCP'
 CONTROL-KEY = '578.152738'
 DATA-LIST = 'COPYBRCHCOPYPUBLEND.'
 DATA-AREA = zero
 ENDP = 'END.'

AFTER THE THIRD CALL COMPLETES
 DATA-AREA = '11 PUBL'



VTI1-3478

Figure 5-17. The READ VARIABLE REVERSE Function



If the Reference Field contains LKxx, TOTAL retrieves the record logically first in the chain. If the Reference Field contains an Internal Reference Point, TOTAL uses it to point to the record from which to obtain the forward pointer to the next record.

When the function has successfully completed, the Reference Field contains the Internal Reference Point of the record just read.

When the logical end of the chain has been processed, "END." is contained in the Reference Field.

Programming Considerations:

When the chain has been completely processed and the Reference Field contains "END.", it should be cleared before executing another variable function to avoid getting the IVRP status code.

An example of coding the Read Variable function is given in figure 5-18. In this example, the value LKCP in the Reference Field enables TOTAL to use the Control Key 578.152738 to access the Master Record BOOK, from which the pointer to the logical beginning of the chain record A is obtained. The data in record A is then read into the Data Area specified by the application program and is followed by the data in record B, C, and D.

5.2.12 The Request Location Function

This function operates by randomizing on the contents of the Control Key Field for the specified master data set, and places the resultant Internal Reference Point in the Data Area. No I/O operations are performed.

Using this function, the user may, among other things, build a file of Control Keys and their respective Internal Reference Points. This file may be sorted on internal Reference Point and the data then processed at maximum speed against the data set.

Required Parameters:

RQLOC, STATUS, DATA-SET, CONTROL-KEY, DATA-AREA, ENDP
--



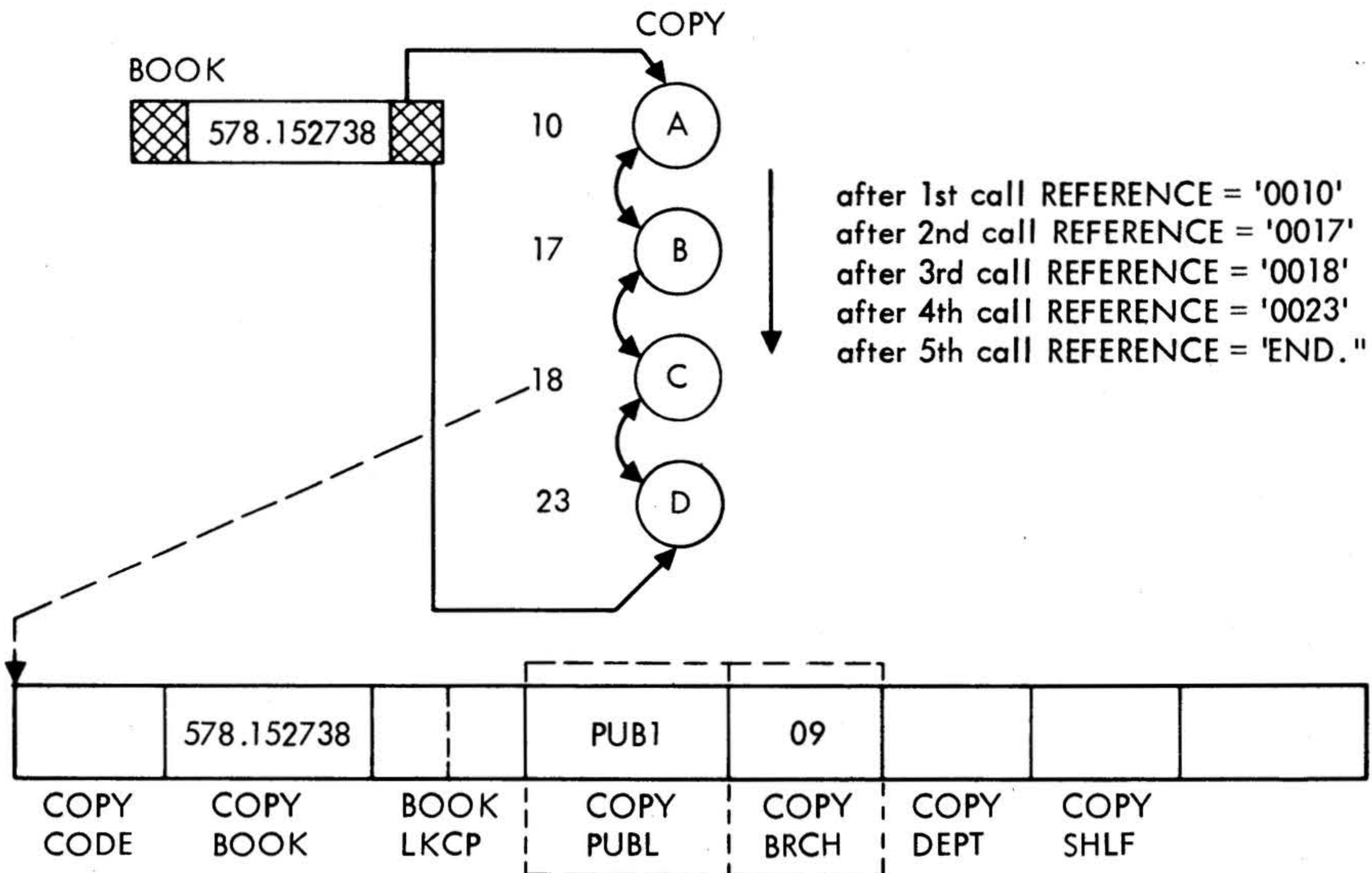
DATA MANAGEMENT LANGUAGE

READV	READ VARIABLE FUNCTION
-------	------------------------

EXAMPLE:

FUNCTION = 'READV'
 STATUS = blanks
 DATA-SET = 'COPY'
 REFERENCE = 'LKCP'
 LINKAGE-PATH = 'BOOKLKCP'
 CONTROL-KEY = '578.152738'
 DATA-LIST = 'COPYBRCHCOPYPUBLEND.'
 DATA-AREA = zero

AFTER THE THIRD CALL COMPLETES
 DATA-AREA = '09PUB1'



VTI1-3479

Figure 5-18. The READ VARIABLE Function



RQLOC: Request Location Function Mnemonic

The user must place this mnemonic into the Operation Field.

STATUS: Status Code

Significant status codes which may be returned are:

FNTF: The requested file cannot be found in the Data Base Descriptor.

Programming Considerations:

- a. The specified data set must be a master data set.
- b. The data set need not be opened.
- c. The data area is defined as four bytes in length.

5.2.13 The Sign-Off Function

This function operates by physically closing any data sets which remain open.

Required Parameters:

SINOF, STATUS, SCHEMA, ENDP

SINOF: Sign-Off Function Mnemonic

The user must place this mnemonic into the Operation Field.

Programming Considerations:

- a. All subsequent commands except a Sign-On will return a status code of NOSO.
- b. A new Sign-On Command may specify the same or different options than the previous Sign-On.
- c. The SINOF format is identical to SINON except for the function.



- d. This should be the last statement, logically, in a user program prior to termination.
- e. For an explanation of SCHEMA, refer to the SINON function.
- f. Since SINOF will be the last logical TOTAL function performed in an application program, any decision process involving a new "****" status return would be limited to a message displaying the error status along with the individual status for each file within the SCHEMA.
- g. SINOF automatically signs off for all files opened for this run.

5.2.14 The Sign-On Function

This function must be the first CALL to the TOTAL System presented by a user program. It also allows the user program to state what data sets this program is to process, what mode of access this program has, and the type of file needed by the program.

Required Parameters:

SINON, STATUS, SCHEMA, ENDP

SINON: Sign-On Function Mnemonic

The user must place this mnemonic into the Operation Field.

STATUS: Status Code

Significant status codes which may be returned are:

EXSO: This sign-on was preceded by another sign-on without an interim SINOF.

DUPO: This file has already been opened. Fatal Condition.

SCHEMA: Explicit options and files needed for this program run.

This parameter "points to" a field defined by the user in the following format and containing all below stated values:



- a. Program Name: 8-character program name of this program (not to be confused with VORTEX task name).
- b. Data Base:
Descriptor Name: 6-character DBMOD name
- c. Access Mode: 6-character field containing general intention of this program:

[RDONLY
UPDATE]

RDONLY: Only read functions will
be permitted.
UPDATE: Entire set of DML functions
available.

- d. Logging option: 2-character field containing option to log or not to log

{NL}
{LG}

Only NL can be coded.

- e. Realm: A group of 12-character entries for each data set in the data base required for this program and terminated by "END." literal.
 - 1. File name: 4-character field containing a name of a data set as in the DBMOD.
 - 2. File mode: 4-character field containing the mode of file sharing needed.

[SHRE
PRIV
OPEN]

SHRE: This file may be shared among
concurrent programs. (read
only).
PRIV: This file is exclusively assigned
to this program and no other program
may have access to it during any
program run. (UPDATE).
OPEN: This option is for debug only. This
option ignores the file status if
already opened, i.e., it allows to
reopen already locked files.



Note: This option must be used very cautiously because files may be locked by other tasks.

3. File status: 4-character field used for unique file status at OPEN time. The file status value is put in by TOTAL.

Programming Considerations:

- a. A Sign-On must be the first TOTAL command executed.
- b. A second Sign-On may be issued after a Sign-Off, e.g., to change access mode, etc.
- c. If any of the status fields used in the REALM entry are not '****', then the general status will contain the proper error indicator. Checking of each REALM status is not required.

An example of the SINON function is given in figure 5-19, showing typical entries for the SCHEMA and REALM parameters. In the example, the RDNLY entry in the SCHEMA parameter ensures that only read functions will be permitted for the program INVOICES, and the three SHRE entries in the REALM field allow the data-sets CUST, ORNM, and CORD to be shared among concurrent programs in the Read Only mode.

5.2.15 The Write Master Function

This update function operates by randomizing on the contents of the Control Key Field to retrieve the record to be updated. The data elements in the Data Area are moved to the record which is then rewritten.

Required Parameters:

WRITM, STATUS, DATA-SET, CONTROL-KEY, DATA-LIST,
DATA-AREA, ENDP

WRITM: Write Master Function Mnemonic

The user must place this mnemonic into the Operation Field.



DATA MANAGEMENT LANGUAGE



EXAMPLE:

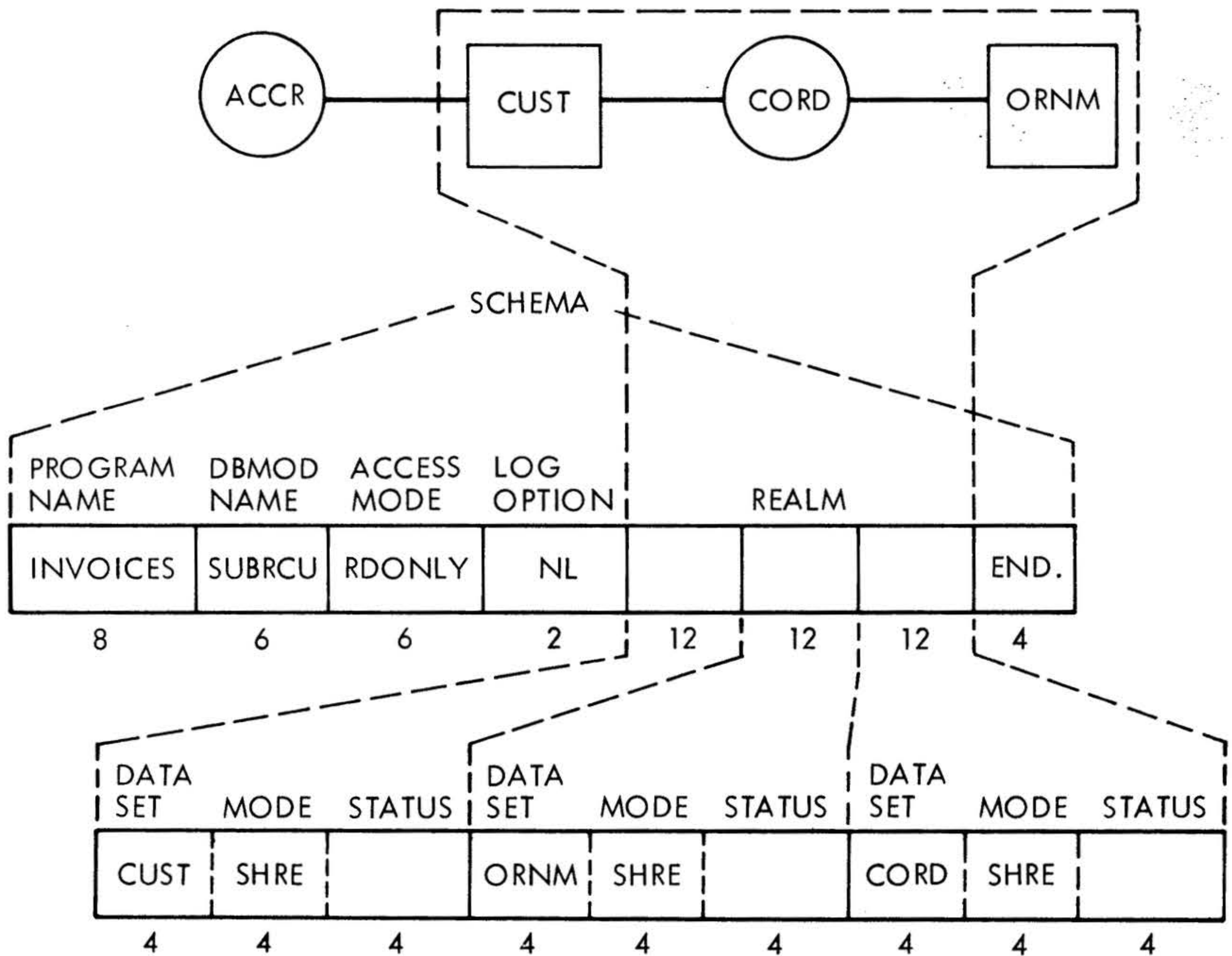
PROGRAM NAME=INVOICES

DBMOD NAME=SUBRCU

NO UPDATING WILL BE MADE DURING THE RUN

WILL ONLY USE DATA SETS: CUST, CORD, AND ORNM

THE LAYOUT OF THE AREA WOULD BE AS SHOWN.



VT11-3480

Figure 5-19. The SIGN-ON Function, showing the use of the SCHEMA and REALM Parameters



STATUS: Status Code

Significant status codes which may be returned are:

- BCTL: The Control Key Field contains blanks.
- MRNF: The specified record is not on the data set.
- UCTL: The contents of the Control Key Field do not match the corresponding field in the Data Area.

Programming Considerations:

The Control Key Field may not be changed.

An example of the coding for the Write Master function is given in figure 5-20. In the example shown, TOTAL randomizes on the contents of the Control Key field 102935 to retrieve the record CUST. The data 5023 MAPLE from the application program's Data Area is then written into the data element field CUSTADDR in the record CUST, as specified by the parameter DATA-LIST.

5.2.16 The Write Variable Function

This update function operates by rewriting the record whose Internal Reference Point is in the Reference Field to update the record.

Required Parameters:

WRITV, STATUS, DATA-SET, REFERENCE, LINKAGE-PATH,
CONTROL-KEY, DATA-LIST, DATA-AREA, ENDP

WRITV: Write Variable Function Mnemonic

The user must insert this mnemonic into the Operation Field.

STATUS: Status Code

Significant status codes which may be returned are:

IVRC: The Data-List contains element names not valid for the record code.

REFERENCE: (An Internal Reference Point)



DATA MANAGEMENT LANGUAGE

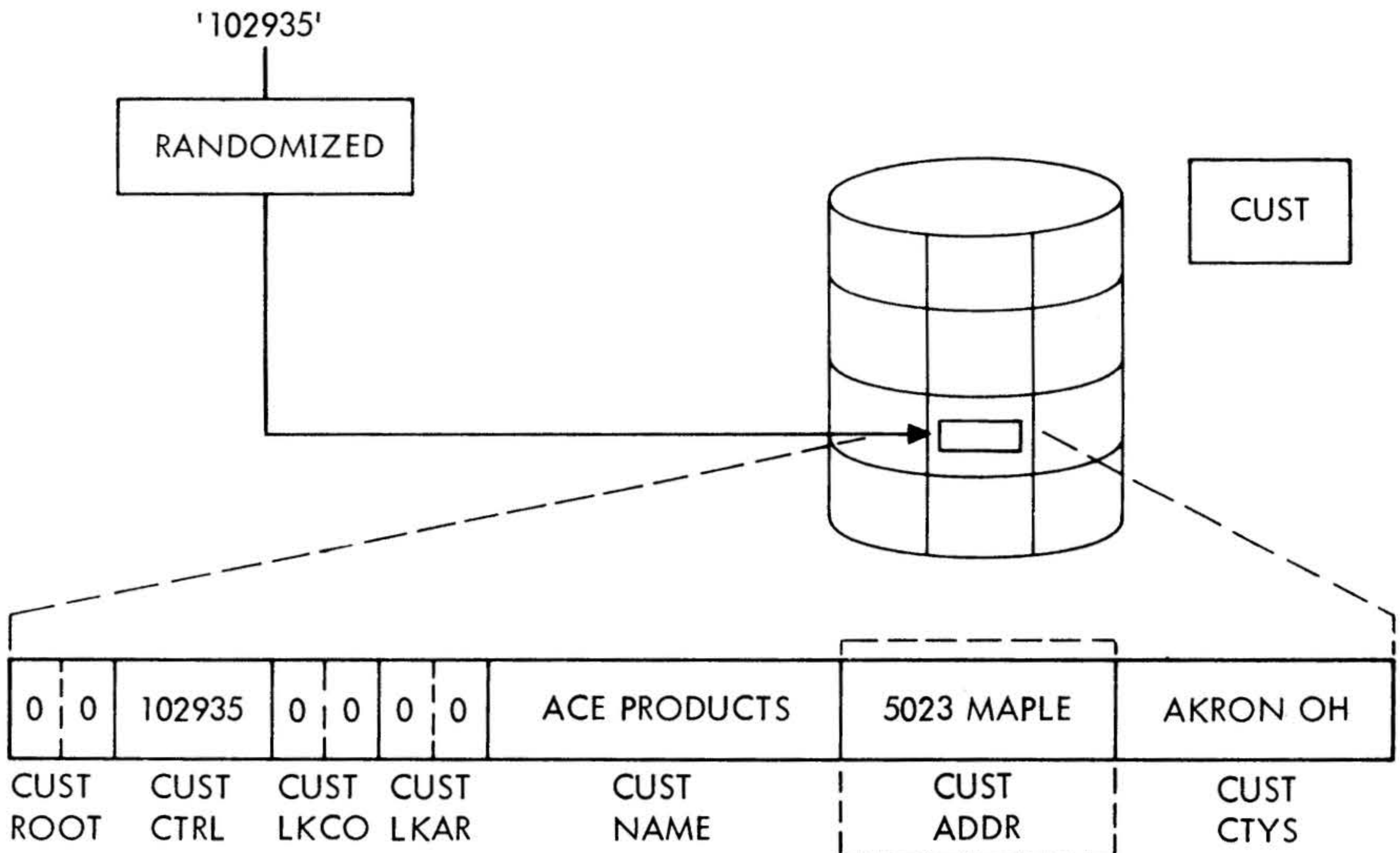
WRITM	WRITE MASTER FUNCTION
-------	-----------------------

EXAMPLE:

FUNCTION = 'WRITM'
 STATUS = blanks
 DATA-SET = 'CUST'
 CONTROL-KEY = '102935'
 DATA-LIST = 'CUSTADDRND.'
 DATA-AREA = '5023 MAPLE'
 ENDP = 'END.'

AFTER THE FUNCTION COMPLETES

'5023 MAPLE' has been written to the record



VT11-3481

Figure 5-20. The WRITE MASTER Function



The contents of the Reference Field are used to point to the specific record to be updated. Therefore, it may contain only an Internal Reference Point.

When the function has successfully completed, the Reference Field will still contain the Internal Reference Point of the record.

Programming Considerations:

Linkage control fields and record codes cannot be modified since the WRITV will not permit linkage maintenance.

An example of the coding for the Write Variable function is given in figure 5-21. In the example, the contents of the COPYBRCH and COPYSHLF elements of record B in Data-Set COPY are rewritten, because the Internal Reference Point value of record B is 0017 and this is the value given in the Reference Field.

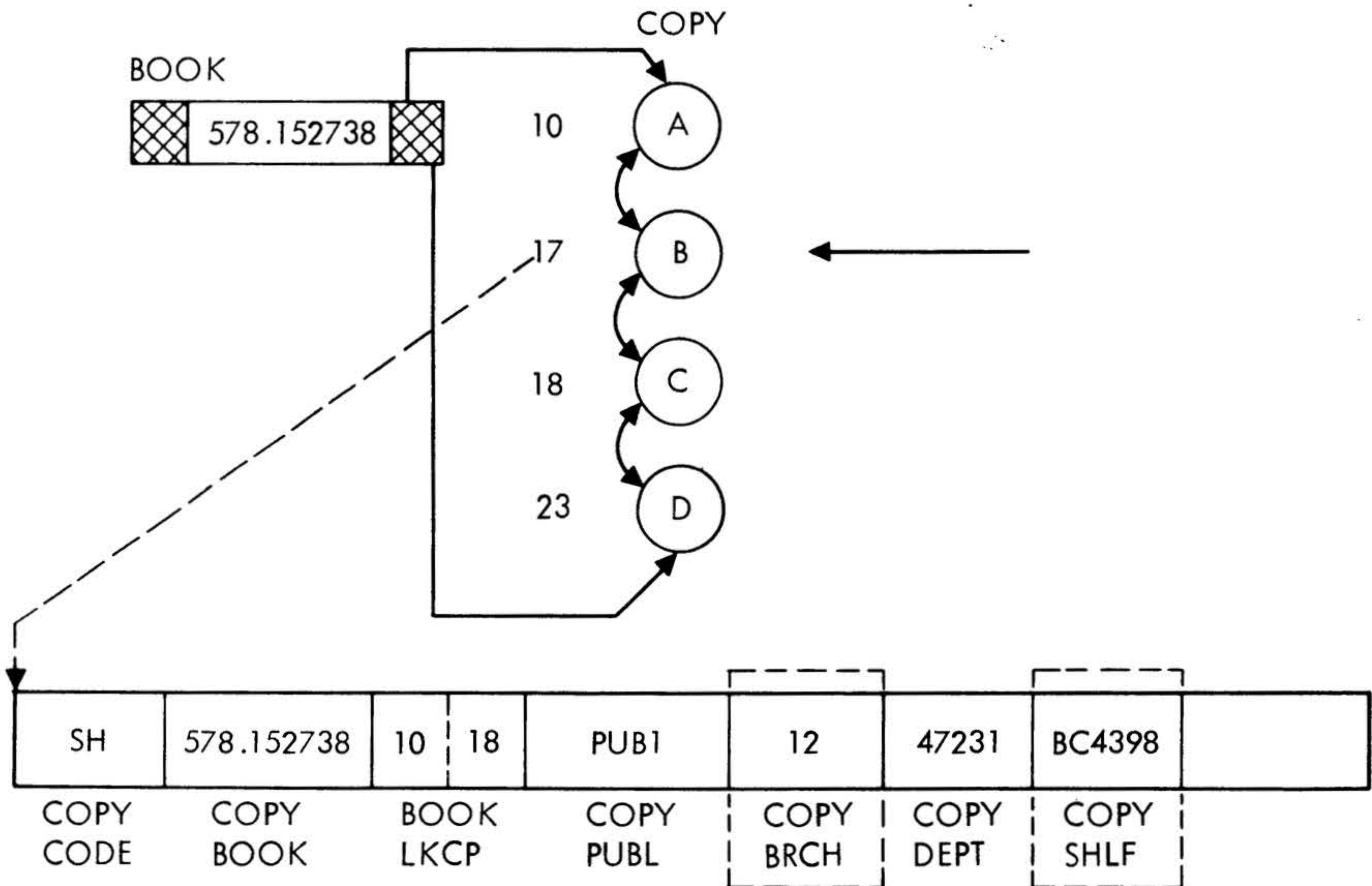


DATA MANAGEMENT LANGUAGE

WRITV	WRITE VARIABLE FUNCTION
-------	-------------------------

EXAMPLE:

FUNCTION = 'WRITV'
 STATUS = blanks
 DATA-SET = 'COPY'
 REFERENCE = '0017' (BINARY)
 LINKAGE-PATH = 'BOOKLKCP'
 CONTROL-KEY = '578.152738'
 DATA-LIST = 'COPYBRCHCOPYSHLFEND.'
 DATA-AREA = '12BC4398'
 ENDP = 'END.'



VTI1-3482

Figure 5-21. The WRITE VARIABLE Function



varian data machines



SECTION 6 PROGRAMMING GUIDELINES

The TOTAL user is not only concerned with the correct and efficient use of the Data Management Language (DML), but also with the Data Base Definition Language (DBDL).

This section attempts to summarize for the programmer certain common methods and pitfalls that have tended to recur in many data base system environments. Its use should assist programmers and programmer/analysts in the development of reasonably correct and efficient programs, in the shortest possible time.

Familiarity with TOTAL terminology with DML functions and formats is assumed during the following discussions. Ideally, the reader will have participated in a basic TOTAL education class, prior to using these guidelines.

6.1 LOGICAL UNIT CONVENTION

Input to DBGEN and DBFMT is via the PI logical unit. Printed output is written on the LO logical unit. Error SNAPSHOT dumps are given on the DO logical unit. Output from DBGEN (the DBMOD source) is given on the SS logical unit. All the above logical units may be reassigned by the operator. For a detailed explanation of VORTEX logical-units refer to the VORTEX II Reference Manual.

6.2 GENERAL PROGRAMMING GUIDELINES

Because all program communication with TOTAL is accomplished with the use of the CALL statement, programming guidelines can be discussed in terms of the CALL statement and CALL parameters.

6.2.1 The CALL Statement

While coding the CALL statement the in-line method is acceptable. It is suggested that TOTAL CALL's be placed into executable subroutines. This method has the advantage of reducing the number of CALL statements that must be coded, permits the CALL statement to be more general in nature, and aids in the 'debugging' process. In most cases, any program could be designed with the following six CALLS:



- a. one "CALL" statement for all master functions
- b. one "CALL" statement for all variable functions
- c. one "CALL" for SIGN-ON
- d. one "CALL" for SIGN-OFF
- e. one "CALL" for Serial (RDNXT) processing
- f. one "CALL" for 'RQLOC' function

In other words no more than six CALLS should have to be coded into a program. In most cases, e and f are used only in special programs.

Example:

Prepare to execute a "CALL" to Read Variable (READV)

Step 1: Move the CALL parameters into the proper areas pointed to by the (fixed) CALL pointers.

e.g.,
Move 'READY' to FUNC (FUNCTION)
Move 'LKXY' to REFER (REFER)
Move 'MOODLKXY' to LNKPTH (LINK PATH)
Move key to CTRFLD (CONTROL FIELD)

Also assume that:

STAT is a pointer to Status field
FNAME is a pointer to File name
ELEMETS is a pointer to the DATA elements
BUF is a pointer to the data buffer
ENDP is a pointer to 'END.'

then:

Step 2: CALL DATBAS, FUNC, STAT, FNAME, REFER, LNKPTH, CTRFLD, ELEMETS, BUF, ENDP

Step 3: Check STAT, if other than "****" go to error routine. If STAT = "****", continue processing.



6.2.2 CALL Statement Parameters

It is recommended that standard parameter names be used by ALL TOTAL programmers in their CALL statement. This is, of course, to ease program maintenance problems by making programs more easily understood by others, and to permit the increased use of the source statement library and other general techniques. While any set of names is acceptable, the following are meaningful, concise, and somewhat standard for the most used parameters.

<u>Name</u>	<u>Size (in bytes)</u>	
FUNC	(5)	
STAT	(4)	
SCHEMA	(M)	
FNAME	(4)	
REFER	(4)	
QUAL	(M)	
LKPATH	(8)	
CTRLFD	(M)	
FFFFEL	(M)	Where FFFF is the file name involed (Element List)
FFFFBF	(M)	Where FFFF is the file name involed (Buffer)
ENDP	(4)	Where ENDP is a pointer to "END."

(M is assumed to be a variable data length).

6.3 PROGRAMMING LANGUAGES

TOTAL supports DML calls from the following languages: DASMR, FORTRAN, COBOL, and RPG II.

Refer to section 5.2 as to the proper call format.

6.4 COMMON PROGRAMMING CONSIDERATIONS

6.4.1 Initialization, File Sharing, and Termination Requirements

a. Initialization:

In order to access a TOTAL data base, one required stage must precede any issuance of DML commands that reference the data base. The program must SINON to TOTAL, activating the TOTAL nucleus and data base descriptor module and OPENing of files specified in the REALM clause.



If a successful SINON (STATUS="****") does not occur for some reason, no other commands will be accepted (STATUS="NOSO" or 'FNOP').

Files must be logically requested for use via the SINON, prior to the issuance of any DML functions that reference these files. Note that if variable entry files only appear to be referenced, TOTAL requires that the associated single entry files may also have to be requested by the program, since TOTAL will perform "hidden" accesses to these files, if this becomes necessary.

b. File Sharing:

Each program, via the REALM component of the SINON parameter, may designate some (or all) files as shared (SHRE) with other programs or private (PRIV) to the exclusive use (and possible update) by that program alone.

A file must be designated as private if its contents may be altered by some updating operation. It is recommended that a file be designated as private even in the case in which a file is not altered by the program but may be altered by some other program. Consider the following example:

Program A declares file x as shared because it intends to use it for READ only. After some time, program B signs on and declares file x as private. Program A reads records which may be changed by program B. However, as long as B is still signed on, no other program may sign on and designate file x for use. To prevent this occurring, program A could have designated file x as private and by doing so, blocking any sharing attempts by program B.

c. Termination:

All that is required for termination from the TOTAL system, is the issuance of the SINOF function. This will not only release all DML facilities, but it will also logically and physically close all active data files.

6.4.2 Checking the STATUS Parameter

It should be emphasized that the contents of the STATUS parameter must be verified after each and every DML "CALL" statement. This is the primary communication cell between TOTAL and user programs, and any contents other than "****" in this parameter has definite significance.



There are two classes of STATUS values other than "*****" (good status), with which the programmer must be concerned:

a. Informative:

These returns are usable as indicators, in that they normally show input errors or file status information, rather than permanent errors. Among these are "MRNF", "DUPM", "IMDL", "LOAD" (on close only).

b. Fatal Errors

Any returns other than those outlined above indicate some severe error in either the parameters themselves, or the execution of this DML CALL. The problem program will normally be terminated for analysis. TOTAL will automatically close the data base and then return to the user program.

TOTAL produces a SNAPSHOT dump (see 7.10.3) of the TEXT area (see appendix C) on logical unit DO in case the error was FUNC or IPAR. In all other cases the user should obtain a memory dump; the TEXT block should be examined to determine the cause of the error.

In test situations, there is a possibility that coding errors may generate some of the status values bypassed by this type of check, on every transaction. For this reason, it is good practice to assure that if bad status is returned repeatedly (no matter what its classification), the program is terminated. For example, if 'MRNF' is returned 50 times in a row, there is a good chance that some coding or file is not correct in some way.

6.4.3 Parameter List Definitions

It is usually wise to conform to minimum conventions, in identifying the parameters in TOTAL DML calls, for example:

SINON	DATA	'SINON'
READM	DATA	'READM'
ENDP	DATA	'END.'
STAT	DATA	'*****'



This will help insure program readability, and make debugging a little easier. In many installations, the ELEMENT LIST and I/O parameters, which identify the data elements to be used from a data base by a problem program, is also standardized.

Example:

ELEMENT LIST

NAME	DATA	'CUSTCTRL'
ADDR	DATA	'CUSTADDR'
END	DATA	'END.'

I/O

NAME	BSS	20
ADDR	BSS	30

It is then possible to keep such definitions in a source statement library (under centralized control), and thus ensure that problem programs in a given system reference the same data by the same names. This can be invaluable in program maintenance.

6.5 STANDARD SINGLE-ENTRY FILE PROCESSING

The various master data-set commands have been described in sufficient detail in the previous sections to make them usable by the programmer. The following sections describe various techniques which can be used to increase overall efficiency in single-entry file processing, and to assist the programmer in avoiding common pitfalls.

6.5.1 The RQLOC Function

In cases where a large portion of any single-entry file will be involved in a given process, there may be significant advantages to be gained by ordering the input transactions, wherever possible, e.g., for loading of data base (section 6.9).

Since single-entry file records are randomized in storage, the only ordering that can be meaningful is the order of physical relative locations occupied by given keys.

For this reason, the RQLOC function was provided to allow the programmer to determine the physical location of a record, given its key, through a direct "port" into TOTAL's randomizing routines. By pre-determining the set of values for a given input transaction stream, a pre-sort of transactions in RQLOC value sequence will ensure minimum RMD self-contention, and



maximum buffer usage in any given program situation.

The following should be considered when using the RQLOC function:

- a. The transaction volume is a factor in determining whether the time taken for the pre-sort and RQLOC processing becomes greater than the arm contention time. Each situation should be judged separately.
- b. Single-entry file accesses are made in variable-entry file processing, even if they are hidden (as they are in ADDVC's, for example). This accessing should also be taken into account, along with the explicit single-entry file CALLS's (ADD-M, DEL-M, etc.).
- c. It may be advantageous to sort keys in descending RQLOC value sequence, whenever synonyms are present in the single-entry file.
- d. RQLOC does not require any I/O.

6.5.2 Data Elements that Should not be Referenced

The following data elements should never appear in application program DML CALL's:

----ROOT, ----LKXX

The ROOT element is used only by TOTAL for internal purposes. ROOT can only be read, and cannot be written by the application program.

The linkage fields are automatically updated and maintained by TOTAL, and should not be referenced or updated in the solution of application problems. Usually such a reference will be disallowed, and will produce a status of ENTF.

Linkage paths to a master file may be both written and read by the application program. Linkage paths to a variable file cannot be written or read by the application program.

6.5.3 Special Note on the ADD-M Function

The most frequent coding errors that occur in the issuance of this important DML function are usually generated by a bad correlation between the data value in the CONTROL FIELD parameter (used for record location) and the value in the mmmmCTRL data element I/O area (used to write the imbedded data key into the data record, and required for this function).



To avoid any problems, always define these two values as the same location within the program. This will guarantee correlation, and ensure that the key is always written on the physical record.

Example (COBOL FORMAT):

```
ENTER ASSEMBLER.  
CALL 'DATBAS' USING ADD-M, STAT, FILE, CTRL, ELEM, IO, ENDP
```

Where:

```
01  IO.  
    02      CTRL      PICTURE X(5)  
    02      DATA     PICTURE X(100).  
01  ELEM.  
    02      XXXX      PICTURE X(8)  VALUE IS 'CUSTCTRL'.  
    02      YYYY      PICTURE X(8)  VALUE IS 'CUSTDATA'.  
    02      ENDX      PICTURE X(4)  VALUE IS 'END.'.
```

6.5.4 Structural Maintenance During Serial Processing

In some instances, a user may wish to perform ADD/DELETE logic to a single-entry file, while processing it serially.

Caution should be exercised in performing this maintenance function due to the fact that the program may not subsequently have serial access to certain records, after the operation is performed. For example, the current record (retrieved serially) may be an anchor record of some synonym, which has not yet been read. If the current record is deleted, TOTAL will automatically optimize the file, and may move the synonym physically, so that it is unavailable for the next or a subsequent serial access. In fact, a reset of the serial marker must be made in order to retrieve it.

To avoid this situation occurring, it is suggested that structural maintenance be performed after the serial processing is complete, to ensure that all records will be available for program analysis.

Update in place logic (WRITM) executes correctly, since no record movement ever takes place.



6.6 STANDARD VARIABLE-ENTRY FILE PROCESSING

If any portion of the TOTAL DML can be said to be complex, it will be associated with variable-entry file commands. For this reason, this section will cover this type of processing in more detail than was outlined for single-entry files.

6.6.1 Basic Considerations

Certain misconceptions about the basic functioning and structure of variable-entry files tend to surface rather frequently, even among experienced TOTAL users, so a list of facts is presented here that should be remembered:

- a. All linkages are updated when an ADD type function is performed, and this is handled automatically by TOTAL. The LINKAGE PATH parameter in the DML call is simply used as the first point of reference for TOTAL's internal processing and is assumed to be the primary path.
- b. Variable-entry records carry the symbolic pointer (control field) of every master record to which they are linked, along with linkage fields. The pointer is symbolic (i.e., of the same format in every respect to its attached single-entry master key), to eliminate re-organizations when files are moved on RMD devices.
- c. Variable-entry record for a given file all have the same length, even though they may differ in format (see CODED RECORDS, discussed later). Variable length records are not supported in any TOTAL architecture, for reasons of efficiency.
- d. If a given variable-entry record is linked to say, 2 other files, the control field value and definition for these other files must be presented in the I/O area and the element list parameters on any ADD function request. Failure to do this will force a "BCTL" (blank or unidentified control field specification) STATUS return.
- e. Records should be read before they are written or deleted.

6.6.2 The REFER Parameter

In order to provide flexibility in manipulating linked records in variable-entry files, TOTAL DML requires that a second communication cell be provided, called REFER. It is essentially a positional indicator for list processing.



The parameter may have several values, some provided by the program, and some by TOTAL. In order to clarify the use of this parameter, these values, and how they change is described below: (Note that "LKXX" will be used to denote the last four characters of some user linkage path parameter.)

REFER Values

"LKXX" This indicates that the first (or last) record of a list is to be processed. This value is inserted by a user program with only one exception, described later. The literal consists of the last four characters of the current linkage path data element name.

This value is valid only for READV, READR, and ADDVC functions.

"END." This literal is inserted by TOTAL to indicate that the last access to this list, exceeded the bounds of the list (i.e., that no more records exist on the given linkage path for this control field).

Only TOTAL may place this value in REFER.

This value must be cleared from REFER before another function CALL is issued.

This value type can only be returned from the issuance of READV or READR functions.

Table 6-1 illustrates the use of REFER in relation to other associated parameters.

Table 6-1. Relationship of REFER to Other Parameters

Condition	Function	Set REFER To	REFER returned by TOTAL
Read first record in chain Read next record in chain Read next (end of chain) Read next (after end of chain)	READV READV READV READV	LKXX (do not disturb) (do not disturb) END.	RRLOC Current Rec RRLOC Current Rec END. END.
Read last record in chain Read previous record Read previous (end of chain)	READR READR READR	LKXX (do not disturb) (do not disturb)	RRLOC Current Rec RRLOC Current Rec END.
Read directly a record	READD	RRLOC desired rec	RRLOC Current Rec
Update Current Rec Update Current Rec	WRITV WRITV	(do not disturb) disturbed	no change no change
Add before Current	ADDVB	RRLOC Current	RRLOC new record
Add record after Current	ADDVA	RRLOC Current	RRLOC new record
Add record at end of chain	ADDVC	ignored	RRLOC new record
Delete a variable record Delete first record in chain	DELVD DELVD	(do not disturb) (do not disturb)	RRLOC previous record LXKK





After every READV or READR, first the programmer checks STAT for "*****" then checks REFER for END.

COBOL Example:

```
CALL 'DATBAS' USING FUNCTION          STAT          FILE-NAME
                        REFER          LINKAGE PATH  CTRL-FIELD
                        FFF-ELEMENT    FFF-DATA     ENDP.
IF STAT NOT="*****" GO TO ERROR-END-OF-JOB.
IF REFER = 'END.'"   GO TO END-OF-CHAIN.
```

REFER Changes

It is important to reiterate the conditions under which the value of REFER is modified by TOTAL, so that program logic can be produced accordingly. The following list indicates the significant REFER value changes after the execution of DML variable-entry functions:

- a. READV with REFER = "LKXX"
If the STATUS value is "*****", then REFER will now contain the relative address of the current record in this variable-entry file, or "END." if no records exist for this control field value. If bad STATUS was returned, REFER will be unchanged.
- b. READV with REFER is a binary number
For good STATUS returns, the value may be another binary number (corresponding to the address of a new record), or "END." if an end-of-list condition has been reached.
- c. ADDVC with REFER is "LKXX"
For good STATUS returns, REFER will now contain the address of the record just added.
- d. ADDVA, ADDVB with REFER is a binary number
For good STATUS, REFER will be modified to point to the record just added.
- e. DELVD with REFER is binary number
After a successful deletion, REFER will point to the previous record in this list, i.e., the record prior in the list to one just deleted. If the record just deleted was the first in this list, REFER will contain "LKXX."



- f. Note that TOTAL modifies REFER in the above ways, within the user's working storage area. It is therefore important to ensure that this field is not modified unnecessarily.
- g. The address value in REFER is a 4 byte binary number that corresponds to a relative record number from the start of this file. It is not an absolute device address.
- h. When attempting DELVD's in the reverse direction (i.e., issuing READR, DELVD, READR, DELVD, etc.), recognize that every second record will be skipped, due to the value of REFER. The user, in this case, should issue READR, DELVD, READD, DELVD, READD, etc., to effect a full list reverse delete.
- i. Under no circumstances issue the following sequence of commands: READV, READV, DELVD, DELVD, DELVD, --; nor READR, DELVD, DELVD, etc. A record must always be read prior to deleting it.

6.6.3 Efficiency Considerations in Variable-Entry File Processing

The majority of programs written to use the TOTAL DML are reasonably efficient, even when written with only a basic knowledge of the DML functions and their workings.

The more sophisticated programmer, however, will want to concern himself with the most optimum methods of utilizing the DML to meet a system requirement. For this reason, the following collection of hints and facts are presented as guides to efficient programming.

- a. VARIABLE ADDS (ADDVC, ADDVB, ADDVA):
The ADDVB and ADDVA functions allow records to be accessed in some logical sequence on one given linkage path. In order to use these, however, the program must read records in a list just to determine the position from which the ADD is to be performed, and then issue the appropriate function. TOTAL itself provides no built-in data sequencing, other than that of chronology.

These functions, then, are far less efficient, in overall time, than the ADDVC function.



b. SPACE MANAGEMENT:

Whenever possible, TOTAL attempts to keep all records within a list as physically close together as it can in secondary storage.

Since multiple lists within one variable-entry file are allowed, it is obvious that this optimization cannot be maintained for all such lists at the same time.

So that TOTAL may optimize lists for its storage allocation, TOTAL always assumes that the linkage path specified in a given ADD function identifies the link list to optimize upon for that given ADD (ADDVC, ADDVB, ADDVA). It is to the user's advantage to assure that the linkage path specified in any ADD is always the same.

In order to optimize retrieval times, programs should also specify this same linkage path in the DML READ command.

c. FUNCTION EQUIVALENCE:

Due to the flexibility of the DML, and of TOTAL's internal processing, it is possible to equate certain function CALL's to other combinations of CALL's. In other words, the same effect can be achieved in different ways, at almost no additional cost in time.

The most usable equivalences are the following:

1. An ADDVC is functionally equivalent to a READM followed by an ADDVC.
2. A READV is functionally equivalent to a READM followed by a READV.

Depending on the given programming situation, it may be advantageous to use one or the other of these possible forms.

6.6.4 Coded Variable-Entry Records

The facility to redefine variant formats in variable-entry records provides a powerful means of storing logically related data elements together in a data base. It also allows certain record formats to be selectively linked to given relationships.



The implementation of this facility involves the use and specification of a 2-byte record code, which must physically appear in each record of the variable-entry file, and furthermore be defined at DBGEN time.

The following is a list of guidelines to the use of this record code facility.

- a. When ADDing a coded record, the record code must be provided in the I/O area, and the "----CODE" element name must be provided as a component of the ELEMENT-LIST parameter in the DML CALL as the first name given.
- b. When READing along a linkage path that connects multiple record types, the user program must determine the format of the retrieved records, by verifying the contents of the "----CODE" element, or by utilizing the code-directed READ feature which may be utilized to retrieve only specific record code types.

Record codes on a variable-entry file cannot be modified using the WRITV function.

6.6.5 List Maintenance in the Batch Mode

Whenever data base files are to be processed primarily in the batch mode, there are certain steps the programmer may take in optimizing the execution times of such processes. Some of these steps are:

- a. Sequence transaction input in RQLOC sequence first (refer to the RQLOC function). This should be done as a matter of course.
- b. Sequence transactions within key in RQLOC sequence, when variable-entry affected transactions will be input (to minimize variable-entry data set access contention).
- c. Consider logically sequencing data during update, in heavily accessed variable-entry files, so that input transactions may also be sequenced in this way. This will require the use of the ADDVB and ADDVA functions. Even though, as mentioned previously, this may impact the execution time of initial update processing, it may be worthwhile particularly when data element values change often (WRITV processing) in major update cycles.



- d. Whenever possible, attempt to group all pending changes on a given list, so that one pass of the list (using READV, READR) will be sufficient to complete this list's processing activities. This will help alleviate situations in which a list is completely read each time a new transaction enters the system, thus forcing redundant data accesses against unmodified data records.

6.7 TESTING DATA BASE PROGRAMS

As is true of other programs, tasks that use the TOTAL DML must be tested prior to becoming productive, in order to insure their structural and logical integrity to the best degree possible.

Program testing in the data base environment is perhaps a little more complex than in, say, a sequential tape or disc environment, due to the inter-relationships inherent in a data base and the fact that a program's actions may affect multiple data sets in only one CALL statement. Furthermore, there is always a temptation to test the program on the "live" or productive data base, but it should be obvious that severe repercussions can result from uncontrolled testing of this nature.

However, with the proper precautions and some simple guidelines, the testing of programs can be simplified, while maintaining the integrity of a live or productive data base.

Some guidelines to program testing are noted here:

- a. Whenever possible, construct a small data base for testing purposes, with the same logical characteristics (file names, DBD names, elements) as the live data base, but containing only a subset of its data (i.e., 5 to 10 percent). This base can be updated periodically to reflect most recent data relationships, through a user-written utility (strongly recommended).

After a test run against this base, a user utility should exist to sample the data that should have been affected, for verification purposes. The test base may then be "reset" to its original condition by restoring it from its original backup copy.

- b. Anytime a program in test is to be active against any data base, it should be guaranteed that a current backup of possibly affected files exists prior to the run.



6.8 BUFFERING TOTAL DATA SETS

All data sets require memory storage for physical records obtained from or transferred to them. In many situations, it is required that a permanent allocation of storage be made, and this can be wasteful, when relatively few or cyclic references are made.

For this reason, TOTAL allows the shared allocation of data sets to buffer areas, to minimize storage.

Single-Task Buffering

Perhaps one of the most pressing questions in the specification of a Data Base description, is: Which files should be shared together in the same buffer areas? Some hints on this will be provided below. One fact should be remembered however, and this is, if main storage is a problem, shared buffers minimize the use of storage, but may trade this off for processing speed. It is unreasonable to expect the ultimate buffer assignments.

- a. Do not share buffers among files with vastly different blocking factors. TOTAL will force the buffer to be as large as the largest block size of all shared files.
- b. Do not share a buffer among single entry files associated with a variable entry file that is subject to structural maintenance (ADD's, DELETES).
- c. The order of priority by which TOTAL fills buffers within a buffer pool is as follows:
 1. If a record is already in the buffer, no I/O operation is performed.
 2. If a record is not found and there is a buffer not used yet, a READ is performed.
 3. If a record is not found and all buffers are used but there is a buffer that does not contain updated data, a READ is performed.
 4. If a record is not found and all buffers contain updated data, a WRITE to save the buffer contents is performed followed by a READ.



6.9 DATA BASE LOADING

One important aspect of data base construction will be the initial loading of the TOTAL data base files. In many cases, due to data volumes and/or the linkage requirements, the speed at which the data base load occurs becomes an important consideration. Obviously, long loading times increase the chance that some malfunction will destroy whatever has been accomplished, and require restart of the process.

Several techniques have therefore been developed that assist the user in minimizing data base loading times, and they are discussed here:

Basic Techniques:

General Comments:

- a. Load single-entry files first, prior to loading the variable-entry files with which they are associated.
- b. Use a special "load mode" Data Base Description module, that minimizes data element definitions for the loading process, (i.e., define all data parts under one element name).
- c. Avoid sharing I/O areas between files during the loading process.
- d. Sequence the loading of files so as to ensure that during any particular loading process, the load routines will not be accessing data sets on the same physical volume. (i.e., avoid arm self-contention of the RMD during loading.)
- e. Once files are loaded they should be immediately backed up with a file copy to guarantee their integrity, and to avoid re-loading in the case of failure.

Single-Entry Loading:

- a. Sort the data in RQLOC sequence before adding to the file (RMD), to avoid RMD self-contention (refer to section 6.5.1).
- b. Allocate about 5 to 10 percent more record space on the file than will actually be used. This will help ensure that synonym space can be found within the home address block, and further minimize synonym processing.



Variable-Entry File Loading:

- a. Sort the input records in the RQLOC sequence and then the control field value on which these records will primarily be accessed in productive processing. This will help minimize single-entry file I/O involvement, while also ensuring physical proximity of related records.
- b. Select a proper load limit value, so as to guarantee a proper list distribution, and to avoid cluttering at load time.

6.10 SERIAL PROCESSING

The serial processing DML function in TOTAL is designed as a method for large scale file or list access when the retrieved records are not known uniquely by key, and when only data that actually exists on the file(s) is to be processed.

The user is cautioned, however, that this facility may be misused, and that trade-offs do exist, particularly with respect to processing time, in its use. In order to comply with these trade-offs, a brief look at the uses of this function and some basic considerations is initially offered here:

Typical Uses of the RDNXT Function:

- a. To scan a single-entry file, in order to set or reset a status element for all, or most records.
- b. To scan a single or variable-entry file in order to produce a data set suitable for report processing.
- c. To access all single-entry records in order to reformat linkage data only (during variable-entry file expansion or logical modification).
- d. To retrieve all "live" data from data base files for the purposes of backup.

Basic Consideration:

- a. The serial function operates serially (not sequentially), and, therefore, do not use sequential access methods.
- b. The serial function will automatically read all records, even though it returns control to the problem program only when records contain data.



- c. The serial function provides full data independence in its operation.

Serial Processing Alternatives:

- a. Sequential type access to TOTAL data sets using normal sequential file definitions.

In this method, TOTAL files are read as normal data sets, and logical records, in their entirety, are accessed. It will be the user's responsibility to bypass unused records, and to capture only the data portion of 'live' data records, by avoiding all control information.

- b. Modified SORT access to TOTAL data sets.

Particularly when report processing is to be performed, TOTAL data sets may be input to a sort program and an exit may be used to avoid unused records and/or control information for the actual sort phase.

These alternatives will allow extremely fast access to all records in TOTAL files, but as was mentioned earlier, do cause a heavy reliance on physical data formats, which are subject to change, and bypass the data independence feature of the TOTAL Data Base Management System.

It is suggested that the user of small to medium size data bases may find that the serial function provides the performance levels required, while users with large data files that require large scale access, may wish to consider the alternatives if attempting to minimize execution times.

It should be easy to see, therefore, that using the serial function in TOTAL will result in execution times for its use much less than random access times, through somewhat greater than sequential access techniques. This difference is primarily caused by the need for data independence, and TOTAL's one buffer per file mechanism.

There are other techniques that may be used in lieu of the serial function, depending on a user's particular needs. These techniques accomplish a savings in time only at the expense of data independence, and require an awareness of the physical file organization which is subject to change.



SECTION 7 OPERATING VORTEX TOTAL DBMS

Operation of the VORTEX TOTAL Data Base Management System consists of the successful generation and formatting of the data base and loading of the data files. In addition, efficient operation of the system requires adequate provision for data base back up and recovery. Provision must also be made for maintenance of the data base to enable data file changes to be incorporated, and the correction of errors through diagnostic error messages and debugging aids.

7.1 CATALOG OF APPLICATION PROGRAM

The application program's OM (Object Module) is bound together with TOTAL and DBMOD by LMGGEN to create the run time (Load Module) application program. This program may be cataloged either on the background or foreground library. The order of a cataloging should be: TOTAL,DBMOD,application OM.

Example:

/JOB,CREATE	
/LMGEN	
TIDB,PROGDB,1,0	
LD,OM,D,TOTAL	TOTAL module on OM library
LD,25,,DBMOD1	DBMOD on LUN 25
LD,25,,PROG	User OM on LUN 25
LIB	
END,BL,E	Background library
/FINI	

7.2 TOTAL FILES

There are two functional types of Data Sets or files in the TOTAL DBMS: the single entry Master data set and the Variable entry data set.

The Master data set is an independent entity which can stand alone or have one or more dependent (variable) data sets attached to it. Each data set consists of records, and in the Master data set each record has a unique control key which enables the record to be accessed directly according to the value of this control key.

The Variable data set must be linked to at least one Master data set. Records in a Variable data set are logically accessed through a particular master record and then from one variable record to the next via the linkage path related to the master



record. A sample data base for a business application showing typical Master data sets and Variable data sets and the links between the data sets and the records contained in the data sets is given in figure 7-1.

7.2.1 TOTAL Logical Files and VORTEX Files

The difference between TOTAL Logical Files and VORTEX physical files is that a VORTEX file is limited to one partition (logical unit) while the TOTAL Logical File may extend on one or more logical units (partitions). Thus, one TOTAL file may contain more than one VORTEX file. Furthermore it is not necessary that all VORTEX files reside on the same RMD. For each DBDL Drive Statement a VORTEX file is generated.

Variable-file space management requires a series of control records inserted at regular intervals throughout the TOTAL program. One control record is placed in front and one behind the file which needs to be segmented. They are used to define the control intervals for the VORTEX files. The control interval is 480 sectors long. The load limit in percent is computed and maintained for each control interval (see LOAD-LIMIT directive). Each variable logical TOTAL file starts a control interval.

The following is an example of TOTAL coding using the Drive Statement:

Example:

TOTAL FILE: BASE

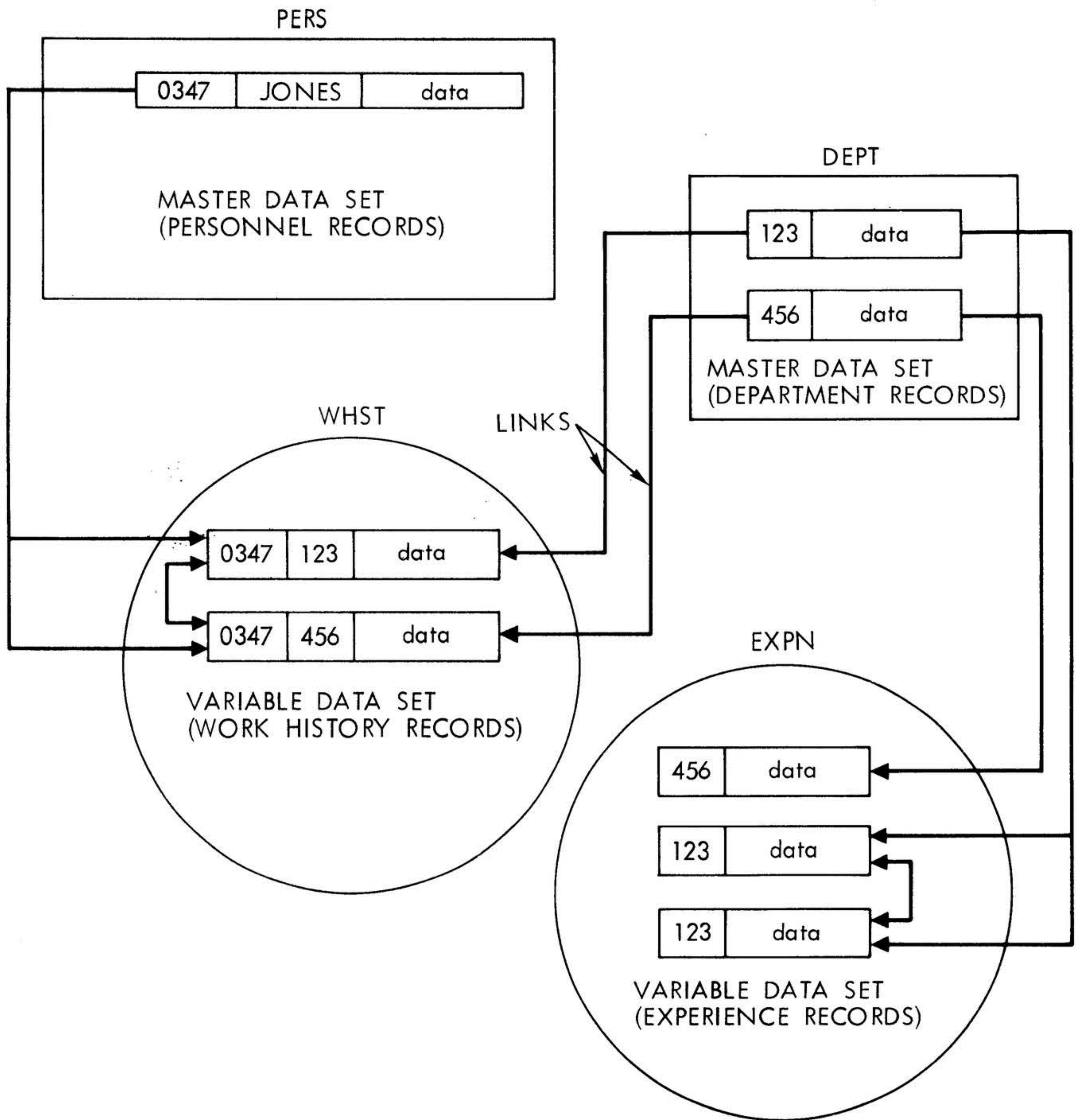
<u>Corresponding VORTEX File</u>	<u>Drive Statement</u>
BASE01	13,200
BASE02	14,100
BASE03	15,400

The first control record of the TOTAL file is the first physical record of BASE01.

The second control record is logically the 481st record and is the 181st record of BASE03.

7.2.2 Disc Utilization

Master files: In order to minimize the number of synonyms, file growth should be anticipated by declaring the file at least 20 to 25 percent larger than its initial capacity.



VTI1-3484

Figure 7-1. TOTAL Data Set Relationships



Variable files: The size of variable files is controlled by the LOAD-LIMIT directive. If more than 80 percent of threshold is required (default), the required number should be supplied. This is important if variable chains are very long and tend to overflow the control interval boundary.

7.3 BUFFER SHARING

A general discussion of buffering TOTAL Data Sets is given in section 6.8. Program efficiency may be increased by the planned use of buffer sharing which minimizes the use of storage, however there may be some sacrifice of processing speed. Buffer sharing is accomplished by specifying the same IOAREA for all files that are to share the same buffer pool. Refer to section 6.8 for the priority for buffer selection. It should be noted that an actual I/O WRITE operation is performed only if the buffer is required and some data in it has been updated. In all other cases a WRITE request only sets the update flag.

7.4 CREATING THE DATA BASE

The TOTAL system consists of three distinct programs:

- a. DBGEN - the Data Base Generator program
- b. DBFMT - the Data Set Format program
- c. TOTAL - the Data Base Management program

DBGEN accepts the Data Base Definition Language (DBDL) statements and generates an assembly language Data Base Descriptor Module (DBMOD). DBFMT accepts the data set format control card(s), creates the data sets, and preformats the specified RMD areas according to the physical parameters.

TOTAL is initiated by CALL instructions from the application program, communicates with the Data Base Descriptor Module, and acts on the preformatted disc area.

7.4.1 Data Base Generation

Generation of the data base is accomplished by means of the Data Base Generation program DBGEN. DBGEN reads user prepared DBDL statements and generates source statements which in turn (after assembly) produce the Data Base Descriptor Module (DBMOD).



To produce the assembly language Data Base Descriptor Module using the DBGEN program, proceed as follows:

- a. Complete and keypunch Data Base Definition Language (DBDL) specification cards.
- b. Execute the DBGEN program via the DBGEN directive. The program reads the DBDL specifications from the PI logical unit, writes onto the SS logical unit assembly language source program statements, and prints the data base documentation listing on the LO logical unit.
- c. The assembly language source program statements are input to the DASMR Assembler and an object Data Base Descriptor Module is created on the BO logical unit.
- d. DBFMT object module is LMGEned together with the user DBMOD object module and is catalogued as the data base formatter program (see section 4.1).

The above steps are shown in flow chart form in figure 2-1.

7.4.2 Parameter Card Format

The parameter card format is described in section 3.1.3.

7.5 FORMATTING THE DATA BASE

Formatting is accomplished by means of the Data Set Format program DBFMT. DBFMT reads format parameter statements and pre-formats the data sets, utilizing a Data Base Descriptor Module.

In use, DBFMT creates and initializes the data sets and establishes all necessary control records. DBFMT is a parameter statement driven program; these parameters name the Data Base Descriptor Module and the data sets to be formatted. DBFMT uses the foreground file maintenance program V\$FGFM to create the files. LMGEn then links DBFMT and the user's DBMOD to create the user formatter program.

The formatter reads data base control directives from the PI logical unit, and creates and formats the requested data sets on the RMD. Diagnostics are printed out on the LO logical unit.

```
bFORMAT dbname file1,file2,.....,filen,END.
```

where

b is a space; FORMAT starts in column 2



dbname is the data base name

file1....filen are data set (file) names

The last parameter must be END.

Continuation cards may be used if there is not enough space on one card for all the file names that are required. However FORMAT dbname must be duplicated for each card.

Data Base Formatting is shown schematically in figure 7-2. A detailed flow chart is given in figure 4-1. Examples of the use of FORMAT control directives are given in section 4.

7.6 DATA BASE EXECUTION

Execution of the application programs is accomplished using the Data Base Interface Module (DATBAS) and the TOTAL Data Base Management Module.

DATBAS serves as an interface between the user application program and the TOTAL and Data Base Descriptor modules.

TOTAL provides the data management capability of the system, interpreting and executing the various DML commands from the user application program.

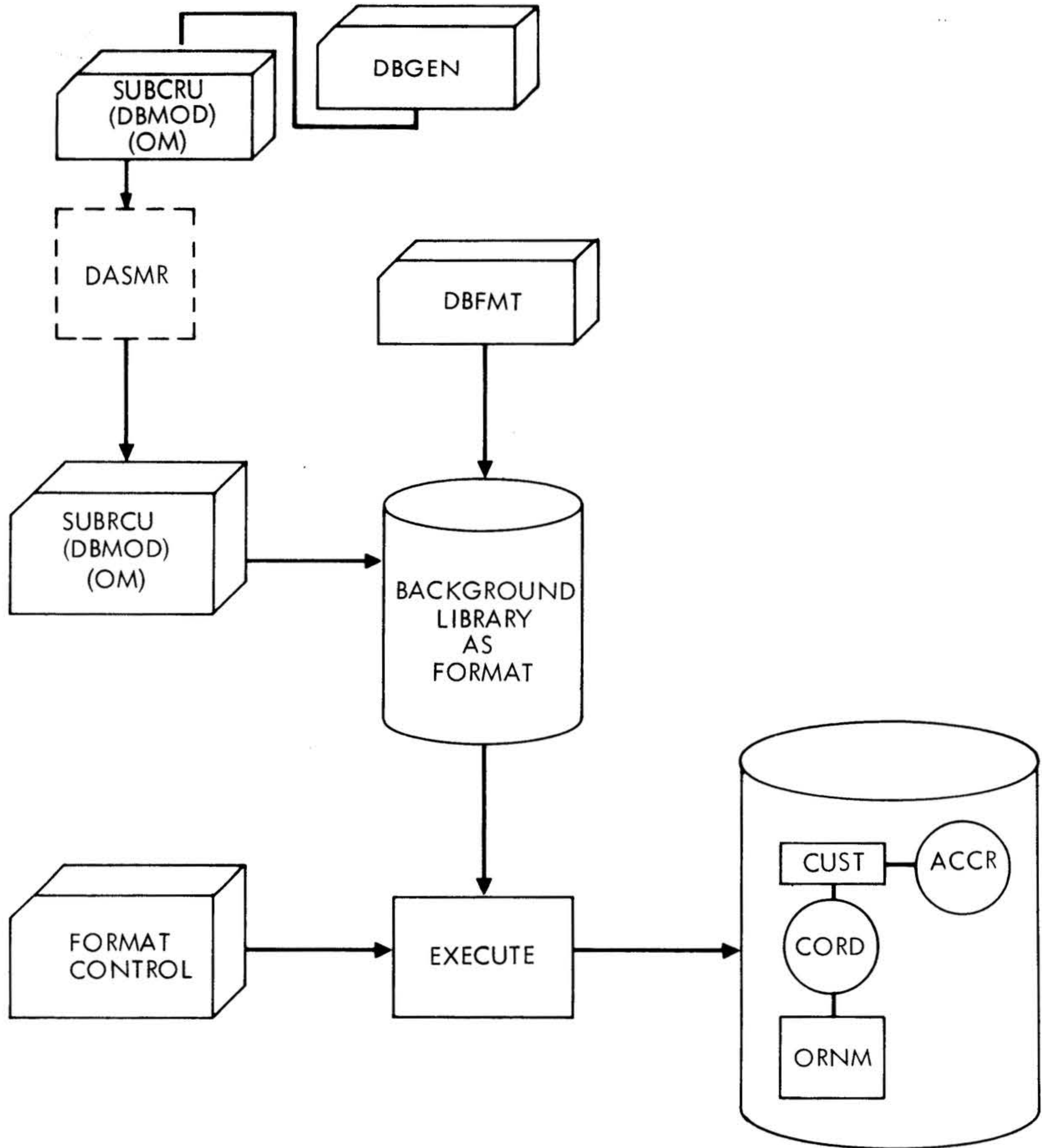
A flow chart showing the TOTAL DBMS operation is given in figure 2-2. TOTAL can be used to load data on to files already formatted by means of CALL statements from the application program, or through a TOTAL utility program. See section 5.2 for a description of the TOTAL calling sequence.

7.7 DATA BASE RECOVERY

Recovery of the Data Base is accomplished by:

- a. Unloading the files to be backed up.
- b. Restoring the files.
- c. Re-running the transactions.

It is recommended that a back-up or history file be maintained, so that the data base can be recovered in the case of a catastrophic error. The history file can be on magnetic tape. FMUTIL can be used to dump files.



- PRE-FORMATS ALL RECORDS
- CREATES AND INITIALIZES SPACE MANAGEMENT RECORDS
- CREATES AND INITIALIZES FILE CONTROL RECORDS

VTI1-3485

Figure 7-2. Formatting TOTAL Data Sets



7.8 DATA SET CHANGES

7.8.1 Single Entry Data Set Changes

When physical changes to a single entry data set occur, such as an increase in TOTAL logical records and expansions in record length, the user must:

- a. Unload the affected data set
- b. Define the physical changes to the DBGEN
- c. Reload the data

The following procedures are necessary to accomplish these changes:

- a. "RDNXT" through the single entry file requesting all elements (including the links) except for the ROOT; i.e., "mmmmROOT".
- b. Write each record to a sequential file.
- c. Redefine the DBGEN for your data base with the necessary changes.

NOTE: Be sure to change the physical entries within the DBGEN for this file.

- d. Reformat the data set changing FORMAT input to reflect the changes to the data set.
- e. Write a program to read the sequential file created in step b, pass the control field (key) to RQLOC, append the relative disc address to the record, and write the record to another sequential file.
- f. Sort the output of step e by relative disc address (4-character binary field) and the control field.
- g. Input the sorted file to a load program. This load program should use the identical element list and I/O areas as defined in step a.

The single entry data set can be reloaded with or without the RQLOC technique.

7.8.2 Variable Entry Data Set Changes

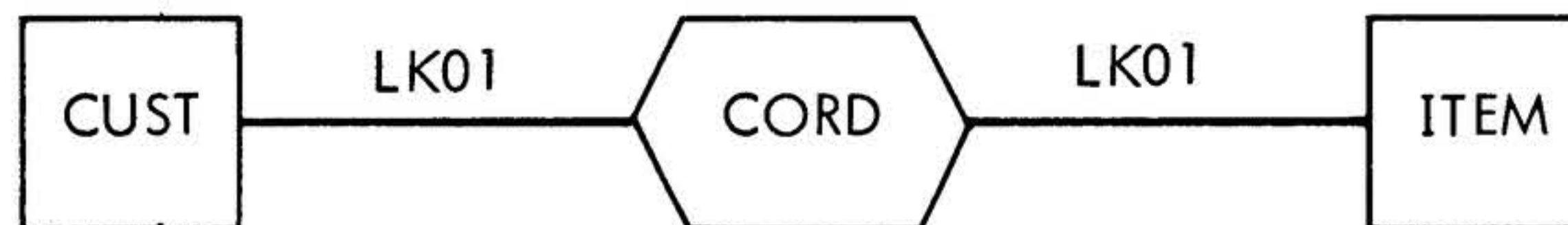
When physical changes to a variable entry data set occur, such as an increase in TOTAL logical records and expansions in record length, the user must:



- a. Unload the affected data set
- b. Define the physical changes to the DBGEN
- c. Reload the data

The following procedures are necessary to accomplish these changes:

- a. "RDNXT" the variable entry file requesting the control fields and all data portions of the record. Do not request the linkage fields. Write each record to a sequential file.
- b. Zero out the linkage path to this variable entry file from the single entry file(s) to which it links. For example:



You would "RDNXT" through the CUST file requesting the elements CUSTCTRL and CUSTLK01, move zeros to this 8-character linkage path, and issue a WRITM. This would also be done for the ITEMLK01 element of the ITEM file.

- c. Redefine the DBGEN with the new changes to the variable entry file.

NOTE: Be sure to change the physical entries within the DBGEN for this file.

- d. Reformat the data set changing FORMAT input to reflect the changes to the data set.
- e. Read the sequential file created in step a and issue an ADDVC; this relinks the variable entry records to the single entry data set(s). The ADDVC should be issued using the same base control as was used to originally load the file. This ensures that the records within a chain are maintained in the same sequence as they were prior to the reload.

7.9 USING THE UTILITY PROGRAMS

TOTAL files can be copied between RMD and magnetic tape using either one of the following utility routines:

IOUTIL
FMUTIL

IOUTIL is a general purpose utility program that allows the user to dump files from one media to another e.g., tape-to-disc, disc-to-disc, etc. FMUTIL is a file maintenance utility program which may be used to dump or load files, singly or by entire partition, to or from a magnetic tape. Both these programs handle multi-reel files. For details as to the usage of these routines, refer to the VORTEX II reference manual.

7.10 DEBUGGING TOTAL DBMS

TOTAL permits a large degree of flexibility for the program to manipulate and handle its own data, but when an obvious mistake has occurred, TOTAL will not allow the invalid command on the erroneous data to change the data base. The command will be terminated, its effect nullified if partially completed, and an error status returned to the program. Some of the specific error checks include:

- a. An attempt to add a record with an invalid blank control key
- b. An attempt to add a duplicate master
- c. A requested file or element does not exist in the data base
- d. An incorrect file type has been specified
- e. A master delete has been attempted before all variable records are deleted.

There are also numerous internal checks to ensure that files are opened properly, locked correctly, and that all network linkage paths are maintained in the proper manner.

In nearly all situations, TOTAL will trap a user error or oversight, stop the command, and return an appropriate error status code. These status returns are a very useful debugging tool, and they are covered again under that topic.

7.10.1 Status Codes

After execution of any DML command, a status code is moved to the user's status field to indicate the result of the operation.



The status codes fall into three categories:

- a. Successful completion.
- b. Informative; some user action may be required.
- c. Fatal; the requested function has not been completed.

The status codes thus aid the programmer in debugging and indicate the action to be taken if there is a problem. A detailed explanation of the interpretation of the status codes and a complete list is given in appendix B.

7.10.2 Diagnostics

If there are errors in the Data Base Definition Language statements which would cause an erroneous Data Base Descriptor Module, the DBGEN program will print appropriate diagnostic messages along with the DBDL statement listing, and the output of source statements will be suppressed.

Certain error conditions will cause immediate termination of the DBGEN program, while other conditions allow further processing of the DBDL input. All output messages are output to the LO logical unit. The programmer can use these diagnostics to debug the input to the DBGEN program. A complete list of the diagnostic error messages is given in appendix B.

7.10.3 Automatic SNAPSHOT (SNAP) Dump

Another method of diagnosing errors in the TOTAL DBMS is by using the SNAPSHOT dump. This dump is output on the DO logical unit automatically by TOTAL when certain FATAL errors occur. The dump will contain the TEXT block which includes a complete parameter, file, and record list as well as some other TOTAL pointers and indicators. By searching the TEXT block for clues to errors in the program, the programmer will be able to determine which statements have caused the program malfunction to occur.

A description and sample of the TEXT block is given in appendix C.



varian data machines



APPENDIX A SAMPLE APPLICATION PROGRAM

A.1 PROBLEM DESCRIPTION

The following example will demonstrate the use of the TOTAL data management system. Assume we are to design a data base for a distributor, named X Company which purchases items in large quantities from large manufacturers and sells them in smaller quantities to local companies.

The X Company wants to develop a TOTAL information system to accommodate the following applications:

- a. Order Acknowledgement, Writing and Control.
- b. Invoicing
- c. Finished Inventory Control
- d. Purchase Order Writing and Control
- e. Accounts Receivable

We will follow the recommended steps in designing the data base module and the resulting applications. The steps are as follows:

- a. Determine the data sets required by the application to be developed.
- b. Categorize the data sets as to Single Entry or Variable Entry.
- c. Determine the data elements and data fields required in the established data sets.
- d. Develop a data base schematic using squares to represent single entry data sets and circles for variable entry data sets.
- e. Indicate the desired relationships by drawing connectors from single entry to variable entry data sets. Qualify the relationships as to ALL or selected record codes.
- f. Code the TOTAL Data Base Generation statements.
- g. Generate the data base module.
- h. Develop application programs.



A.2 DATA SET DEFINITION AND CLASSIFICATION

A.2.1 Determining the Required Data Sets

Using the sample applications, we determine that the following data sets are required:

- a. Customers
- b. Vendors
- c. Inventory Items
- d. Order Numbers
- e. Dates
- f. Open Customer Orders
- g. Open Purchase Orders
- h. Open Accounts Receivable

A.2.2 Categorizing Data Sets as to Single Entry or Variable Entry

Single Entry data sets are created to reflect the company's assets, to serve as entries into a pool of information, and to automatically edit identifying information.

Variable Entry data sets are created as a result of the recognition of business transactions. They indicate and monitor the effect of transactions on the single entry data sets while a business function is being performed. For example, a customer order is a transaction which inter-acts with customer and inventory information as the physical functions of allocating stock and shipping the order take place within the company. Eventually, the customer order generates another variable entry data set (Accounts Receivable) which shows the interaction of customer, invoices, and cash payments.

Using the sample X Company, we categorize the data sets as follows:

Single entry:

- a. Customer (CUST)
- b. Vendor (VEND)
- c. Inventory Item (INVT)



- d. Order Numbers (ORNM)
- e. Dates (DATE)

Variable Entry:

- a. Customer Orders (CORD)
- b. Purchase Orders (PORD)
- c. Accounts Receivable (ACCR)

A.3 DEVELOPING DATA BASE RELATIONSHIP SCHEMATICS

To determine the data base relationships required in the data base, we must set the objectives of the TOTAL information system for Company X. We want to use the computer for more than record keeping and document preparation. We want the system to process orders from our customers as they are received and to assist us in proper allocation of our inventories. We want the system to present management, immediately, with all information pertinent to solving business exceptions. For example, if orders for an item exceed its availability, we want to know all inhouse and unshipped demands against that particular item so that we can allocate according to the importance of the customer demands.

We want to be able to identify immediately all open purchase orders outstanding for a particular item in the event of excessive demands on our on hand stock.

The following is a summary of the relationships we will require:

- a. Customers to their orders to us
- b. Vendors to our orders to them
- c. All orders received on a particular day
- d. All orders to ship on a particular day
- e. Detail to all orders
- f. Customers to their open invoices and cash payments
- g. Items to open customer orders
- h. Items to open purchase orders

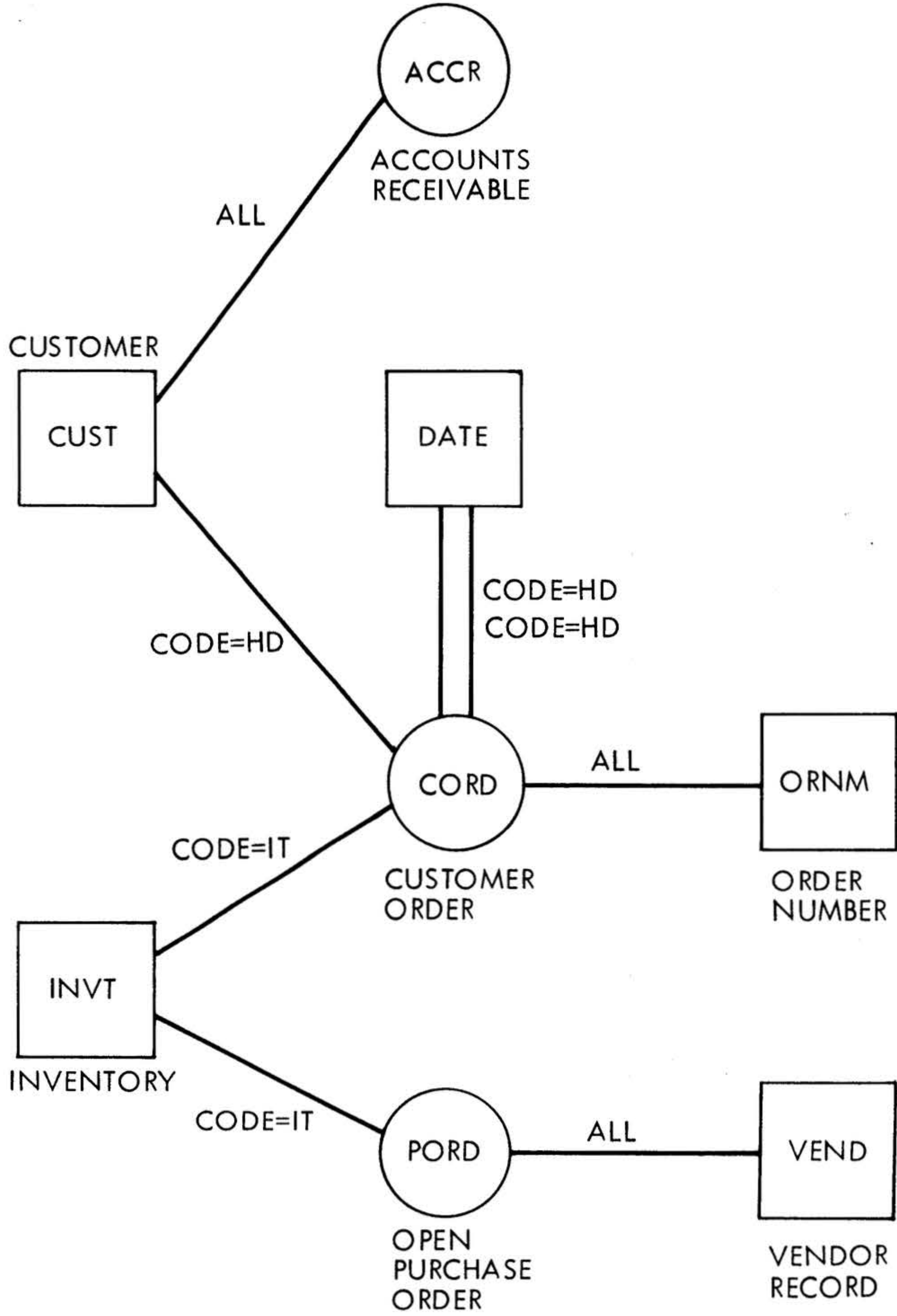


A.4 DATA BASE SAMPLE PROBLEM SCHEMATIC

A schematic of a data base sample problem is shown in figure A-1.

A.5 DETERMINING THE DATA RECORDS AND DATA ELEMENTS REQUIRED

Using the data sets developed and the information obtained by studying the business functions of Company X, the information requirements presented in figure A-2 are developed.



VTI1-3486

Figure A-1. Data Base Sample Problem



1. CUSTOMER RECORD - SINGLE ENTRY CUST

				← CUSTDATA →			
data:	CUSTROOT	CUSTCTRL customer number	CUSTLKCO link to CORD	CUSTLKAR link to ACCR	CUSTNAME name	CUSTADDR address	CUSTCTYS city/state
bytes:	8	6	8	8	30	30	20

2. VENDOR RECORD - SINGLE ENTRY VEND

				← VENDDATA →			
data:	VENDROOT	VENDCTRL vendor number	VENDLKPO Link open purchase order	VENDNAME vendor name	VENDADDR vendor address	VENDCTYS city/state	VENDLTME vendor lead time
bytes:	8	7	8	25	30	30	5

VT11-3487

Figure A-2. Information Requirements (Sheet 1 of 5)

Figure A-2. Information Requirements (Sheet 2 of 5)

VTI2-455

3. INVENTORY ITEM RECORD - SINGLE ENTRY INVT

					← INVTDAT1 → 40		← INVTDAT2 → 8		
	INVTROOT	INVTCTRL	INVTLKCO	INVTLKPT	INVTDESC	INVT COST	INVT PRIC	INVT ONHD	INVT ORDR
data:		Inventory item number	Link to customer items CORD	Link to open vendor items VEND	Item description	Item cost	Item price	Item on hand	Item on order
bytes:	8	8	8	8	30	5	5	4	4

4. ORDER NUMBER RECORD - SINGLE ENTRY ORNM

	ORNMROOT	ORNMCTRL	ORNMLKCO
data:		order number	Line to order records CORD
bytes:	8	12	8

5. DATE RECORD - SINGLE ENTRY DATE

	DATEROOT	DATECTRL	DATELKDR	DATELKDS
data:		date	Line to date received	Line to date to ship
bytes:	8	4	8	8



6. CUSTOMER ORDER - VARIABLE ENTRY CORD (RECORD CODES HD, IT, CM)

	BASE				REDEFINED								
	Record Code	Key 1	Link Path	Data	Key 2	Key 2	Key 3	LP1	LP2	LP3	Data		
data:	CORDCODE 'HD'	CORDORNM order number	to ORNM	CORDLINE line on order	CORDCUST customer record	CORDDATR date received	CORDDATS date to ship	CUSTLKCO via key 1	DATEUKDR via key 2	DATELKDS via key 3	CORDVALE value	CORDTOTI total	total # of items
bytes:	2	12	8	2	6	4	4	8	8	8	5	2	10

	Base				Redefined				
	Record Code	Key	Link Path	Data	Key 2	LP2			
data:	record code 'IT'				CORDITEM	INVTLKCO	CORDQTYX quantity	CORDPRCE price	CORDSLBS shipping weight
bytes:	2	12	8	2	8	8	4	5	4

	Base				Redefined
	Record Code	Key	Link Path	Data	
data:	record code CM				CORDCOMT comment record
bytes:	2	12	8	2	55

VTI2-456

Figure A-2. Information Requirements (Sheet 3 of 5)

VTI2-457

Figure A-2. Information Requirements (Sheet 4 of 5)

7. ACCOUNTS RECEIVABLE - VARIABLE ENTRY ACCR (RECORD CODES BL, CK)

Base				Redefined, ACCRDAT1					
Record Code	Key	Link Path	Data						
data: ACCRCODE	ACCRCUST	CUSTLKAR	ACCRSEQS	ACCRINUM	ACCRNDAT	ACCRGDAT	ACCRNAMT	ACCRGMAT	ACCRPAID
Code 'BL'	Customer Control	Link to CUST	Sequence	Invoice Number	Net, due date	Gross due date	Net amount	Gross amount	Amount paid
bytes: 2	6	8	2	6	6	6	5	5	5

Base				Redefined			
Same as above							
data: Code 'CK'				ACCRINO	ACCRCHKN	ACCRDREC	ACCRCAMT
				Invoice number	Check number	Date received	Check amount
bytes: 2	6	8	2	6	6	6	5



8. PURCHASE ORDER - VARIABLE ENTRY PORD (RECORD CODES HD. IT)

	Record Code	Key	LP		Redefined			
					PORDDAT1			
data:	PORDCODE 'HD'	PORDVEND Order Vendor	VENDLKPO Link to VEND	PORDPNUM Purchase Order No.	PORDLINE Purchase Order Line No.	PORDRDAT Released date	PORDDUDT Due date	PORDCARR Order carrier
bytes:	2	7	8	5	2	6	6	15

	Same as above				Redefined			
					Key 2	LP2	Data	
data:	Code 'IT'			PORDPNUM Purchase Order No.	PORDLINE Purchase Order Line No.	PORDITEM Item number	INVTLKPT Link to ITEM	PORDIQTY Item quantity
bytes:	2	7	8	5	2	8	8	5

VT11-3491

Figure A-2. Information Requirements (Sheet 5 of 5)



A.6 DATA BASE GENERATION

The following is an output listing from the data base generation program:

PAGE 1

CINCOM SYSTEMS, INC. DATA BASE GENERATION

```
BEGIN=DATA=BASE=GENERATION
DATA=BASE=NAME=ORDRDB
OPTIONS=OUTPUT=Y
SHARE=IO
IUAREA=MAS1=2
IUAREA=VAR1=4
IUAREA=MAS2=6
IUAREA=MAS3
END=IO
```

PAGE 2

CINCOM SYSTEMS, INC. DATA BASE GENERATION

```
BEGIN=MASTER=DATA=SET
DATA=SET=NAME=CUST
IUAREA=MAS1
MASTER=DATA
CUSTROOT=8
CUSTCTRL=6
CUSTLKCO=8
CUSTLKAR=8
CUSTDATA=80
.1.CUSTNAME=30
.1.CUSTADDR=30
.1.CUSTCTYS=20
END=DATA
LUGICAL=RECORD=LENGTH=120
TOTAL=LOGICAL=RECORDS=1118
DRIVE=11,600
END=MASTER=DATA=SET
LOGICAL=RECORDS=PER=SECTOR= 2
TOTAL=SECTORS= 559
UNUSED SECTORS IN LUN 11
```

CUSTOMER NUMBER		
LINK TO CUSTOMER ORDER HEADER RECORDS		
LINK TO INVOICES AND PAYMENTS		
NAME		
ADDRESS		
CITY AND STATE		
* CALCULATED *		
* CALCULATED *		
SECTORS		41



CINCOM SYSTEMS, INC. DATA BASE GENERATION

BEGIN=MASTER=DATA=SET
 DATA=SET=NAME=VEND
 IOAREA=MAS2
 MASTER=DATA
 VENDROOT=8
 VENDCTRL=7
 VENDLKPO=8
 VENDDATA=90
 .1.VENDNAME=25
 .1.VENDADDR=30
 .1.VENDCTYS=30
 .1.VENDLTME=5
 END=DATA

VENDOR NUMBER
 LINK TO OPEN PURCHASE ORDER

LOGICAL=RECORD=LENGTH=120
 TOTAL=LOGICAL=RECORDS=5586
 DRIVE=56,2800

END=MASTER=DATA=SET		
LOGICAL=RECORDS=PER=SECTOR=	2	* CALCULATED *
TOTAL=SECTORS=	2793	* CALCULATED *
UNUSED SECTORS IN	LUN	SECTORS
	56	7



CINCOM SYSTEMS, INC. DATA BASE GENERATION

BEGIN=MASTER=DATA=SET
 DATA=SET=NAME=INVT
 IUAREA=MAS1
 MASTER=DATA
 INVTRQOT=8
 INVTCTRL=8
 INVTLKPT=8
 INVTLKCO=8
 INVTDAT1=40
 .1.INVTDESC=30
 .1.INVTCOST=5
 .1.INVTPRIC=5
 INVTDAT2=8
 .1.INVTONHD=4
 .1.INVTORDR=4
 END=DATA

INVENTORY ITEM NUMBER

LOGICAL=RECORD=LENGTH=80
 TOTAL=LOGICAL=RECORDS=3198
 DRIVE=56,1200
 END=MASTER=DATA=SET

LOGICAL=RECORDS=PER=SECTOR=	3	* CALCULATED *
TOTAL=SECTORS=	1066	* CALCULATED *
UNUSED SECTORS IN	LUN	SECTORS
	56	134

CINCOM SYSTEMS, INC. DATA BASE GENERATION

BEGIN=MASTER=DATA=SET
 DATA=SET=NAME=ORNM
 IUAREA=MASS
 MASTER=DATA
 ORNMROOT=8
 ORNMCTRL=12
 ORNMLKCO=8
 END=DATA

ORDER NUMBER
 LINK TO ORDER RECORDS

LOGICAL=RECORD=LENGTH=30
 TOTAL=LOGICAL=RECORDS=4611
 DRIVE=56,600
 END=MASTER=DATA=SET

LOGICAL=RECORDS=PER=SECTOR=	8	* CALCULATED *
TOTAL=LOGICAL=RECORDS=	4616	* CALCULATED *
TOTAL=SECTORS=	577	* CALCULATED *
UNUSED SECTORS IN	LUN	SECTORS
	56	23



CINCOM SYSTEMS, INC. DATA BASE GENERATION

BEGIN=MASTER=DATA=SET
 DATA=SET=NAME=DATE
 IUAREA=MAS2
 MASTER=DATA
 DATEROOT=8
 DATECTRL=4
 DATELKDR=8
 DATELKDS=8
 END=DATA

DATE YYMMDD5 PACKED
 LINK TO DATE RECEIVED
 LINK TO DATE TO SHIP

LOGICAL=RECORD=LENGTH=30
 TOTAL=LOGICAL=RECORDS=1755
 DRIVE=133,300
 END=MASTER=DATA=SET

LOGICAL=RECORDS=PER=SECTOR=	8	* CALCULATED *
TOTAL=LOGICAL=RECORDS=	1760	* CALCULATED *
TOTAL=SECTORS=	220	* CALCULATED *
UNUSED SECTORS IN	LUN	SECTORS
	133	80



CINCOM SYSTEMS, INC. DATA BASE GENERATION

BEGIN=VARIABLE-ENTRY-DATA-SET

DATA-SET-NAME=PORD

IOAREA=VAR1

BASE=DATA

PURDCODE=2

RECORD CODES HD,IT

PUROVEND=7

BASE VENDOR NUMBER

VENDLKPO=8=PORDVEND

LINK TO VEND

PURDPNUM=5

PD NUMBER

PURDLINE=2

LINE NUMBER

PURDDAT1=27

VARIABLE DATA

RECORD-CODE=HD

REDEFINITION PART STATS HERE

.1.PORDRDAT=6

.1.PORDDUDI=6

.1.PORDCARR=15

RECORD-CODE=IT

.1.PORDITEM=8

INVTLKPT=8=PORDITEM

.1.PCRDTQTY=5

END=DATA

LOGICAL-RECORD-LENGTH=60

TOTAL-LOGICAL-RECORDS=20000

DRIVE=133,5200

END=VARIABLE-ENTRY-DATA-SET

LOGICAL-RECORDS-PER-SECTOR=

4

* CALCULATED *

TOTAL-SECTORS=

5000

* CALCULATED *

UNUSED SECTORS IN

LUN

SECTORS

133

200



CINCOM SYSTEMS, INC. DATA BASE GENERATION

BEGIN-VARIABLE-ENTRY-DATA-SET

DATA-SET-NAME=CORD

IDAREA=VAR1

BASE=DATA

CURCODE=2

RECORD CODES HD,CM,IT

CURDORN=12

BASE ORDER NUMBER

CURDLIN=2

LINE ON ORDER

OKNMLKCO=8=CORDORN

CURDDATA=55

VARIABLE DATA AREA

RECORD-CODE=HD

ONE PER ORDER

.1.CORDCUST=6

CUSTOMER

.1.CORDDATR=4

DATE RECEIVED

.1.CORDDATS=4

DATE TO SHIP

CUSTLKCO=8=CORDCUST

DATELKDR=8=CORDDATR

DATELKDS=8=CORDDATS

.1.CORDVALE=5

TOTAL VALUE OF ORDER

.1.CORDTOT1=2

TOTAL NUMBER OF ITEMS

.1.CORDTERM=10

TERMS OF ORDER

RECORD-CODE=IT

.1.CORDITEM=8

ITEM REQUIRED

INVTLKCO=8=CORDITEM

.1.CORDQTYX=4

QUANTITY

.1.CORDPRCE=5

SPECIAL PRICE

.1.CORDSLBS=4

SHIPPING WEIGHT

RECORD-CODE=CM

.1.CORDCOMT=55

COMMENT

END=DATA

LOGICAL-RECORD-LENGTH=80

TOTAL-LOGICAL-RECORDS=98000

DRIVE=132,32000

DRIVE=133,1000

END-VARIABLE-ENTRY-DATA-SET

LOGICAL-RECORDS-PER-SECTOR=

3

* CALCULATED *

TOTAL-LOGICAL-RECORDS=

98001

* CALCULATED *

TOTAL-SECTORS=

32667

* CALCULATED *

UNUSED SECTORS IN

LUN

SECTORS

133

333



CINCOM SYSTEMS, INC. DATA BASE GENERATION

BEGIN-VARIABLE-ENTRY-DATA-SET

DATA-SET-NAME=ACCR

IUAREA=VAR1

BASE=DATA

ACCRCODE=2

RECORD CODES BL,CK

ACCRUST=6

BASE CUSTOMER CONTROL

CUSTLKA=8=ACCRUST

ACCRSEQS=2

ACCRDAT1=33

RECORD-CODE=BL

.1.ACCRINUM=6

.1.ACCRNDAT=6

.1.ACCRGDAT=6

.1.ACCRNMAT=5

.1.ACCRGMAT=5

.1.ACCRPATD=5

RECORD-CODE=CK

CHECKS

.1.ACCRRIND=6

.1.ACCRCHKN=6

.1.ACCRDREC=6

.1.ACCRCAMT=5

END-DATA

LOGICAL-RECORD-LENGTH=60

TOTAL-LOGICAL-RECORDS=37737

DRIVE=56,10000

END-VARIABLE-ENTRY-DATA-SET

LOGICAL-RECORDS=PER-SECTOR=

4

* CALCULATED *

TOTAL-LOGICAL-RECORDS=

37740

* CALCULATED *

TOTAL-SECTORS=

9435

* CALCULATED *

UNUSED SECTORS IN

LUN

SECTORS

56

565

EPILOGUE

END-DATA-BASE-GENERATION



A.7 RPG II SAMPLE PROGRAM ORDPRO

The RPG II program listed on the following pages shows how TOTAL is used to add order transactions into an order entry system using "ORDRDB" as its DBMOD. The order transactions are input in no sequence. This program (a) inserts transactions in sequence using the READV, ADDVB, and ADDVC commands, (b) updates transactions using the READV and WRITV commands, and (c) creates a new order number master record using the ADD-M command.

A customer order record can be one of three types. It may be a header record, record code = HD; an item record, record code = IT; or a comment record, record code = CM. The order transactions are input through the card reader. The Input Specifications define the layout for the three types of transaction records. The first two columns of each input record contain the record code. However, the first record will contain "SINON" in columns 1-5, and columns 6-27 will contain some of the data necessary to "sign-on" the data base. The last card must contain "SINOF" in columns 1-5 which indicates that the data base should be "signed-off."

The first task this program performs is to execute the subroutine "INIT." This builds the proper schema in order that the SINON function may be executed. It also builds all data-lists that will be needed in this program. This subroutine will be executed only once. This program uses five of the files defined in the data base - CUST, INVT, DATE, CORD, and ORNM. Since these data sets will be updated, each data set must use "PRIV" as its usage mode. The data, which TOTAL uses to "sign-on" the data base, is established by source statements 143-149 of this program. Since the result fields are defined one after the other, the "sign-on" data will be contained in a contiguous area of core storage which is a requirement. Once all files have been opened by TOTAL, transaction processing may begin. All input transaction records are added to the variable entry CORD file which will be linked to all associated single entry files. The data-area which contains the input data as defined by the data-list is defined in this program in the Input Specifications by the field ORDREC. The data-area, in which TOTAL places the data it retrieves from a particular record, is a 4-byte area which is defined in this program in statement numbers 138-139. Also included in this "dummy section," statement numbers 136-139, is where the Data Base Descriptor Module for this program is defined. This section of the program should never be executed.



```

M
ORDPR()

RG 004
0001 FCDL  IPF F 80 80 READ42
0002 FPRFL 0 F 132 PRINTER
C*****
C***** THE INPUT SPECIFICATIONS DEFINE THE POSSIBLE *****
C***** CARD FORMATS THAT MAY BE INPUTTED *****
0003 ICDFL AA 01 1 CH 2 CD
0004 I 1 2 RECCD
0005 I 3 14 ORDNUM
0006 I 15 16 LNNUM
0007 I 17 22 CUSNUM
0008 I 23 26 DATREC
0009 I 27 30 DATSHP
0010 I 31 35 ORDVAL
0011 I 36 37 TOTITM
0012 I 38 47 DTERMS
0013 I 1 79 ORDREC
0014 I BB 02 1 CI 2 CT
0015 I 1 2 RECCD
0016 I 3 14 ORDNUM
0017 I 15 16 LNNUM
0018 I 17 24 ITMNUM
0019 I 25 28 ITMQNT
0020 I 29 33 ITMPRI
0021 I 34 37 ITMWGT
0022 I 1 79 ORDREC
0023 I CC 03 1 CC 2 CM
0024 I 1 2 RECCD
0025 I 3 14 ORDNUM
0026 I 15 16 LNNUM
0027 I 17 71 COMM
0028 I 1 79 ORDREC
0029 I DD 04 1 CS 2 CI 3 CN
0030 I AND 4 CO 5 CV
0031 I OR 05 1 CS 2 CI 3 CN
0032 I AND 4 CO 5 CF
0033 I 1 5 CCCD
0034 I 6 27 SCHEMA
0035 I 1 79 ORDREC
C*****
C*****
C***** THE FOLLOWING SECTION PRINTS EACH INPUT CARD *****
C***** AND DETERMINES, BY THE INDICATOR *****
C***** SETTINGS, WHICH FUNCTION *****
C***** IS TO BE PERFORMED *****
0036 C SETOF 21
0037 C SETOF 98
0038 C SFTOF 99
0039 C EXSR CDPRT
0040 C MOVE ' ' STATS 4
0041 C 04 GOTO SINON
0042 C 05 GOTO SINOF
0043 C MOVE 'LKCD' REFER 4
0044 C GOTO FNDPOS
C*****
C*****

```



```

C***** THE FOLLOWING SECTION EITHER SIGNS-ON OR *****
C***** SIGNS-OFF THE DATA BASE. *****
0045 C          SINON      TAG
0046 C          EXSR INIT
0047 C          SINOF      TAG
0048 C          EXIT DATBAS
0049 C          RLABL      CDCD
0050 C          RLABL      STATS
0051 C          RLABL      REALM
0052 C          RLABL      ENDP
0053 C          STATS     COMP '*****'          21
0054 C    21          GOTO END
0055 C          SETON      LR
0056 C          GOTO END

C*****
C*****
C***** THE FOLLOWING SECTION READS THE VARIABLE 'CORD' *****
C***** FILE. IT ALSO DETERMINES WHETHER THE *****
C***** CARD INPUT SHOULD MODIFY, ADD *****
C***** BEFORE, OR ADD AFTER THE *****
C***** RECORD READ FROM 'CORD'. *****
0057 C          FNDPOS     TAG
0058 C          MOVE 'READV'  FUNCT  5
0059 C          EXIT DATBAS
0060 C          RLABL      FUNCT
0061 C          RLABL      STATS
0062 C          RLABL      CORD
0063 C          RLABL      REFER
0064 C          RLABL      ORNMLK
0065 C          RLABL      ORDNUM
0066 C          RLABL      ELEMI
0067 C          RLABL      LINENO
0068 C          RLABL      ENDP
0069 C          MOVE CORD    DATAST  4
0070 C          STATS     COMP 'MRNF'          20
0071 C    20          GOTO ADDMAS
0072 C          STATS     COMP '*****'          21
0073 C    N21        SETON          99
0074 C    N21        GOTO FND
0075 C          REFER     COMP 'END.'          22
0076 C    22          MOVE 'LKCO'    REFER
0077 C    22          MOVE 'ADDVC'   FUNCT
0078 C    22          GOTO ADDVAR
0079 C          LINENO    COMP LNNUM          23
0080 C    23          MOVE 'ADDVB'   FUNCT
0081 C    23          GOTO ADDVAR
0082 C          LINENO    COMP LNNUM          24
0083 C    24          MOVE 'WRITV'   FUNCT
0084 C    24          GOTO MODIFY
0085 C          GOTO FNDPOS

C*****
C*****
C***** THE FOLLOWING SECTION EITHER EXECUTES THE *****
C***** 'ADDVA', 'ADDVB', OR 'ADDVC' *****
C***** TOTAL COMMAND. *****
0086 C          ADDVAR     TAG
0087 C          EXIT DATBAS
0088 C          RLABL      FUNCT
0089 C          RLABL      STATS
0090 C          RLABL      CORD
0091 C          RLABL      REFER

```



```

0092      C          RLABL          ORNMLK
0093      C          RLABL          ORDNUM
0094      C          RLABL          ECORCD
0095      C          RLABL          ORDREC
0096      C          RLABL          ENDP
0097      C          MOVE CURD      DATAST
0098      C          STATS          COMP '*****'          21
0099      C  N21          SETON          99
0100      C          GOTO END
C*****
C*****
C***** THE FOLLOWING SECTION MODIFIES THE RECCRD READ *****
C***** FROM 'CORD' BY USING THE 'WRITE' COMMAND. *****
0101      C          MODIFY          TAG
0102      C          CODENO          COMP RECCD          25
0103      C  N25          SETON          98
0104      C  N25          GOTO END
0105      C          EXIT DATBAS
0106      C          RLABL          FUNCT
0107      C          RLABL          STATS
0108      C          RLABL          CORD
0109      C          RLABL          REFER
0110      C          RLABL          ORNMLK
0111      C          RLABL          ORDNUM
0112      C          RLABL          ECORCD
0113      C          RLABL          ORDREC
0114      C          RLABL          ENDP
0115      C          MOVE CORD      DATAST
0116      C          STATS          COMP '*****'          21
0117      C  N21          SETON          99
0118      C          GOTO END
C*****
C*****
C***** THE FOLLOWING SECTION ADDS A NEW 'ORNM' MASTER *****
C***** RECORD IF ONE DOES NOT EXIST FOR THE *****
C***** VARIABLE 'CORD' RECORD TO BE ADDED. *****
0119      C          ADDMAS          TAG
0120      C          MOVE 'ADD-M'    FUNCT
0121      C          EXIT DATBAS
0122      C          RLABL          FUNCT
0123      C          RLABL          STATS
0124      C          RLABL          ORNM
0125      C          RLABL          ORDNUM
0126      C          RLABL          EORNCT
0127      C          RLABL          ORDNUM
0128      C          RLABL          ENDP
0129      C          MOVE ORNM      DATAST
0130      C          STATS          COMP '*****'          21
0131      C  N21          SETON          99
0132      C  N21          GOTO END
0133      C          MOVE 'LKCO'     REFER
0134      C          MOVE 'ADDVC'    FUNCT
0135      C          GOTO ADDVAR
C*****
C*****
C***** THE FOLLOWING SECTION IS A DUMMY SECTION USED *****
C***** TO DEFINE THE DATA-AREA AS WELL AS TO *****
C***** SPECIFY WHICH DBMOD IS TO BE USED. *****
C***** THIS SECTION SHOULD NEVER *****
C***** BE EXECUTED. *****
0136      C          DUMMY          TAG

```



```

0137 C EXIT SUBROK
0138 C MOVE ' ' LINENO 2
0139 C MOVE ' ' CODENO 2
C *****
C *****
0140 C END TAG
0141 CLR EXCPT
C *****
C *****
C ***** THE FOLLOWING SECTION IS EXECUTED ONCE TO *****
C ***** INITIALIZE DATA FIELDS NECESSARY FOR *****
C ***** THE EXECUTION OF THE TOTAL *****
C ***** COMMANDS IN THE PROGRAM. *****
0142 CSR INIT BEGSR
0143 CSR MOVE SCHEMA REALM 22
0144 CSR MOVFL'CUSTPRIV'FILE1 12
0145 CSR MOVFL'INVTPRIV'FILE2 12
0146 CSR MOVFL'DATEPRIV'FILE3 12
0147 CSR MOVFL'CORDPRIV'FILE4 12
0148 CSR MOVFL'ORNMPRIV'FILE5 12
0149 CSR MOVE 'END.' ENDP 4
0150 CSR MOVE 'ORNMLKCO'ORNMLK 8
0151 CSR MOVE 'CORD' CORD 4
0152 CSR MOVE 'ORNM' ORNM 4
0153 CSR MOVE 'CORDLINE'ELEM1 8
0154 CSR MOVE 'CORDCODE'ELEMB 8
0155 CSR MOVE 'END.' ELEM1 4
0156 CSR MOVE 'CORDCODE'ECORCO 8
0157 CSR MOVE 'CORDORNM'FCORON 8
0158 CSR MOVE 'CORDLINE'FCORLN 8
0159 CSR MOVE 'CORDDATA'ECORDA 8
0160 CSR MOVE 'END.' EENDP 4
0161 CSR MOVE 'ORNMCtrl'EORNCT 8
0162 CSR MOVE 'END.' UENDP 4
0163 CSR ENDSR
0164 CSR C DPRNT BEGSR
0165 CSR SETON 90
0166 CSR EXCPT
0167 CSR SETOF 90
0168 CSR ENDSR
C *****
C *****
0169 DPRFL E 1 90
0170 O ORDREC 79
0171 O 100 'INPUT RECORD'
0172 DPRFL D 1 21
0173 O STATS 4
0174 O 50 'SUCCESSFUL COMPLETION'
0175 O D 1 99
0176 O STATS 4
0177 O 25 'UNRECOVERABLE ERROR,'
0178 O 35 'FUNCT WAS '
0179 O FUNCT 41
0180 O 54 'DATA SET WAS'
0181 O DATAST 59
0182 O D 1 98
0183 O 23 'INVALID LINE CHANGE -- '
0184 O 42 'ORDER NUMBER IS - '
0185 O ORDNUM 50
0186 O 65 ' , CODES ARE - '
0187 O CODENO 67

```



```
0188      0              72 ' AND '  
0189      0              RECCD   74  
0190      0      F I      LR  
0191      0              STATS   4  
C*****  
C*****
```



A.8 COBOL SAMPLE PROGRAM

A COBOL sample program is provided on the following pages.

VARIAN DATA V-75 COBOL

```

1 000100 IDENTIFICATION DIVISION.
2 000200 PROGRAM-ID. ORDRPRG.
3 000300 REMARKS.
4 000400     THIS PROGRAM ILLUSTRATES THE USE OF TOTAL THE DATA
5 000500     BASE MANAGEMENT SYSTEM. TO ADD ORDER TRANSACTIONS
6 000600     INTO AN ORDER ENTRY SYSTEM. THE ORDER TRANSACTIONS
7 000700     ARE INPUTTED IN NO SEQUENCE. THIS PROGRAM -
8 000800
9 000900           1. INSERTS TRANSACTIONS IN SEQUENCE USING THE
10 001000           READV, ADDVC, OR ADDVB COMMANDS.
11 001100
12 001200           2. UPDATES TRANSACTIONS USING THE
13 001300           READV AND WRITV COMMANDS.
14 001400
15 001500           3. CREATES A NEW ORDER NUMBER MASTER RECORD
16 001600           USING THE ADD-M COMMAND.
17 001700*
18 001800 ENVIRONMENT DIVISION.
19 001900 CONFIGURATION SECTION.
20 002000 SOURCE-COMPUTER. V75.
21 002100 OBJECT-COMPUTER. V75.
22 002200 INPUT-OUTPUT SECTION.
23 002300 FILE-CONTROL.
24 002400     SELECT TRANS-IN ASSIGN TO READER.
25 002500*
26 002600 DATA DIVISION.
27 002700 FILE SECTION.
28 002800 FD TRANS-IN
29 002900     LABEL RECORDS ARE OMITTED
30 003000     DATA RECORDS ARE ORDER-RECORD,
31 003010     ORDER-RECORD-2,
32 003020     ORDER-RECORD-3.
33 003040 01 ORDER-RECORD.
34 003050     05 COMMON-DATA.
35 003060         10 RECORD-CODE     PIC X(02).
36 003070         10 ORDER-NUMBERX   PIC X(12).
37 003080         10 LINE-NUMBER     PIC 99.
38 003090     05 REDEFINED-AREA-1.
39 003100         10 HEADER-RECORD.
40 003110             15 CUSTOMER-NUMBER     PIC X(06).
41 003120             15 DATE-RECEIVED       PIC 9999.
42 003130             15 DATE-SHIP           PIC 9999.
43 003140             15 ORDER-VALUE        PIC 999V99.

```



VARIAN DATA V-75 COBOL

44	003150		15	TOTAL-LINES		PIC 99.
45	003160		15	ORDER-TERMS		PIC X(10).
46	003170		15	FILLER		PIC X(33).
47	003180	01		ORDER-RECORD-2.		
48	003190		05	FILLER		PIC X(16).
49	003200		05	REDEFINED-AREA-2.		
50	003210		10	ITEM-RECORD.		
51	003220		15	ITEM-NUMBER		PIC X(08).
52	003230		15	ITEM-QUANTITY		PIC 9(04).
53	003240		15	ITEM-PRICE		PIC 999V99.
54	003250		15	ITEM-WEIGHT		PIC 999V9.
55	003260		15	FILLER		PIC X(43).
56	003270	01		ORDER-RECORD-3.		
57	003280		05	FILLER		PIC X(16).
58	003290		05	REDEFINED-AREA-3.		
59	003300		10	COMMENT-RECORD.		
60	003310		15	COMMENT		PIC X(55).
61	003320		15	FILLER		PIC X(09).
62	005400	**				
63	005500			WORKING-STORAGE SECTION.		
64	005600	01		READV	PIC X(05)	VALUE 'READV'.
65	005700	01		ADD-M	PIC X(05)	VALUE 'ADD-M'.
66	005800	01		ADDVC	PIC X(05)	VALUE 'ADDVC'.
67	005900	01		ADDVB	PIC X(05)	VALUE 'ADDVB'.
68	006000	01		WRITV	PIC X(05)	VALUE 'WRITV'.
69	006100	01		STAT	PIC X(04)	VALUE '****'.
70	006200	01		ENDP	PIC X(04)	VALUE 'END.'.
71	006300	01		REFER	PIC X(04)	VALUE 'LKXX'.
72	006400	01		ORNMLKCU	PIC X(08)	VALUE 'ORNMLKCU'.
73	006500	01		CORD	PIC X(04)	VALUE 'CORD'.
74	006600	01		ORNM	PIC X(04)	VALUE 'ORNM'.
75	006700	01		SINON	PIC X(05)	VALUE 'SINON'.
76	006800	01		SINOF	PIC X(05)	VALUE 'SINOF'.
77	006900	01		SCHEMA.		
78	007000		07	TASKNAME	PIC X(08)	VALUE 'ORDRPGX'.
79	007100		07	DBMODULE	PIC X(06)	VALUE 'ORDRDB'.
80	007200		07	AKCESS	PIC X(06)	VALUE 'UPDATE'.
81	007300		07	LOGOPT	PIC X(02)	VALUE 'NL'.
82	007400		07	REALM.		
83	007500		09	FILE1	PIC X(12)	VALUE 'CUSTPRIV****'.
84	007600		09	FILE2	PIC X(12)	VALUE 'INVTPRIV****'.
85	007700		09	FILE3	PIC X(12)	VALUE 'DATEPRIV****'.
86	007800		09	FILE4	PIC X(12)	VALUE 'CORDPRIV****'.



VARIAN DATA V-75 COBOL

87	007900	09	FILES	PIC X(12) VALUE 'ORNMPRIV****'.
88	008000	09	TERMINATOR	PIC X(04) VALUE 'END.'.
89	008100	01	DATA-SETS	PIC X(04).
90	008200	01	FUNCT	PIC X(05).
91	008300	01	LOCATION-SEGS.	
92	008400	05	FILLER	PIC X(08) VALUE 'CORDCODE'.
93	008500	05	FILLER	PIC X(08) VALUE 'CORDLINE'.
94	008600	05	FILLER	PIC X(04) VALUE 'END.'.
95	008700	01	CORD-SEGS.	
96	008800	05	CURDCODE	PIC X(08) VALUE 'CORDCODE'.
97	008900	05	CORDORNM	PIC X(08) VALUE 'CORDURNM'.
98	009000	05	CORDLINE	PIC X(08) VALUE 'CORDLINE'.
99	009100	05	CORDDATA	PIC X(08) VALUE 'CORDDATA'.
100	009200	05	FILLER	PIC X(04) VALUE 'END.'.
101	009300	01	ORNM-SEGS.	
102	009400	05	ORNMCTRL	PIC X(08) VALUE 'ORNMCTRL'.
103	009500	05	FILLER	PIC X(04) VALUE 'END.'.
104	009600	01	LINE-CODE.	
105	009700	05	CODE-NO	PIC X(02).
106	009800	05	LINE-NO	PIC 99.
107	009900	01	ORDER-NUMBER	PIC X(12).



VARIAN DATA V-75 CUBUL

```

108 010000 PROCEDURE DIVISION.
109 010100
110 010200
111 010300 SIGN-UN-FUNCTION.
112 010400     ENTER ASSEMBLER.
113 010500     CALL 'DATBAS' USING SINON STAT SCHEMA ENDP.
114 010600     IF STAT = '****'
115 010700         GO TO SIGN-UN-OK.
116 010800     DISPLAY 'ERROR ON SINON,          STAT = ' STAT.
117 010900         GO TO THAIS-IT.
118 011000 SIGN-UN-OK.
119 011100     OPEN INPUT TRANS-IN.
120 011200 READ-TRAN-IN.
121 011300     READ TRANS-IN RECORD
122 011400         AT END GO TO END-OF-JOB.
123 011500     IF RECORD-CODE = 'HD' OR
124 011600         RECORD-CODE = 'IT' OR
125 011700         RECORD-CODE = 'CM'
126 011800         MOVE 'LKCO' TO REFER
127 011900     ELSE GO TO INVALID-RECORD.
128 012000     DISPLAY 'INPUT RECORD ' ORDER-RECORD.
129 012100     MOVE ORDER-NUMBERX TO ORDER-NUMBER.
130 012200 FIND-POSITION.
131 012300     ENTER ASSEMBLER.
132 012400     CALL 'DATBAS' USING READV STAT CORD REFER ORNMLKCU
133 012500         ORDER-NUMBER LOCATION-SEGS LINE-CODE ENDP.
134 012600     MOVE CORD TO DATA-SETS.
135 012700     MOVE READV TO FUNCT.
136 012800     IF STAT = 'MRNF'
137 012900         GO TO ADD-NEW-ORDER-NU.
138 013000     IF STAT NOT = '****'
139 013100         GO TO ERROR-ROUTINE.
140 013200     IF REFER = 'END.'
141 013300         MOVE 'LKCO' TO REFER
142 013400         GO TO ADDVC-ROUTINE.
143 013500     IF LINE-NU IS GREATER THAN LINE-NUMBER
144 013600         GO TO ADDVB-ROUTINE.
145 013700     IF LINE-NU = LINE-NUMBER
146 013800         GO TO CHANGE-THE-LINE.
147 013900     GO TO FIND-POSITION.
148 014000 ADDVC-ROUTINE.
149 014100     ENTER ASSEMBLER.
150 014200     CALL 'DATBAS' USING ADDVC STAT CORD REFER ORNMLKCU

```



VARIAN DATA V-75 CUBOL

```
151 014300 ORDER-NUMBER CORD-SEGS ORDER-RECORD ENDP.
152 014400 MOVE CORD TO DATA-SETS.
153 014500 MOVE ADDVC TO FUNCT.
154 014600 IF STAT NOT = '****'
155 014700 GO TO ERROR-ROUTINE.
156 014800 GO TO READ-TRAN-IN.
157 014900 ADDVB-ROUTINE.
158 015000 ENTER ASSEMBLER
159 015100 CALL 'DATBAS' USING ADDVB STAT CORD REFER ORNMLKCU
160 015200 ORDER-NUMBER CORD-SEGS ORDER-RECORD ENDP.
161 015300 MOVE CORD TO DATA-SETS.
162 015400 MOVE ADDVB TO FUNCT.
163 015500 IF STAT NOT = '****'
164 015600 GO TO ERROR-ROUTINE.
165 015700 GO TO READ-TRAN-IN.
166 015800**
167 015900 CHANGE-THE-LINE.
168 016000 IF CODE-NO NOT = RECORD-CODE
169 016100 GO TO INVALID-LINE-CHANGE.
170 016200 ENTER ASSEMBLER
171 016300 CALL 'DATBAS' USING WRITV STAT CORD REFER ORNMLKCU
172 016400 ORDER-NUMBER CORD-SEGS ORDER-RECORD ENDP.
173 016500 MOVE CORD TO DATA-SETS.
174 016600 MOVE WRITV TO FUNCT.
175 016700 IF STAT NOT = '****'
176 016800 GO TO ERROR-ROUTINE.
177 016900 GO TO READ-TRAN-IN.
178 017000 ADD-NEW-ORDER-NO.
179 017100 ENTER ASSEMBLER.
180 017200 CALL 'DATBAS' USING ADD-M STAT ORNM ORDER-NUMBER
181 017300 ORNM-SEGS ORDER-NUMBER ENDP.
182 017400 MOVE ORNM TO DATA-SETS.
183 017500 MOVE ADD-M TO FUNCT.
184 017600 IF STAT NOT = '****'
185 017700 GO TO ERROR-ROUTINE.
186 017800 MOVE 'LKCO' TO REFER
187 017900 GO TO ADDVC-ROUTINE.
188 018000 ERROR-ROUTINE.
189 018100 IF STAT = 'MRNF'
190 018200 GO TO INVALID-RECORD.
191 018300 DISPLAY 'UNRECOVERABLE ERROR, STAT =-STAT- FUNCT WAS'
192 018400 FUNCT 'DATA SET WAS' DATA-SETS.
193 018500 GO TO END-OF-JOB.
```



VARIAN DATA V-75 COBOL

```
194 018600 INVALID-LINE-CHANGE.  
195 018700     DISPLAY 'INVALID LINE CHANGE'.  
196 018800     DISPLAY 'ORDER NUMBER' ORDER-NUMBER.  
197 018900     DISPLAY 'LINE NUMBER' LINE-NUMBER.  
198 019000     DISPLAY 'CODES-CODE-NO-' RECORD-CODE.  
199 019100     GO TO READ-TRAN-IN.  
200 019200 INVALID-RECORD.  
201 019300     DISPLAY 'INVALID RECORD ' ORDER-RECORD.  
202 019400     GO TO READ-TRAN-IN.  
203 019500 END-OF-JOB.  
204 019600     CLOSE TRANS-IN.  
205 019700 SIGN-OFF-FUNCTIONS.  
206 019800     ENTER ASSEMBLER.  
207 019900     CALL 'DATBAS' USING SINOF STAT SCHEMA ENDP.  
208 020000     DISPLAY 'STAT ON SINOF = ' STAT.  
209 020100 THATS-IT.  
210 020200     STOP RUN.
```



A.9 FORTRAN SAMPLE PROGRAM

A FORTRAN sample program is provided on the following pages.

```

PAGE      1          CMFGSMP  VORTXT1 FTN IV

1          NAME CMFGSMP
2          *
3          *
4          *   - A SAMPLE PROGRAM ILLUSTRATING THE USE OF
5          *   THE 'DIAL' DATA BASE MANAGEMENT SYSTEM
6          *
7          *   - THE PROGRAM PERFORMS VARIOUS MAINTENANCE
8          *   AND RETRIEVAL FUNCTIONS
9          *
10         *   - THE DATA BASE UTILIZED IS TYPICAL OF A
11         *   SIMPLE MANUFACTURING ENVIRONMENT
12         *
13         * PART - PART MASTER DATA SET
14         * PILL - BILL OF MATERIAL VARIABLE DATA SET
15         * ROUT - ROUTING VARIABLE DATA SET
16         * WCTR - WORK CENTER MASTER DATA SET
17         *
18         *
19         *
20         *
21         *
22         *
23         *
24         *
25         *
26         *
27         *
28         *
29         *
30         *
31         *
32         *
33         *
34         *
35         *
36         *
37         *
38         *
39         *
40         *
41         *
42         *
          EXTERNAL DATBAS

```

```

PPPPPPPPPPPPPP          BRBRBR
P          P  LKBM/ALI  BR          BR
P          P-----R          B
P  PART  P          R  PILL  B
P          P-----R          B
P          P  LKWII/ALI  BR          BR
PPPPPPPPPPPPPP          BRBRBR
          T
          I T A
          K T L
          R T I
          T T
          T
          RRRRRR          WWWWWWWWWWWW
          RP          RP          W          W
P          R  LKRT          W          W
P  PULL  R-----W  WCTR  W
P          R  CUDF=SR          W          W
          RP          RR          W          W
          RRRRRR          WWWWWWWWWWWW

```



PAGE 2

CMFGSMP VORTXT1 FTN IV

```

43      DIMENSION KSCHFM(37)
44      DIMENSION CPART(7),CRILL(7),CWCTRL(5),CPOIII(9)
45      DIMENSION KPARTA(16),KBTLIA(11),KWCTRA(12),KROUTA(27)
46      DIMENSION KKEY(5), KFUNC(3), TPATH(2), KCKEY(5), KPKEY(5)
47      DIMENSION TRREF(10), KIRPRI(5,10), TIRQTY(10)
48      DIMENSION KXNUM(20), KINUM(20)
49      DIMENSION KARDX2(40), CARDX4(20)
50      EQUIVALENCE (KARDX2(1),CARDX4(1))
51      * * SCHEMA FOR SINON/SINOF * *
52      DATA KSCHFM/2HMF,2HGS,2HAM,2HPL,2HMF,2HGD,2HBS,2HIIP,2HDA,2HTE,
53      1          2HNI,2HPA,2HRT,2HPR,2HIV,2HXX,2HXX,
54      1          2HWC,2HIR,2HPR,2HIV,2HXX,2HXX,
55      1          2HRI,2HLL,2HPR,2HIV,2HXX,2HXX,
56      1          2HRU,2HUT,2HPR,2HIV,2HXX,2HXX,
57      1          2HEN,2HD./
58      * * ELEMENT LISTS * *
59      DATA CPARTL/4HPART,4HCTRL,4HPART,4HDESC,4HPART,4HLLCD,4HEND./
60      DATA CBTLIL/4HBTLI,4HPARN,4HBTLI,4HCOMP,4HBTLI,4HDTYP,4HEND./
61      DATA CWCTRL/4HCWCTR,4HCTRL,4HCWCTR,4HDESC,4HEND./
62      DATA CROUTL/4HROUT,4HCODE,4HROUT,4HPART,4HROUT,4HSEDN,
63      1          4HROUT,4HDATA,4HEND./
64      * * ELEMENT AREAS * *
65      DATA KPARTA/16*2H /, KRILLA/11*2H /
66      DATA KWCTRA/12*2H /, KPOIIIA/27*2H /
67      * * ERROR CODES, CONSTANTS, ETC * *
68      DATA CENDP/4HEND./, TSTAT/4HXXX/, TSTAR/4H***./
69      DATA TITLF/4HFFF/, TREFR/4HLKXX/, TQIAL/4HREGN/, SREFR/4HLKXX/
70      DATA CBFGN/4HREGN/, CLKRM/4HLKBM/, CLKWU/4HLKWU/, CLKRT/4HLKRT/
71      DATA CMRNF/4HMRNF/, CNSMR/4HNSMR/, CDUPM/4HDUPM/, CTMDL/4HIMDI/
72      DATA CCARD/4HCARD/, CCONS/4HCONS/, CASTR/4H* /
73      DATA KAD/2HAD/, KRD/2HRD/, KDI/2HDL/, KWR/2HWR/, KAL/2HAL/
74      DATA KMD/2HMD/, KSR/2HSB/, KSW/2HSW/, KIR/2HTB/, KIW/2HTW/
75      DATA KBP/2HRP/, KBW/2HRW/, KSR/2HSR/, KCM/2HCM/
76      DATA KDV/2HDV/, KC/2HC /, KR/2HR /, KSPAC/2H /
77      DATA KKEY/5*2H /, KFUNC/3*2H /, TPATH/4HFFF/, 4HLKXX/
78      DATA TRREF/10*4HLKXX/
79      DATA TIRQTY/10*0/, TQTY/0/, IHLLCD/0/
80      DATA KXNUM/2H01,2H02,2H03,2H04,2H05,2H06,2H07,2H08,2H09,2H10,
81      1          2H11,2H12,2H13,2H14,2H15,2H16,2H17,2H18,2H19,2H20/
82      DATA KINUM/1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20/
83      ITN=4
84      IOUT=5

```



```
PAGE      3          CMFGSMP  VORTXT1 FTN IV

85          WRITE (TOUT,1)
86  * * STNON TO TOTAL * *
87          CALL DATBAS(5HSTNON,TSTAT,KSCHM,CENDP)
88          IF (TSTAT.EQ.TSTAR) GO TO 100
89          WRITE (TOUT,4) TSTAT
90          WRITE (TOUT,6) KSCHM
91  * * STNOF FROM TOTAL * *
92  90      CONTINUE
93          CALL DATBAS(5HSTNOF,TSTAT,KSCHM,CENDP)
94          IF (TSTAT.EQ.TSTAR) GO TO 91
95          WRITE (TOUT,5) TSTAT
96          WRITE (TOUT,6) KSCHM
97  91      WRITE (TOUT,2)
98          STOP
99  * * INPUT FROM CARDS, OUTPUT TO PRINTER * *
100  95     ITN=4
101         IOUT=5
102         GO TO 100
103  * * INPUT AND OUTPUT TO CONSOLE * *
104  96     ITN=3
105         IOUT=3
106         GO TO 100
107  * * ERROR STATUS * *
108  99     WRITE (TOUT,9) TSTAT
109  * *
110  * * READ INPUT TRANSACTION * *
111  100    READ (ITN,8) KARDX2
112         WRITE (TOUT,7) KSPAC
113         WRITE (TOUT,7) KARDX2
114  * * CHECK FIRST 4 CHARACTERS OF TRANSACTION CODE * *
115         IF (KARDX4(1).EQ.CPARTL(1)) GO TO 200
116         IF (KARDX4(1).EQ.CWCIRL(1)) GO TO 400
117         IF (KARDX4(1).EQ.CBTLL(1)) GO TO 600
118         IF (KARDX4(1).EQ.CROUTL(1)) GO TO 800
119         IF (KARDX4(1).EQ.CENDP) GO TO 90
120         IF (KARDX4(1).EQ.CASTR) GO TO 100
121         IF (KARDX4(1).EQ.CCARD) GO TO 95
122         IF (KARDX4(1).EQ.CCONS) GO TO 96
123  * * INVALID TRANSACTION CODE * *
124  101    WRITE (TOUT,3)
125         GO TO 100
126  * *
```



```

PAGE      4          CMFGSMP  VORTXII FTN IV

127      * * *
128      * * *
129      * * *
130      * * *
131      * * PROCESS 'PART' TRANSACTIONS * *
132      * * *
133      * * *
134      * * *
135      * * *
136      200  CONTINUE
137          DO 201 T=1,8
138          KEY(T-3)=KARDX2(T)
139      201  CONTINUE
140          IF (KARDX2(3).EQ.KRD) GO TO 210
141          IF (KARDX2(3).EQ.KAD) GO TO 220
142          IF (KARDX2(3).EQ.KDI) GO TO 230
143          IF (KARDX2(3).EQ.KWR) GO TO 240
144          IF (KARDX2(3).EQ.KAI) GO TO 250
145          GO TO 101
146      * * *
147      * * READ A PART MASTER * *
148      * * *
149      210  CONTINUE
150          CALL DATBAS(5HREADM,ISTAI,CPARIL,KKEY,CPARII,KPARTA,CFNDP)
151          IF (ISTAI.EQ.CMRNF) GO TO 1024
152          IF (ISTAI.NE.TSTAR) GO TO 99
153          WRITE(1OUT,42) (KPARTA(T),I=1,5),(KPARTA(T),I=6,15),KPARTA(16)
154          GO TO 100
155      * * *
156      * * ADD A PART MASTER * *
157      * * *
158      220  CONTINUE
159          DO 221 T=1,8
160          KPARTA(T-3)=KARDX2(T)
161      221  CONTINUE
162          DO 222 T=9,18
163          KPARTA(T-3)=KARDX2(T)
164      222  CONTINUE
165          KPARTA(16)=1
166          CALL DATBAS(5HADDM,ISTAI,CPARIL,KKEY,CPARII,KPARTA,CFNDP)
167          IF (ISTAI.EQ.CDUPM) GO TO 1025
168          IF (ISTAI.NE.TSTAR) GO TO 99

```



```

PAGE      5          CMFGSMP  VORTXT1 FTN IV

169          GO TO 1021
170      * * *
171      * * DFLFIF A PART MASTER * *
172      * * *
173      230  CONTINUE
174          CALL DATBAS(5HDFL-M,ISIAI,CPARIL,KKEY,CPARII,KPARTA,CFNDP)
175          IF (ISIAI.EQ.CMRNF) GO TO 1024
176          IF (ISIAI.EQ.CIMDI) GO TO 1020
177          IF (ISIAI.NE.TSTAR) GO TO 99
178          GO TO 1022
179      * * *
180      * * UPDAIF A PART MASTER * *
181      * * *
182      240  CONTINUE
183          CALL DATBAS(5HDFADM,ISIAI,CPARIL,KKEY,CPARII,KPARTA,CFNDP)
184          IF (ISIAI.EQ.CMRNF) GO TO 1024
185          IF (ISIAI.NE.TSTAR) GO TO 99
186          DO 241 T=9,18
187          KPARTA(T-3)=KARDX2(T)
188      241  CONTINUE
189          CALL DATBAS(5HWRTM,ISIAI,CPARII,KKEY,CPARII,KPARTA,CFNDP)
190          IF (ISIAI.NE.TSTAR) GO TO 99
191          GO TO 1023
192      * * *
193      * * SERTAILY READ ALL PART MASIFRS * *
194      * * *
195      250  CONTINUE
196          IOUAL=CREGN
197      251  CONTINUE
198          CALL DATBAS(5HRDNXI,ISIAI,4HPART,TUIAI,CPARTL,KPARTA,CFNDP)
199          IF (ISIAI.NE.TSTAR) GO TO 99
200          IF (IOUAL.EQ.CFNDP) GO TO 100
201          WRITE(1OUT,12) (KPARTA(T),I=1,5),(KPARTA(T),I=6,15),KPARTA(16)
202          GO TO 251
203      * * *
204      * * *
205      * * *
206      * * *
207      * * *
208      * * *
209      * * *
210      * * *

```




```

PAGE      6      CMFGSMP      VORTXTI FTN IV

211      * * *
212      * * *
213      * * *
214      * * *
215      * * PROCESS 'WCTR' TRANSACTIONS * *
216      * * *
217      * * *
218      * * *
219      * * *
220      400 CONTINUE
221          DO 401 T=4,5
222          KKEY(T-3)=KARDX2(T)
223      401 CONTINUE
224          IF (KARDX2(3).EQ.KRD) GO TO 410
225          IF (KARDX2(3).EQ.KAD) GO TO 420
226          IF (KARDX2(3).EQ.KDI) GO TO 430
227          IF (KARDX2(3).EQ.KWR) GO TO 440
228          IF (KARDX2(3).EQ.KAI) GO TO 450
229          GO TO 101
230      * * *
231      * * READ A WCTR MASTER * *
232      * * *
233      410 CONTINUE
234          CALL DATBAS(5HREADM,ISTAT,CWCTR1,KKEY,CWCTR1,KWCTRA,LFNDP)
235          IF (ISTAT.EQ.CMPNF) GO TO 1020
236          IF (ISTAT.NE.TSTAR) GO TO 99
237          WRITE(1OUT,44) (KWCTRA(T),I=1,2),(KWCTRA(T),I=3,12)
238          GO TO 100
239      * * *
240      * * ADD A WCTR MASTER * *
241      * * *
242      420 CONTINUE
243          DO 421 T=4,5
244          KWCTRA(T-3)=KARDX2(T)
245      421 CONTINUE
246          DO 422 T=6,15
247          KWCTRA(T-3)=KARDX2(T)
248      422 CONTINUE
249          CALL DATBAS(5HADD-M,ISTAT,CWCTRL,KKEY,CWCTRL,KWCTRA,LFNDP)
250          IF (ISTAT.EQ.CDHPM) GO TO 1030
251          IF (ISTAT.NE.TSTAR) GO TO 99
252          GO TO 1026

```



PAGE 7 CMFGSMP VORTXII FTN IV

```

253 * * *
254 * * DFLFIF A WCTR MASTER * *
255 * * *
256 430 CONTINUE
257 CALL DATBAS(5HDFL-M,ISIAI,CWCTRI,KKEY,CWCTRI,KWCIRA,CFNDP)
258 IF (ISIAI.EQ.CMPNF) GO TO 1029
259 IF (ISIAI.EQ.CIMDI) GO TO 1020
260 IF (ISIAI.NE.TSTAR) GO TO 99
261 GO TO 1027
262 * * *
263 * * UPDATE A WCTR MASTER * *
264 * * *
265 440 CONTINUE
266 CALL DATBAS(5HREADM,ISIAI,CWCTRL,KKEY,CWCTRI,KWCIRA,CFNDP)
267 IF (ISIAI.EQ.CMPNF) GO TO 1029
268 IF (ISIAI.NE.TSTAR) GO TO 99
269 DO 441 T=4,15
270 KWCTRA(T-3)=KARDX2(T)
271 441 CONTINUE
272 CALL DATBAS(5HWRTM,ISIAI,CWCTRL,KKEY,CWCTRI,KWCIRA,CFNDP)
273 IF (ISIAI.NE.TSTAR) GO TO 99
274 GO TO 1028
275 * * *
276 * * SERIALY READ ALL WCIP MASIFRS * *
277 * * *
278 450 CONTINUE
279 IQUAL=CREGN
280 451 CONTINUE
281 CALL DATBAS(5HRDNXI,ISIAI,4HWCTR,TQIIAI,CWCTRL,KWCTRA,CFNDP)
282 IF (ISIAI.NE.TSTAR) GO TO 99
283 IF (IQUAL.EQ.CFNDP) GO TO 100
284 WRITE(ROUT,44) (KWCTRA(T),I=1,2),(KWCTRA(T),I=3,12)
285 GO TO 451
286 * * *
287 * * *
288 * * *
289 * * *
290 * * *
291 * * *
292 * * *
293 * * *
294 * * *

```



```

PAGE      R          CMFGSMP  VORTXT1 FTN IV

295      * * *
296      * * *
297      * * *
298      * * *
299      * * PROCESS 'BILL' TRANSACTIONS * *
300      * * *
301      * * *
302      * * *
303      * * *
304      600  CONTINUE
305          DO 601 T=4,8
306          KKEY(T-3)=KARDX2(T)
307      601  CONTINUE
308          TRFR=CLKRM
309          IF (KARDX2(3).EQ.KAD) GO TO 620
310          IF (KARDX2(3).EQ.KDI) GO TO 630
311          IF (KARDX2(3).EQ.KWR) GO TO 640
312          IF (KARDX2(3).EQ.KAL) GO TO 650
313          IF (KARDX2(3).EQ.KMD) GO TO 660
314          IF (KARDX2(3).EQ.KSR) GO TO 670
315          IF (KARDX2(3).EQ.KSW) GO TO 680
316          IF (KARDX2(3).EQ.KIR) GO TO 710
317          IF (KARDX2(3).EQ.KIW) GO TO 720
318          GO TO 101
319      * * *
320      * * ADD A BTLL OF MATERIAL VARIABLE * *
321      * * *
322      620  CONTINUE
323          TRFR=CLKRM
324      621  CONTINUE
325          CALL DATBAS(SHREADV,ISIAI,CBILI,TRFR,RHPARTIKRM,KKEY,
326          1          CBILI,KBTLA,CFNDP)
327          IF (ISIAI.EQ.CMRNF) GO TO 1024
328          IF (ISIAI.NE.TSTAR) GO TO 99
329          IF (TRFR.EQ.CENDP) GO TO 623
330          DO 622 T=9,13
331          IF (KBTLA(T-3).NE.KARDX2(1)) GO TO 621
332      622  CONTINUE
333          GO TO 1035
334      623  CONTINUE
335          DO 624 T=4,8
336          KBTLA(T-3)=KARDX2(T)

```



```
PAGE      9          CMEGSMP  VORTXTI  FTN  IV

337      KRILLA(T+2)=KARDX2(T+5)
338      624  CONTINUE
339      KRILLA(11)=1
340      DO 625 T=1,20
341      IF (KARDX2(14).EQ.KYNUM(1)) KRILLA(11)=KINUM(T)
342      625  CONTINUE
343      IREFR=CIKRM
344      CALL DATBAS(5HADDVC,TSTAT,CRILLI,TRFFR,RHPARTLKRM,KKEY,
345      1      CRILLI,KBTLIA,CENDP)
346      IF (ISTAT.EQ.CMRNF) GO TO 1024
347      IF (ISTAT.NE.TSTAR) GO TO 99
348      GO TO 1031
349      * * *
350      * *  DFLIF A RILL OF MATERIAL VARTABLE * *
351      * * *
352      630  CONTINUE
353      IREFR=CIKRM
354      631  CONTINUE
355      CALL DATBAS(5HREADV,ISTAT,CRILLI,TRFFR,RHPARTLKRM,KKEY,
356      1      CRILLI,KBTLIA,CENDP)
357      IF (ISTAT.EQ.CMRNF) GO TO 1024
358      IF (ISTAT.NE.TSTAR) GO TO 99
359      IF (IREFR.EQ.CENDP) GO TO 1034
360      DO 632 T=9,13
361      IF (KRILLA(T-3).NE.KARDX2(1)) GO TO 631
362      632  CONTINUE
363      CALL DATBAS(5HDFLVD,ISTAT,CRILLI,TRFFR,RHPARTLKRM,KKEY,
364      1      CRILLI,KBTLIA,CENDP)
365      IF (ISTAT.NE.TSTAR) GO TO 99
366      GO TO 1032
367      * * *
368      * *  UPDAIF A RILL OF MATERIAL VARTABLE * *
369      * * *
370      640  CONTINUE
371      IREFR=CIKRM
372      641  CONTINUE
373      CALL DATBAS(5HREADV,ISTAT,CRILLI,TRFFR,RHPARTLKRM,KKEY,
374      1      CRILLI,KBTLIA,CENDP)
375      IF (ISTAT.EQ.CMRNF) GO TO 1024
376      IF (ISTAT.NE.TSTAR) GO TO 99
377      IF (IREFR.EQ.CENDP) GO TO 1034
378      DO 642 T=9,13
```



```

PAGE 10          CMFGSMP  VORTXT1 FTN IV

379             IF (KRILLA(T-3).NE.KARDX2(1)) GO TO 641
380 642 CONTINUE
381             KRILLA(11)=1
382             DO 643 T=1,10
383             IF (KARDX2(14).EQ.KXNUM(1)) KRILLA(11)=KINUM(T)
384 643 CONTINUE
385             CALL DATBAS(5HWPITV,ISTAT,CRILL,TRFR,RHPARTLKRM,KKEY,
386             1          CRILL,KBTLA,CENDP)
387             IF (ISTAT.NE.TSTAR) GO TO 99
388             GO TO 1033
389 * * *
390 * * SURELY READ ALL BTLI VARIABLES * *
391 * * *
392 650 CONTINUE
393             IOUAL=CREGN
394 651 CONTINUE
395             CALL DATBAS(5HRONXI,ISTAT,4HBTLI,TQIAT,CBTL,CRILLA,CENDP)
396             IF (ISTAT.NE.TSTAR) GO TO 99
397             IF (IOUAL.EQ.CENDP) GO TO 100
398             WRITE(1OUT,46) (KRILLA(T),I=1,5),(KRILLA(T),I=6,10),KRILLA(11)
399             GO TO 651
400 * * *
401 * * DELETE ALL BTLI OF MATERIAL VARIABLES FOR A PART * *
402 * * *
403 660 CONTINUE
404             IREFR=CIKRM
405 661 CONTINUE
406             CALL DATBAS(5HREADV,ISTAT,CRILL,TRFR,RHPARTLKRM,KKEY,
407             1          CRILL,KBTLA,CENDP)
408             IF (ISTAT.EQ.CMRNE) GO TO 1024
409             IF (ISTAT.NE.TSTAR) GO TO 99
410             IF (IREFR.EQ.CENDP) GO TO 1032
411             CALL DATBAS(5HDFLVD,ISTAT,CRILL,TRFR,RHPARTLKRM,KKEY,
412             1          CRILL,KBTLA,CENDP)
413             IF (ISTAT.NE.TSTAR) GO TO 99
414             GO TO 661
415 * * *
416 * * READ A SINGLE LEVEL BTLI OF MATERIAL * *
417 * * *
418 670 CONTINUE
419             IREFR=CIKRM
420             IPATH(1)=CPARTL(1)

```



PAGE 11

CMFGSMP VORTXII FTN IV

```
421      IPATH(2)=CLKBM
422      675  CONTINUE
423      CALL DATBAS(5HREADV,ISTAT,CRIILL,TREFR,TPATH,KKEY,
424      1      CRIILL,KBTLLA,CENDP)
425      IF (ISTAT.EQ.CMRNF) GO TO 1024
426      IF (ISTAT.NE.TSTAR) GO TO 99
427      IF (IREFR.EQ.CENDP) GO TO 100
428      WRITE(1OUT,46) (KRILLA(T),I=1,5),(KRILLA(T),I=6,10),KRILLA(11)
429      GO TO 675
430      * * *
431      * * READ A SINGLE LEVEL WHERE USED LIST * *
432      * * *
433      680  CONTINUE
434      IREFR=CLKWU
435      IPATH(1)=CPARTL(1)
436      IPATH(2)=CLKWU
437      GO TO 675
438      * * *
439      * * READ AN INDENTED BILL OF MATERIAL * *
440      * * *
441      710  CONTINUE
442      SREFR=CLKRM
443      IPATH(1)=CPARTL(1)
444      IPATH(2)=CLKBM
445      J=6
446      K=10
447      GO TO 730
448      * * *
449      * * READ AN INDENTED WHERE USED * *
450      * * *
451      720  CONTINUE
452      SREFR=CLKWU
453      IPATH(1)=CPARTL(1)
454      IPATH(2)=CLKWU
455      J=1
456      K=5
457      730  CONTINUE
458      LEVL=1
459      IOTY=1
460      CALL DATBAS(5HREADM,ISTAT,CPARTL,KKEY,CPARTI,KPARTA,CENDP)
461      IF (ISTAT.EQ.CMRNF) GO TO 1024
462      IF (ISTAT.NE.TSTAR) GO TO 99
```



```

PAGE 12          CMFGSMP  VORTXTI FTN IV

463      WRITE(IOUT,42) (KPARIA(T),I=1,5),(KPARIA(T),I=6,15),KPARIA(16)
464      TREFR=SREFR
465      734 CONTINUE
466      CALL DATBAS(SHREFADV,TSTAT,CRILL,TRFR,TPATH,KKEY,
467      1          CRILL,KBTLLA,CENDP)
468      IF (ISTAT.NE.TSTAR) GO TO 99
469      IF (TREFR.EQ.CENDP) GO TO 750
470      736 CONTINUE
471      TRFR(LFVL)=TREFR
472      ITBTY(I EVL)=TWTY
473      DO 738 I=1,5
474      KTBPR(T,I EVL)=KKEY(I)
475      738 CONTINUE
476      L=0
477      DO 740 I=1,K
478      L=L+1
479      KKEY(I)=KBTLLA(T)
480      740 CONTINUE
481      CALL DATBAS(SHREFADM,ISTAT,CPARTI,KKEY,CPARTI,KPARTA,CENDP)
482      IF (ISTAT.NE.TSTAR) GO TO 99
483      IOTY=KBTLLA(11)*ITBTY(I EVL)
484      WRITE(IOUT,55) LFVL,TWTY
485      WRITE(IOUT,42) (KPARIA(T),I=1,5),(KPARIA(T),I=6,15),KPARIA(16)
486      TREFR=SREFR
487      CALL DATBAS(SHREFADV,ISTAT,CRILL,TRFR,TPATH,KKEY,
488      1          CRILL,KBTLLA,CENDP)
489      IF (ISTAT.NE.TSTAR) GO TO 99
490      IF (TREFR.EQ.CENDP) GO TO 754
491      LFVL=I EVL+1
492      IF (LFVL.GT.10) GO TO 1056
493      GO TO 736
494      750 CONTINUE
495      IF (LFVL.EQ.1) GO TO 100
496      LFVL=I EVL-1
497      754 CONTINUE
498      TRFR=TRFR(LFVL)
499      IOTY=ITBTY(LFVL)
500      DO 756 I=1,5
501      KKEY(T)=KTBPR(T,I EVL)
502      756 CONTINUE
503      GO TO 734
504      * * *

```



PAGE 13

CMFGSMP VORTXTI FTN IV

```
505 * * *
506 * * *
507 * * *
508 * * *
509 * * PROCESS 'ROUT' TRANSACTIONS * *
510 * * *
511 * * *
512 * * *
513 * * *
514 R00 CONTINUE
515     DO P01 T=4,8
516     KKEY(T-3)=KARDX2(T)
517 R01 CONTINUE
518     TREFR=CLKRT
519     IF (KARDX2(3).EQ.KAD) GO TO 820
520     IF (KARDX2(3).EQ.KDL) GO TO 830
521     IF (KARDX2(3).EQ.KWR) GO TO 840
522     IF (KARDX2(3).EQ.KAI) GO TO 850
523     IF (KARDX2(3).EQ.KBP) GO TO 870
524     IF (KARDX2(3).EQ.KBW) GO TO 880
525     GO TO 101
526 * * *
527 * * ADD A ROUTING VARIABLE * *
528 * * *
529 R20 CONTINUE
530     TREFR=CLKRT
531 R21 CONTINUE
532     CALL DATBAS(5HREADV,ISTAT,CROUTL,TREFR,8HPARTLKRT,KKEY,
533     1          CROUTL,KROUTA,CENDP)
534     IF (ISTAT.EQ.CMPNF) GO TO 1024
535     IF (ISTAT.NE.TSTAR) GO TO 90
536     IF (TREFR.EQ.CENDP) GO TO 823
537     IF (KROUTA(7).LT.KARDX2(9)) GO TO 821
538     IF (KROUTA(7).GT.KARDX2(9)) GO TO 822
539     GO TO 1040
540 R22 CONTINUE
541     KFUNC(3)=KB
542     GO TO 824
543 R23 CONTINUE
544     KFUNC(3)=KC
545     TREFR=CLKRT
546 R24 CONTINUE
```




```

PAGE 141          CMFGSMP  VORTXT1 FTN IV

547          DO 825 T=1,8
548          KRUIIA(T-2)=KARDX2(T)
549      825     CONTINUE
550          KRUIIA(7)=KARDX2(9)
551          IF (KARDX2(10).EQ.KSPAC) GO TO 827
552          KRUIIA(1)=KSK
553          KRUIIA(8)=KARDX2(10)
554          KRUIIA(9)=KARDX2(11)
555          DO 826 T=12,23
556          KRUIIA(T-2)=KARDX2(T)
557      826     CONTINUE
558      829     KFUNC(2)=KDV
559            KFUNC(1)=KAD
560            CALL DATBASE(KFUNC,ISTAT,CRUIII,TRFFR,RHPARTI,KRI,KKEY,
561      1          CRUIII,KROUTA,CENDP)
562            IF (ISTAT.EQ.CMRNF) GO TO 1029
563            IF (ISTAT.NE.TSTAR) GO TO 99
564            GO TO 1036
565      827     CONTINUE
566            KRUIIA(1)=KCM
567            DO 828 T=12,31
568            KRUIIA(T-1)=KARDX2(T)
569      828     CONTINUE
570            GO TO 829
571      * * *
572      * * DEFIE A ROUTING VARIABLE * *
573      * * *
574      830     CONTINUE
575            IREFR=LIKRI
576      831     CONTINUE
577            CALL DATBASE(SHREADV,ISTAT,CRUIII,TRFFR,RHPARTI,KRI,KKEY,
578      1          CRUIII,KROUTA,CENDP)
579            IF (ISTAT.EQ.CMRNF) GO TO 1024
580            IF (ISTAT.NE.TSTAR) GO TO 99
581            IF (IREFR.EQ.CENDP) GO TO 1039
582            IF (KRUIIA(7).NE.KARDX2(9)) GO TO 831
583      832     CONTINUE
584            CALL DATBASE(SHDFLVD,ISTAT,CRUIII,TRFFR,RHPARTI,KRI,KKEY,
585      1          CRUIII,KROUTA,CENDP)
586            IF (ISTAT.NE.TSTAR) GO TO 99
587            GO TO 1037
588      * * *

```



PAGE 15 CMFGSMP VORTXTI FTN IV

```

589 * * UPDATE A ROUTING VARIABLE * *
590 * * *
591 R40 CONTINUE
592 TREFR=CLKRT
593 R41 CONTINUE
594 CALL DATBAS(5HREADV,ISTAT,CROUTL,TREFR,8HPARTLKRT,KKEY,
595 1 CROUTL,KROUTA,CENDP)
596 IF (ISTAT.EQ.CMRNF) GO TO 1024
597 IF (ISTAT.NF.TSTAR) GO TO 99
598 IF (KROUTA(1).EQ.KSR) GO TO 843
599 DO R42 T=12,31
600 KROUTA(T-4)=KARDX2(T)
601 R42 CONTINUE
602 CALL DATBAS(5HWRITV,ISTAT,CROUTL,TREFR,8HPARTLKRT,KKEY,
603 1 CROUTL,KROUTA,CENDP)
604 IF (ISTAT.NF.TSTAR) GO TO 99
605 GO TO 1038
606 R43 DO R44 T=12,23
607 KROUTA(T-2)=KARDX2(T)
608 R44 CONTINUE
609 GO TO 842
610 * * *
611 * * SERTAILY READ ALL ROUT VARIABLES * *
612 * * *
613 R50 CONTINUE
614 IQUAL=CREGN
615 R51 CONTINUE
616 DO R52 T=8,27
617 KROUTA(T)=KSPAC
618 R52 CONTINUE
619 CALL DATBAS(5HRDNXT,ISTAT,4HROUT,IQUAL,CROUTL,KROUTA,CENDP)
620 IF (ISTAT.NF.TSTAR) GO TO 99
621 IF (IQUAL.EQ.CENDP) GO TO 100
622 WRITE(IOUT,48) KROUTA(1),(KROUTA(T),I=2,6),KROUTA(7),
623 1 (KROUTA(T),I=8,27)
624 GO TO 851
625 * * *
626 * * READ ALL ROUTINGS FOR A PART * *
627 * * *
628 R70 CONTINUE
629 TREFR=CLKRT
630 IPATH(1)=CPARTL(1)

```



```

PAGE      16              CMFGSMP  VORTXTL FTN IV

631          IPATH(2)=CLKRT
632      * * *
633      875  CONTINUE
634          DO 876 T=8,27
635          KRUITA(T)=KSPAC
636      876  CONTINUE
637          CALL DATBAS(SHREADV,ISIAI,CROUITL,TRFR,TPATH,KKEY,
638      1          CROUIT,KRUTA,CENDP)
639          IF (ISIAI.EQ.CMPNF) GO TO 877
640          IF (ISIAI.NE.TSTAR) GO TO 99
641          IF (TRFR.EQ.CENDP) GO TO 100
642          WRITE(OUT,48) KRUTA(1),(KRUITA(T),I=2,6),KRUTA(7),
643      1          (KRUITA(T),I=8,27)
644          GO TO 875
645      877  CONTINUE
646          IF (IPATH(1).EQ.CPART(1)) GO TO 1024
647          GO TO 1029
648      * * *
649      * * READ ALL ROUTINGS FOR A WCTR * *
650      * * *
651      880  CONTINUE
652          IREFR=CLKRT
653          IPATH(1)=CWCTRL(1)
654          IPATH(2)=CLKRT
655          GO TO 875
656      * * *
657      * * *
658      * * *
659      * * *
660      * * *
661      * * *
662      * * *
663      * * *
664      * * *
665      * * *
666      * * *
667      * * *
668      * * *
669      * * *
670      * * *
671      * * *
672      * * *

```



```
PAGE 17          CMFGSMP  VORTXII FTN IV

673  * * *
674  * * *
675  * * LOW LEVEL CODE UPDATE ROUTINE * *
676  * * *
677  * * *
678  * * *
679  * * *
680  1200  DO 1202 I=1,5
681         KKEY(T)=KRILLA(T)
682         KPKFY(1)=KBTLIA(1)
683  1202  CONTINUE
684         CALL DATBAS(SHREADM,ISTAT,CPARTI,KKEY,CPARTI,KPARTA,CENDP)
685         IF (ISTAT.NE.TSTAR) GO TO 99
686         IHLLCD=KPARTA(16)
687         IHLLCD=IHLLCD+1
688         DO 1204 I=1,5
689         KKEY(T)=KRILLA(T+5)
690         KCKFY(1)=KBTLIA(I+5)
691  1204  CONTINUE
692         CALL DATBAS(SHREADM,ISTAT,CPARTL,KKEY,CPARTI,KPARTA,CENDP)
693         IF (ISTAT.NE.TSTAR) GO TO 99
694         IF (KPARTA(16).LI.IHLLCD) GO TO 1206
695         GO TO 1270
696  1206  KPARTA(16)=IHLLCD
697         CALL DATBAS(SHWRTM,ISTAT,CPARTL,KKEY,CPARTL,KPARTA,CENDP)
698         IF (ISTAT.NE.TSTAR) GO TO 99
699         LFVI=1
700         TREFR=CLKRM
701         IPATH(1)=CPARTL(1)
702         IPATH(2)=CLKBM
703  1210  CONTINUE
704         CALL DATBAS(SHREADV,ISTAT,CRIILL,TREFR,TPATH,KKEY,
705         1          CRIILI,KBTLIA,CENDP)
706         IF (ISTAT.NE.TSTAR) GO TO 99
707         IF (TREFR.EQ.CENDP) GO TO 1240
708  1220  DO 1222 I=1,5
709         IF (KCKFY(1).NE.KRILLA(T+5)) GO TO 1224
710  1222  CONTINUE
711         GO TO 1260
712  1224  TRRFF(LFVI)=TREFR
713         ITBOTY(LEVEL)=IHLLCD
714         DO 1226 I=1,5
```



PAGE 18

CMFGSMP VORTXTI FTN IV

```

715      KTBPRT(T,LEVL)=KKEY(I)
716      1226 CONTINUE
717      DO 1228 I=1,5
718      KKEY(T)=KRILLA(T+5)
719      1228 CONTINUE
720      IHLICD=THLLCD+1
721      CALL DATBAS(5HRFADM,ISTAT,CPARTL,KKEY,CPARTL,KPARTA,CFNDP)
722      IF (ISTAT.NF.TSTAR) GO TO 99
723      IF (KPARTA(16).LT.IHLICD) GO TO 1230
724      GO TO 1250
725      1230 KPARTA(16)=THLLCD
726      CALL DATBAS(5HWRTM,ISTAT,CPARTL,KKEY,CPARTL,KPARTA,CFNDP)
727      IF (ISTAT.NF.TSTAR) GO TO 99
728      TRFR=CIKRM
729      CALL DATBAS(5HRFADV,ISTAT,CRILL,TRFR,TPATH,KKEY,
730      1          CRILL,KBILLA,CFNDP)
731      IF (ISTAT.NF.TSTAR) GO TO 99
732      IF (TRFR.EQ.CENDP) GO TO 1250
733      LFVL=LEVL+1
734      IF (LFVL.GT.10) GO TO 1056
735      GO TO 1220
736      1240 IF (LFVL.EQ.1) GO TO 1270
737      LFVL=LEVL-1
738      1250 TRFR=TRKFF(LFVL)
739      IHLICD=TRQTY(LFVL)
740      DO 1252 I=1,5
741      KKEY(T)=KTBPRT(T,LEVL)
742      1252 CONTINUE
743      GO TO 1210
744      1260 CONTINUE
745      CALL DATBAS(5HDFLVD,ISTAT,CRILL,TRFR,TPATH,KKEY,
746      1          CRILL,KBILLA,CFNDP)
747      IF (ISTAT.NF.TSTAR) GO TO 99
748      WRITE (TOUT,53)
749      DO 1262 I=1,5
750      KKEY(I)=KSPAC
751      1262 CONTINUE
752      GO TO 1210
753      1270 GO TO 2062
754      * * *
755      * * *
756      * * *

```



```

PAGE      19                CMFGSMP  VORTXT1 FTN IV

757      * * *
758      * * *
759      * * MTSC MESSAGES * *
760      * * *
761      * * *
762      * * *
763      * * *
764      1      FORMAT(16H *START MFGSAMP)
765      2      FORMAT(16H *F O T MFGSAMP)
766      3      FORMAT(20H ** INVALID REQUEST)
767      4      FORMAT(18H ** SINON FRRUR ,A4)
768      5      FORMAT(18H ** SINOF FRRUR ,A4)
769      6      FORMAT(2H ,57A2)
770      7      FORMAT(2H ,40A2)
771      8      FORMAT(40A2)
772      9      FORMAT(18H ** FRRUR STATUS ,A4)
773      * * *
774      20     FORMAT(18H *INVALID DELETE )
775      1020   WRITE (TOUIT,20)
776              GO TO 100
777      21     FORMAT(18H *PART ADDED      )
778      1021   WRITE (TOUIT,21)
779              GO TO 100
780      22     FORMAT(18H *PART DELETED   )
781      1022   WRITE (TOUIT,22)
782              GO TO 100
783      23     FORMAT(18H *PART UPDATED   )
784      1023   WRITE (TOUIT,23)
785              GO TO 100
786      24     FORMAT(18H *PART NOT FOUND )
787      1024   WRITE (TOUIT,24)
788              GO TO 100
789      25     FORMAT(18H *DUPLICATE PART )
790      1025   WRITE (TOUIT,25)
791              GO TO 100
792      26     FORMAT(18H *WCTR ADDED     )
793      1026   WRITE (TOUIT,26)
794              GO TO 100
795      27     FORMAT(18H *WCTR DELETED   )
796      1027   WRITE (TOUIT,27)
797              GO TO 100
798      28     FORMAT(18H *WCTR UPDATED   )

```



```

PAGE 20          CMFGSMP  VORTXTL FTN IV

799  1028 WRITE (TUIT,28)
800          GO TO 100
801  29  FORMAT(18H *WCTR NOT FOUND )
802  1029 WRITE (TUIT,29)
803          GO TO 100
804  30  FORMAT(18H *DUPLICATE WCTR )
805  1030 WRITE (TUIT,30)
806          GO TO 100
807  31  FORMAT(18H *BILL ADDED      )
808  1031 WRITE (TUIT,31)
809          GO TO 1200
810  2062 CONTINUE
811          GO TO 100
812  32  FORMAT(18H *BILL DELETED   )
813  1032 WRITE (TUIT,32)
814          GO TO 100
815  33  FORMAT(18H *BILL UPDATED   )
816  1033 WRITE (TUIT,33)
817          GO TO 100
818  34  FORMAT(18H *BILL NOT FOUND )
819  1034 WRITE (TUIT,34)
820          GO TO 100
821  35  FORMAT(18H *DUPLICATE BILL )
822  1035 WRITE (TUIT,35)
823          GO TO 100
824  36  FORMAT(18H *RUIT ADDED     )
825  1036 WRITE (TUIT,36)
826          GO TO 100
827  37  FORMAT(18H *RUIT DELETED   )
828  1037 WRITE (TUIT,37)
829          GO TO 100
830  38  FORMAT(18H *RUIT UPDATED   )
831  1038 WRITE (TUIT,38)
832          GO TO 100
833  39  FORMAT(18H *RUIT NOT FOUND )
834  1039 WRITE (TUIT,39)
835          GO TO 100
836  40  FORMAT(18H *DUPLICATE RUIT )
837  1040 WRITE (TUIT,40)
838          GO TO 100
839  * * *
840  42  FORMAT(10H  PART= ,5A2,8H  DESC= ,10A2,8H  (LFD= ,12)

```



```

PAGE      21                CMFGSMP  VORTXT1 FTN IV

841      * * *
842      44      FORMAT(10H      WCIR= ,2A2,8H  DESC= ,10A2)
843      * * *
844      46      FORMAT(10H      PARN= ,5A2,8H  COMP= ,5A2,8H  QTY= ,T2)
845      * * *
846      48      FORMAT(10H      CODE= ,A2,8H  PART= ,5A2,8H  SEQN= ,
847      1      A2,8H  DATA= ,20A2)
848      * * *
849      56      FORMAT (22H  ** LEVEL TABLE FULL )
850      1056  WRITE (TUNIT,56)
851      GO TO 100
852      * * *
853      55      FORMAT (8H  *LEVEL=,T2,6H  QTY=,T4)
854      * * *
855      53      FORMAT (38H  ** CONTINUITY ERROR, RECORD DELETED )
856      * * *
857      END

```

ENTRY/COMMON BLOCK NAMES

013264 R CMFGSMP

EXTERNAL NAMES

011643 E DATRAS
013114 E \$WR
000000 E V\$RFRB
000000 E V\$RFR1
000000 E V\$RFRN
013120 E \$ND
011662 E \$UI
001025 E \$I3
010430 E \$I1
000776 E \$ST
001033 E \$KD
011722 E \$DD

SYMBOL TABLE

013207 R 000001
013221 R 000002
013167 R 000004
011643 E DATRAS
000002 R KSCHEM
000047 R CPART1
000065 R CP11LI
000103 R LWLTRI
013171 R 000005



APPENDIX B DIAGNOSTICS/STATUS CODES

B.1 DATA BASE GENERATION

The DBGEN program, while analyzing DBDL statements, may determine various error conditions which would lead to an erroneous Data Base Descriptor module. If this should occur, appropriate messages will be printed along with the DBDL statement listing and the output of source statements will be suppressed. Certain error conditions will cause immediate termination of the DBGEN program, while other conditions allow further processing of the DBDL input. All output messages are output to LO logical unit.

The message: MISSING STATEMENT:

Followed by a DBDL statement, will cause immediate termination of the DBGEN run. Correct the error and resubmit.

The message: BYPASSING FILE:

Followed by an indicative message, will cause the DBGEN to ignore the remainder of the statements for that file, but will continue processing with the next file definition. Output is suppressed. Correct the error and resubmit.

The message: STATEMENT IGNORED:

Indicates the current DBDL statement has been ignored by DBGEN, usually because of a prior error condition. Correct the prior and resubmit.

The following messages are among those used in combination with BYPASSING FILE: and are indicative of the error condition. Each message is followed by a description.

DUPLICATE NAME

The same name has been used for more than one data set, I/O area, data element, or linkage path.

I/O AREA NOT DEFINED

The I/O area named for a data set has not been previously defined in the prologue.



NO I/O AREAS DEFINED

Between SHARE-IO and END-IO, there are no I/O areas.

CODE=2 MUST BE FIRST ENTRY

The VVVV CODE=2 ENTRY when specified must be the first element in the variable file definition following BASE-DATA.

DATA BASE TOO LARGE

Indicates there is not enough storage available for DBGEN's internal tables.

EXCEEDS MAX PARTITION SECTORS

A drive statement entry specified a number of sectors in excess of the maximum partition size.

DRIVE TABLE FULL

The maximum number of drive statements permissible for a file was exceeded.

NO DRIVE STATEMENT

All files must have at least one drive statement.

RECORDS/SECTOR ERROR

The logical record length specified or calculated is not a multiple of VORTEX sector size.

SECTORS/RECORD ERROR

The logical record length specified is not a multiple of VORTEX sectors.

SECTOR NUMBER ERROR

The number of sectors specified on drive statements is less than required to accommodate TOTAL records in the file.

**NO DASMR OUTPUT**

No output of DASMR source resulted because either "OPTIONS-OUTPUT=N" has been specified, or an error was encountered in processing files.

INVALID STATEMENT

An unrecognized statement or statement has appeared in an illogical order.

NUMERIC FORMAT ERROR

An expected numeric field contains invalid characters or an invalid delimiter.

INVALID ELEMENT NAME

An element name does not begin with the data set name or otherwise contains invalid characters.

ELEMENT LENGTH ERROR

The length of sub-elements is greater than the length of a parent element.

RECORD CODE INVALID

The record-code entry is invalid at this point or the record-code itself is invalid (**).

NO VVVV CODE=2 ENTRY

The record-code entry is invalid unless a prior definition has been made for vvvvcode=2.

MASTER LINK MISSING

The linkage path specified in a variable entry definition was not previously defined in a master file.

LINK FIELD MISSING

The element specified as a variable entry control field was not previously defined.



MMMMLKXX=8
MMMMROOT=8
MMMCTRL=
VVVVCODE=2

The statement contains an improper character, delimiter, or length.

B.2 DATA BASE FORMAT STATUS CODES

The DBFMT program provides printed output describing the results of processing each file. A list of possible messages follows:

bFORMAT dbname FILE1, FILE2,...,FILEn

The above line displays the parameter card requesting the format function. 'b' in the first column represents a blank space.

MMMM FILE FORMATTED AND CLOSED

NNN TOTAL SECTORS WRITTEN

The above lines indicate formatting of the requested data set has been successfully completed. MMMM is the name of a TOTAL file.

MMMM NOT FOUND IN FILE TABLE

The above line indicates the requested data set cannot be found in the DBMOD. MMMM is the name of a TOTAL file.

INVALID FORMAT STATEMENT

The above message indicates the requested parameter card is in error.

INVALID DESCRIPTOR

---- NOT FORMATTED

The above lines indicate the requested DBMOD is not valid.



'FFFFTB LN CREATE IO ERROR'
'FFFFTB LN OPEN IO ERROR'
'FFFFTB LN WRITE TO ERROR'
'FFFFTB LN RETRY TIME OUT'
'FFFFTB LN DIRECTORY STRUCTURE ERROR'
'FFFFTB LN INSUFFICIENT SPACE'

where FFFF is the TOTAL file name

TB is the FCB tie breaker number assigned when DBMOD was built

LN is the logical unit number of the VORTEX TOTAL file

The preceding messages indicate I/O errors, clock errors, directory structure errors, and file creation errors.

END DATA BASE FORMAT

The above line is the normal end of job message.

B.3 DML COMMAND DIAGNOSTIC STATUS CODES

All DML messages are listed in table B-1.

B.3.1 Status Code Testing

After execution of any DML Command, a status code will be moved to the user's status field to indicate the result of the operation, such as:

- a. Successful completion of the operation.
- b. Command failure because of:
 1. An incorrect command parameter.
 2. A violation of conditions (e.g., an attempt to update a locked data-set).
 3. A redundant command (e.g., Sign-On of an active task).

It is up to the user program to test the status field after each DML command and to take appropriate action. If the command failed, the program should be terminated and the problem corrected. If the status code indicates some special condition other than failure, the program should include logic to handle and rectify the situation.



Table B-1. DML Diagnostic Messages

Status Code	DML Command	ADD-M	ADDVA	ADDVB	ADDVC	DEL-M	DELVD	RDNXT	READD	READM	READR	READV	RQLOC	SINOF	SINON	WRITM	WRITV	MARKL	WRITD	QUIET	STATUS CODE DEFINITION	
1.	BCTL	X	X	X	X	X			X	X	X	X				X					Blank (or zero) control field	
2.	DUPM	X																			Duplicate Master Record	
3.	DUPO														X						Duplicate Open	
4.	ENTF	X	X	X	X			X	X	X	X	X					X	X				Element (data field) not found
5.	EXSO														X						Extra SINON	
6.	FATL		X	X	X					X	X										Internal Core damage - fatal condition	
7.	FNOP	X	X	X	X	X	X	X	X	X	X	X					X	X				File not open
8.	FNTF	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	File not found	
9.	FTYP	X	X	X	X	X	X		X	X	X	X					X	X				Invalid file type
10.	FULL	X	X	X	X																File is full	
11.	FUNC	← SNAPDUMP/EXIT →															X	X	X	Function error		
12.	IDBM	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X					Invalid Data Base Module
13.	IMDL				X																Invalid Master Delete	
14.	IOER	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X					I/O Error
15.	IPAR	← SNAPDUMP/EXIT →																		Invalid Parameter		
16.	IRLC	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X					Invalid Record Location
17.	IVEL		X	X	X			X	X		X	X								X		Invalid Element
18.	IVRC		X	X	X				X		X	X								X		Invalid Record Code
19.	IVRP		X	X	X		X	X	X		X	X								X		Invalid Reference Parameter
20.	IVTF														X						Invalid Total File	
21.	LOAD													X	X						File above load point	
22.	LOCK															X					File is locked by another task	
23.	MLNF		X	X	X					X	X										Master Linkage Path not found	
24.	MRNF		X	X	X	X	X		X	X	X						X				Master Record not found	
25.	NOSO	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	No previous SINON	
26.	UACM															X					Undefined ACCESS MODE	
27.	UCTL	X																			Unequal control field	
28.	ULGO														X						Undefined Logging Option	
29.	UPDE	X	X	X	X	X	X									X	X				Update error	

NOTES:

1. An X means the corresponding command will respond to the corresponding status code.
2. MARKL, WRITD, and QUIET are not implemented.



The status code field is defined by the user and named in the command parameter list. Normally, it is a 4-byte alphanumeric field.

It should be noted that for certain commands, several status codes may result from multiple violations or conditions. However, only the status codes of highest severity (or the first of equal status codes) encountered will be retained and passed to the user. A secondary error may occur during backout of a failing function; only the first error is reported to the user.

B.3.2 Explanation of Terms

A few terms used in the following listing of status codes are defined here for clarification:

Serial	Refers to physical order as opposed to logical order, e.g., a Serial Read retrieves records in physical order.
Primary	Refers to the current controlling linkage path as specified in the Link Path parameter.
Secondary	Refers to any linkage path other than the Primary.
Base	Refers to the Base Data portion of a coded variable record, i.e., that portion which is common to all records in the data set and does not depend on a record code for its identification.
Coded	Refers to the redefined portion of a variable record, i.e., elements which are defined under a specific record code.

B.3.3 Status Code Listing

The status codes fall into three categories:

- a. Successful completion ('****')
- b. Informative; some user action may be required.
- c. Fatal; the requested function has not been completed. Any modifications to the data base performed before the error was detected were "backed out" to the condition immediately prior to the request. The data base is closed down automatically. User program will then receive control.
- d. Any status codes that are FATAL are marked with an F after the heading.



varian data machines

BCTL	<u>Blank or Zero Control Field.</u> An attempt was made to add, read or write a master record with a blank control field: a. In a master data set command parameter. b. In a variable data set command parameter or I/O area.
DUPM	<u>Duplicate Master Record.</u> A master record with the same key already exists on the data set.
DUPO	<u>Duplicate Open of a Data Set (F).</u> TOTAL sensed at SINON command time, a request for a data set that this program has already OPENED.
ENTF	<u>Element Not Found.</u> An element name in the command parameter list is incorrect or the requested element does not exist in the data base descriptor currently in use.
EXSO	<u>Extra SINON.</u> More than one sign-on.
FATL	<u>Fatal Error.</u> Internal memory backout attempt failed. Memory or data base is destroyed.
FNOP	<u>File Not Open (F).</u> The requested data set in the command parameter list has not been opened. This same data set was not specified in the SINON command.
FNTF	<u>File Not Found (F).</u> The data set name in the command parameter list is misspelled or the data set does not exist in the data base descriptor module now in use.
FTYP	<u>Invalid File Type.</u> An attempt was made to process a master data set with a variable function or vice-versa.
FULL	<u>File Loaded to Capacity.</u> The number of records loaded equals the number of TOTAL-LOGICAL-RECORDS specified in the data base descriptor less any control records. This status code is returned when the excess record addition is attempted.
FUNC	<u>Invalid Function Code (F).</u> The function code in the command parameter list is misspelled. A SNAPSHOT dump is given.
IDBM	<u>Invalid Data Base Module (F).</u> When the name or the definition of DB is not correct.
IMDL	<u>Invalid Master Delete.</u> One or more variable entry records are still linked to this master record. These variable



entry records must be deleted before the master record can be deleted.

- IOER I/O error. Selected by VORTEX driver. FATAL condition. If an I/O error occurs, there are two possible causes:
(a) The error was detected while in the process of writing.
(b) The error was detected before any attempt to write. In (a) TOTAL already executed an internal SINOF which left the appropriate files locked. The only recovery left is reloading the file(s) from the backup storage and re-starting the update process.
- In (b) it is up to the user application program to decide on the proper action (e.g. SINOF and backup, ignore, etc).
- IPAR Invalid Parameter (F). Parameters are missing or incorrect in the command list.
- IRLC Invalid Record Location (F). The content of the Reference Field, a linkage path or the generated master address is invalid.
- a. The address generated is outside of the file limits.
 - b. FATAL condition.
- INVEL Invalid Element Name.
- IVRC Invalid Record Code. The record code being processed is invalid.
- a. Upon encountering an ADDVC, ADDVA, or ADDVB and with record-codes defined, the user I/O area contains an invalid record code.
 - b. Using a READV, READR, or READD and with record-codes defined, the user I/O area has an invalid record code.
- IVRP Invalid Reference Parameter. TOTAL has encountered an invalid reference parameter in a variable entry function prior to execution. If this function was allowed to continue, an 'IRLC' condition would have been returned with FATAL shutdown.
- IVTF Invalid TOTAL file. File name during Sign On does not match DBGEN. Format may not be the same as run time. (Print last record on MASTER or first record on VARIABLE to look for name.)



- LOAD** Variable Entry File Loaded Beyond Load Limit.
- a. The reference variable entry file has been loaded to the load limit as specified in the data base description.
 - b. Subsequent addition may still be done until the physical load limit is reached, which will result in a FULL status code.
- LOCK** Data Set Locked (F).
- a. A program using the data base is in UPDATE mode previously aborted and left the data set(s) locked. It is necessary to restore the data base from backup and reprocess everything since that backup. Failure to restore the data base will cause unpredictable results.
 - b. Another copy of TOTAL has already locked this specific data set. If the LOCK status is ignored, further processing will return the NOSO status code.
 - c. This status is returned in the GENERAL command status as well as the file area of the REALM section.
- MLNF** Master Link Not Found. A master linkage name is invalid:
- a. READV, READR, or READD cannot find requested linkage path name as stated.
 - b. ADDVC, ADDVA, or ADDVB cannot find associated linkage path names in defined masters.
- MRNF** Master Record Not Found. The master record corresponding to a given control field cannot be found.
- In a variable entry function, this will correspond to a primary control field.
- NOIO** No Assigned I/O Area (F). TOTAL sensed, while trying to open a file from the SINON command, that no I/O area exists in the data base descriptor being used.
- Severe problem with DBGEN output.
- NOSO** No SINON. A function without a former SINON has been issued.
- UACM** Undefined Access Mode (F). The access Mode parameter in the SINON command parameter is invalid.



- UCTL Unequal Control Field. The control field referenced in the command parameter list does not match that in the user I/O area.
- UPDE Update Mode Error. An add, delete or write function was requested against a file(s) whose access mode was stated as RONLY or whose file mode was SHRE in the SINON command.



varian data machines



APPENDIX C
'TEXT' BLOCK

The 'TEXT' portion of the TOTAL core dump is used to interpret the dump. It contains the following sections:

- Address in DATBAS for TOTAL to return to
- Parameter list pulled from user call
- Parameter count for a particular function
- Save area lists
- I/O macros
- Statistics
- DBMOD prefix
- Task general information
- Status
- Sequential function dependent text
- Work (Variable) fields
- File table definition
- IOTABLE(IOPOOL) entry description
- Buffer prefix definition
- General file descriptor
- Master file unique description
- Variable file unique description
- Variable file linkpath table
- Element table
- Relative labels for file/cylinder control records
- I/O macro control block expansion
- File partition control blocks

The 'TEXT' block listing is provided on the following pages.



VORTEXII DASMR DATBAS 2039 HOURS

```

      89      EJEC
000000 000000 R 90 TEXT  EQU  *
      91      DATA  (*)

      93 *****
000001 152305 A 94      DATA  'TEXT'
000002 154324 A

      95 *      ADDRESS (IN DATBAS) FOR TOTAL TO RETURN TO
      96 FRETRN EQU  *
      97 ARETRN EQU  3      * RX *
000003 000000 A 98      DATA  0
000004 150301 A 99 PARM  DATA  'PARM'
000005 151315 A

100 *      PARAMETER LIST PULLED FROM USER CALL
101 *****
102 *
000006 000006 A 103 PA    EQU  0      VARIABLE  MASTER  RDNXT  SINON/OFF
000006 000000 A 104      DATA  0      * RX *
      105 PB    EQU  PA+1    STATUS
000007 000000 A 106      DATA  0
      107 PC    EQU  PB+1    FILE
000010 000000 A 108      DATA  0
      109 EPD   EQU  *
000011 000011 A 110 PD    EQU  PC+1    REFER  KEY  REFER  END
000011 000000 A 111      DATA  0
      112 PE    EQU  PD+1    LINKPATH ELEMENT LIST
000012 000000 A 113      DATA  0
      114 PF    EQU  PE+1    KEY  I/O AREA
000013 000000 A 115      DATA  0
      116 PG    EQU  PF+1    ELEMENT  END
000014 000000 A 117      DATA  0
      118 PH    EQU  PG+1    I/O AREA
000015 000000 A 119      DATA  0
      120 PI    EQU  PH+1    END
000016 000000 A 121      DATA  0
122 *      PARAMETER COUNT FOR THIS PARTICULAR FUNCTION
123 *
124 *      USED BY DATBAS TO CALCULATE RETURN ADDRESS IN USER PROGRAM
000017 000017 A 125 PARMCT EQU  PI+1 *
000017 000000 A 126      DATA  0

128 *****

```



VURTXII DASMR DATBAS 2039 HOJRS

129 * SALTS (SAVE AREA LISTS)
 130 * THESE SAVE WORDS ARE USED TO SAVE REGISTERS
 131 * THE NAMES (LABELS) ARE DEFINED AS FOLLOWS--SALTNX
 132 * WHERE 'N' IS THE LEVEL NUMBER (LOWEST LEVEL IS 1, INDICATING NO
 133 * OTHER SUBROUTINES-LOWER LEVELS-ARE USED-PERFORMED)
 134 * AND X IS THE REGISTER INDICATOR--A--A REGISTER
 135 * B--B REGISTER
 136 * C--X REGISTER

		137	*****				
	000020	A	138	SALT	EQJ	PARMCT+1	
000020	151701	A	139		DATA	'SALT'	
000021	146324	A					
	000022	R	140	ESALT1	EQJ	*	
	000022	A	141	SALT1A	EQJ	SALT+2	* RX *
000022	000000	A	142		DATA	0	
	000023	R	143	RETRN1	EQJ	*	
	000023	A	144	SALT1B	EQJ	SALT1A+1	* RX *
000023	000000	A	145		DATA	0	
	000024	A	146	SALT1C	EQJ	SALT1B+1	* RX *
	000024	R	147	EALT1C	EQJ	*	
	000024	A	148	SALT1X	EQJ	SALT1B+1	* RX *
000024	000000	A	149		DATA	0	
	000025	R	150	EALT2A	EQJ	*	
	000025	A	151	SALT2A	EQJ	SALT1C+1	* RX *
000025	000000	A	152		DATA	0	
	000026	R	153	RETRN2	EQJ	*	
	000026	A	154	SALT2B	EQJ	SALT2A+1	* RX *
000026	000000	A	155		DATA	0	
	000027	R	156	EALT2C	EQJ	*	
	000027	A	157	SALT2C	EQJ	SALT2B+1	* RX *
000027	000000	A	158		DATA	0	
	000030	A	159	SALT3A	EQJ	SALT2C+1	* RX *
000030	000000	A	160		DATA	0	
	000031	R	161	RETRN3	EQJ	*	
	000031	A	162	SALT3B	EQJ	SALT3A+1	* RX *
000031	000000	A	163		DATA	0	
	000032	A	164	SALT4A	EQJ	SALT3B+1	* RX *
000032	000000	A	165		DATA	0	
	000033	R	166	RETRN4	EQJ	*	
	000033	A	167	SALT4B	EQJ	SALT4A+1	* RX *
000033	000000	A	168		DATA	0	
	000034	A	169	SALT5A	EQJ	SALT4B+1	* RX *



VORTXII DASM R DATBAS 2039 HOURS

```

000034 000000 A 170 DATA 0
          000035 R 171 RETRN5 EQU *
          000035 A 172 SALT5B EQU SALT5A+1 * RX *
000035 000000 A 173 DATA 0
          000036 A 174 SALT6A EQU SALT5B+1 * RX *
000036 000000 A 175 DATA 0
          000037 R 176 RETRN6 EQU *
          000037 A 177 SALT6B EQU SALT6A+1 * RX *
000037 000000 A 178 DATA 0
          000040 A 179 SALT7A EQU SALT6B+1 * RX *
000040 000000 A 180 DATA 0
          000041 H 181 RETRN7 EQU *
          000041 A 182 SALT7B EQU SALT7A+1 * RX *
000041 000000 A 183 DATA 0
          000042 R 184 RETRN8 EQU *
          000042 A 185 SALT8B EQU SALT7B+1 * RX *
000042 000000 A 186 DATA 0

```

```

188 *****
189 * I / O M A C R O S
190 *****

```

```

          000043 R 191 IOPEN EQU *
          000043 A 192 TKOPEN EQU SALT8B+1 * RX *
          193 OPEN FCB,LUN,WAIT,REW
000043 006505 A
000044 000404 A
000045 100000 A
000046 003000 A
000047 000000 A
000050 000000 A
000051 000000 A
000052 006037 A 194 LDXE TEXT
000053 000000 R
000054 006706 A 195 IJMP ZERO,REGB
000055 000000 A
          196 *
          000056 A 197 TKREAD EQU TKOPEN+11
          198 READ FCB,LUN,WAIT,BINARY
000056 006505 A
000057 000404 A
000060 100000 A
000061 000000 A

```




VORTXII DASMR DATBAS 2039 HOURS

```

000062 000000 A
000063 000000 A
000064 000000 A
000065 006037 A 199      LDXE      TEXT
000066 000000 R
000067 006706 A 200      IJMP      ZERO,REG8
000070 000000 A
                201 *
                000071 A 202 TKWRIT EQU      TKREAD+11
                203      WRITE   FCB,LUN,WAIT,BINARY
000071 006505 A
000072 000404 A
000073 100000 A
000074 000400 A
000075 000000 A
000076 000000 A
000077 000000 A
000100 006037 A 204      LDXE      TEXT
000101 000000 R
000102 006706 A 205      IJMP      ZERO,REG8
000103 000000 A
                206 *
                207 *
                000104 A 208 LGWRIT EQU      TKWRIT+11
                209 *LOG   WRITE   FCB,LUN,WAIT,BINARY
                210 *LOG   IJMP      ZERO,REG8
                211 *
                213 *****
                214 *      S T A T I S T I C S
                215 *****
                216 *      COUNT OF LOGICAL READS,I.E. CALL'S TO SBREAD
000104 000104 A 217 LREADS EQU      LGWRIT      * RX *
000104 000000 A 218      DATA      0
                219 *      COUNT OF LOGICAL WRITES,I.E. CALL'S TO SBWRIT
                000105 A 220 LWRITS EQU      LREADS+1    * RX *
000105 000000 A 221      DATA      0
                222 *      COUNT OF PHYSICAL READS,I.E. ACTUAL READ I/O'S
                000106 A 223 PREADS EQU      LWRITS+1    * RX *
000106 000000 A 224      DATA      0
                225 *      COUNT OF PHYSICAL WRITES,I.E. ACTUAL WRITE I/O'S
                000107 A 226 PWRITS EQU      PREADS+1    * RX *
    
```



VORTXII DASMR DATBAS 2039 HOURS

```

000107 000000 A 227 DATA 0
                228 * COUNT OF TOTAL FUNCTIONS ISSUED BY USER
000110 000110 A 229 FUNCNT EQU PWRITS+1 * RX *
000110 000000 A 230 DATA 0

                232 * COUNT OF LOGICAL RECORDS WRITTEN TO LOG FILE
000111 A 233 LOGCNT EQU FUNCNT+1 * RX *
                234 *LOG DATA 0

                236 *****
                237 * DBMOD PREFIX
                238 *****
                239 * SIX CHARACTER DBMOD NAME
000111 000111 A 240 DBMODN EQU LOGCNT * RX *
000111 000000 A 241 DATA 0,0,0
000112 000000 A
000113 000000 A

                242 * ADDRESS OF DBMOD
000114 A 243 ADBMOD EQU DBMODN+3 * RX *
000114 010666 R 244 DATA (ENDATB)
                245 * DATE DBMOD WAS GENERATED BY DBGEN
000115 A 246 DBMOAT EQU ADBMOD+1 * RX *
000115 000000 A 247 DATA 0,0
000116 000000 A
000117 000000 A 248 DATA 0,0
000120 000000 A

                249 * TIME DBMOD WAS GENERATED BY DBGEN
000121 A 250 DBMTIM EQU DBMOAT+4 * RX *
000121 000000 A 251 DATA 0,0
000122 000000 A

                252 * ADDRESS OF FILE TABLE
000123 A 253 AFLTAB EQU DBMTIM+2 * RX *
000123 000000 A 254 DATA 0
                255 * ADDRESS OF SAVE AREA
000124 R 256 ESAVEA EQU *
000124 A 257 ASAVEA EQU AFLTAB+1 * RX *
000124 000000 A 258 DATA 0
                259 * ADDRESS OF LOG BUFFER AREA
000125 A 260 ALOGRA EQU ASAVEA+1 * RX *
000125 000000 A 261 DATA 0
                262 * ADDRESS OF LOG FILE FCB
000126 A 263 ATLOG EQU ALOGRA+1 * RX *

```



VORTXII DASMR DATBAS 2039 HOURS

```

000126 000000 A 264 DATA 0
000017 A 265 AATLOG EQU 15 ADDRESS EQU FOR A(ATLOG) IN DBMOD PREFIX
267 *****
268 * TASK GENERAL INFORMATION
269 *****
270 * INTERNAL TASK I.D. (VORTEX II T.I.D.B.)
000127 A 271 WTASKN EQU ATLOG+1 * RX *
000127 000000 A 272 DATA 0
273 * THE FUNCTION TABLE ENTRY FOR THIS FUNCTION
000130 R 274 WFUNCE EQUJ *
000130 A 275 WFUNCT EQUJ WTASKN+1 * RX *
000130 000000 A 276 DATA 0,0
000131 000000 A
000132 000000 A 277 DATA 0,0
000133 000000 A
000134 000000 A 278 DATA 0,0
000135 000000 A
000136 000000 A 279 DATA 0
280 * ADDRESS OF THE FILE DESCRIPTOR FOR FILE NAMED IN PARM 3
000137 A 281 WFFLOC EQU WFUNCT+7 * RX *
000137 000000 A 282 DATA 0 PARAMETER 3
283 * ADDRESS OF FILE DESCRIPTOR CURRENTLY BEING ACCESSED
000140 A 284 WFFADR EQU WFFLOC+1 * RX *
000140 000000 A 285 DATA 0
286 * ACCESS MODE SPECIFIED IN USER SINON
000141 A 287 WACMOD EQU WFFADR+1 * RX *
000141 000000 A 288 DATA 0 SINON(RD,UP,RE)
289 * LOG OPTIONS SPECIFIED IN USER SINON
000142 A 290 WLGOPT EQU WACMOD+1 * RX *
000142 000000 A 291 DATA 0
292 * (L-LOG,N-NOLOG)
000143 A 293 * ADDRESS OF USER PROGRAM'S REALM (IN SCHEMA)
000143 000000 A 294 WASCHE EQU WLGOPT+1 * RX *
295 DATA 0
296 * DISPLACEMENT VALUE OF COMMON CONTROL RECORD
297 * USED BY SINON/SINOF
000144 A 298 WCCRAD EQU WASCHE+1 * RX *
000144 000000 A 299 DATA 0
301 *****
302 * STATUS * RX *
    
```



VORTXII DASHR DATBAS 2039 HOURS

```

303 *****
304 * 4 CHARACTER STATUS CODE (IF ANY)
000145 R 305 ESTATS EQU *
000145 A 306 WSTATS EQU WCCRAD+1 * RX *
000145 000000 A 307 DATA 0,0
000146 000000 A
308 * DOCUMENTATION NUMBER ,ADDRESS WITHIN TOTAL WHICH PRODUCED THE
309 * STATUS CODE FOUND IN WSTATS
000147 A 310 DOCNUM EQU WSTATS+2 * RX *
000147 000000 A 311 DATA 0
312 * RECOVERY SWITCH
000150 A 313 RECOVS EQU DOCNUM+1 * RX *
000150 000000 A 314 DATA 0
315 * IF 'FU' FUNCTION HAS BEGUN TO PROCESS AND ERRORS MUST BE
316 * BACKED OUT
317 * IF 'RE' RECOVERY HAS ALREADY BEEN REQUESTED AND ANYFURTHER
318 * ERROR IS A 'FATAL'
000151 A 319 KBUG EQU RECOVS+1 * RX *
000151 141325 A 320 DATA 'BUG#' * RX *
000152 143643 A
000153 R 321 BUGLOC EQU *
000153 A 322 DOWNSW EQU KRUG+2
000153 000000 A 323 DATA 0 * RX *
000154 A 326 FUNC EQU KRUG+3
000154 143325 A 327 DATA 'FUNC' * RX *
000155 147303 A
328 * GENERAL FUNCTION TEXT--COMMON TO MOST FUNCTIONS
329 * RELATIVE RECORD NUMBER OF SPACE FOUND FOR AN ADD
000156 A 330 WSPACE EQU FUNC+2 * RX *
000156 000000 A 331 DATA 0
000157 000000 A 332 DATA 0
333 * MASTER FUNCTION DEPENDENT TEXT
334 * WORK (VARIABLE) FIELDS USED BY:RRNCAL
335 * RELATIVE RECORD NUMBER CALCULATED BY RRNCAL
000160 A 336 WHASH EQU WSPACE+2 * RX *
337 * ADDRESS OF KEY TO BE HASHED
000160 A 338 RNAKEY EQU WHASH * RX *
000160 000000 A 339 DATA 0
340 * LENGTH OF KEY TO BE HASHED
000161 A 341 RNLKEY EQU WHASH+1 * RX *

```



VORTXII DASMR DATBAS 2039 HOURS

```

000161 000000 A 342 DATA 0
343 * 'EVEN OR ODD' SWITCH TO TELL RRNCAL IF ON OR OFF A WORD
344 * BOUNDARY.
000162 000162 A 345 SWEORD EQU RNLKEY+1 * RX *
000162 000000 A 346 DATA 0
347 * MASTER SYNONYM SWITCH
348 * IF 'B' THE SLOT IS BLANK
349 * IF 'N' THE SLOT IS OCCUPIED BY A SYNONYM OF ANOTHER 'HOME'
350 * IF 'S' THERE IS A VALID 'HOME' AND IT HAS SYNONYMS
351 * IF 'H' THERE IS A VALID HOME, BUT NO SYNONYMS
000163 000163 A 352 WMSYSW EQU SWEORD+1 * RX *
000163 000000 A 353 DATA 0
354 * SYNONYM CHAIN 'END'
355 * THE RELATIVE RECORD NUMBER OF THE LAST RECORD IN THE CHAIN
000164 000164 A 356 WSYNEN EQU WMSYSW+1 * RX *
000164 000000 A 357 DATA 0,0
000165 000000 A
358 * WORKAREA TO HOLD RRN OF 'LOW-SIDE' IN MASTER SPACE SEARCH
000166 000166 A 359 MFSLOW EQU WSYNEN+2 * RX *
000166 000000 A 360 DATA 0,0
000167 000000 A
361 * ONE DEFINED FOR 32 BIT ARITHMETIC
000170 000170 A 362 MFSLO1 EQU MFSLOW+2 * RX *
000170 000000 A 363 DATA 0,1
000171 000001 A
364 * ONE DEFINED FOR 32 BIT ARITHMETIC
000172 000172 A 365 MFSHI1 EQU MFSLO1+2 * RX *
000172 000000 A 366 DATA 0,1
000173 000001 A
367 * WORKAREA TO HOLD RRN OF 'HIGH-SIDE' IN MASTER SPACE SEARCH
000174 000174 A 368 MFSHIH EQU MFSHI1+2 * RX *
000174 000000 A 369 DATA 0,0
000175 000000 A
370 * VARIABLE FUNCTION DEPENDENT TEXT
371 * THE RELATIVE RECORD NUMBER OF THE CCR 'THIS RRN' IS 'IN'
000176 000176 A 372 WVCCR EQU MFSHIH+2 * RX *
000176 000000 A 373 DATA 0,0
000177 000000 A
374 * SWITCH TO TELL IF THIS FILE HAS BASE, CODED, OR BOTH RECORD TYPES
000200 000200 A 375 WVCODS EQU WVCCR+2 * RX *
000200 000000 A 376 DATA 0
377 * REFER PULLED FROM USER PARAMETER LIST

```



VORTXII DASHR DATBAS 2039 HOURS

```

000201 A 378 WVREFR EQU WVCODS+1 * RX *
000201 000000 A 379 DATA 0
000202 000000 A 380 DATA 0
381 * ADDRESS OF LINKPATH TABLE
000203 A 382 WVALNK EQU WVREFR+2 * RX *
000203 000000 A 383 DATA 0
384 * ADDRESS OF LAST BASE LINKPATH ENTRY IN TABLE (IF ANY)
000204 A 385 WVBAS EQU WVALNK+1 * RX *
000204 000000 A 386 DATA 0
387 * ADDRESS OF 1ST CODED LINKPATH ENTRY IN TABLE (IF ANY)
000205 A 388 WVACOD EQU WVBAS+1 * RX *
000205 000000 A 389 DATA 0
390 * ADDRESS OF LAST CODED LINKPATH ENTRY IN TABLE (IF ANY)
000206 A 391 WVECOD EQU WVACOD+1 * RX *
000206 000000 A 392 DATA 0
393 * ADDRESS OF 1ST LINKPATH IN RECORD (CODED OR BASE)
394 * ADDRESS OF 'CONTROLLING' LINKPATH (I.E. THE ONE REFERENCED IN PE
000207 A 395 WVCTLK EQU WVECOD+1 * RX *
000207 000000 A 396 DATA 0
397 * WORK AREA FOR LOOKUP OF CODED ETC. LINKPATH NAMES
000210 A 398 WVWORK EQU WVCTLK+1 * RX *
000210 399 BSS 5
400 * SWITCH USED BY LINK AND DLINK TO SHOW HISTORY OF PROCESSING
000215 A 401 WLNKSW EQU WVWORK+5 * RX *
000215 000000 A 402 DATA 0
403 * ADDRESS OF LINKPATH NOW BEING PROCESSED IN SAVE AREA RECORD
000216 A 404 WSAVLK EQU WLNKSW+1 * RX *
000216 000000 A 405 DATA 0
406 * ADDRESS OF LINKPATH NOW BEING PROCESSED IN BUFFER RECORD
000217 A 407 WBUFLK EQU WSAVLK+1 * RX *
000217 000000 A 408 DATA 0
409 * ADDRESS OF LINKPATH ENTRY IN TABLE, CORRESPONDING TO WSAVLK
000220 A 410 WSLKEN EQU WBUFLK+1 * RX *
000220 000000 A 411 DATA 0
412 * ADDRESS OF LINKPATH ENTRY IN TABLE, CORRESPONDING TO WBUFLK
000221 A 413 WBLKEN EQU WSLKEN+1 * RX *
000221 000000 A 414 DATA 0
415 * SEQUENTIAL FUNCTION DEPENDENT TEXT
416 *****
000222 A 417 IOTAB EQU WBLKEN+1 * RX *
000222 144717 A 418 DATA 'IOTABL' * RX *
000223 152301 A

```



VORTXII DASMR DATBAS 2039 HOURS

```

000224 141314 A
      419 *****
      000225 A 420 WPRTY EQU IOTAB+3 * RX *
      000225 A 421 WIOSEC EQU WPRTY * RX *
000225 000000 A 422 DATA 0
      000226 A 423 WIOTAB EQU WIOSEC+1 * RX *
000226 000000 A 424 DATA 0
      000227 A 425 WIOLEN EQU WIOTAB+1 * RX *
000227 000000 A 426 DATA 0
      000230 A 427 WIOBAD EQU WIOLEN+1 * RX *
000230 000000 A 428 DATA 0
      000231 A 429 WIOMOD EQU WIOBAD+1 * RX *
000231 000000 A 430 DATA 0
      000232 A 431 WIOOVR EQU WIOMOD+1 * RX *
000232 000000 A 432 DATA 0
      000233 A 433 WIORBM EQU WIOOVR+1 * RX *
000233 000000 A 434 DATA 0
      000234 R 435 EIOLOC EQU *
      000234 A 436 WIOLOC EQU WIORBM+1 * RX *
000234 000000 A 437 DATA 0
      000235 A 438 WRDRCD EQU WIOLOC+1 * RX *
000235 000000 A 439 DATA 0,0
000236 000000 A

      441 *****
      000237 A 442 MISC EQU WRDRCD+2 * RX *
000237 153717 A 443 DATA 'WORK' * RX *
000240 151313 A

      444 *****
      445 * WORK (VARIABLE) FIELDS USED BY: SCMPN *
      446 * SCOMPE *
      447 * TABLUK *
      000241 R 448 ECOMPA EQU *
      000241 A 449 WCOMPA EQU MISC+2 * RX *
000241 000000 A 450 DATA 0
      000242 R 451 ECOMPB EQU *
      000242 A 452 WCOMPB EQU WCOMPA+1 * RX *
      000242 A 453 FOUND EQU WCOMPB * RX *
000242 000000 A 454 DATA 0
      455 * WORK (VARIABLE) FIELDS USED BY: MOVE4 *
      456 * SCLEAR WITH *
      457 * SZERO *
    
```



VORTXII DASMR DATBAS 2039 HOURS

		458 *			SBLKRD		
		459 *			SBLKSA		
		460 *			SMOVEW		
		461 *			CMPBYT		
		462 *			MOVBYT		
		463 *			CHKBYT		
	000243 R	464 EVBDAT	EQJ	*			
	000243 A	465 MVBDAT	EQJ	FOUND+1	* RX *		
	000243 R	466 EMBDAT	EQJ	*			
	000243 A	467 CMRDAT	EQJ	MVBDAT	* RX *		
000243	000000 A	468	DATA	0			
	000244 R	469 EATO	EQJ	*			
	000244 A	470 ATD	EQJ	MVBDAT+1	* RX *		
000244	000000 A	471	DATA	0			
	000245 R	472 EAFROM	EQJ	*			
	000245 A	473 AFROM	EQJ	ATD+1	* RX *		
000245	000000 A	474	DATA	0			
		475 *	WORK (VARIABLE) FIELDS USED BY: ANY LOGIC INVOLVING DOUBLE PRECISION ARITHMETIC				
		476 *					
		477 *	ONE DEFINED FOR 32 BIT ARITHMETIC				
	000246 A	478 WRKONE	EQJ	AFROM+1	* RX *		
000246	000000 A	479	DATA	0,1			
000247	000001 A						
	000250 A	480 WORK	EQJ	WRKONE+2	* RX *		
000250	000000 A	481	DATA	0,0			
000251	000000 A						
	000252 R	482 EWORK1	EQJ	*			
	000252 A	483 WORK1	EQJ	WORK+2	* RX *		
000252	000000 A	484	DATA	0,0			
000253	000000 A						
	000254 A	485 WORK2	EQJ	WORK1+2	* RX *		
000254	000000 A	486	DATA	0,0			
000255	000000 A						
	000256 A	487 WTOTAL	EQJ	WORK2+2	* RX *		
000256	000000 A	488	DATA	0,0			
000257	000000 A						
	000260 A	489 WORK3	EQJ	WTOTAL+2	* RX *		
000260	000000 A	490	DATA	0,0			
000261	000000 A						
		491 *	WORK (VARIABLE) FIELDS USED BY: TABLUK FUNCTION LOOKUP				
	000262 A	492 TLUKFN	EQJ	WORK3+2	* RX *		
	000263 A	493 TAFUNT	EQJ	TLUKFN+1	* RX *		



VURTXII DASHR DATBAS 2039 HOURS

```

000262 001012 R 494 DATA (FUNTAB+1) * RX *
000263 000007 A 495 TLFUNT DATA 7
000264 000002 A 496 TFNARG DATA 2
497 * WORK (VARIABLE) FIELDS USED BY:SFFILE FOR TABLUK *
000265 A 498 TLUKFL EQU TLUKFN+3 * RX *
000265 A 499 TAFILE EQU TLUKFL * RX *
000265 000000 A 500 DATA 0 * RX *
000266 000003 A 501 TLFILF DATA 3
000267 000002 A 502 DATA 2 LENGTH OF ARGUMENT
503 * WORK (VARIABLE) FIELDS USED BY:SELECT FOR TABLUK *
000270 A 504 TLUKEL EQU TLUKFL+3 * RX *
000270 A 505 TAELEM EQU TLUKEL * RX *
000270 000000 A 506 DATA 0 * RX *
000271 000006 A 507 TLELEM DATA 6
000272 000002 A 508 DATA 2 LENGTH OF ARGUMENT
509 * WORK (VARIABLE) FIELDS USED BY:LINKPATH LOOKUP FOR TABLUK *
000273 A 510 TLUKLK EQU TLUKEL+3 * RX *
000273 A 511 TALINK EQU TLUKLK * RX *
000273 000000 A 512 DATA 0 * RX *
000274 000010 A 513 TLLINK DATA 8
000275 000004 A 514 DATA 4 LENGTH OF ARGUMENT
515 * WORK (VARIABLE) FIELDS USED BY: SFLKBF FOR TABLE LOOKUP
000276 A 516 TLUKLC EQU TLUKLK+3 * RX *
000276 A 517 TALKCD EQU TLUKLC * RX *
000276 000000 A 518 DATA 0 * RX *
000277 000010 A 519 TLLKCD DATA 8
000300 000005 A 520 TLCLEN DATA 5
521 * WORK (VARIABLE) FIELDS USED BY: TABLE LOOKUP RECORD CODES
000301 A 522 TLUKCD EQU TLUKLC+3 * RX *
000301 A 523 TACUDE EQU TLUKCD * RX *
000301 000000 A 524 DATA 0
000302 R 525 TLENCO EQU *
000302 000010 A 526 DATA 8
000303 R 527 TLCODE EQU *
000303 000001 A 528 DATA 1
000304 A 529 TLUKEC EQU TLUKCD+3 * RX *
000304 A 530 TAELCD EQU TLUKEC * RX *
000304 000000 A 531 DATA 0 * RX *
000305 000006 A 532 TLELCD DATA 6
000306 000003 A 533 TLEARG DATA 3
535 * WORK (VARIABLE) FIELDS USED BY: SELECT *
    
```



VORTXII DASM R DATBAS 2201 HOURS

```

6468          EJEC
6469 *****
6470 *
6471 *          FILE TABLE DEFINITION
6472 *
6473 *****
000000 A 6474 TFTNME EQU      0          2 WORD NAME OF THE FILE
6475 *
000002 A 6476 TFTADD EQU      2          1 WORD ADDRESS OF THE GENERAL FILE DESCRIP
6477 *                               TION FOR THIS FILE
6478 *
6479 * CTF TIC DATA 3  THIS IS THE LENGTH OF ONE FILE TABLE ENTRY WORDS

```

10/26/75

VORTXII DASM R DATBAS 2201 HOURS

```

6480          EJEC
6481 *****
6482 *
6483 *          INTABLE (IOPDOL) FNTRY DESCRIPTION
6484 *
6485 *          AT SINON THE POOL ENTRY HAS SOME VALUES ESTABLISHED
6486 *          BY THE DBGEN, USING THESE VALUES, THE MBEGIN LOGIC MUST
6487 *          INITIALIZE THE BUFFER PREFIXES
6488 *****
6489 *
000000 A 6490 PL1ST  EQU      0          ADDRESS OF THE FIRST BUFFER PREFIX
6491 *
000001 A 6492 PLAST  EQU      1          BEFORE SGON - THE COUNT OF BUFFERS IN POOL
6493 *          AFTER SGON - THE ADDRESS OF THE LAST
6494 *          BUFFER PREFIX
000002 A 6495 PLPRTY EQU      2          BEFORE SGON - THE LENGTH OF 1 BUFFER
6496 *          INCLUDING THE LENGTH OF THE PREFIX
6497 *          AFTER SGON - THE FIRST BYTE IS THE HIGHEST
6498 *          PRIORITY BUFFER IN THE POOL AS
6499 *          FOUND BY THE BUFFER SEARCH LOGIC
6500 *          IN SBREAD.
000002 A 6501 PLCK  EQU      PLPRTY    AFTER SGON - THE SECOND BYTE IS A LOCK
6502 *          SWITCH
6503 *          -L- LOCKED
6504 *          -N- NOT LOCKED
000003 A 6505 PLRUFF EQU      3          THE ADDRESS OF THE BUFFER CORRESPONDING TO
6506 *          THE 'PLPRTY' SET ABOVE

```



VORTXII DASMR DATBAS 2201 HOURS

```

6507          EJEC
6508 *****
6509 *
6510 *          BUFFER PREFIX DEFINITION
6511 *          THE ACTUAL BUFFER BEGINS IMMEDIATELY AFTER THE PREFIX
6512 *
6513 *****
000000 A 6514 TIDLEN EQU      0          LENGTH OF THE BUFFER CURRENTLY IN USE
6515 *          -NOT-INCLUDING THE PREFIX
000001 A 6516 TIDNXT EQU      1          ADDRESS OF NEXT BUFFER PREFIX OR
6517 *          POOL CONTROL BLOCK
000002 A 6518 TIDBAD EQU      2          ADDRESS OF THIS BUFFER PREFIX
6519 *          -----
000003 A 6520 TIDAFD EQU      3          ADDRESS OF FILE DESCRIPTOR CURRENTLY
6521 *          USING THIS BUFFER
000004 A 6522 TIDMOD EQU      4          ADDRESS OF MODULE (FCB) THIS BUFFER WAS
6523 *          READ FROM
6524 *          NOTE- A BUFFER COULD BE READ FROM MULTIPLE FCB'S IF THEY CROSS
6525 *          A BOUNDARY WITH MULTIPLE SECTORS -THEY MUST- BE ADJACENT
6526 *          -----
6527 *          SO WE NEED ONLY KNOW IF WE NEED 2 FCB'S AND THEN WHICH
6528 *          FCB GETS WHICH SECTOR'S
000005 A 6529 TIDOVR EQU      5          THE NUMBER OF SECTORS TO WRITE TO
6530 *          THE FCB SPECIFIED IN TIDMOD-THE REST ARE
6531 *          TO BE WRITTEN TO THE NEXT FCB IN THE TABLE
6532 *          IF THIS IS -NOT- AN OVERLAPPED BUFFER THE WORD IS ZERO
000006 A 6533 TIDRRM EQU      6          RELATIVE SECTOR # WITHIN THE MODULE (FCB)
6534 *          SPECIFIED
000007 A 6535 TIDLNC EQU      7          ADDRESS OF CURRENT LOGICAL RECORD WITHIN
6536 *          THIS BUFFER -THIS IS 'BYTE FORMAT'
000010 A 6537 TIDRCO EQU      8          RELATIVE RECORD NUMBER OF THE RECORD
6538 *          CURRENTLY REQUESTED IN THE BUFFER
6539 *
000012 A 6540 TIDUPD EQU      10         1ST BYTE IS AN UPDATE STATUS SWITCH
6541 *          -U- DESIGNATES THIS BUFFER WAS UPDATED
6542 *          -N- DESIGNATES THIS BUFFER WAS NOT UPDATED
000012 A 6543 TIDLCK EQU      TIDUPD    2ND BYTE IS A LOCK BYTE FOR THIS BUFFER
6544 *          -L- LOCKED
6545 *          -N- NOT LOCKED
6546 *
6547 *
6548 *          THE BUFFER PREFIX IS 11 WORDS LONG
    
```



VORTXII DASMR DATBAS 2201 HOURS

```

6549          EJEC
6550 *****
6551 *
6552 *          GENERAL FILE DESCRIPTOR TABLE
6553 *
6554 *****
000000 A 6555 TGFNAM EQU      0          2 WORDS NAME OF DATA BASE FILE
6556 *
000002 A 6557 TGFDAD EQU      2          ADDRESS OF OPERATING SYSTEM FILE DESCRIPTOR
6558 *          THIS IS ACTUALLY THE ADDRESS OF THE MODULE
6559 *          TABLE, SINCE THE FCB(FILE CONTROL BLOCK)
6560 *          IS REQUIRED FOR SECTIONING THE LOGICAL FILE
6561 *
000003 A 6562 TGFAIO EQU      3          ADDRESS OF I/O TABLE ENTRY FOR BUFFER
6563 *          ASSIGNED TO THIS FILE (POOL ENTRY)
000004 A 6564 TGFTRD EQU      4          2 WORDS CONTAINING THE TOTAL # OF LOGICAL
6565 *          RECORDS IN THE FILE
000006 A 6566 TGFRCY EQU      6          1 WORD COUNT OF THE # OF LOGICAL RECORDS
6567 *          PER CYL
000007 A 6568 TGFRBK EQU      7          1 WORD COUNT OF THE # OF LOGICAL RECORDS
6569 *          WITHIN A SECTOR, IF THE LOGICAL RECORD
6570 *          IS LARGER THAN THE SECTOR SIZE THIS IS THE
6571 *          NUMBER OF SECTORS WITHIN THE RECORD
000010 A 6572 TGFLRZ EQU      8          1 WORD COUNT OF THE # OF BYTES IN A LOGICAL
6573 *          RECORD
000011 A 6574 TGFLRZ EQU      9          1 WORD COUNT OF THE # OF BYTES IN A SECTOR
6575 *
000012 A 6576 TGFTYP EQU     10          FIRST BYTE OF THIS WORD IS THE ONE
6577 *          CHARACTER FILE TYPE 'M' IS A MASTER
6578 *          CHARACTER FILE TYPE 'V' IS A VARIABLE
000012 A 6579 TGFOCS EQU     10          SECOND BYTE OF THIS WORD IS A ONE
6580 *          CHARACTER OPEN/CLOSE SWITCH 'O' IS OPEN
6581 *          'C' IS CLOSED
000013 A 6582 TGFSPA EQU     11          2 WORD FILE SPACE STATUS
6583 *          IF SPACE AVAILABLE ****
6584 *          IF FILE ABOVE LOAD LOAD
6585 *          IF FILE FULL FULL
000015 A 6586 TGFUMD EQU     13          1 WORD USAGE MODE
6587 *          'U' THE FILE IS UPDATE
6588 *          'R' THE FILE IS READONLY
000016 A 6589 TGFELT EQU     14          ADDRESS OF ELEMENT TABLE
6590 *****

```



VORTXII DASHR DATABS 2201 HOURS

```

6591 *      COMMON FILE CONTROL RECORD *
6592 *      THE FOLLOWING AREA CONTAINS THE PORTION OF THE FILE CONTROL *
6593 *      RECORD THAT IS THE SAME FOR BOTH MASTER AND VARIABLE FILE TYPES*
6594 *****
000017 A 6595 TGFFIL EQU      15      NAME OF FILE AS STORED IN CONTROL RECORD
000021 A 6596 TGFLNK EQU      17      LOCK WORD
000022 A 6597 TGFTID EQU      18      TASK I.D. OF TASK WITH LOCK ON FILE
000023 A 6598 TGFCNT EQU      19      LOGICAL RECORD COUNT FOR THIS FILE 2 WORDS
6599 *****
6600 *
6601 * MASTER FILE UNIQUE DESCRIPTION *
6602 *
6603 *****
000025 A 6604 TMFPRM EQU      21      2 WORD BINARY # CONTAINING THE CLOSEST
6605 *      PRIME # TO THE TOTAL # OF RECORDS IN THE
6606 *      FILE
000027 A 6607 TMFKLN EQU      23      1 WORD LENGTH (IN BYTES) OF THE KEY
6608 *
6609 *
6610 *****
6611 *
6612 * VARIABLE FILE UNIQUE DESCRIPTION *
6613 *
6614 *****
000025 A 6615 TVFLNK EQU      21      ADDRESS OF VARIABLE LINKAGE TABLE
6616 *
000026 A 6617 TVFCCN EQU      22      2 WORD RELATIVE RECORD # OF THE CURRENT
6618 *      CYLINDER CONTROL RECORD IF NO CCR IN USE
6619 *      THIS CONTAINS 'NONE'
000030 A 6620 TVFNXT EQU      24      2 WORD RELATIVE RECORD # OF NEXT AVAILABLE
6621 *      RECORD IN THIS CYLINDER
000032 A 6622 TVFUSE EQU      26      1 WORD COUNT OF THE NUMBER OF RECORDS USED
6623 *      IN THIS CYLINDER
000033 A 6624 TVFLMT EQU      27      1 WORD 'LOAD' LIMIT THE # OF RRNS ABOVE
6625 *      WHICH NEW CHAINS SHOULD NOT BE ADDED'
000034 A 6626 TVFLCY EQU      28      2 WORD RRN OF THE CCR IN THE FIRST CYL NOT
6627 *      'AT OR ABOVE' LOAD LIMIT IF ALL CYL'S ARE
6628 *      ABOVE LOAD THE HIGH ORDER BYTE IS 'L' AND
6629 *      THIS IS THE FIRST CYL. 'NOT FULL', IF ALL
6630 *      CYL.'S ARE FULL THIS CONTAINS 'FULL'
000036 A 6631 TVFCUP EQU      30      FIRST BYTE IS AN UPDATE INDICATOR FOR THIS
6632 *      CCR 'N' MEANS IT HAS NOT BEEN UPDATED
    
```



VORTXIT DASHR DATBAS 2201 HOURS

```

6633 *                               'U' MEANS IT HAS BEEN UPDATED
6634 *
6635 *
6636 *      VARIABLE FILE LINKPATH TABLE
6637 *
6638 *
10000 A 6639 TVLPNM EQU      0      4 WORDS (8 CHARACTERS) OF LINKPATH NAME
6640 *      THE FIRST 2 WORDS (4 BYTES) ARE THE MASTER
6641 *      FILE NAME THIS LINK IS CONNECTED TO, THE
6642 *      NEXT 2 WORDS(4 BYTES) IS THE LINKPATH NAME
10004 A 6643 TVLCOD EQU      4      1 WORD RECORD CODE UNIQUE TO THIS LINKPATH
6644 *      IF THE LINKPATH IS COMMON TO ALL RECORD
6645 *      TYPES THIS IS '***'
10005 A 6646 TVLVLD EQU      5      1 WORD COUNT IN BYTES REPRESENTING THE
6647 *      DISPLACEMENT TO THIS LINK FROM THE BEGIN-
6648 *      NING OF THE LOGICAL RECORD
00006 A 6649 TVLVKD EQU      6      AS TVLVLD FOR THE KEY IN THE VARIABLE REC-
6650 *      ORD
00007 A 6651 TVLMLO EQU      7      AS 'TVLVLD' BUT DISPLACEMENT FOR LINKPATH
6652 *      IN MASTER RECORD
6653 *
6654 *
6655 *      ELEMENT TABLE
6656 *
6657 *
1000 A 6658 TELNAM EQU      0      2 WORDS(4 BYTES) WHICH IS THE LAST 4 BYTES
6659 *      OF THE ELEMENT NAME - THE FIRST 4 ARE
6660 *      ASSUMED - THEY MUST (BY DBGEN) BE THE NAM
6661 *      OF THE FILE
1002 A 6662 TELCOD EQU      2      1 WORD RECORD CODE UNIQUE FOR THIS ELEMENT
6663 *      IF THIS ELEMENT IS COMMON TO ALL RECORDS
6664 *      IT IS '***'
1003 A 6665 TELFTY EQU      3      FIRST BYTE IS THE 'FIELD TYPE' I.E.
6666 *      'K' IS A KEY
6667 *      'L' IS A LINKPATH
6668 *      'R' IS A ROOT IN MASTER RECORDS
6669 *      ' ' (BLANK) IS A NORMAL ELEMENT
6670 *      PRETTY USELESS EXCEPT IN MASTER FILE RECORDS
1003 A 6671 TELLEV EQU      3      SECOND BYTE IS A ONE CHARACTER LEVEL #
6672 *
1004 A 6673 TELDSP EQU      4      1 WORD DISPLACEMENT(IN BYTES) FROM BEGIN-
6674 *      NING OF THIS RECORD TO THIS ELEMENT

```



VORTXIT DASM R DATBAS 2201 HOURS

```

1005 A 6675 TELLEN EQU      5          1 WORD LENGTH (IN BYTES) OF THIS ELEMENT
6676 *****
6677 *          RFLATIVE LABELS FOR FILE/CYLINDER CONTROL RECORDS          *
6678 *****
6679 *
1000 A 6680 VCRNXT EQU      0
1002 A 6681 VCRUSE EQU      2
1003 A 6682 VCRLMT EQU      3
1004 A 6683 VCR1ST EQU      4
1014 A 6684 VCREND EQU      12
6685 *
1000 A 6686 CCRNAM EQU      0          FILE NAME 2 WORDS
1002 A 6687 CCRLCK EQU      2          LOCK WORD 1 WORD
1003 A 6688 CCRTID EQU      3          TASK I.O. DARNING LOCK 1 WORD
1004 A 6689 CCRCNT EQU      4          COUNT OF RECORDS USED IN THIS FILE
6690 *          2 WORDS
1006 A 6691 MCREND EQU      6
    
```

10/26/75

VORTXIT DASM R DATBAS 2201 HOURS

```

6692          EJEC
6693 *****
6694 *
6695 *          I/O MACRO CONTROL BLOCK EXPANSION          *
6696 *
6697 *****
6698 *
1000 A 6699 IOHJSR EQU      0          2 WORD JSR X TO I/O SUPERVISOR
6700 *
1002 A 6701 IOBSTS EQU      2          1 WORD STATUS OF I/O REQUEST -SIGN BIT ON
6702 *          INDICATES COMPLETION,LOW ORDER (RIGHT-
6703 *          MOST) BIT ON INDICATES UNRECOVERABLE I/O
6704 *          ERROR
6705 *
1003 A 6706 IOBLUN EQU      3          LOW ORDER 8 BITS ARE LOGICAL UNIT #
6707 *
1004 A 6708 IOBFCB EQU      4          ADDRESS OF FCB
6709 *
1005 A 6710 IOBITD EQU      5          ADDRESS OF TIDB
6711 *
1006 A 6712 IOBIOC EQU      6          I/O CONTROL THREAD ADDRESS
    
```



VORTXII DASM R DATBAS 2201 HOURS

```

6713      EJEC
6714 *****
6715 *
6716 *      FILE PARTITION CONTROL BLOCKS - (MODULE TABLE)
6717 *
6718 *  A LOGICAL FILE (TOTAL FILE) MAY CONSIST OF MULTIPLE NON-CONTIGUOUS
6719 *      DISK AREAS CALLED PARTITIONS - NO PARTITION MAY EXCEED 32,768
6720 *      SECTORS - ALL SECTORS ARE 120 WORDS (240 BYTES) LONG
6721 *      THEREFORE ALL BUFFERS ALLOCATED ARE 120 WORDS LONG
6722 *      IN ORDER TO PROVIDE SOME ERROR CHECKING EACH PARTITION WILL HAVE
6723 *      AN OPERATING SYSTEM FILE GENERATED FOR IT. THESE FILES REQUIRE
6724 *      A CONTROL BLOCK (FILE CONTROL BLOCK) THESE FILES WILL BE
6725 *
6726 *      GENERATED WITH SIX BYTE NAMES, WHERE THE FIRST FOUR BYTES ARE
6727 *  THE TOTAL FILE NAME AND THE LAST TWO ARE NUMERIC PARTITION SEQUENCE
6728 *      NUMBERS FROM 00 TO 99.
6729 *      ALTHOUGH RECORDS-PER-BLOCK WILL REMAIN A VALID INPUT CONTROL
6730 *      IT WILL BE IGNORED - NO STORAGE WILL BE GENERATED AND IT CAN
6731 *      HAVE NO EFFECT
6732 *      IF THE LOGICAL RECORD SIZE IS LESS THAN 240 BYTES ONLY ONE
6733 *      BUFFER WILL BE GENERATED (OF 120 WORDS - SECTOR SIZE)
6734 *      IF THE LOGICAL RECORD SIZE IS GREATER THAN 240 BYTES DBGEN WILL
6735 *      GENERATE AS MANY BUFFERS OF 120 WORDS AS ARE NECESSARY
6736 *      TO EQUAL OR EXCEED THAT RECORD LENGTH
6737 *
000000 A 6738 TMDFCB EQU      0      THIS IS THE FILE CONTROL BLOCK GENERATED
6739 *      FOR THIS PARTITION
6740 *****
000000 A 6741 FCBRLN EQU      0      LOGICAL RECORD LENGTH
6742 *
000001 A 6743 FCHADD EQU      1      DATA ADDRESS
6744 *
000002 A 6745 FCHACS EQU      2      ACCESS MODE (FIRST BYTE)
6746 *
000002 A 6747 FCRKEY EQU      2      PROTECT KEY (SECOND BYTE)
6748 *
000003 A 6749 FCBASN EQU      3      CURRENT SECTOR NUMBER
6750 *
000004 A 6751 FCBEN EQU      4      CURRENT END OF FILE SECTOR NUMBER
6752 *
000005 A 6753 FCBIST EQU      5      FIRST SECTOR #
6754 *

```




VORTXII DASMR DATBAS 2201 HOURS

000006	A	6755	FCHLST	EDJ	6	LAST SECTOR #, NUMBER OF SECTORS IN THIS PARTITION
		6756	*			
000007	A	6757	FCHNAM	EDJ	7	FILE NAME-3 WORDS, THE FORMAT IS DESCRIBED IN COMMENTS PREFIXING THIS DEFINITION
		6758	*			
		6759	*			
000012	A	6760	TMDLOW	EDJ	10	2 WORDS LOWEST RRN IN THIS PARTITION
		6761	*			
000014	A	6762	TMDHIH	EDJ	12	2 WORDS HIGHEST RRN IN THIS PARTITION
		6763	*			
000016	A	6764	TMDNUM	EDJ	14	2 WORDS NUMBER OF LOGICAL RECORDS IN THIS PARTITION
		6765	*			
000020	A	6766	TMDLUN	EDJ	16	LOGICAL UNIT NUMBEROVE
		6767	*			

10/26/75

VORTXII DASMR DATBAS 2201 HOURS

		6768	EJEC			
		6769	*****			
		6770	*			
		6771	*	LOG RECORD PREFIX DESCRIPTION		
		6772	*			
		6773	*****			
000000	A	6774	TLGLEN	EDJ	0	LENGTH OF LOG RECORD IN BYTES / WORD
		6775	*			
000001	A	6776	TLGSER	EDJ	1	2 WORD SERIAL # OF LOG RECORD
		6777	*			
000003	A	6778	TLGPNM	EDJ	3	4 WORD PROGRAM NAME
		6779	*			
000007	A	6780	TLGOTB	EDJ	7	3 WORD DBMOD NAME
		6781	*			
000012	A	6782	TLGFNM	EDJ	10	2 WORD FILE NAME IF APPROPO.ELSE '*****'
		6783	*			
000014	A	6784	TLGRRN	EDJ	12	2 WORD RELATIVE REC # OR SNOB/SNOF/QUIE/ MARK
		6785	*			
000016	A	6786	TLGREC	EDJ	14	ACTUAL RECORD IMAGE OR DATE FOR SINOX OR USER MESSAGE FOR QUIET / MARK
		6787	*			



VORTXII DASMR DATBAS 2201 HOURS

```

6788      EJEC
6789 *****
6790 *      USER BEGIN PARAMETER TABLE
6791 *      THIS IS THE FORMAT DEFINITION OF THE USER PROGRAM'S
6792 *      'SINON SCHEMA'
6793 *
6794 *****
000000 A 6795 TURPGM EQU      0      4 WORD (8 BYTE) LOGICAL NAME OF USER PROGR
6796 *
000004 A 6797 TUBDTB EQU      4      3 WORD (6 BYTE) NAME OF DATA BASE MODULE
6798 *
000007 A 6799 TUBACS EQU      7      3 WORD (6 BYTE ACCESS MODE) AS FOLLOWS:
6800 *      'ROONLY' DENOTES USER PROGRAM WILL NOT
6801 *      UPDATE THE DATA BASE
6802 *      'UPDATE' DENOTES THE USER PROGRAM WILL
6803 *      UPDATE THE DATA BASE
000012 A 6804 TUBOPT EQU     10      1 WORD (2 BYTE) LOG OPTION FIELD AS FOLLOW
6805 *      'LG' DENOTES LOGGING REQUESTED
6806 *      'NL' NO LOGGING REQUESTED
6807 *      THE FOLLOWING 3 ENTRIES REPRESENT A SET, REPEATED FOR EACH FILE
6808 *      ACCESSED BY THE USER
6809 *
000000 A 6810 TURFIL EQU      0      2 WORD NAME OF FILE
6811 *
000002 A 6812 TUBMOD EQU      2      2 WORD ACCESS MODE FOR THE SPECIFIED FILE
6813 *      'SHRE' DENOTES THE USER WILL NOT UPDATE
6814 *      THE FILE
6815 *      'PRIV' DENOTES THE USER WILL UPDATE THE
6816 *      FILE
6817 *
000004 A 6818 TURSTA EQU      4      2 WORD FIELD TO RECIEVE THE STATUS CODE ON
6819 *      RETURN FROM SINON
6820 *****
010670 R 6821 ENDATB EQU      *
6822      END

```

0 ERRORS ASSEMBLY COMPLETE