HONEYWELL EDP

SOFTWARE MANUAL

SERIES 200

SERIES 200/OPERATING SYSTEM MOD 1 (MASS STORAGE RESIDENT)

GENERAL SYSTEM:	SERIES 200/OPERATING SYSTEM - MOD 1 (MASS STORAGE RESIDENT)
SUBJECT:	Functional Description of and Programming Procedures for the Components of the Mod 1 Mass Storage Resident Operating System.
SPECIAL INSTRUCTIONS:	This interim manual supersedes the bulletin entitled Preliminary Description of Series 200/ Operating System - Mod 1 (Mass Storage Resident), Order No. 427, dated June 20, 1966. More complete and definitive information will be made available in a forthcoming series of manuals fully describing the Mod 1 Mass Storage Resident Operating System.

SPECIFICATIONS OF SOFTWARE COMPONENTS DESCRIBED HEREIN REMAIN SUBJECT TO CHANGE IN ORDER TO ALLOW THE INTRODUCTION OF DESIGN IMPROVEMENTS.

DATE: December 1, 1966

FILE NO.: 123.0005.131C.0-427*

9375
41266
Printed in U.S.A.

*Underscoring denotes File Number.

PREFACE

This interim manual contains functional descriptions of system components and programming information for the Series 200/Operating System - Mod 1 (Mass Storage Resident). More complete information is forthcoming in a series of manuals (which will supersede this document) that fully describe the programming and operating considerations for the Mass Storage Operating System.

Section I introduces the Mass Storage Operating System and includes descriptions of certain features which are not yet available. These features are described herein to indicate the scope of the operating system and to place them in their proper perspective. Subsequent sections contain descriptions of system components, as well as programmer's preparation information. Section II describes the Supervisor, Section III describes the Data Management Subsystem, Section IV describes the Program Development Subsystem, and, finally, Section V describes the Utility Routines. A series of appendices provides additional information peculiar to the Mass Storage Operating System.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont).

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I

INTRODUCTION

The Series 200 Operating System Mod 1 (Mass Storage Resident) is an integrated software data processing system. It includes the sophisticated software necessary for simplified programming and efficient operations, and brings the advantages of mass storage to users of the Series 200 Computer Systems. The operating system runs with all Series 200 systems that have at least 12K characters of main memory and a mass storage device.

In the operating system, data is handled by macro statements used with the Easycoder assembly language, and by the mass storage language elements incorporated into the COBOL and FORTRAN Compilers. The user controls the operating system by instructions to a monitor program. The monitor program runs the jobs, supervises multi-programming, and calls other system programs as they are required.

OPERATING SYSTEM OBJECTIVES

The operating system, whose components are shown in Figure 1-1, is designed to achieve four major objectives: provide assistance to the user, increase system throughput, reduce response time, and provide for flexibility and orderly growth of the computing system.

Assisting the user is accomplished by providing data and program management facilities, providing easily understood programming languages and translators that convert these languages into executable programs, providing standard modes of operation so that the programmer is not normally required to prepare large amounts of control information, and by providing facilities for operator communication with the system through specific, clear instructions.

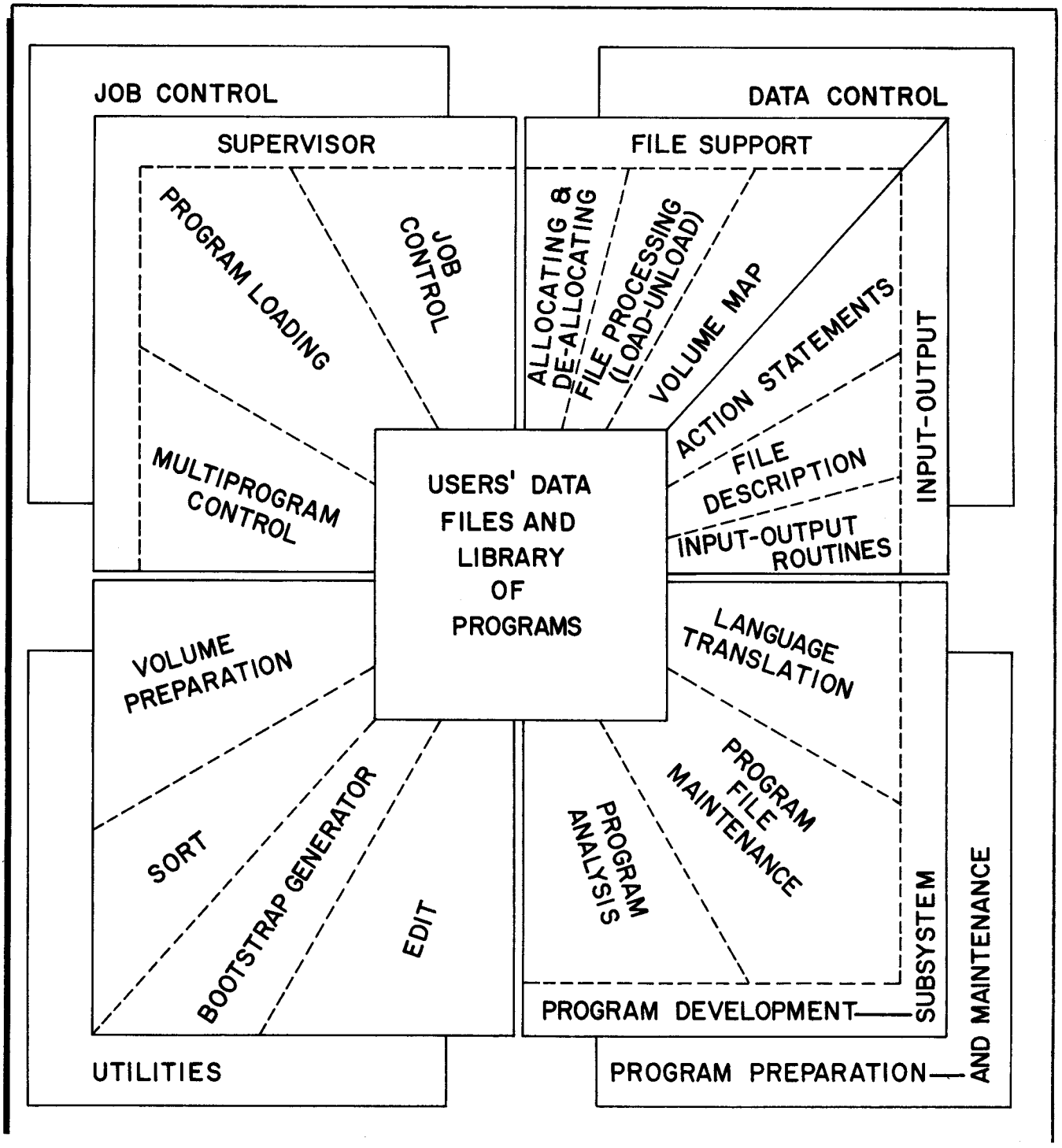


FIGURE 1-1. Components of the Mass Storage Operating System

Throughput, the total amount of work performed by the system over a period of time, is increased in the following ways:

1. The operating system can process a continuous stream of jobs without delays via the automatic transition from one job to the next. Delays (due to operator mounting and demounting of input/output volumes) are reduced because all system programs and libraries of user's programs can be resident on the same on-line mass storage volume and may use work areas on the same volume.

2. The operating system uses the direct access abilities inherent in the mass storage device to locate programs and files without time consuming searches.

3. The operator system efficiently uses the physical resources of the computer system. It can overlap central processor operations with input/output operations.

4. It can control the allocation of central processor time, switching from one program to another while awaiting the completion of an input/output operation. (This process is termed multi-programming.)

Response time (turn-around time) is the interval between the time a job is submitted for processing and the time that a result is available. Response time is reduced in the following ways:

1. Operations are unbatched; i.e., initiated, performed, and completed on a single job at a time. This provides output to the user without time consuming delays caused by waiting for other jobs to be completed (as in batched operations).

2. In an integrated system, input/output operations can be performed concurrently with normal processing. This eliminates the delays caused by the transition from operation to operation and by human

activity involved in performing such operations on an off-line peripheral system.

Flexibility and orderly growth of the computing system are provided for through a variety of programming facilities. The open-end design of the operating system allows the user's programs and data files to be incorporated easily into the system. Because the system is made up of independent modules, the facilities of the system can be combined in a variety of ways.

FUNCTIONAL DESCRIPTION

The operating system provides supervisory, data managing, program development, and service functions. Each of the four major functions is performed by a subsystem of the operating system. These subsystems are named for the function they perform. That is, the subsystem that performs supervisory functions is the Supervisor, the subsystem that performs data managing functions is the Data Management subsystem, etc. The basic functions of the Supervisor are controlling the sequence of jobs and finding and loading the programs to perform a job. The primary function of the Data Management subsystem is the creation, maintenance, and input/output processing of all files. The Program Development subsystem's primary functions are the creation and maintenance of libraries of programs, and the source language translation to machine language. The Service subsystem performs functions such as sorting and editing generally required in any EDP installation.

Supervisor

All processing done by the operating system is done under the general control of the Supervisor. The Supervisor controls processing by performing its functions of Job Control, Program Loading, and Multi-program Control.

JOB CONTROL

Job Control is, simply, the automatic sequencing from one job to the next. The Supervisor performs this function based on information read from the Job Control File. The Job Control File is the input stream of control information identifying a job and defining its requirements. The Supervisor reads the Job Control File and then activates the appropriate element of the system to complete the job. When the job is completed, the activated element informs the Supervisor, which then reads the input from the Job Control File again. This sequence of events goes on until there is no more input in the Job Control File.

PROGRAM LOADING

Program Loading consists of locating the appropriate system program or user program in the machine language program file on the mass storage device, loading it into core memory, and starting its execution. The Supervisor determines which program to load by reading the Job Control File. Several options are provided to control the searching, loading, and starting sequence so that the programmer has complete freedom to set up exactly the sequence of functions he desires.

MULTI-PROGRAM CONTROL

Multi-program Control consists of controlling the simultaneous execution of two programs. The Supervisor controls the sharing of central processor time between the two programs by automatically switching execution from one program to the other, and performing the necessary housekeeping operations. This type of multi-programming is called foreground/background operation. The foreground program user the peripheral interrupt feature and, to run effectively in the multi-program environment, should be peripheral bound. An example of a foreground (peripheral) program is one that would perform a communications job such as on-line inquiry or real-time updating. The background program does not use the peripheral interrupt feature, such

as most of the systems programs of the operating system. The background program is processed during the data transfer time of the foreground program. This facility for Multi-program Control is similar to that offered by Interrupt Control D of the Operating System - Mod 1 (Tape Resident).

Data Management

The Input/Output routines and the File Support routines make up the Data Management subsystem. Mass Storage I/O routines of the Data Management subsystem are a set of macros that enable the programmer to control the I/O operations for the mass storage device performed by a given program. The File Support routines are used to create and reorganize the files stored on the mass storage device. This includes structuring the data into one of the Honeywell standard file organizations.

DATA MANAGEMENT CONCEPTS

The fundamental concept of data management is that all data in a system is organized according to established rules. These rules govern the organization of data into files. The type of file organization governs the access modes that can be used on that file.

File Organizations

The operating system provides individual sets of rules for organizing three types of files; Sequential, Indexed Sequential, and Direct Access Files.

SEQUENTIAL FILES: The operating system accepts files in which data are organized sequentially. In a sequential file, items are arranged in any logical succession perscribed by the user and are accessed in logical and physical order. The user may, optionally, establish several collections of sequentially arranged items in one sequential file. This option is called partitioning and, when used, each individual collection of items is known as a member of the file. Access to the beginning of a sequential file and

to the beginning of a member of a partitioned sequential file is direct.

INDEXED SEQUENTIAL FILES: The operating system also accepts files in which the data are organized in sequence and includes indexes of item keys and addresses. The sequence of items in an indexed sequential file is by item key fields. The indexes are a series of item keys and addresses that enable the user to access items either sequentially or directly by item keys. The user does not process or maintain these indexes. One of the benefits of this type of file organization is that items can be inserted in sequence and deleted without copying the entire file.

DIRECT ACCESS FILES: In the direct access type of file organization, the file is created by the user supplying a mass storage address indicating where the item is to be loaded.

Access Modes

The types of file organizations available with the operating system enable two access modes to be used; Sequential and Direct Access.

SEQUENTIAL ACCESS: Sequential access refers to obtaining or placing items sequentially (in succession). This method of accessing items can be used with either the Sequential file organization, with the Direct Access file organization, or with the Indexed Sequential file organization.

DIRECT ACCESS: Direct Access refers to obtaining or placing items individually or directly. This method can be used with the Indexed Sequential files and with the Direct Access files.

MASS STORAGE I/O CONTROL ROUTINES

The I/O Control routines are a set of macros that enable the programmer to control the I/O operations for a mass storage device. These macros are

named according to their functions as Control Macros, Communication Macros, and Action Macros.

Control Macro

The Control Macro is that portion of the I/O that performs the actual I/O processing. This macro is specialized when assembled to reduce the amount of coding required in main memory and eliminate or include coding as directed by the programmer. The elimination of coding from main memory frees up space for the execution of programs and the inclusion of only the required coding assures the programmer that certain operations will not be performed inadvertently.

Communications Macros

The Communications Macros enable the user to communicate with the I/O routines. For instance, when the system is instructed to open a file for processing and the specified file cannot be found, an exit as specified in the communications area is made to a user supplied routine that determines whether or not the search should continue.

Action Macros

The Action Macros perform such functions as opening and closing files and getting and putting items. This saves the programmer the time and trouble of coding these subroutines himself, and by judicuous specialization via the Control macro, saves main memory locations. For instance, the coding to open a file contains instructions for opening both sequential and direct access files. If only sequential files are to be processed, the instructions that apply only to opening a direct access file can be eliminated.

FILE SUPPORT

The File Support routines are used for the creation and maintenance of all files. These routines are named for the functions they perform.

Allocate/Deallocate Routine

The Allocate/Deallocate routine allocates space for a file and automatically formats that space to accommodate the file, and deallocates files, thus making space available for new files.

Load/Unload Routine

The Load/Unload routine loads and unloads files onto and off of a mass storage device to or from magnetic tape, punched cards, another mass storage device, or the printer.

Map Routine

The Map routine maps the mass storage volumes to provide a tool for determining where on a volume a new file can be written.

Program Development

The Program Development subsystem is an integrated set of routines that assist the user in the process of program creation, translation, modification, and testing. The user makes one request for a connected series of operations on a single program or library routine and the Program Development element automatically controls the sequencing of the various operations in the job. For example, one request might perform the following: updating a program in a source language library, COBOL compilation, storing the output in an executable program library, and execution of the program for testing. To accomplish its functions, the Program Development element is made up of language translators and library maintenance routines.

LANGUAGE TRANSLATORS

The operating system provides languages that enable the programmer to express procedures in forms that can be easily learned and readily used, and translators that convert such programs into a machine-executable format. All language translators in the operating system produce the same format of machine-executable code and can store their outputs on a common file. The Easycoder Assembly, COBOL and FORTRAN language translators are provided for use with the operating system.

Easycoder Assembly

Easycoder Assembly is a symbolic, machine-oriented assembly language with facilities for inclusion and specialization of macro routines. The language level is comparable with Easycoder D of the Operating System - Mod 1 (Tape Resident).

COBOL

COBOL is a business data processing language that is close to normal English language usage. The language level is comparable to COBOL B of the Basic Programming System.

FORTRAN

FORTRAN is a scientific language similar to usual mathmetical notation. The language level is comparable to FORTRAN D of the Operating System - Mod 1 (Tape Resident).

LIBRARY MAINTENANCE ROUTINES

The Program Development element has two library maintenance routines: one to maintain libraries of source language programs, and one to maintain libraries of machine-executable programs. The source language library update routine can add, delete, and replace routines and can correct individual

statements in a source language program. The machine-executable program library update routine can add, delete, or replace routines in this library. This library is created from the output of the language translators.

Service Routines

The operating system provides service routines to perform common data processing functions. Routines are provided to perform volume preparation, sort, and mass storage edit functions.

VOLUME PREPARATION

The volume preparation routine prepares the mass storage volume for use under the Data Management conventions. Volume preparation must be performed once for every mass storage volume at the time the volume is entered into the operating system.

SORT

The sort routine involves ordering the sort key and the address of the associated items. Additional data fields may be extracted, if desired, or the original source item in the sort ordered sequence may be accessed.

EDIT

The editing and printing of selected areas of a mass storage volume is accomplished by the edit routine.

BASIC EQUIPMENT REQUIREMENTS

The basic equipment required to use the operating system is listed below. Some elements require more than the basic configuration.

Series 200 Central Processor[1] (any model) with control panel.

Advanced Programming Instructions (Feature 010, 011, or 1011).

12,288 characters of main memory, of which about 1,500 are required for the Supervisor.

1 mass storage control unit, (Type 255, 257 or 257A).

1 mass storage device for system residence, (Type 256, 258, 259 or 259A).

The Honeywell Series 200/Operating System-Mod 1 (Mass Storage Resident) is designed to operate ultimately with several different mass storage devices. Each control unit for each device allows as many as eight mass storage devices to be connected to the Series 200 computer.

The equipment in the preceding list is the minimum equipment requirement. The systems files can be contained on this equipment.

The required configuration offers the following capabilities:

1. Supervisor -- Sequential job control from card reader, program segment loading from mass storage, and communication via the control panel.
2. Input/Output Routines -- All supported file organizations. The main memory required is assigned to the user's program so that if more memory is available, additional functions can be performed in the user's program.
3. File Support Routines -- All file support routines.
4. Program Development Subsystem -- Assembly and executable and source language program file maintenance.
5. Volume Preparation.
6. Mass Storage Sort.
7. Mass Storage Edit.

[1]This includes the Type 201-Ø and 201-1 Central Processors.

Table 1-1. Devices For System Files

FILE	EQUIPMENT
System Residence	1 mass storage device
Job Control	1 card reader
Operation Control	1 control panel, or 1 console Type 220
Operator Information	1 printer, or 1 console Type 220
Input	1 card reader (may be same as Job Control)
List	1 printer (may be same as Operator Information) or 1 mass storage device (may be same as System Residence)
GO	1 card punch, or 1 mass storage device (may be same as System Residence)
Library	1 mass storage device (may be same as System Residence)
Work Files	1 mass storage device (may be same as System Residence)

NOTE: A standard mass storage volume may be used to run all system programs in the Mass Storage Operating System. This volume is formatted for all the permanent and work files required for the system.

The equipment required for major capabilities not included in the preceding paragraph is listed below.

1. Supervisor Options - Additional main memory is required for use of each of the following options:

   Multi-Program Control.

   Communication via console keyboard/typewriter.

   Program segment loading above location 32,767.

2. COBOL - 12,288 characters of main memory. Edit Instructions (Feature 013 or 1013).

3. FORTRAN - 20,480 characters of main memory. Edit Instructions (Feature 013 or 1013).

Peripheral Devices

The Mass Storage Operating System supports certain peripheral devices by providing input/output routines to perform necessary operations on files stored on or accessed through those devices. These routines may be requested and used in Easycoder, COBOL, and FORTRAN language programs.

Equipment supported includes:

Mass Storage (Types 256, 258, 259, or 259A)
Magnetic Tape (Type 204B)
Card Reader
Card Punch
Printer

SECTION II

SUPERVISOR

All operations in the Mass Storage Operating System are performed under the general control of the Supervisor. The Supervisor performs three main functions: job control, loading and starting execution of program segments, and multi-program control. (Multi-program control is not part of the first release of the Mass Storage Operating System, and is not described in this edition of the manual.)

JOB CONTROL

Job Control is the automatic sequencing from one job to the next. The Supervisor performs this function based on information read from the Job Control File. The Job Control File is the input stream of control information identifying a job and defining its requirements. The Supervisor reads a statement from the Job Control File and then activates the appropriate element of the system to complete the job. When the job is completed, the activated element informs the Supervisor which then reads from the Job Control File again. This sequence of events continues until the input in the Job Control File is exhausted.

PROGRAM LOADING

Program Loading consists of locating the appropriate system program or user program in the machine language program file on the mass storage device, loading it into core memory, and starting its execution. The Supervisor determines which program to load by reading the Job Control File. Several options are provided to control the searching, loading, and starting sequence so that the programmer has complete freedom to set up exactly the sequence of functions he desires.

FUNCTIONS OF THE SUPERVISOR

The Supervisor is initially bootstrapped into main memory from mass storage. This operation is performed once; the Supervisor can then function continuously without being reloaded.

Once in memory, the Supervisor is ready to perform its functions of job control and program segment loading. The Supervisor is controlled by the user in one of two ways: (1) Job and program sequencing are controlled by job control statements. (2) Segment loading within a program is controlled by programmed calls to the Supervisor.

Executing a Job Or Program

The Supervisor starts by reading the Job Control File, which is normally on punched cards and is read into memory via the card reader. The Supervisor searches for an Execute statement. The Execute statement directs the Supervisor to locate, load, and start a named program segment.

Alternatively, the operator may enter job control statements through the control panel. (A later extension will provide the ability to enter job control statements through a Type 220 Console keyboard).

Loading a Program Segment

When the current segment of the program has completed operation, it transfers control to the Supervisor to load another segment. To do this, the program executes a macro call[1] to the Supervisor. The Supervisor then searches the directory for the mass storage address of the specified segment, loads this program segment into the locations specified by assembly or compilation, and starts execution of the program segment.

[1] Macro calls will be defined at a later date.

Exiting From a Program

When a program has completed operation and does not wish to load and execute another segment, it issues a macro call to the Supervisor. The Supervisor then reads the job control file for a new Execute statement and starts the new job or new program.

STRUCTURE OF THE SUPERVISOR

The Supervisor is a segmented program, part of which is permanently resident in main memory. Other segments are called in from mass storage as they are required. The Supervisor occupies two areas in main memory: the communication area and the floating area.

Communication Area

The communication area is an area fixed in lower memory that includes location Ø and locations 61 through 189. The index registers, locations 1 through 60, are available to the user. The communication area contains the necessary information for the Supervisor to perform its searching, loading, and starting functions. The content of the communication area is shown in Table 2-1.

Floating Area

The floating area is an area in upper memory whose size depends on the selection of Supervisor features made at system generation time. This portion of the Supervisor can be "floated" to the upper memory locations in two ways: (1) At execution time, when the Supervisor can relocate itself to the highest bank (unit of 4,096 characters) of memory. (2) At system generation time, when the Supervisor can be specialized and assembled to reside permanently in the highest bank of memory.

The advantage of "floating" the Supervisor is that it is possible to provide a common origin for all programs that operate within the system. The highest address used by the communication area is location 189 (decimal). Programs may be assembled above this area. On the other hand, if the remainder of the Supervisor were to be placed immediately behind the communication area, the origin of operated programs would vary because the size of the Supervisor varies.

Although the "floating" portion of the Supervisor does vary in size, the "high memory address" field in the communication area (locations 187 to 189, see Table 2-1 below), always contains the highest memory location available to operating programs. Thus, the user always knows the lowest and highest memory addresses available to operating programs.

Part of the floating area contains resident routines that are always in memory. A second part is an overlay area where less frequently required routines of the Supervisor are brought in as they are needed.

EXECUTABLE PROGRAM FILE

Programs to be operated under control of the Supervisor are stored in the executable program file on mass storage. An executable program file is a partitioned sequential file, composed of two areas, a directory area, and a program data area containing the program segments themselves.

Directory

The directory is a table (the member index) giving the name and mass storage location for every program segment in the file. Each item in the directory refers to one program segment.

The capacity of the directory, which determines the number of program segments that may be contained in the file, is specified by the user when the executable program file is created through a file support allocation run.

Program Segments

The data area of the executable program file contains the program segments or "loading units." A loading unit is the portion of code located and loaded as the result of one programmed call to the Supervisor. The area allocated to the data portion of the executable program file determines the total amount of code that can be stored in the file.

COMMUNICATING WITH SUPERVISOR

The following paragraphs discuss the two methods by which the user communicates with the Supervisor. (1) Job and program sequence control by means of job control statements, and (2) segment loading within a program and exiting by means of programmed macro calls to the Supervisor.

EXECUTE Statement

The execution of a job or of a program is requested by an Execute Statement in the job control file. The Execute statement directs the Supervisor to locate, load, and start a named program segment. The format of the Execute Statement is:

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ____ _____ PAGE ____ OF ____

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				EX	progsegname, [haltprasegname]	

Where: progsegname = The program and segment name of the program segment to be executed.

haltprogsegname = The Supervisor halts after the program and segment name are loaded. The brackets [ ] indicate that this parameter is optional.

The program segment name consists of eight (8) characters; the first six (6) are the program name and the last two (2) are the segment name. The characters must be chosen from the letters A through Z and the digits 0 through 9; no other characters may appear.

Program segment names for system programs supplied by Honeywell for the Mass Storage Operating System always begin with an asterisk (*) to prevent any duplication with user program names.

Loading a Program Segment

When the calling program is ready to load the next segment, it executes a macro call. This macro sets parameters in the communication area and branches to the Supervisor. The branch is to the Normal Call entrance (location 130 of the Communication Area).

The parameters associated with a Normal Call depend on the current contents of the communication area. These parameters are discussed below in detail starting on page 2-7, under the paragraph headings "Searching," "Loading," and "Starting."

Macro routines will eventually be supplied to communicate with the Supervisor. Until these macros are properly defined, the user may communicate with the Supervisor by means of coded program calls. Some examples of such calls are given on page 2-21 under the heading entitled "Program Calls for Segment Loading."

DETAILED DESCRIPTION OF SUPERVISOR

The following information is needed only if the programmer writes his own instructions for communicating with the Supervisor, instead of using the macros that are provided for loading the next segment or returning to job control.

The Communication area for Supervisor occupies locations 0 and 61-189 (decimal). The programmer, through the communication area, supplies the necessary parameters to the Supervisor so that the functions of searching, loading, and starting execution can be performed. Table 2-1 is a summary of the communication area with its permissible values and the reset conditions. Table 2-2 is a summary of Supervisor parameters according to function. The expression "Initial Values" in Tables 2-1 and 2-2 refers to the contents of the corresponding field at the time when the Supervisor is first brought into memory. "Reset" refers to a value entered by the Supervisor into the field just before control is returned to the program.

Punctuation marks within the communication area must not be altered. Each field is originally loaded with a word mark at its high-order or left-most end.

Later sections define the Supervisor parameters, their locations, their initial and permissible values, the conditions under which they are reset, and the resulting loader actions. A parameter summary is given in Table 2-2.

Searching

Normally, the Supervisor searches the executable program file directory for the mass storage address of the called program segment. The record at this address is then read and checked to see if it is a segment header record. If there is no directory entry for the called program segment, or if the first record is not a segment header record, the Supervisor halts.

If the Supervisor was directed to load from a specified mass storage address (i.e., search mode 07, see Table 2-3 for this and other Search Mode Designations) the directory search routine is not executed.

Table 2-1. Supervisor Communications Area

LOCATIONS		FUNCTION AND POSSIBLE VALUES	INITIAL VALUE	AT RETURN TO JOB CONTROL	AFTER LOADING
Decimal	Octal				
64	100	Job Control Device Ø - Card Reader 1 - Control Panel			
65-67	101-103	Revision Number of Unit Last Loaded	△		
68-73	104-111	Program Name	△		
74-75	112-113	Segment Name	△		
77-84	115-124	Halt Name	△		
85	125	ID Character From Console Call Card (*)	△		
86-89	126-131	Fixed Start Ø - Entrance to Job Control Function	Branch to Supervisor		
102-105	146-151	Exit to Owncode Routine	Branch to Supervisor	Branch to Supervisor	Branch to Supervisor
107-109	153-155	Relocation Augment	ØØØ	ØØØ	ØØØ
111	157	Search Mode - 20 - Search and Load by program and Segment Name 60 - Search and Load by program and Segment Name by visibility. 22 - Search by program and segment name, do not load; supply mass storage address to calling unit. 62 Search by program and segment name and by visibility, do not load, supply mass storage address to calling unit. 07 - Do not search; load by known mass storage address.	2Ø	2Ø	

Table 2-1 (contd). Supervisor Communications Area

LOCATIONS		FUNCTION AND POSSIBLE VALUES	INITIAL VALUE	AT RETURN TO JOB CONTROL	AFTER LOADING
Decimal	Octal				
112	160	Start Mode N - Normal R - Return to Calling Program. S - Special	N	N	
113-18	161-166	Visibility Mask	-ØØØØØ (visibility A)		
119-121	167-171	Special Starting Location	ØØØ		
122-125	172-175	Owncode Return Before Distribution	Branch to Supervisor Distribution Routine		
126-129	176-201	Owncode Return After Distribution	Branch to Supervisor Starting Routine		
130-138	202-212	Program Return for Segment Loading (3-character mode)	Store Bc Register and Branch to Supervisor Search Routine		
139-141	213-215	Program Return to Job Control Function (3-character mode)	Fixed Start Ø		
142-146	216-222	Current Date	△		
147	223	Trapping Mode Ø4 ON - ØØ OFF	Ø		
187-189	273-275	Highest Memory Location Available to User Programs			

Table 2-1 (contd). Supervisor Communications Area

LOCATIONS		FUNCTION AND POSSIBLE VALUES	INITIAL VALUE	AT RETURN TO JOB CONTROL	AFTER LOADING
Decimal	Octal				
61-63	75-77	Reserved for use of the Operating System. These locations are not available to the user.			
76	114				
90-93	132-135				
94-97	136-141				
98-101	142-145				
106	152				
110	156				
148-186	224-272				

Table 2-2. Summary of Supervisor Parameters

	PARAMETER	DEC. LOC. FROM	DEC. LOC. TO	OCT. LOC. FROM	OCT. LOC. TO	VALUES	INITIAL NAME	RESET AT CONSOLE CALL	RESET AFTER LOADING
SEARCHING	SEARCH MODE	111		157		20-Program and segment name. 60-Program and segment name and visibility. 22-Program and segment name; do not load. 62-Program and segment name and visibility. Do not load. 07-Load by known mass storage address.	2Ø	2Ø	
	PROGRAM NAME	68	73	1Ø4	111				
	SEGMENT NAME	74	75	112	113				
	VISIBILITY	113	118	161	166		4Ø ØØ ØØ ØØ ØØ ØØ		
	DEVICE	76		114			Ø		
LOADING	RELOCATION AUGMENT	1Ø7	1Ø9	153	155		Ø	Ø	Ø
	HALT NAME	77	84	115	124				
STARTINH	START MODE	112		16Ø		N=Normal S=Special R=Return	N	N	
	SPECIAL START LOCATION	119	121	167	171		Ø		
	TRAPPING MODE	147		223		ØØ=Off Ø4=On	ØØ		

Table 2-3. Search Mode (Location 111) Designators

CONTENTS	FUNCTION
20	Search for and load the program segment with the specified program and segment name.
22	Search the executable program file directory for the specified program and segment name and supply the calling program with the mass storage address of the called program segment. This address is conveyed through the Program Name parameter locations 68 to 73 of the communications area in the format CCTTRR (cylinder, track, record).
60	Search for and load the program segment with the specified program name, segment name, and visibility.
62	Search the directory for the specified program name, segment name, and visibility and supply the calling program with the mass storage address of the called program segment (as in search mode 22).
07	Load a program segment at the mass storage address specified in locations 68 to 73 of the communication area. This address has the same format shown above under search mode 22. The use of search mode 07 implies that search mode 22 or 62 had been previously used to supply the calling program with mass storage address of the called program segment.

The initial value of the Search Mode is 2Ø. It is reset to 2Ø when return is made to job control.

NOTE: Three search modes used by the Tape Resident Operating System-Mod 1 (Tape Loader-Monitor C) are not applicable to the Mass Storage Supervisor. They are:

00 - Search for and load the segment with the specified segment name within the current program.

40 - Search for and load the segment with the specified segment name and visibility within the current program.

01 - Search and load the n^{th} segment of specified visibility.

The Supervisor treats these codes as follows:

00 - Converted to search mode 20.

40 - Converted to search mode 60.

01 - Converted to search mode 20.

PROGRAM NAME (LOCATIONS 68-73)

This field contains the program name to be used as a search key in search modes 20, 22, 60, and 62. The program name of the called program segment is inserted in these locations by the Execute Statement in the job control file, or by a programmed call to the Supervisor. When a program segment is loaded, its program name is placed in this field by the Supervisor. If the search mode is 22 or 62 (so that no loading occurs) and after locating the program segment, the Supervisor places in this field the mass storage address of the first record of the requested program segment.

SEGMENT NAME (LOCATIONS 74-75)

This field contains the segment name to be used as a search key in search modes 20, 22, 60, and 62. The segment name of the called program segment is inserted in these locations by an Execute Statement in the job control file, or by a macro call to the Supervisor. When a program segment is loaded, its segment name is placed in this field by the Supervisor.

VISIBILITY MASK (LOCATIONS 113-118)

This field contains the visibility mask to be used in search modes 60 and 62. A visibility match is obtained if there is at least one bit position containing a "1" in both the visibility mask and the visibility key of the requested program segment. The initial value is visibility A (40 00 00 00 00 00 octal).

Loading

If the search is successful, the Supervisor proceeds to load the called program segment into memory. Loading consists of reading and then distributing and punctuating each successive record of the program segment. Each record is read into a buffer. From there, the instructions and constants are distributed to specific memory locations and punctuated as specified by control characters in the record.

Between the reading and distributing phases of loading, it is possible to execute own-coding routines. After reading a record into the buffer, the Supervisor always branches to the own-code location. An own-code routine may then do one of two things: (1) return to use the Supervisor's own distribution routine (Own-code return before distribution), or (2) it may do its own distribution and return to the read routine of the Supervisor (Own-code after distribution).

A program segment may be loaded into an area higher than that for which it was translated (assembled or compiled) by using the relocation augment. However, the Supervisor does not perform address adjustment; the program segment is loaded into the new area exactly as it was translated.

At loading time, the program and segment names of the program segment loaded are placed in locations 68 through 75. After a program segment has been loaded, the Supervisor resets the Relocation Augment to Ø and the Owncode Exit to assume no own-coding.

RELOCATION AUGUMENT (LOCATIONS 107-109)

The relocation augment field contains a value to be added to the address at which all the code of the called program segment is to be loaded. This augment is applied to instructions, constants, the addresses of areas to be cleared, and the normal start location of the program segment. Note that the code itself is not altered; the program segment is merely loaded into a new area. The initial value is Ø. It is reset to Ø after a program segment has been loaded and when a program exits.

HALT NAME (LOCATIONS 77-84)

The halt name field provides space for a program name (locations 77-82) and segment name (locations 83-84). After the program segment with this name has been loaded, the Supervisor halts. When the RUN button is pressed, the Supervisor continues as directed by the starting parameters. "Halt Name" is checked against the name of the program segment just loaded. If it is equal, the Supervisor halts. "Halt Name" is a single field. The only word mark is at location 77.

EXIT TO OWNCODING

A calling program may execute own-coding during the loading of a called program segment by setting up an appropriate branch in the communication area. The starting address of the own-code routine must be placed in locations 103-105. This is the A address of a branch instruction. No punctuation is present at these locations, and no punctuation may be placed there by a calling program. The branch is made immediately after reading each record. Before the branch, the monitor sets index register X5 to the address in main memory of the first character in the record. This Exit is initially set to assume that there is no own-coding. It is reset to this same value whenever a program completes execution and exits.

OWNCODE RETURN POINTS

The owncode routine must return to the Supervisor with a branch to one of the two own-code return points in the communication area (see below).

Owncode Return Before Distribution

If the return is made to location 122, the Supervisor performs record distribution in the normal way. Under this option the settings of X5 and X6 must not be altered by the owncode routine.

Owncode Return After Distribution

If the return is made to location 126, the Supervisor bypasses normal record distribution and reads the next record. Under this option, the owncode routine must recognize the last record of the called program segment and must not return to location 126 after obtaining it. Instead, X5 must be set to the address of a location containing the character 61, followed by the three-character starting address of the program segment just loaded. Then return is made to location 122.

Starting

After loading a called program segment, the supervisor may return to the calling program segment or branch to a normal or a special location in the called program segment. The branch is always made in 3-character address mode. Before executing the branch, the Supervisor uses the rightmost four bits of the Trapping Mode Indicator (location 147 decimal) to set up and execute a CAM instruction that sets the trapping mode indicator of the central processor.

STARTING MODE (LOCATION 112)

The Start Mode parameter specifies which of the three alternative locations the Supervisor will transfer control to after loading. It must contain one of the three following values:

N - Branch to the location specified as the normal starting location in the called program segment. The relocation augment is added before the branch is made.

S - Branch to the address given by the parameter "special start location" (119-121). The relocation augment is not added to this address.

R - Branch to the location immediately following the one from which the call to the loader was made. The relocation augment is not added to the address.

The initial value is N. It is reset to N when a program exits.

SPECIAL START LOCATION (LOCATIONS 119-121)

This field, which is used only with start mode "S," specifies the address to which control is to be transfered by the Supervisor after loading. This may be any memory location up to 32,768. The initial value is 0. It is never reset by the Supervisor.

TRAPPING MODE (LOCATION 147)

This field contains a character whose low order four bits are substituted into the variant character of a CAM instruction. This instruction is executed immediately before the Supervisor starts the called program, to establish the trapping mode that will be in effect when the called program segment is started. The field should contain one of the following two values:

00 -- No Item Mark Trapping

04 -- Item Mark Trapping

The initial value is 00.

If the trapping mode is specified, the operation code of any instruction which contains an item or record mark is both extracted and executed as if

it were a Change Sequencing Mode (CSM) instruction, regardless of the operation code present. The CAM and CSM instructions and the trapping mode are described in detail in the Honeywell Series 200 Programmers Reference Manual (Models 200/1200/2200/4200), Order No. 139. This facility provides for automatic changes in program sequence without executing programmed instructions to initiate such changes.

The programmer is urged not to use the trapping mode in his programs because the program test facility uses the item mark trapping feature to initiate dumps. A program using item mark trapping will not be able to use certain dump facilities.

Returning to the Supervisor

PROGRAM RETURN FOR SEGMENT LOADING (LOCATION 130)

The Program Return for Segment Loading is used to load the next segment of a program or job automatically without returning to the job control routine. The program segment making the call changes the appropriate parameter values in the communication area and then branches to location 130. When this return is used, the supervisor does not reset any parameter values; any changes must be made by the program before it branches to 130.[1]

PROGRAM EXIT LOCATION (LOCATION 139)

The Program Exit Location is the location to which a program branches (indirectly) after having completed its processing. The program, without changing values in the communication area, may branch indirectly to location 139. The Supervisor then resets the Start Mode parameter to N, the Search Mode to 20, the Relocation Augment to 0, and Exit to Own-coding to assume no own-coding and then reads the next statement in the Job Control File.

[1] Note that the Relocation Augment was reset when the calling segment was loaded (see Table 2-1).

Locations 139 through 141 normally contain the address of the control routine in the Supervisor. But when a series of programs are to be executed as a system, with a user-written control program, the user control program may change the contents to the address of some routine within itself. In this case, all of the program segments in the series should terminate with an indirect branch to location 139. This has the effect of returning control to the system's control program, allowing it to determine which program segment it wants to be loaded next and to make the appropriate call. After a systems run, the control program should restore the contents of locations 139 through 141 to the initial value.

OTHER FEATURES

Fixed Starts

The communication area contains four branch instructions that are used for console starts. The first instruction, Fixed Start 0, (Locations 86-89), is a branch to the job control routine of the supervisor. It is equivalent to a program exit. Job Control resets the Start Mode parameter to N, Search Mode to 20, Relocation Augment to 0, and Exit to Owncoding to assume no owncoding; and then reads the next statement in the job control file.

Revision Number (Locations 65-67)

Before starting to load a program segment, the Supervisor moves the programs revision number into this field. This is provided for use or reference by other programs or by the operator.

Current Date (Locations 142-146)

The operator may enter the current date into locations 142 through 1465, for reference by other programs. Locations 142 and 143 specify the year (00 to 99), and locations 144 through 146 specify the day of the year (001 to 366). The initial value of this field is 00000.

Upper Limit of Available Memory (Locations 187-189)

This field contains the address of the highest memory location that may be used by any program. The floating portion of the Supervisor resides above this address. Any program computing the amount of memory available must take account of this value.

Relocation Bank Indicator

The Supervisor preserves the relocation bank indicator in location 76. (Table 2-4 shows the acceptable relocation bank indicators.) The indicator was used when this version of the Supervisor was first brought into memory and shows the bank in which the Supervisor resides.

Table 2-4. Relocation Banks

INDICATOR	BANK	LAST ADDRESS USED BY SUPERVISOR (OCTAL)
Ø2	12K	Ø27777
Ø3	16K	Ø37777
Ø4	20K	Ø47777
Ø5	24K	Ø57777
Ø6	28K	Ø67777
Ø7	32K	Ø77777

Program Calls for Segment Loading

Once the first segment of a program has been loaded and started, subsequent segments may be loaded and started, using program calls performed by instructions in the program segment currently running. The program segment making the call first moves the desired parameter values into the communication area and then transfers control to the Supervisor. The return branch to the supervisor is made to location 130 (Return for Segment Loading). This loads the requested program segment without returning to the job control routine: there is no reset of parameter values or reference to statements in the job control file.

EXAMPLES OF SEGMENT LOADING

Example 1: - Loading a specified Program Segment

Call the program segment named PROCES AA and start PROCES AA at its normal starting location (see coding example below). Note that the coding does not include entries for Loading Device, Search Mode, or Start Mode, since the desired values for these parameters are the initial values established by the Supervisor.

EASYCODER
CODING FORM

PROBLEM EXAMPLE I PROGRAMMER DATE PAGE OF

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 14	15 20	21 62 63 80
				MCW	SGNAME,75
				MCW	
				B	13Ø
			PRNAME	DCW	@PROCES@
			SGNAME	DCW	@AA@

Example 2: - Loading a Specified Program Segment by Visibility

Call the program segment named INITPR NN that also is identified by either visibility C or visibility D and start the specified segment at its normal starting location (see coding example below).

EASYCODER
CODING FORM

PROBLEM EXAMPLE II PROGRAMMER DATE PAGE OF

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 14	15 20	21 62 63 80
				MCW	SG,75
				MCW	
				MCW	SRCH,1,1,1
				MCW	VIS18,1,18
				B	13Ø
			PR	DCW	@INITPR@
			SG	DCW	@NN@
			SRCH	DCW	#1CGØ
			VIS18	DCW	#GC14ØØØØØØØØØØ

Example 3: - Relocation, Special Start

Call the program unit named AAAMEM S1, relocate the program unit 2500 octal locations higher, and start AAAMEM S1 at octal location 2510 (see coding example below). (DSA's and the operand addresses of instructions are not altered by the Relocation Augment.)

EASYCODER
CODING FORM

PROBLEM EXAMPLE III PROGRAMMER DATE PAGE OF

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62 63 ... 80
				MCW	SN,75,
				MCW	
				MCW	RELOC,109
				MCW	STMODE,112
				MCW	SPST,121
				B	130
			P.N.	DCW	@AAAMEM@
			SN	DCW	@S1@
			STMODE	DCW	@S@
			RELOC	DCW	#3CØØ,25ØØ
			SPST	DCW	#3CØØ,2510

PROGRAMMER'S PREPARATION INFORMATION

1. Since the Supervisor resides in high memory and has a variable starting location, some care must be taken to ensure that no program overlaps the Supervisor. In particular, programs which operate with a variable amount of memory must take into account the address stored in locations 187-189 when computing the memory available.

2. A Supervisor assembled in address mode 3 can only load programs into the first 32K of memory and always starts programs in address mode 3.

3. The Supervisor uses, and does not restore, index registers X5 and X6. These registers have word marks at their high order locations after loading, but the user is cautioned that the address mode used by the

Supervisor does not necessarily correspond to that used by the loaded program segment, and the word marks may not be where the loaded program segment desires.

X6 contains the address at which the last character of the called unit was loaded. X5 contains the address of the control character in the buffer that terminated the loading operation. The three characters at the locations immediately following the address specified by X5 will contain the normal starting address of the unit just loaded.

4. Owncode routines must not destroy the contents of index registers X5 and X6.

5. The Supervisor does not use nor disturb any locations below 61_{10} with the exception of index registers X5 and X6 and location 0.

EQUIPMENT REQUIREMENTS

Series 200 Central Processor with Control Panel

Advanced Programming Instructions (Feature 010, 011, or 1011)

The number of storage locations used is dependent upon the system generation process. The smallest version (3 character, single buffer, without typewriter options) will require 1,350 locations. In addition, 130 locations are used for the communication area (Ø and 61-189).

1 Mass Storage Control Unit (Type 255, 257, or 257A)

1 Mass Storage Device (Type 256, 258, 259, or 259A)

Index registers X5 and X6

Additional Usable Equipment

1 Card Reader

SECTION III

DATA MANAGEMENT

Data Management is the element of the Mass Storage Operating System that provides the necessary input/output routines associated with data files, and routines for their creation and maintenance. More than this, however, the Data Management element provides a specific set of rules, or conventions, governing data management concepts and file organization. All elements of the operating system follow these conventions. This section describes in detail these features of the Data Management element.

DATA MANAGEMENT CONVENTIONS

The Data Management conventions include the general concepts relating to data and volume conventions, which lead directly to the more specific rules of file organization. The data conventions define the basic units of data and the relationships between them. These relationships lead directly into the conventions established for the allocation of space to store a data file on a volume. The volume conventions are concerned with the preparation of volumes that includes labeling and establishing directories.

Data Conventions

This paragraph defines the units of data and explains the relationships between them and between certain units of data and the physical capabilities of the storage device.

UNITS OF DATA

Item

An item is the basic unit of logically related information for a data processing program. In this sense, an item can possibly be a single policy in an insurance policy file, or perhaps, an individual's account in a master payroll file.

Record

A record is that data physically located between two gaps on the track of a mass storage device.

Block

A block is the sum of physical records transferred to or from main memory by a single data transfer instruction. For convenience, the term block can be considered as synonymous with a buffer.

File

A file is a collection of logically related items. This is the largest single unit of data that can be stored and retrieved by the operating system.

RELATIONSHIPS BETWEEN UNITS OF DATA

Record-To-Track

All records on a track must be the same size.

Record-To-Block

A block may be one or more records long.

Item-To-Block

There may be any number of items in a block. When the number of items per block leaves unused character spaces in the block, these are filled with 77_8.

Block-To-Track

A block may be entirely within one track or it may start on one track and end on the next track.

Allocation Conventions

The unit of allocation is the basic element in the designation of the area of mass storage assigned to a data file. The description of a unit of allocation is of the form:

$C_1T_1C_2T_2$

Where: C_1 is the first cylinder of the unit of allocation.

T_1 is the first track used on all cylinders from C_1 to C_2 inclusive.

C_2 is the last cylinder of the unit of allocation.

T_2 is the last track used on all cylinders from C_1 to C_2 inclusive.

If a unit of allocation for a file were Ø6-Ø1-11-Ø5, it could be shown graphically as in figure 3-1.

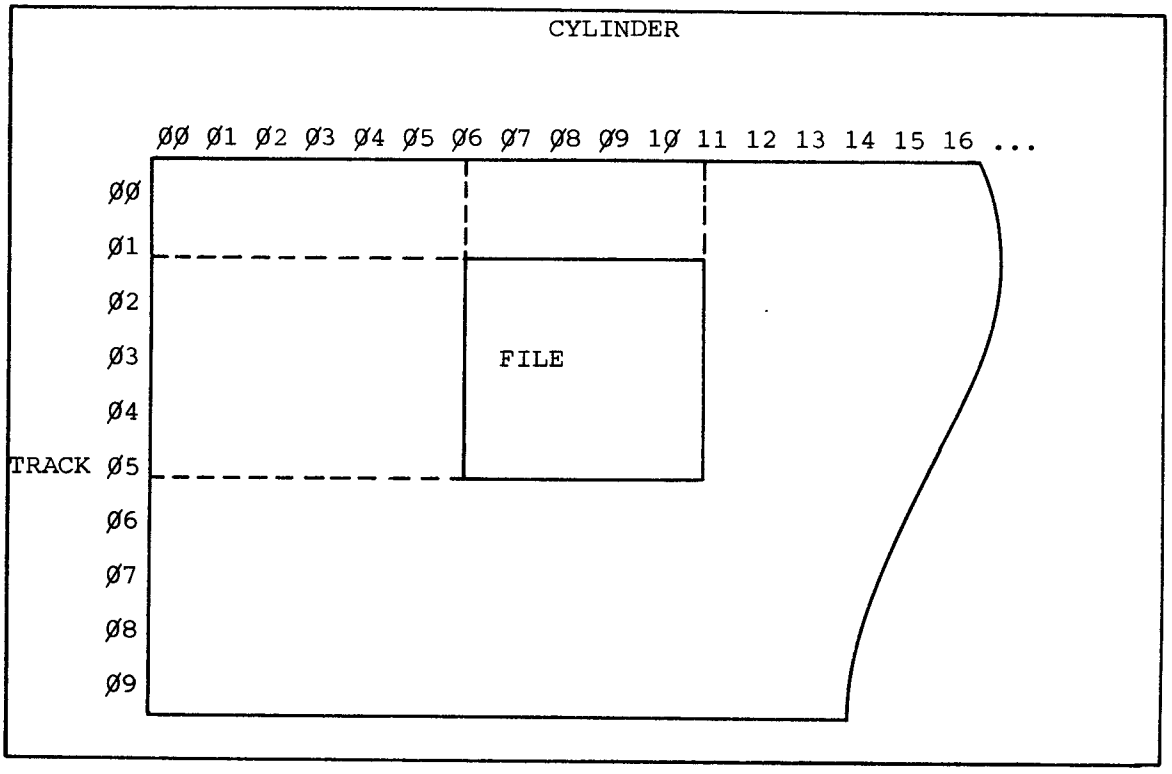


Figure 3-1. Illustration of Unit of Allocation

A single data file may have more than one unit of allocation. When this is the case, the number of tracks per cylinder in each unit of allocation for that file must be the same. An acceptably allocated file is shown in figure 3-2 and an unacceptable allocation of a file is shown in figure 3-3.

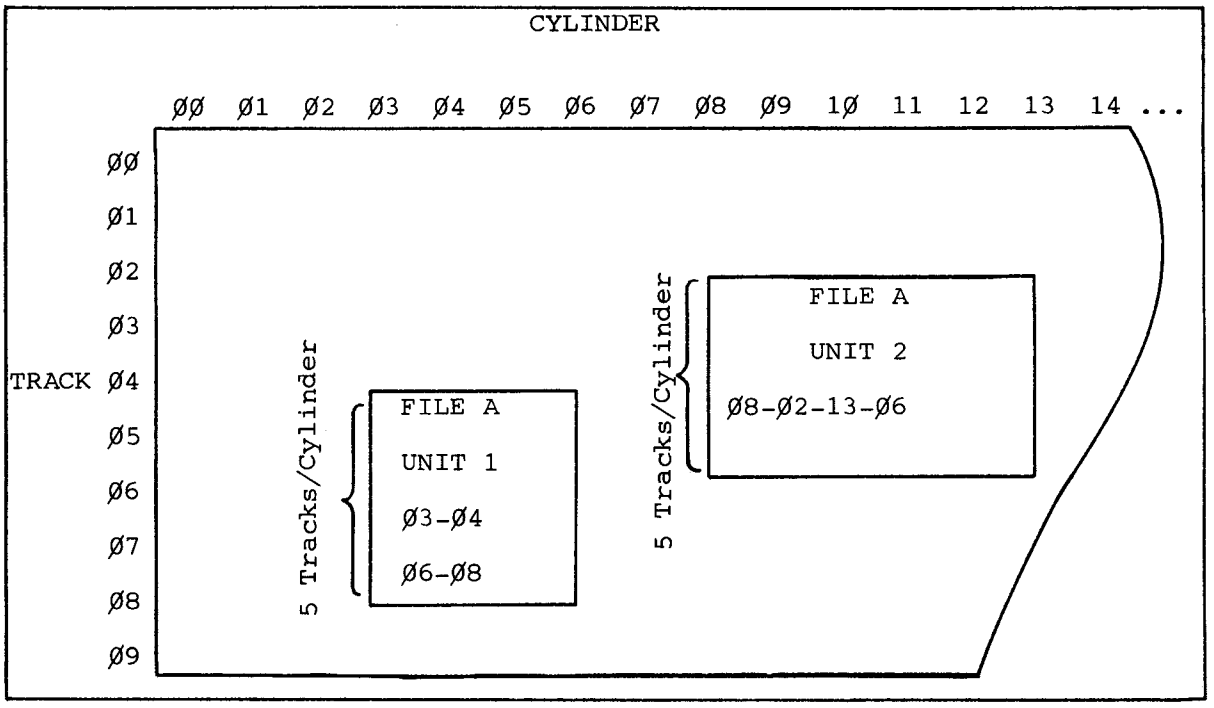


Figure 3-2. Acceptable Allocation Of A File

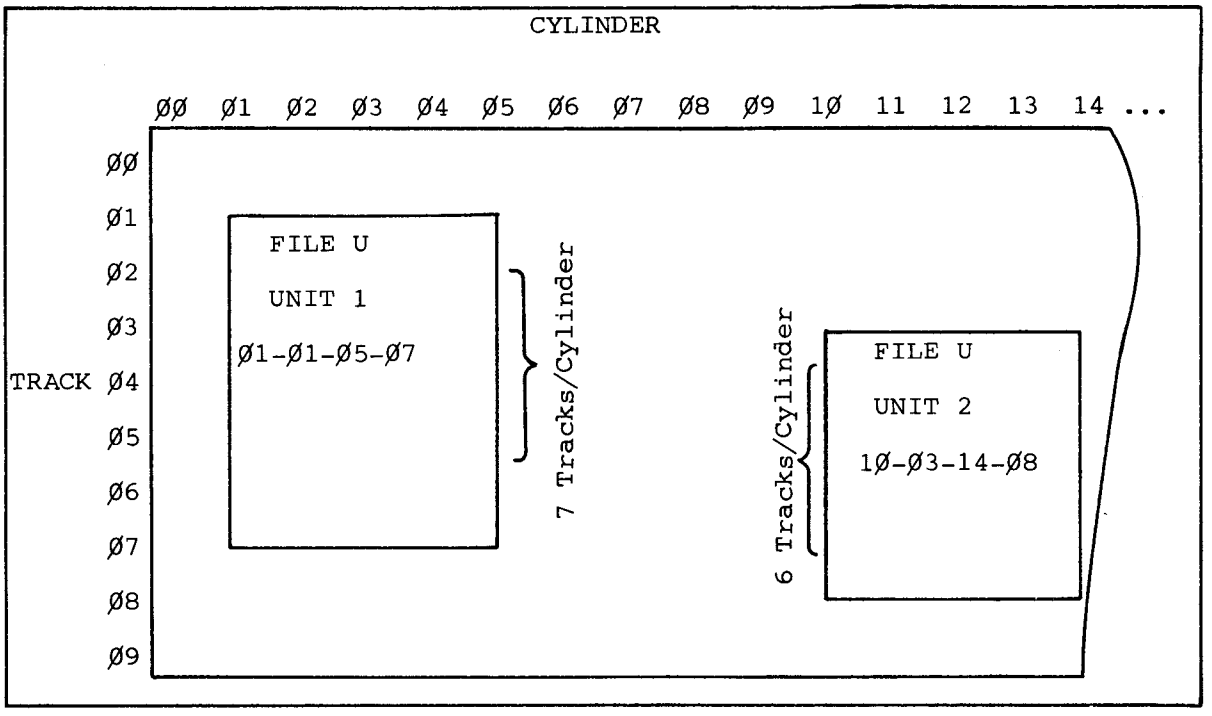


Figure 3-3. Unacceptable Allocation Of A File

The status of the units of allocation for a given mass storage device is maintained by the operating system in such a way that it will not allocate space for a new file whenever the new file's units of allocation are in conflict with those of any other file previously stored on the device.

In the preceding illustrations, a cylinder was shown as if it had been rolled out flat. Figure 3-4 shows an overall view of a cylinder in an exploded view of a disk.

The method of determining the required unit of allocation for any file is described in the appendices. Space allocation for Sequential File Organization is described in Appendix G and for Direct Access file organization in Appendix H. In general, it is recommended for the disk that 1Ø tracks per cylinder be allocated to each file.

Volume Conventions

The volume conventions are concerned with formatting and volume preparation, bootstrap records, volume labels, and volume directories. Each of these are discussed individually in the following paragraphs.

FORMATTING AND VOLUME PREPARATION

All mass storage volumes used in the Series 200 must be formatted before data can be stored on them. Formatting establishes the size of each record on a track. All records on a given track are equal in size. Whenever the size of the records on a track is to be changed, the track must be reformatted. The facility for automatically reformatting tracks is a feature of the operating system.

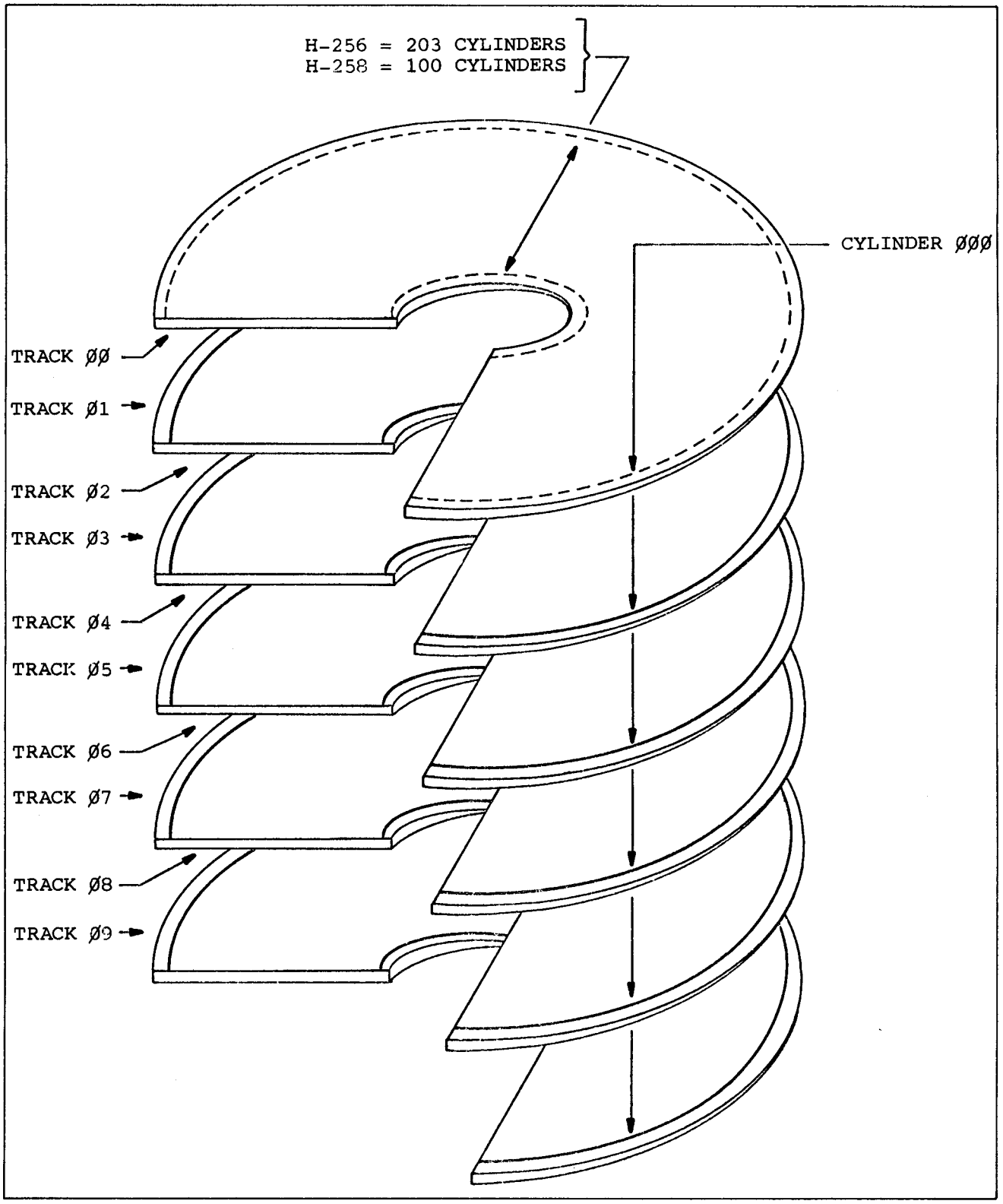


Figure 3-4. Overall Concept of a Cylinder

Initially, however, each volume is formatted with 250 character records on all tracks. This size record is used for all Honeywell system files (such as machine language, source language, and work files). User's data files in which the records are other than 250 characters requires that the volume be reformatted. This is accomplished automatically by the File Support routines.

BOOTSTRAP RECORDS

The bootstrap records are recorded on the first track (Track Ø) of each volume. This track is not available to the user for storage of data.

VOLUME LABEL

The volume label is the unique identification of the volume. This record is 250 characters long and is recorded as the first record (Record Ø) on the second track (Track 1) of each volume. The volume label is described in detail in table 3-1.

Table 3-1. Volume Label Description

FIELD	POSITION	NAME AND LENGTH	DESCRIPTION
1	1-5	Identification 5 Characters	1VOLΔ
2	6-11	Volume Serial Number 6 Characters	This field contains the unique identification of the volume.
3	12	Device Type 1 Character	11_8 Disk with 100 cylinders 13_8 Disk with 203 cylinders
4	13-200	Reserved 188 Characters	Reserved for use of the operating system.

VOLUME DIRECTORY

The volume directory is a list of all files that are stored in whole or in part on the volume. Table 3-2 contains a complete description of the volume directory. Three sequential files make up the list:

1. File Name Index (*VOLNAMES*)
2. File Description Index (*VOLDESCR*)
3. File Allocation Index (*VOLALLOC*)

The first file (*VOLNAMES*) is an index of file names and references the other two files for additional information. This index contains the names of all files allocated on this magazine and the addresses of the associated entries in the File Description Index and the File Allocation Index. The item size of the File Name Index is 30 characters. Its format is shown in Table 3-2.

The second file (*VOLDESCR*) is a complete description of each file, including general information, labeling information, and information pertinent to the particular organization and structure of the file. The item size of the File Description Index is 100 characters. Its format is shown in Table 3-2.

The third file (*VOLALLOC*) is a list of the mass storage areas allocated to each file stored on the volume, i.e., the units of allocation. Each unit is one item. Since a data file may consist of many units, the allocation item (unit) referenced by the File Name Index may itself reference another allocation item, etc. The size of an item is 20 characters. The format of the File Allocation Index is shown in Table 3-2.

Table 3-2. Volume Directory Description

FIELD	POSITION	NAME AND LENGTH	DESCRIPTION
FILE NAME INDEX (*VOLNAME*)			
1	1-10	FILE NAME 10 characters	The unique name assigned to the file.
2	11-14	RESERVED 4 characters	Reserved for future use.
3	15-22	FILE DESCRIPTION ADDRESS 8 characters	Address of the entry in the File Description Index describing Index the file named in (1). In the format CCTTRRII.
4	23-30	ALLOCATION ADDRESS 8 characters	Address of the first entry in the Allocation Index for the file named in (1). In the format CCTTRRII.
FILE DESCRIPTION INDEX (*VOLDESCR*)			
1	1	FILE TYPE 1 character	Identifies the file organization 01 = Sequential 11 = Partitioned Sequential 02 = Direct Access 03 = Indexed Sequential
2	2-3	ITEM SIZE 2 characters	Number of characters per item, in binary.
3	4-5	RECORD SIZE 2 characters	Number of characters per record, in binary.
4	6-7	BLOCKING FACTOR 2 characters	Number of items per block, in binary.
5	8-9	RECORDS PER BLOCK 2 characters	Number of physical records per block, in binary.
6	10-11	RECORD PER TRACK 2 characters	Number of physical records per track, in binary (does not include TLR).
7	12	CYLINDER OVERFLOW 1 character	Number of tracks per cylinder assigned for overflow.
8	13	GENERAL OVERFLOW 1 character	General overflow indicator 00 = No general overflow 77 = The last cylinder of each unit of allocation is used for general overflow.
9	14-21	RESERVED 8 characters	Reserved for future use

Table 3-2 (cont). Volume Directory Description

FIELD	POSITION	NAME AND LENGTH	DESCRIPTION
Labeling Information			
10	22-26	CREATION DATE 5 characters	Date file was last created in the form YYDDD
11	27-29	CREATION NO. 3 characters	The number of times this file has been reorganized in decimal.
12	30-34	MODIFICATION DATE 5 characters	Date this file was last modified (i.e. opened for O/P or I/O). In the form YYDDD.
13	35-37	MODIFICATION NO. 3 characters	Number of times this creation of the file has been modified in decimal.
14	38-42	EXPIRATION DATE 5 characters	The date on which this file may be deleted, in the form YYDDD.
15	43-50	RESERVED (Unavailable to User) 8 characters	
16	51-53	ITEM COUNT 3 characters	Total number of active items in the file, in binary.
17	54-63	RESERVED 10 characters	Reserved for future use.
File Definition Information - Sequential Organization			
18	64-65	INDEX LENGTH 2 characters	Number of blocks set aside for the number index, in binary.
19	66-68	BLOCKS IN FILE 3 characters	Total number of data blocks available to this file, in binary.
20	69-100	RESERVED 32 characters	Reserved for future use.
File Definition Information - Direct Access Organization			
18	64-65	KEY LENGTH 2 characters	Number of characters in the key, in binary.
19	66-68	KEY DISPLACEMENT 3 characters	Number of positions from the LHE of the item of LHE character of key, in binary. If the key is the first field in the item, field 19 = ØØ.

Table 3-2 (cont). Volume Directory Description

FIELD	POSITION	NAME AND LENGTH	DESCRIPTION
20	69-70	BLOCKS/BUCKET 2 characters	Binary number of blocks in a bucket.
21	71-100	RESERVED 30 characters	Reserved for future use.
FILE ALLOCATION INDEX (*VOLALLOC*)			
1	1	STATUS 1 character	Status indication for this item 77_8 = unused 40_8 = last unit for this file 60_8 = more units follow on this volume 20_8 = more units follow on another volume
2	2-4	RESERVED 3 characters	Reserved for future use.
3	5-12	ALLOCATION UNIT 8 characters	Boundaries of this unit of allocation, in the binary form CCTTCCTT.
4	12-20	NEXT UNIT ADDRESS 8 characters	If field 1 = 6Ø, field 4 = ØØØØØØØØ where the next unit of allocation is the next physical item in this index. Otherwise, field 4 is the address of the item in this file containing the next unit of allocation (in the form CCTTRRII).

File Organization

A file is a collection of one or more logically related items. Files may be organized in different ways to satisfy different requirements. An application with a high degree of serial processing requires a file organization different from an application that requires direct access to any item in the file. Three types of file organizations are offered by the operating system: Sequential Organization, Indexed Sequential Organization and Direct Access Organization. The Sequential and Direct Access file organizations are described in succeeding paragraphs. The Indexed Sequential Organization will be described in a later publication.

FACTORS GOVERNING THE ORGANIZATION OF FILES

Mass Storage processing has great advantages to offer for the storage of large volumes of data and for fast accessing of any item of data. But, in order to use these advantages, the files must be effectively organized. File organization is the systematic arrangement of data records on a storage medium in a way that will make the effective use of storage capacity and, at the same time, permit easy and efficient retrieval of data for processing.

There are three major systems objectives which will determine the best type of file organization for a particular application:

1. Maximum use of storage space.
2. Minimize the time required for accessing items.
3. Minimize processing time required for creating and maintaining files.

System Considerations

Efficient file organization is based on thorough system planning for the particular application and on complete and accurate definition of the data to be stored. In particular, this depends on:

1. The volume of data involved.
2. The frequency and size of the peak volumes.
3. The frequency of access of data in the file and how this varies, both between items and between particular fields within items.
4. The type of processing and addressing techniques used to access the various files.
5. Whether mass storage is the sole storage medium or whether some data will be stored on magnetic tape or on punched cards.
6. Whether the existing item keys are useable, or, if not, what the cost of conversion or modification would be.
7. How much expansion or modification of the files is forseen.
8. The inquiry requirements.
9. The total reporting requirements and the desired sequence for reports.
10. Whether the associated records will be referrenced individually, or whether they will be consolidated.
11. The extent and complexity of file maintenance requirements.
12. Whether a particular file will be processed in more than one processing mode.
13. Whether total systems approach is envisioned, or whether each application will be processed individually.

The overriding consideration of efficient file organization is to keep techniques as simple and as standard as possible within the limitations imposed by the particular application.

Storage Layout Considerations

The specific considerations that must be looked into before organizing a file include:

1. The precise data layout, which in the first instance should merely include all data required. From this rough draft should be prepared the final layout, which will have fields arranged in order of access frequency and degree of essential reference, with associative fields grouped together where possible. This final re-organization is designed primarily to minimize accessing and processing time.

2. The record length to be employed. This is a factor of the data length but also of the storage medium since the ideal record length for processing efficiency will be one that fits in conveniently with track length. Cases that require special consideration are those in which records are either under or over one track in length. Ideally, the length chosen should be sufficient to cover all records, but where there is considerable variation, then an optimum size must be chosen to reduce unused storage to a reasonable minimum. For records which exceed this limit, either by variation in field length or in the multiples of fixed length fields, continuation records must be linked or chained in the form of trailer records.

3. The blocking factor to be employed. Ideally, each physical record should occupy one track, which may contain several items. This makes maximum use of storage by eliminating inter-record gaps,

which reduce file storage efficiency. But large single track records require larger buffers and allow less time for pre-update processing within normal latency time. A multi-record track layout makes less efficient use of storage and requires a separate access to each record. But, less memory is required by the smaller buffers and more latency time is available for update processing.

OVERALL EFFICIENCY

Processing efficiency will be a factor, first of the agreed system objectives, and secondly, of the system layout considerations previously outlined. Since efficinecy depends on so many inter-related factors, it will be the result of a compromise. For example, it may be necessary to sacrifice some storage utilization to improve the speed of access and maintenance, or vice versa. Overall efficiency is the prime objective. To achieve this, every factor must be carefully examined both individually and in relation to the other factors.

SEQUENTIAL FILE ORGANIZATION

The Sequential file is organized for items to be accessed in a logical sequence. This type of file organization is intended primarily for an application in which most of the items are processed each time the file is used. The data is one physically continuous stream of items. Processing is in logical sequence which conforms to physical sequence. The end of data is signified by an item starting with □EOD¢ (7625462477_8). All tracks allocated to the file are used for data. Items are fixed length and all characters in an item are data.

There is an option available to break the file into a number of smaller files. This option is called partitioning and is described in detail in Appendix E of this manual.

DIRECT ACCESS FILE ORGANIZATION

A Direct Access file provides fast access to items that are not to be retrieved sequentially. Its organization is flexible and a user may tailor it to his specific needs. The organization of a Direct Access file is principally in terms of user defined areas called buckets. A bucket is defined in terms of blocks. It may be composed of any number of blocks ranging from one to a maximum number of blocks per cylinder. A bucket cannot cross cylinders. A small bucket may provide faster access but it also increases the possibility of overflow. Conversly, a large bucket may increase the access time to a given item but it decreases the possibility of overflow. There is no ordering of items within a bucket and access to a bucket is made through a user supplied address.

Data Area

The data area for a cylinder used in a Direct Access file is the number of tracks on the cylinder within the unit of allocation minus those tracks specified for the cylinder overflow. Within the data area, a file is divided into buckets. The size of the buckets is used defined in terms of blocks. A bucket may be one or more blocks in size but cannot exceed the total number of blocks in the cylinder data area.

The bucket address is the address of the first record in the first block of the bucket. When a bucket contains more than one item, there need be no logical relationship between the items except through some means (randomization or otherwise) the address of that bucket was specified as belonging to that item.

Because a block may cross tracks, buckets can cross tracks. Buckets cannot cross cylinders, however, because a given bucket is processed as though it flows from the data area into the cylinder overflow area and then into the general overflow area.

Cylinder Overflow Area

The cylinder overflow area is a user specified number of tracks set aside at the end of the unit of allocation of each cylinder in the file. This area is used to store the overflow of data from the bucket or buckets that comprise the data area.

General Overflow Area

The general overflow area is an optional area set aside to store the overflow from the cylinder overflow area. This optional feature is included to avoid costly termination in the middle of a run. When the operating system is forced to use the general overflow area, suitable notice is provided. Frequent use of the general overflow area not on the same cylinder as the bucket would be very costly in terms of time. For this reason, the general overflow area is not recommended for normal use. The general overflow area, when used, will always be the last cylinder of each unit of allocation.

Overflow Options

It is not necessary to use all overflow areas. If any are not used the path taken to store overflow items would vary as shown in figure 3-5.

1. If no overflow areas are used, any overflow causes an exit to the user.
2. If only cylinder overflow is specified, overflow goes first to the cylinder overflow area and then to the user.

3. If only general overflow is specified, overflow goes first to the general overflow area and then to the user.

4. If both cylinder and general overflow are specified, overflow goes first to the cylinder overflow area, then to the general overflow area, then to the user.

LOG I/O PATH	LEVEL 1 TO BUCKET	LEVEL 2 CYLINDER OVERFLOW	LEVEL 3 GENERAL OVERFLOW	LEVEL 4 USER DATA EXIT PARAMETER 42 OF LOGICAL I/O MCA
ITEM	YES (1)	NO	NO	YES (4)
ITEM	YES (1)	YES (2)	NO	YES (4)
ITEM	YES (1)	NO	YES (3)	YES (4)
ITEM	YES (1)	YES (2)	YES (3)	YES (4)

Figure 3-5. Data Path For Overflow Options In Direct Access Files

RELATIONSHIPS BETWEEN DIRECT ACCESS ORGANIZATION AND KEYS

The word "key" can be used in association with other words such as "acual key", "relative key" and "item key". Because of this, these terms are defined as follows.

Actual Key

The actual, physical address of the bucket in terms of cylinder, track and record and expressed as CCTTRR.

Relative Key

The number of a bucket relative to the beginning of the file. The first bucket in a file is numbered ØØØ.

Item Key

The identification field of an item. This field must be continguous characters but its length and location within the item is specified by the user.

RELATIONSHIP BETWEEN DIRECT ACCESS FILE PROCESSING AND KEYS

Directly processing an item normally involves the operating system's use of a bucket address and an item key field provided by the user. The bucket address may be direct, using an actual key or it may be relative, using a key relative to the bucket's numeric position within a file. Using either an actual or a relative key, the beginning of the bucket is located and then the bucket is searched for the item containing the specified item key. When the search of the bucket does not produce the desired item, the search continues through the cylinder overflow area and the general overflow area if available.

NULL ITEMS

Because it is frequently necessary to insert or delete items in a Direct Access file, it is important to be able to distinguish between an item and an "empty hole". To accomplish this, a status character is appended to every item in the file. This character indicates whether or not an item is null (inactive), or whether it is active or deleted. When a Direct Access file is allocated and before data is recorded in the file, all items have their status character set to indicate null items.

NOTE: When allocating a Direct Access file, it should be remembered that the status character must be included in the item size parameter.

Table 3-3 shows a single cylinder of a Direct Access file organization. The buckets are numbered as though this cylinder were the first cylinder used in a particular file, and the numbers representing items in the file are intended to show that there is no necessary relationship between actual key and item key.

Table 3-3. Single Cylinder In Direct Access File Organization

CYLINDER

BUCKET ØØ	1ØØ	2Ø3	NULL	NULL	
Blocks ØØ--Ø4	NULL	NULL	NULL	NULL	
	NULL	NULL	1Ø6	5Ø3	BUCKET Ø1
	126	315	124	516	Blocks Ø5--Ø9
	315	411	612	214	
BUCKET Ø2	505	NULL	611	NULL	
	NULL	NULL	NULL	NULL	
Blocks 1Ø--14	NULL	NULL	ØØ6	111	
	312	4Ø6	411	NULL	BUCKET Ø3
	NULL	NULL	NULL	NULL	Blocks 15-19
	333	Ø12	Ø57	664	
BUCKET Ø4	NULL	NULL	NULL	NULL	
Blocks 2Ø--24	NULL	NULL	3Ø2	4Ø5	BUCKET Ø5
	117	224	NULL	NULL	Blocks 25--29
	NULL	NULL	NULL	NULL	
BUCKET Ø6	ØØ2	414	415	Ø67	
	2Ø5	123	518	6Ø9	
Blocks 3Ø--34	1Ø9	299	NULL	NULL	CYLINDER
	NULL	NULL	NULL	NULL	OVERFLOW
	NULL	NULL	NULL	NULL	AREA

INPUT/OUTPUT CONTROL

The input/output control (I/O control) facilities provided by the operating system reduce to a minimum the amount of coding the programmer must write to process his files. Using the I/O macros, a programmer can control the entire I/O process, communicate with the system to alter the process, and specify the processing modes and actions required by the particular application.

Every macro function available to the user is fully identified and the method for writing each macro call is specified in this section of the manual. This section also identifies and defines the processing modes available to the user and relates them to the applicable file organizations. Because some of the functions performed by certain action macros depend on the processing mode chosen for a particular type of file organization, a table is included that identifies each action macro available for each type of file processing mode and type of file organization.

File Processing Modes

There are three distinct processing modes available: input/output processing, input only processing and output only processing.

INPUT/OUTPUT PROCESSING MODE

The input/output processing mode can be used with the Sequential and Direct Access file organizations. In the input/output processing mode, the user can both read data items from the file (input) and write data items to the file (output).

INPUT ONLY PROCESSING MODE

The input only processing mode also can be used with all the available types of file organizations. In this processing mode, the file can only supply data items to main memory (input) and cannot receive data items from main memory. Because of this, certain action macros normally available for use with a particular type of file organization become inapplicable.

OUTPUT ONLY PROCESSING MODE

The output only processing mode can be used only with the Sequential file organization. In this type of file processing, the file can only accept data items from main memory (output) and cannot supply data items to main memory. As in the input only processing mode, the output only processing mode renders certain action macros normally available for use with a Sequential file inapplicable.

GENERAL USAGE OF THE PROCESSING MODES

The processing modes refer to the direction of data flow with respect to main memory. In the input/output mode, data is transfered into and out of main memory to and from mass storage. In the input only mode, data is transfered into main memory from mass storage. In the output only mode, data is transfered from main memory to mass storage. Specialization of the processing mode by user specified parameters if assembly time allows excess coding to be deleted from the program being assembled. For example, if a master file was being created on mass storage, the output only processing mode could be used. This would enable the transfer of data from main memory to mass storage and the coding required for input/output or output only processing would not be assembled into the program creating

the file. As another example, the contents of a file can be protected from accidental destruction by opening the file in the input only mode. This would prohibit an accidental transfer of data from main memory to mass storage. Also, in the file processing program, the coding required to perform the input/output and output only functions would not be required. As a last example, if a file update were being performed, the input/output mode could be used and the program would not require the coding for output only processing.

Input/Output Macros

The input/output macros are summarized in Table 3-4. This table contains a comprehensive list of all macro calls and the general function performed by each macro. In addition, each macro is identified as control, communication or action and the applicable file organization and processing modes are shown.

Table 3-4. Input/Output Mac os

MACRO TYPE	MACRO CALL	TYPE OF FILE PROCESSED	FILE PROCESSING MODE	GENERAL FUNCTION PERFORMED
CONTROL	MIOC	SEQUENTIAL	INPUT/OUTPUT INPUT ONLY OUTPUT ONLY	Provides general control of the entire input/output process.
		DIRECT ACCESS	INPUT/OUTPUT INPUT ONLY	
COMMUNICATION	MCA	SEQUENTIAL	INPUT/OUTPUT INPUT ONLY OUTPUT ONLY	Sets up a communication area in which all values necessary to describe a file and the processing options are stored. Pertinent portions of this information are available to the user and can be altered by him.
		DIRECT ACCESS	INPUT/OUTPUT INPUT ONLY	
	MLCA	SEQUENTIAL		Used to alter any applicable field of information in the communication area.
		DIRECT ACCESS		
	MUCA	SEQUENTIAL		Used to interrogate any applicable field of information in the communication area.
		DIRECT ACCESS		
ACTION	MSOPEN	SEQUENTIAL	INPUT/OUTPUT INPUT ONLY OUTPUT ONLY	Opens a file for processing.
			INPUT/OUTPUT INPUT ONLY	

Table 3-4 (cont). Input/Output Macros

MACRO TYPE	MACRO CALL	TYPE OF FILE PROCESSED	FILE PROCESSING MODE	GENERAL FUNCTION PERFORMED
ACTION (continued)	MSCLOS	SEQUENTIAL	INPUT/OUTPUT INPUT ONLY OUTPUT ONLY	Closes a file after processing.
		DIRECT ACCESS	INPUT/OUTPUT INPUT ONLY	
	MSGET	SEQUENTIAL	INPUT/OUTPUT INPUT ONLY	Retrieves the next sequential item in a file.
		DIRECT ACCESS	INPUT/OUTPUT INPUT ONLY	
	MSPUT	SEQUENTIAL	OUTPUT ONLY	Delivers items sequentially from main memory to mass storage.
	MSREP	SEQUENTIAL	INPUT/OUTPUT INPUT ONLY	Replaces the last item retrieved.
		DIRECT ACCESS	INPUT/OUTPUT INPUT ONLY	
	SETM	PARTITIONED SEQUENTIAL	INPUT/OUTPUT INPUT ONLY OUTPUT ONLY	Begins processing of a specified member in the desired mode.
	ENDM	PARTITIONED SEQUENTIAL	INPUT/OUTPUT INPUT ONLY OUTPUT ONLY	Stops processing of the current member.
	MALTER	PARTITIONED SEQUENTIAL	INPUT/OUTPUT INPUT ONLY OUTPUT ONLY	Changes the specified member of a file as directed.

Table 3-4 (cont). Input/Output Macros

MACRO TYPE	MACRO CALL	TYPE OF FILE PROCESSED	FILE PROCESSING MODE	GENERAL FUNCTION PERFORMED
ACTION (continued)	MSINS	DIRECT ACCESS	INPUT/OUTPUT	Inserts an item into a Direct Access file.
	MSDEL	DIRECT ACCESS	INPUT/OUTPUT	Deletes the last item retrieved from a Direct Access file.
	MSREL	PARTITIONED SEQUENTIAL	INPUT/OUTPUT INPUT ONLY OUTPUT ONLY	Used to free up the area occupied by a Partitioned Sequential file.

Mass Storage Input/Output Control Macro - MIOC

The I/O control macro, MIOC, provides general control of the entire I/O process. More specifically, MIOC is a segmentable series of subroutines that are specialized at assembly time. Assembly time specialization causes the inclusion of all mass storage I/O functions required by the program and eliminates all those functions not required. The coding to perform the functions required by the program is further specialized when the macro call that initiates a given function is written. The macro call that causes the assembly of the control macro functions is MIOC. The specific function of MIOC is the performance of interface functions between the action macros and mass storage.

One MIOC macro call is required for each user written program that uses the mass storage I/O control facilities. The MIOC macro call contains the parameters necessary for specifying options or functions to be included in the program.

When more than one MIOC is to be included in a given program, each MIOC must originate at the same memory location. Different file requirements can thus be handled by various specializations of MIOC. Only one MIOC can be in memory at any given time. The method of achieving tag uniqueness between the various MIOC routines is explained in the description of parameter Ø1 of MIOC.

MIOC FORMAT

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ____ PAGE ____ OF ____

CARD NUMBER	TYPE	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6 7	8 14 15	20 21	62 63	80
1	L	anytag	MIOC	parameter1,...,parameter53,	The Type Field CAN
2					contain a C.
3					

MIOC DESCRIPTION

The Type Field must contain a C in all lines of coding of the MIOC call except the last line. The last line of the MIOC call must contain an L in the Type Field. Note that it is possible to have a one line MIOC call, in which case the Type Field must contain an L.

The Location Field is considered as parameter Ø of the MIOC call and can contain any acceptable assembly tag.

The Operation Code Field contains the MIOC call.

The Operands Field contains the parameters required for MIOC. Note that the function of most MIOC parameters is to insert into or delete from MIOC certain subroutines. Thus, a particular specialization of MIOC is as small as possible. A list of the MIOC parameters and an accompanying description follow.

PARAMETER 1: Unique Character. This parameter specifies a single character that will prefix each tag used by MIOC. The single character can be any one of the following:

Keypunched As	Printed As
+,8,5	%
+,8,6	□
$	$
-,8,5	" (quotation mark)
/	/
Ø,8,5	C_R

NOTE: That the character that prints as % (percent) is not the same as the character (0,8,4) which is the keypunch percent but which prints as a left parenthesis, (.

PARAMETER 2: Sequential File Functions. This parameter indicates whether or not Sequential Files will be processed by this program, and what processing mode (if applicable) will be used.

NOTE: By specifying the file functions, the user tells MIOC what subroutines to include in the program. Only coding applicable to the specified file type or processing mode is included in MIOC. For example, when only sequential files are specified, or when the input only processing mode is specified for direct access files, no coding relevant to the Insert function will be included in MIOC. A list of all applicable action macros for each file type and processing mode is given above in table 3-4.

PARAMETER 3: Sequential File Options. This parameter specifies whether or not the sequential files to be processed by this program are partitioned. If parameter 2 specifies that sequential files will not be processed by this program, parameter 3 is inapplicable.

PARAMETER 4: Direct Access Functions. This parameter specifies whether or not direct access files will be processed by this program, and what processing mode (if applicable) will be used. See NOTE 1.

PARAMETERS 5 THROUGH 9: Parameters 5 through 9 are reserved for the use of the operating system.

PARAMETER 10: Segmentation. This parameter specifies whether or not all MIOC coding is to reside in memory concurrently, or whether the MIOC coding is to be segmented. Exercising the segmentation option enables the user to save main memory locations. When specified, infrequently used functions are called into a common area of memory when required for execution. For example, each file to be processed requires the coding necessary to open it and to close it. These actions, however, are normally performed only once for a given file during a run. Therefore, the coding for these routines can reside on mass storage and be brought into main memory only when needed. When the segmentation option is exercised, the following coding will reside in main memory: MIOC, which includes the coding for the Get, Replace, Delete, and Put functions, and the MCA macro. The coding for the MSOPEN, MSCLOS ENDM, SETM, and MALTER macros will reside on mass storage until required for execution when the segmentation option is exercised. If there is no insert coding required when processing direct access files, this can be indicated and the coding required for this function will not be assembled into the program. When this is the case, the MSINS action macro call becomes invalid.

PARAMETER 11: Insert Coding Residence. This parameter is used to specify whether or not the Insert function coding is required for direct access file processing, and (if applicable) whether or not the coding is to be resident in main memory or on mass storage.

PARAMETER 12: SETM-ENDM Overlay Structure. Parameter 12 is used only when parameter 10 specified that the MIOC coding is to be segmented. Parameter 12 specifies whether or not

the coding for the SETM and ENDM functions is to be segmented so that each routine is a separate overlay, or whether the SETM and ENDM function routines are to be brought into the common overlay area together.

PARAMETER 13. Direct Access Bucket Addressing. Parameter 13 specifies whether the direct access bucket addresses are relative, actual, or both. This parameter can only be used when parameter 4 specified that direct access files are to be processed by this program.

PARAMETER 14: Multiple MIOC Indicator. This parameter is used to specify whether or not the program contains one or more MIOC macros.

PARAMETERS 15 THROUGH 31: Parameters 15 through 31 are reserved for the use of the operating system.

PARAMETER 32: Buffer Modes. This parameter specifies whether the buffering mode to be used with this program is single buffering, double buffering, or both modes.

PARAMETER 33: Item Handling Modes. This parameter specifies the methods of delivering items to the user in this program. The user has the option of specifying a locate item handling mode, a move item handling mode, or both modes. In the locate mode, the item is located and its address is supplied to the user's coding. In the move mode, the item is located and moved to the address specified by the user's coding.

PARAMETERS 34 THROUGH 49: Parameters 34 through 49 are reserved for the use of the operating system.

PARAMETER 50: Physical I/O Requirements. This parameter is used to specify whether or not the user has called the Physical I/O control macro (MPIOC). Normally, the MIOC will call MPIOC for specialization on the basis of parameters 51 through 55 of MIOC. The MPIOC macro is described in detail in Appendix B of this manual.

PARAMETER 51: Suffix. This parameter is used to specify a single character from the list of characters given in parameter 1. This character can be the same as was specified in parameter 1.

PARAMETER 52: Peripheral Control Unit Address. This parameter gives the peripheral control unit address to which the Type 250 control unit is attached.

PARAMETER 53: Write Verification. This parameter is used to specify whether or not an automatic write verification will be performed for any MCA macro in the program.

PARAMETER 54: Control of More than One PCU. This parameter specifies how the PCU number is to be specialized.

PARAMETER 55: RWC Definition. This parameter specifies how the RWC is to be specialized.

Table 3-5 contains a summary of the MIOC parameters.

Table 3-5. MIOC Parameters

PARAMETER NUMBER	NAME	VALUE	DESCRIPTION	REQUIREMENTS
ØØ	BASE	ANYTAG	The user may specify any assembly tag in this field. When MIOC is specialized, this tag will be equated with the lowest memory location that MIOC occupies.	Optional.
Ø1	UNIQUE CHARACTER	(+,8,5) % (+,8,6) □ (-,8,3) $ (-,8,5) " (0,1) / (0,8,5) C_R	This is a single character that is to be contained in each tag used by this MIOC. Note that the character that prints as % is not the same as the character (0,8,4), which is the key punch % but which prints as a left parenthesis, (.	Must be specified.
Ø2	SEQUENTIAL FILE FUNCTIONS	△	Sequential files will not be processed by this program so all coding pertaining only to sequential files can be eliminated.	Optional.
		1	Input/Output, or Input Only and Input/Output processing of sequential files will be done by this program.	When sequential files are to be processed, one of these five options must be specified.
		2	Output Only processing of sequential files will be done by the program.	
		3	Input Only processing of sequential files will be done by this program.	
		4	Input/Output and Output Only, or all three types of processing of sequential files will be done by this program.	
		5	Input Only and Output Only processing of sequential files will be done by this program.	

Table 3-5 (cont). MIOC Parameters

PARAMETER NUMBER	NAME	VALUE	DESCRIPTION	REQUIREMENTS
Ø3	SEQUENTIAL FILE OPTIONS	△	None of the sequential files to be processed by this program are partitioned.	When parameter Ø2 contains any digit between 1 and 5 and at least one of the sequential files to be processed by this program is partitioned PARTITION must be specified. Otherwise, this parameter has no significance.
		PARTITION	At least one of the sequential files to be processed by this program is partitioned.	
Ø4	DIRECT ACCESS FILE FUNCTIONS	△	Direct access files must not be processed by this program.	Optional.
		1	Input/Output, or Input Only and Input/Output processing of direct access files will be done by this program.	When direct access files are to be processed by this program one of these two options must be specified.
		2	Input Only processing of direct access files will be done by this program.	
Ø5 THROUGH Ø9	N.A.	N.A.	Parameters Ø5 through Ø9 are reserved for the use of the operating system and are not available for use by the programmer.	N.A.

Table 3-5 (cont). MIOC Parameters

PARAMETER NUMBER	NAME	VALUE	DESCRIPTION	REQUIREMENTS
1Ø	SEGMENTATION	△	All coding for this specialization of MIOC will reside in memory concurrently. See I/O Control Programmer's Preparation Information, Page 3- , for a full discussion of segmentation.	Optional.
		x	Any letter of the alphabet here indicates that this MIOC is to be segmented. This letter is used as the first character of each segment generated by this MIOC.	
11	INSERT CODING RESIDENCE	△	The **INSERT** function is not used by this program in processing direct access files.	Must be specified when parameter Ø4 is △ or 2.
		RESIDENT	The **INSERT** function is used in the direct access files processed by this program and the coding for the **INSERT** function is to be resident. For a full discusiion of resident coding see I/O Control Programmer's Preparation Information, Page 3-92, of this section.	If the **INSERT** function is to be used by this program, **RESIDENT** must be specified.
12	SETM-ENDM OVERLAY STRUCTURE	△	When parameter 1Ø is assigned a letter and this parameter is △, the coding for the **SETM** and **ENDM** functions is segmented so that each function is a separate overlay.	Optional, Note, however, that if parameter 3 is △ this parameter has no significance.
		COMBINE	When parameter 1Ø is assigned a letter and this parameter is **COMBINE**, the coding for the **SETM** and **ENDM** functions is brought into the common overlay area together as a single segment.	

Table 3-5 (cont). MIOC Parameters

PARAMETER NUMBER	NAME	VALUE	DESCRIPTION	REQUIREMENTS
13	DIRECT ACCESS BUCKET ADDRESSING	△ OR RELATIVE	The direct access bucket addresses used in this program are relative only, and are supplied in binary. For a full discussion of direct access bucket addressing see I/O Control Programmer's Preparation Information, Page 3-98 of this section.	Optional. Note, however, that if parameter Ø4 is △ this parameter has no significance.
		ACTUAL	The direct access bucket addresses used in this program are actual only, and are supplied in binary only.	
		BOTH	The direct access bucket addresses used in this program are both relative and actual, depending on the direct access file being processed by this program.	
14	MULTIPLE MIOC INDICATOR	△	Only one MIOC is included in this program.	When more than one MIOC is in the program, this parameter must be MULTIMIOCS.
		MULTIMIOCS	This program uses more than one MIOC.	
15 THROUGH 31	N.A.	N.A.	Parameters 15 through 31 are reserved for the use of the operating system and are not available for use by the programmer.	N.A.
32	BUFFER MODES	△ OR DOUBLE	This program uses double buffering.	Optional.
		SINGLE	This program uses single buffering.	
		BOTH	This program uses both double and single buffering.	

Table 3-5 (cont). MIOC Parameters

PARAMETER NUMBER	NAME	VALUE	DESCRIPTION	REQUIREMENTS
33	ITEM HANDLING MODES	△ OR LOCATE	The items are to remain in the I/O buffers and their addresses are to be supplied to this program.	Optional
		MOVE	The items are to be moved from the I/O buffers to a work area for processing by this program.	
		BOTH	This program requires that some items only be located and that some be moved into the work area for processing.	
34 THROUGH 49	N.A.	N.A.	Parameters 34 through 49 are reserved for the use of the operating system and are not available for use by the programmer.	N.A.
5Ø	PHYSICAL I/O REQUIREMENTS	△ OR CALL	This program will use the automatic Physical I/O Control facilities and wants them specialized according to parameters 51 through 55.	An MPIOC must be in the program. When this parameter is △ or CALL it will be included automatically. Otherwise, PRESENT must be specified and a separate MPIOC MPIOC must be written for this program.
		PRESENT	This program contains its own MPIOC and the specialization of the MPIOC is as indicated by parameters 51 through 55.	

Table 3-5 (cont). MIOC Parameters

PARAMETER NUMBER	NAME	VALUE	DESCRIPTION	REQUIREMENTS
51*	SUFFIX	x	This single character, which must be the same as parameter Ø1 of this MPIOC, is appended as a suffix to all tags in MPIOC.	Must be specified.
52*	PCU ADDRESS	△	This is the Honeywell recommended address for the mass storage control unit, $Ø4_8$.	If $Ø4_8$ is not acceptable, the user must specify a PCU address.
		xx_8	The address of the mass storage control unit.	
53*	WRITE VERIFICATION	△	Write verification is not required by this program unless otherwise specified.	Must be specified.
		V	The automatic write verification is to be used by this program.	
54*	CONTROL OR MORE THAN ONE PCU	M	The PCU number is to be specialized at execution time from the MCA.	Must be specified. When B is specified, the PCU number cannot be changed without re-assembly.
		B	The PCU number is to be specialized at assembly time.	

Table 3-5 (cont). MIOC Parameters

PARAMETER NUMBER	NAME	VALUE	DESCRIPTION	REQUIREMENTS
55*	RWC DEFINITION	△	The RWC will automatically be specialized depending on parameter 52. When parameter 52 is less than or equal to Ø7, a 56 is generated. When parameter 52 is greater than Ø7, a 76 is generated. This ensures that channels for the appropriate I/O Sector are used.	Must be specified.
		xx_8	RWC to be used for all data transfers. Cannot be changed without re-assembly and must correspond to sector of PCU assignment.	
		VAR	The RWC will be specialized from the communication area for each action macro call.	

*Parameters 51 through 55 are the Physical I/O parameter set. The user must specify here the values of parameters 51 through 55 that he used in his MPIOC call if he specified parameter 5Ø of this MIOC as PRESENT. If the programmer specified parameter 5Ø of this MIOC as △ or CALL, he must specify in parameters 51 through 55 how he wants the automatic MPIOC specialized. Parameters 51 through 55 of MIOC are identical to parameters 1 through 5 of MPIOC. A detailed description of MPIOC is contained in Appendix B of this manual.

Mass Storage Communication Area Macro - MCA

The general function of the communication area macro is to set up a communications area. The MCA macro provides the interface functions between the programmer and the Data Management element of the operating system. The macro call to set up the communication area is MCA. There must be one MCA macro for each file to be processed by a given program. The MCA macro sets up a table (the communication area) in which all the necessary information to identify the file and all the desired processing options are placed. The macros to interrogate and alter the communication area are MUCA and MLCA respectively. Neither of these macros are required to be included in the program. These macros are described in detail in Appendix C of this manual.

The MCA macro automatically generates a Physical I/O Communications Area (MPCA). This area has the same file prefix as specified by the user for MCA in parameter Ø. The Physical I/O Communication Area macro, MPCA, is described in Appendix B of this manual.

MCA FORMAT

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ____ . _____ PAGE ____ OF ____

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62	63 ... 80
1	L		TAG	MCA	PARAMETER1,.....PARAMETER44,	The Type Field can
2						contain a C
3						

MCA DESCRIPTION

The requirements for the MCA Type field are the same as described previously for the MIOC Type field on page 3-29. The Location field is considered as parameter Ø of the MCA call. This field establishes the 1-, 2-, or 3-character prefix to all tags using the communications area set up by the MCA. All action macros using this MCA communications

area incorporate the prefix as parameter 1. The Operation Code field contains the MCA call, and the Operands field contains the parameters required by MCA. A list of the MCA parameters and an accompanying description follow.

PARAMETER 1: Unique MIOC Character. This parameter is used to specify a single character that is identical to the character specified in parameter 1 of MIOC. The allowable characters are given in the description of MIOC Parameter 1 on page 3-29 of this section.

PARAMETER 2: Volume Address. This parameter is used to specify the address of a user supplied table containing the device address of the volume containing the file to be processed. The address specified must be a direct address of the left-most character of the table. The table should contain as many entries as there are devices associated with the file. Each entry should be three characters long and be word marked at its left-most character. There must be a record mark one character to the right of the last entry. The format of each entry is: PPDDxx (octal), where PP = the peripheral control unit number, DD = the device number, and xx = the actual location of the left-most character.

PARAMETERS 3 THROUGH 9: Parameters 3 through 9 are reserved for the use of the operating system.

PARAMETER 1Ø: I/O Buffer Address. This is the direct address of the left-most character of the data transfer buffer provided by the user. There must be three record marked characters to the right of the buffer, which must be as long as one data block. There must be no item marks in the buffer when

Logical I/O is entered and a word mark may not be on the right-most data character in the buffer. When processing direct access files, word marks cannot be in the item key. The key field may, however, be word marked at its left-most location.

PARAMETER 11: Alternate Buffer. This is the actual address of the left-most character of the second data transfer buffer provided by the user. When this parameter is specified by TAG, double buffering will be used. The format of the alternate buffer is the same as described for the buffer of parameter 1Ø. When this parameter is left blank, single buffering is used.

PARAMETER 12: Item-Delivery Mode. Parameter 12 gives the desired item delivery mode. It is used to specify whether an item is to be moved from the data transfer buffer to a user supplied work area (move mode) or whether the address of an item in the data transfer buffer is to be delivered to the user (locate mode).

PARAMETER 13: Item Linkage. This parameter is used to specify the right end direct address of a user-supplied address storage area (index register or DSA). This is used in the move item delivery mode to contain the address of a user provided work area to or from which the item is to be moved, and in the locate item delivery mode to locate for the user the left-most character of the item in the buffer. The address storage area must be the length of one item and cannot contain item marks at the time the I/O is entered.

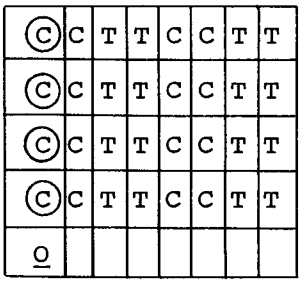
PARAMETER 14: Insert Item Linkage. Parameter 14 specifies the right end direct address of a user supplied address storage area (index register or DSA) in which is contained the address of a user provided work area that contains an item to be inserted into a direct access file. The address work area must be the length of one item and cannot contain item marks when the I/O is entered. The item to be inserted must be placed in the work area by the user. In direct access file processing, the value of parameter 14 may be the same as parameter 13.

PARAMETERS 15 AND 16: Parameters 15 and 16 are reserved for the use of the operating system.

PARAMETER 17: Units of Allocation. This parameter specifies the actual address of a user provided table into which the Units of Allocation for the file referenced by this MCA will be placed when the file is opened for processing.

When the file to be processed has more than one unit of allocation, the user must provide a table in which all the units of allocation for the file will be listed at the time the file is opened for processing. The units of allocation parameter (parameter 17) is the address of this table. When the file has only one unit of allocation, this parameter can be left blank and the system will generate the table and properly load it when the file is opened. The table provided by the user must be large enough to contain all the units of allocation applicable to the file. Each entry in the table must be 8 characters long to accommodate a unit of allocation ($C_1C_1T_1T_1C_2C_2T_2T_2$) and must contain

a word mark in the left-most location of each entry. There must be a record mark in the location immediately to the right of the last entry. A units of allocation table for a file with four units of allocation would look like:



PARAMETER 18: Direct-Access Bucket Addressing. This parameter is used to specify whether the buckets of a direct access file are addressed using relative keys or actual keys.

PARAMETER 19: Parameter 19 is reserved for the use of the operating system.

PARAMETER 20: File Name. This parameter is used to specify the name of the file to be processed. This name must be exactly the same as that assigned to the file and stored in the volume directory. It cannot be more than 1Ø characters long.

PARAMETER 21: Password. This parameter is used as a file security measure that ensures that only those persons who have knowledge of the exact password assigned to the file can process (read or write) the file. The password parameter specifies the right end address of a user supplied field (word mark on the left-most location) in which he must place the password for the file. The password placed in this field must be exactly the same as

stored in the volume directory.

PARAMETERS 22 THROUGH 29: Parameters 22 through 29 are reserved for the use of the operating system.

PARAMETER 30: Physical I/O Suffix. This parameter is used to specify a suffix to all tags written for the MIOC to which this MCA applies. The suffix specified in this parameter must be exactly the same as that specified in the parameter 51 of the MIOC macro call.

PARAMETER 31: Protection. This parameter is used to specify the type of physical protection desired for the file. This parameter is written as a two digit octal number. The physical protection afforded to a file by this parameter is as follows:

OCTAL NUMBER	PROTECTION
ØØ	Enables No Writing
Ø2	Enables Data Write
Ø6	Enables A-File Write
12	Enables B-File Write
16	Enables A- and B-File Write

In order for this parameter to be effective, the corresponding switches on the Type 250 Control Unit must be in the PERMIT position. For a comprehensive explanation of the various types of write permits (enables), refer to Appendix F.

PARAMETER 32: Verification Requirements. This parameter is used to specify whether or not data transfers are to be verified. When the verify option is exercised, all the data transfers from main memory to mass storage automatically will be verified. This ensures that each data transfer in this direction has been read back without errors. When such a read cannot be completed after several automatic correction attempts, appropriate notice is given to the user.

PARAMETERS 33 THROUGH 39: Parameters 33 through 39 are reserved for the use of the operating system.

PARAMETERS 40 THROUGH 44: Exits. Parameters 40 through 44 enable the user to exercise direct control over the operation of the I/O. For example, the user may desire to insert specific information into certain fields of *VOLDESCR* which have been set aside for the user. Another example is, when the I/O is unable to locate a file in *VOLNAMES* the programmer might want to inform the operator of the situation in a manner that is not within the capabilities of the I/O (for instance, through a teletypewriter).

To achieve direct control, the user is provided with several exits. Each exit deals with a particular portion of the I/O. For example, a single exit exists for all situations involving the Volume Directory.

Exits, therefore, to the user are multi-purpose. On the basis of an exit code, the user can take some action through his own coding. When the exit code is of no interest to the user, he can return control to the I/O, requesting that the I/O perform a predefined action. When more than one option is allowed upon return from the user's exit routines, the user sets up another code indicating his desired action or solution. For a further explanation of the exits, see I/O control Programmer's Preparation Information, page 3-100 of this section. The exits available are the following:

VOLUME DIRECTOR EXIT: This exit is taken whenever the information to be conveyed is pertinent to the

Volume Directory. For example, the user wants to look at *VOLDESCR*, or the Open macro is unable to locate the file in a volume.

INDEX EXIT: This exit is taken whenever information is pertinent to a particular file's member index. For example, the user specified SETM input/output and the SETM macro is unable to locate the member.

DATA EXIT: This exit is taken whenever the information to be conveyed is pertinent to this file's data. For example, when the item is an end of data item on an input file, or when there is no more room on an output file this exit will be taken. This exit must be specified whenever the user expects to process to the end of an input file.

DEVICE EXIT: This exit is taken when the information to be conveyed is pertinent to a device currently being used for this file. For example, when a READ or a WRITE error occurs, or when the device is inoperative this exit is taken.

A summary of the MCA parameter is contained in Table 3-6.

Table 3-6. MCA Parameters

PARAMETER NUMBER	NAME	VALUE	DESCRIPTION	REQUIREMENTS
ØØ	FILE PREFIX	1,2,or 3 PREFIX CHARACTERS	These characters prefix all MCA tags. Action macros that are to use the communications area set up by this MCA must include this prefix.	Must be specified.
Ø1	UNIQUE MIOC CHARACTER	4	This character must be the same as specified in parameter Ø1 of MIOC.	Must be specified.
Ø2	VOLUME ADDRESS	TAG	See preceding description of MCA parameter Ø2.	Must be specified.
Ø3 THROUGH Ø9	N.A.	N.A.	Parameters Ø3 through Ø9 are reserved for the use of the operating system and are not available for use to the user.	N.A.
1Ø	INPUT/OUTPUT BUFFER ADDRESS	TAG	See preceding description of MCA parameter 1Ø.	Must be specified.
11	ALTERNATE BUFFER	TAG	See preceding description of MCA parameter 11.	Optional.
		△	This file is single buffered.	
12	ITEM DELIVERY MODE	MOVE	Items are to be moved to or from the I/O buffer from or to the work area.	Optional.
		△ OR LOCATE	The address of the item in the I/O buffer is to be delivered to this program.	Optional
13	ITEM LINKAGE	TAG	See preceding description of MCA parameter 13.	Must be specified.

Table 3-6.(cont). MCA Parameters

PARAMETER NUMBER	NAME	VALUE	DESCRIPTION	REQUIREMENTS
14	INSERT ITEM LINKAGE	TAG	See preceding description of MCA parameter 14.	Optional. Note that MIOC parameter Ø4 must be specified as either 1 or 2.
		△	The Direct Access file this program is processing does not require the insert function coding.	
15 AND 16	N.A.	N.A.	Parameters 15 and 16 are reserved for the use of the operating system and are not available for use by the programmer.	N.A.
17	UNITS OF ALLOCATION	TAG	See preceding description of MCA parameter 17.	Optional.
		△	Because this file was only one unit of allocation, no tag is necessary.	
18	DIRECT ACCESS BUCKET ADDRESSING	△ OR RELATIVE	Buckets are relatively addressed in binary for this file.	Optional.
		ACTUAL	The actual key, in binary, is given for buckets in this field.	
19	N.A.	N.A.	Parameter 19 is reserved for the use of the operating system and is not available for use by the programmer.	N.A.
2Ø	FILE NAME	1Ø CHARACTERS	This is the name of the file, as it is in the volume directory, for which this MCA is building the communications area.	Must be specified.
21	PASSWORD	TAG	See preceding description of MCA parameter 21.	Must be specified. when file is protected by a password.
		△	The password in the volume directory is blank, therefore this file is not protected by a password.	

Table 3-6.(cont.) MCA Parameters

PARAMETER NUMBER	NAME	VALUE	DESCRIPTION	REQUIREMENTS
22 THROUGH 29	N.A.	N.A.	Parameters 22 through 29 are reserved for the use of the operating system and are not available for use by the programmer.	N.A.
3Ø	PHYSICAL I/O SUFFIX	4	This character must be the same as specified in parameter 51 as MIOC.	Must be specified.
31	PROTECTION	44_8	See preceding description of MCA parameter 31 and Appendix F of this manual.	Optional.
		△	The value for this parameter will be $ØØ_8$. Thus, no writing is permitted.	
32	VERIFICATION REQUIREMENTS	△	Data transfers to this file will not be verified.	Optional
		VERIFY	All data transfers to this file (writes) are to automatically be verified.	
33 THROUGH 39	N.A.	N.A.	Parameters 33 through 39 are reserved for the use of the operating system and are not available for use to the programmer.	N.A.
4Ø	VOLUME DIRECTORY EXIT	TAG	See I/O control programmer's preparation information regarding exits on page 3-100 of this section. Also see preceding description of MCA parameters 4Ø through 44.	Optional.
41	INDEX EXIT	TAG		Optional.
42	EVERY INDEX ENTRY EXIT	TAG or △		Optional.
43	DATA EXIT	TAG		Optional.
44	DEVICE EXIT	TAG		Optional.

Action Macros

The mass storage Action macros are summarized in Table 3-7. The following paragraphs describe the functions performed by these macros. The function performed by each macro varies according to the type of file being processed and the mode in which the processing takes place. In the following discussions, the term "Exit" is used as described previously in this section.

ACTION MACRO FUNCTIONS RELATED TO ALL SEQUENTIAL FILES

Open Function

The open function is used to open a file for processing. When the file is not partitioned, it is opened in the processing mode specified in the MIOC macro call. When the file is partitioned, however, the processing mode for the member is not specified until the set member function is executed. The following sequence of events occurs when the open function is executed.

1. The appropriate file description is located in the volume description. If the file name cannot be located in *VOLNAMES*, an exit to the user is made. A return to the open function after this exit indicates that a new volume has been loaded and a new open attempt is to be made.

2. If password checking is specified in the MCA macro, a comparison of the password given by the user in the MCA macro call and that stored for the file is made. When password checking was not specified in the MCA macro, the password field for the file is checked to see that it contains all blanks, i.e., no password. If either of these checks produce a discrepancy, an exit to the user is made.

Table 3-7. Action Macro Calls

PROCESSING MODE	SEQUENTIAL FILE ORGANIZATION ACTION MACRO CALLS	PARTITIONED SEQUENTIAL FILE ORGANIZATION ACTION MACRO CALLS	DIRECT ACCESS FILE ORGANIZATION ACTION MACRO CALLS
INPUT/OUTPUT MODE	MSOPEN MSCLOS MSGET MSREP	MSOPEN MSCLOS MSGET MSREP SETM ENDM MALTER MSREL	MSOPEN MSCLOS MSGET MSREP MSINS MSDEL
INPUT ONLY MODE	MSOPEN MSCLOS MSGET	MSOPEN MSCLOS MSGET SETM ENDM	MSOPEN MSCLOS MSGET
OUTPUT ONLY MODE	MSOPEN MSCLOS MSPUT	MSOPEN MSCLOS MSPUT SETM ENDM MALTER MSREL	NOT APPLICABLE

At this point, processing will halt until the user either clears the password field for the file or includes the proper password for the file in the MCA macro call. After this, the user can again request that the file be opened.

3. When steps 1 and 2 are successfully completed, exit is made to the user's coding so that he can, if he desires, examine the file description, *VOLDESCR*. A return form the user's coding at this point indicates either that the open function is to continue for this file or that, after examination of *VOLDESCR*, the user rejected the file and a new file is to be opened. (In the latter case, steps 1 and 2 will be repeated.) If the open function for the original file is to continue, and that file is to be processed in the input/output or output only mode, *VOLDESCR* is written back to mass storage. The same holds true in the case in which a new file is to be opened after the successful completion of steps 1 and 2 for the new file.

4. All information included in *VOLDESCR* that is required by other I/O functions is moved to the file's communication area. Generally, this is information that was not specified when the MCA for the file was specialized.

5. When the file is being opened in the output only mode and the item delivery mode in the MCA macro is specified as LOCATE, the address of the left-hand end of the first item location in the current buffer is moved to the field specified by the user in parameter 1Ø of the MCA macro. Note that this step does not apply if the file is partitioned.

6. An indicator is set in the communication area, showing the appropriate processing mode. Note that this indicator is not set by the open function when the file is partitioned. When this is the case, the SETM function sets this indicator because the processing mode for the member is not given until the SETM function is executed.

Close Function

The close function is used to close a file after processing. The following sequence of events occurs when the close function is executed.

1. When the file that was being processed was not partitioned and the processing mode used for this file was output only, the close function ensures that all buffers have been written back to the device and that the item following the last item written back is truly an end-of-data item. An end-of-data item is signified by □EOD¢ in its first five character positions.

   When the file that was processed was partitioned and the processing mode used was input/output, the close function ensures that the current buffers have been written back to the device if a replace function was executed for any item in those buffers.

2. An exit to the user's coding is made at this time so that he can, if he wishes, examine *VOLDESCR*.

3. A normal return to the close function from the user's exit routine causes the close function to write back *VOLDESCR* to the device, when the processing mode for the file was either output only or input/output.

Get Function

The get function is used to deliver the next sequential item in the file to the user. This function can only be executed in the input/output and input only modes of processing. Note that "buffer priming" is accomplished with the first get function that is executed. When the get function is executed, the following sequence of events occurs.

1. When the next sequential item is in the current buffer, it is examined to see whether or not it is an end-of-data item. When it is an end-of-data item an exit to the user's coding is made indicating that the end-of-data has been encountered. There should be no return from the exit, a close function or an end member processing function if the file is partitioned should be the next action issued for the file.

2. When the next sequential item is not in the current buffer, the get function determines whether or not the current buffer is to be written back to the device. This determination is based on whether or not a replace function has been issued for any item in the buffer. When this is the case, the current buffer is written back to the device. Note that this step has no significance when processing is in the input only mode.

3. Depending on the buffering mode being used, the get function causes the current buffer to be loaded with the next sequential block from the device.

4. Step 1 is now repeated. If the next item is not an end-of-data item, it is delivered to the user established work area when move item handling mode was specified. When the locate item handling mode was specified, the address of the left-most

character of the next item in the buffer is delivered to the user established address field.

5. When step 4 is completed, the get function returns to the user's main line coding after ensuring that the address of the item just retrieved is available to the user in the communication area in the following format:
D M C C T T R R I I where D = the device number, M = the magazine number (always zero), CCTTRR = a mass storage record address and II = the relative item within the block. (When II = ØØ the current item is the first item in the block.)

NOTE: The mass storage address, CCTTRR, can be presented to Physical I/O with an extended search and read instruction to cause the block containing the item to be re-accessed. (Physical I/O is described in Appendix B of this manual.) This record is either the first record of the block containing the item, a track linking record that points to the first record of the block containing the item or it is the first record of a partial block, which is the last portion of the cylinder previous to the block containing the item. A partial block does not contain valid data.

Replace Function

The replace function is used to replace the item in the file that was retrieved by the last get function. This function, replace, can only be executed in the input/output processing mode. When a replace function is executed the following sequence of events occurs.

1. The replace function sets an indicator in the communication area showing that a replace function has been issued for the item to which the last get function referred. This ensures that the current buffer is written back to the device after it is used but before it is overlaid with a new block.

2. When processing in the move item handling mode, the item in the current buffer is overlaid with the item in the user's work area before the buffer is written back to the device.

Put Function

The put function is used to deliver items sequentially from main memory to the mass storage device. Recall that when processing in the locate item handling mode, either the open or the set member processing function places an initial item delivery address in the user's address field. The put function can only be executed in the output only processing mode and when executed either of the two following actions occurs.

1. When operation is in the move item handling mode, the put function moves the user's item to the current buffer. To do this, the put function first must determine if there is room in the current buffer for another item or if there is not. When no room is available for another item in the current buffer, the put function next determines whether or not there is room in the file for another block of data. When there is no room in the current buffer for another item but the file can accept another block, the put function writes the current block back to the device, sets an indicator pointing to the new current buffer and returns to the user. When the file has no room for another block an exit to the user's coding is made. There can be no return from this exit and the next action issued for the file must be either

a close function or an end member processing function (if the file is partitioned). When either of these is the case, the next action issued for the file will overlay with an end-of-data item the last item for which the put function was issued. This item, however, will remain in the user's work area.

2. When operating in the locate item handling mode, the address of the left-most character location of the next available item is moved to the user's address field by the put function. When this is done, the put function returns to the user's coding.

ACTION MACRO FUNCTIONS RELATED ONLY TO PARTITIONED SEQUENTIAL FILES

Set Member Function

The set member function is used to start processing at the beginning of the specified member in the specified processing mode. The user has the option of requesting an exit after each member index entry is located by the set member function that shows an undeleted member. If this option is exercized, the name portion of the index entry is never interrogated by the set member function. Rather, the set member function supplies the user with the address of the left-most character location of the index entry. It is then up to the user to interrogate this member index entry and decide whether or not this is the member he desires. A return from the user's coding will cause either the continuation of the search of the member index or it will cause the opening of the member. Opening the member, in this case, depends on a valid member status, i.e., that the member is available for processing.

When the option is not exercized, the set member function execution

causes the following sequence of events to occur when processing is to be in the input only mode.

1. The member index entry for an undeleted member whose name is the same as that specified is located. When the name of the desired member cannot be found, an exit to the user is made. (This would be a good time to re-issue the set member function macro call and exercize the option just discussed.)

2. When the name of the desired member is located, the address of the member's first item is set into the communications area for the file.

3. When this is accomplished, the processing mode indicator is set to show input only processing in the communications area.

A normal execution of the set member function in the input/output processing mode is the same as described for the input only mode except that the processing mode indicator in the communications area is set to show input/output processing.

A normal execution of the set member function in the output only mode of processing will cause the following sequence of events to occur.

1. A search of the member index is made for an undeleted member whose name is the same as that specified. When this member is found, a check is made to ensure that the member is available for output only processing. When the member is not available for output only processing, an exit to the user's coding is made. At this point a new set member or close function can be initiated.

2. When the search of the member index reveals that no member exists with the specified name, verification of the fact

that there is room in the member index for another entry is made. If there is no verification of this, an exit to the user's coding is made and a new action must be specified.

3. When verification of the room is made, an indicator is set showing that a new member is being created.

4. With this accomplished, the address of the first item of the unused area is set into the communications area for the file and the processing mode indicator in the communications area is set to show output only processing.

End Member Function

The end member function is used to close a member of a partitioned sequential file after processing. When the member is a new member, i.e., created by the previous set member function, and was created in the output only processing mode, the end member function generates a member index entry for the new member. When the member index entry is generated, the end member function also appropriately decreases the length of the unused area recorded in the member index. In connection with this, it is sometimes necessary for the end member function to create a new end-of-index entry for the member index.

When the processed member is not a new member, and was processed in either the input/output or the input only mode, the end member function ensures that all the buffers have been written back to the device. When the member was processed in the output only mode, the end member function also ensures that an end-of-data item has been generated for the member.

The final operation of the end member function is the setting of an indicator in the file's communications area showing that no member is open.

Alter Member Function

The alter member function is used to change the specified member according to the values of the various parameters in the macro call. The user has the same option with the alter member function described for the set member function. In the normal execution of the alter member function, the following sequence of events occurs.

1. The member index entry for the specified member is located and one of the following operations is performed:
   a. The member's status is changed to "available for output only processing."
   b. The member's status is changed to "unavailable for output only processing."
   c. After verifying that the member is available for output only processing, the member's status is changed to "deleted." When the member's status is not "available for output only processing", exit to the user's coding is made.
   d. The member's current name is overlaid with a new name.
2. When the member index entry for the specified member cannot be located, an exit to the user's coding is made.

Release Function

The release function is used to release the partitioned sequential file specified in the macro call so that no members exist and the complete data area is available for re-use. To do this, however, the file must be opened first. Note that whenever the release function is issued by the programmer, verification of the "availability for processing" status of the active member is not made by the release function. When executed, the release function moves the end-of-index entry in the

member index to the second position in the index and the unused area entry (the first entry in the index) is set to point to the first data block in the file.

The release function eliminates the need for deleting all the members of a file and re-allocating another partitioned sequential file. This can be very beneficial if the members of the file are used for the storage of temporary data or as work areas. The major drawback to this is that the original file has to be large enough initially to accommodate any subsequent member.

ACTION MACRO FUNCTIONS RELATED TO ALL DIRECT ACCESS FILES

Open Function

The open function is used to open the file specified in the macro call, in the specified processing mode. Direct access files cannot be processed in the output only mode, consequently, either the input/output mode or the input only mode must be specified. When the open function is executed, the following sequence of events occurs.

1. The appropriate file description is located in the volume directory. If the file name cannot be located in *VOLNAMES*, an exit to the user is made.
   A return to the open function after this exit indicates that a new volume has been loaded and a new attempt is to be made.

2. If password checking is specified in the MCA macro, a comparison of the password given by the user in the MCA macro call and that stored for the file is made. When password checking was not specified in the MCA macro, the password field for the file is checked to see that it contains all blanks, i.e., no password. If either of these

checks produces a discrepancy, an exit to the user is made. At this point, processing will halt until the user either clears the password field for the file or includes the proper password for the file in the MCA macro call. After this, the user can again request that the file be opened.

3. When steps 1 and 2 are successfully completed, exit is made to the user's coding so that he can, if he desires, examine the file description, *VOLDESCR*. A return from the user's coding at this point indicates that the open function is to continue for the file or that, after examination of *VOLDESCR*, the user rejected the file and a new file is to be opened. (In the latter case steps 1 and 2 will be repeated.)

4. All information included in *VOLDESCR* that is required by other I/O functions is moved to the file's communications area. Generally, this is information that was not specified when the MCA macro was specialized.

5. After the appropriate information is moved into the communications area, an indicator is set showing that the file has been opened. A second indicator is set that shows in which processing mode the file was opened.

6. The actual key of the file's first item is set into the files communications area so that the first processing action is not required to specify a bucket address.

Close Function

The close function is used to close a file when processing has been completed. When the close function is executed, the following sequence of events occurs.

1. The close function writes the buffers back to the device when a replace function or a delete function was issued for an item in the buffers, only if the file was processed in the input/output mode.

2. The item count for the file is updated in *VOLDESCR*.

3. An exit to the user is made so that he can examine *VOLDESCR*, only if the processing mode was input/output. A normal return from the user at this point causes *VOLDESCR* to be written back to the device.

4. An indicator is set in the file's communications area showing that the file is closed.

Get Function

The get function is used to get an item as specified in the macro call. Because getting an item can be done by specifying an item key, a bucket address, both or neither, the operations performed when the get function is executed vary.

When the get function macro call specifies a bucket address and an item key, the get function begins searching the specified bucket for an undeleted item with the specified key. When the item is located it is delivered to the user in either the locate or move item handling mode, depending on which was specified in the macro call. If the desired item is not located in the specified bucket, the cylinder overflow area is searched. When this is done an indicator in the file's communications area is set to show this. If the item is not in the cylinder overflow area, the general overflow area (if any) is searched and an indicator is set in the communications area showing this. If, at the end of the general overflow area (or at the end of

the cylinder overflow area when there is no general overflow area), the desired item is not located or an inactive item is encountered, an exit to the user is made indicating that the item is not in the file.

When only the item key is given in the macro call, the current bucket is searched for the desired item. This search commences with the next sequential item in the current bucket and continues in the same sequence of area searching as just described.

When only the bucket is given in the macro call, the specified bucket is sequentially searched from its beginning for an undeleted item. No significance is placed on this item's key. When located, the item is delivered to the user in either the locate or move item handling mode, as specified. When there is not an undeleted item on the cylinder (from the beginning of the search), the cylinder overflow area is sequentially searched. If there is not an undeleted item in this area, the next subsequent cylinder in the unit of allocation for the file is sequentially searched, in the manner just described. This method of searching continues until either the end of the general overflow area (if any) is reached or until an inactive item is encountered. In either of these cases, an exit to the user is made indicating that there are no undeleted items in the file.

If neither an item key nor a bucket address is given in the macro call, the current bucket, i.e., the bucket to which the last action referred, is sequentially searched, starting with the next sequential item. The sequence of events here is the same as that described for searching when only the bucket address is specified.

Replace Function

The replace function is used to replace in the file the item retrieved by the last get function when processing the file in the input/output mode. This function cannot be used when processing in the input only mode.

When the replace function is executed, an indicator is set in the file's communications area indicating that a replace has been issued for an item in the current block. Next, the replace function verifies that the item to replace the original item is in the current buffer. With this done, the block is written back to the device before it is overlaid.

Insert Function

The insert function is used to insert items into the file as specified by the parameters of the macro call when processing the file in the input/output mode. Like the replace function, the insert function cannot be executed in the input only mode.

When the insert function is executed and the bucket address is given in the macro call parameter, a search for the next available item position is made from the beginning of the specified bucket. Recall that a previous get for an undeleted item sets an indicator in the file's communications area showing the location of available space in a bucket. The insert function interrogates the communications area to determine if the specified bucket has room for another item. If it does have room, the item is placed in the bucket. If, however, the communications area shows that the bucket does not have room, the cylinder overflow area is searched for space to insert the item. This saves a redundant search of the bucket by the insert function. When the cylinder overflow area is entered, an indicator is set in the

file's communications area showing this. When there is no room in this area, the general overflow area (if any) is entered. Also, when this area is entered, an indicator is set. When room for the item is located, the item is moved from the user's item work area, established for inserting items, into the current buffer and this is then written back to the device. This is done regardless of the item handling mode. When there is no room in the file for the item, the item is not moved from the item work area and an exit to the user is made indicating that there is no room for the item in the file.

When the bucket address is not given in the macro call and the insert function is executed, searching begins at the current position in the current buffer. The sequence of events from this point onward is the same as just described.

NOTE: Duplicate item key checking is not incorporated into the insert function since the programmer may check for duplicates simply by issuing a get before each insert.

Delete Function

The delete function is used to delete the current item. To do this, the delete function sets the item's status character to deleted and sets an indicator that ensures that the current block is written back to the device.

MSOPEN	Opens a file for processing.

FILE ORGANIZATION: Sequential and Direct Access.

PROCESSING MODES: Sequential Files; all modes.
Direct Access Files; all modes except the Output Only mode.

FORMAT

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	INDIRECT	LOCATION	OPERATION CODE	OPERANDS
1		L	Any tag	MSOPEN	File-tag, {IN/OUT / IN / OUT / UPDATE / BLANK},
2					
3					
4					
5					

DESCRIPTION:

Location Field

This tag references the first instruction in the generated coding. It can be any acceptable assembly tag; but it is not required and can be omitted if desired.

Operands Field

File-tag: the file-tag is the 1, 2, or 3 character prefix specified in parameter Ø of the appropriate MCA.

IN/OUT: This parameter specifies that the processing mode is to be Input/Output.

IN: this parameter specifies that the processing mode is to be Input Only.

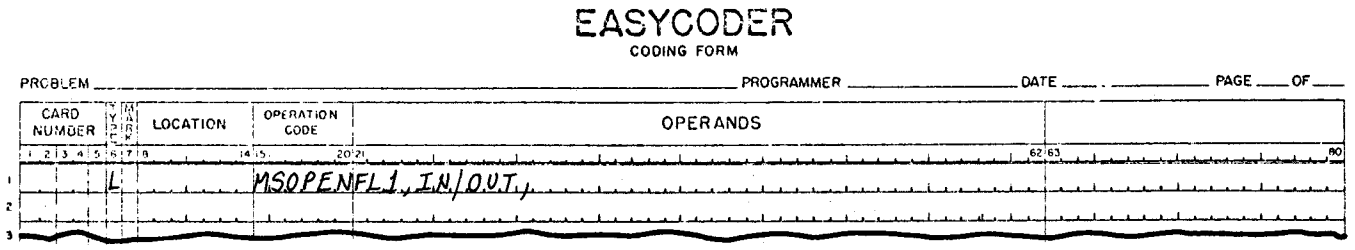
OUT: This parameter specifies that the processing mode is to be Output Only. This cannot be specified for Direct Access files.

UPDATE: this must be specified when the file being opened is a

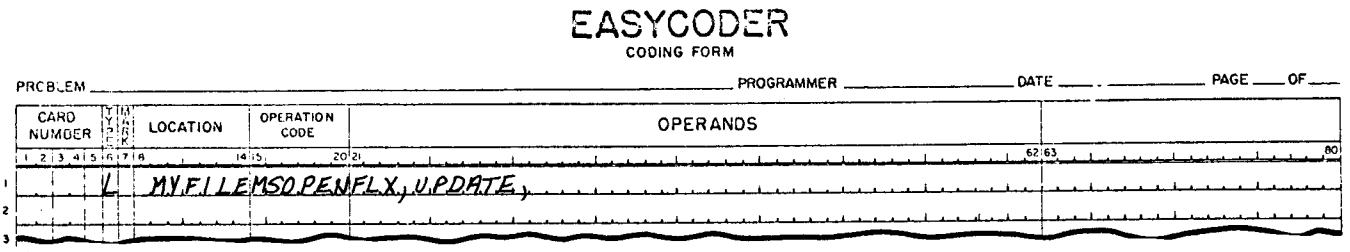
Partitioned Sequential file that is to be processed in the INPUT/OUTPUT or OUTPUT ONLY mode. When the file being opened is a Partitioned Sequential file that is to be processed in the INPUT ONLY mode, this parameter can be left blank. This parameter cannot be used with Non-partitioned Sequential or Direct Access files.

EXAMPLES:

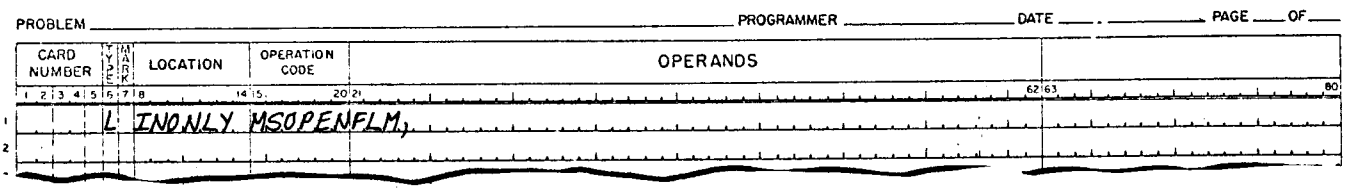
The following coding will cause File-1 (FL1) to be opened for processing in the INPUT/OUTPUT mode.



The following coding will cause the Partitioned Sequential file FLX to be opened for OUTPUT ONLY processing and be tagged MYFILE.



The following coding will cause the Partitioned Sequential file FLM to be opened for INPUT ONLY processing and be tagged INONLY.

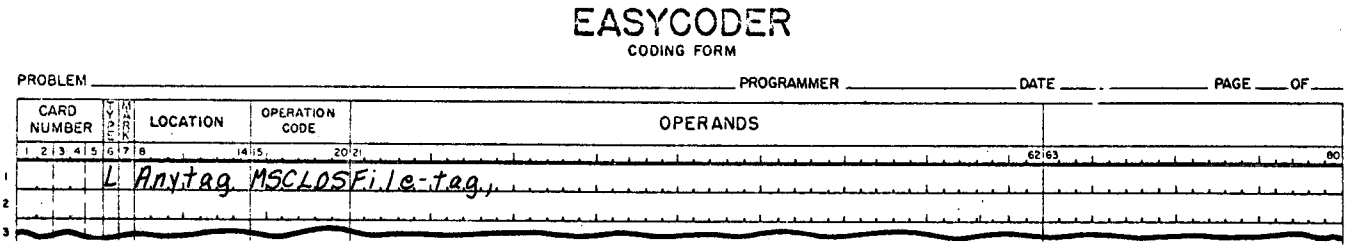


MSCLOS	Closes a file after processing.

FILE ORGANIZATION: Sequential and Direct Access Files.

PROCESSING MODES: The mode in which a file was processed is not significant to the MSCLOS macro.

FORMAT



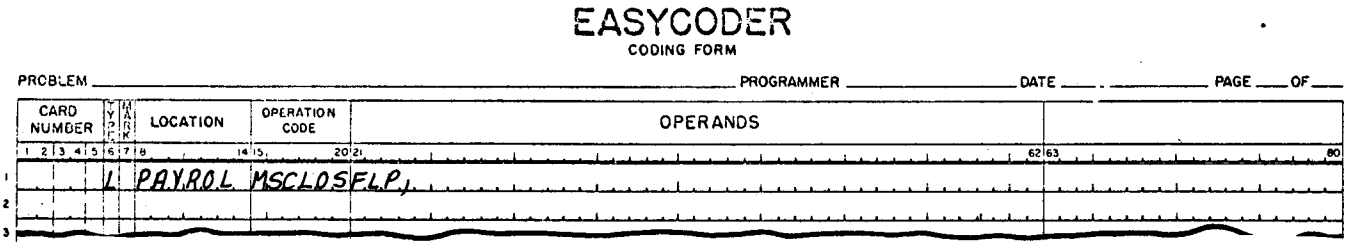
DESCRIPTION:

Operands Field

File-tag: the File-tag is the 1, 2, or 3 character prefix specified in parameter Ø of the appropriate MCA.

EXAMPLE:

The following coding will cause the payroll file FLP, tagged PAYROL, to be closed when processing has been completed.



NOTE: When the file being processed is a Partitioned Sequential file, the MSCLOS macro must be preceded by the ENDM macro.

MSGET	Retrieves the next sequential item in a file.

FILE ORGANIZATION: Sequential and Direct Access files.

PROCESSING MODES: Sequential Files; INPUT/OUTPUT and INPUT ONLY modes.
Direct Access Files; INPUT/OUTPUT and INPUT ONLY modes.

FORMAT

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1		L	Anytag	MSGET	File-tag,{Bucket-tag} Key-tag,
2					{NEXT}
3					
4					

DESCRIPTION:

Operands Field

File-tag: the File-tag is the 1, 2, or 3 character prefix specified in parameter Ø of the appropriate MCA.

Bucket-tag: the Bucket-tag points to a user defined bucket address field. This parameter cannot be used with Sequential Files.

Key-tag: the Key-tag points to a user defined field where the Item Key is located. This parameter cannot be used with Sequential Files.

NEXT: this parameter is specified when neither the bucket or key is specified for Direct Access Files. This parameter cannot be used with Sequential Files.

EXAMPLES:

The following coding will cause the retrieval of the next sequential

item in the sequentially organized accounts receivable file ACC, tagged INCOME.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___.___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62 63 ... 80
1	L		INCOME	MSGET	ACC,
2					
3					

The following coding will cause the retrieval of the next item in a Direct Access File DAF, tagged INVTRY (inventory).

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___.___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62 63 ... 80
1	L		INVTRY	MSGET	DAF,NEXT,
2					

The following coding will cause the retrieval of the next item in a Direct Access File FL8, tagged MYFILE. The bucket address is contained in the field tagged BUCKET and the Item Key is contained in the field tagged ITMKEY.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___.___ PAGE ___ OF ___

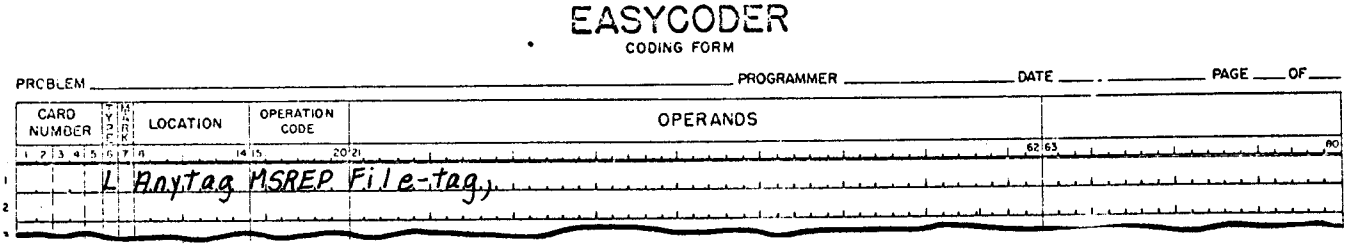
CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62 63 ... 80
1	L		MYFILE	MSGET	FL8,BUCKET,ITMKEY,
2					
3					

MSREP	Replaces the last item retrieved.

FILE ORGANIZATION: Sequential and Direct Access Files.

PROCESSING MODES: Sequential Files; INPUT/OUTPUT mode.
Direct Access Files; INPUT/OUTPUT mode.

FORMAT



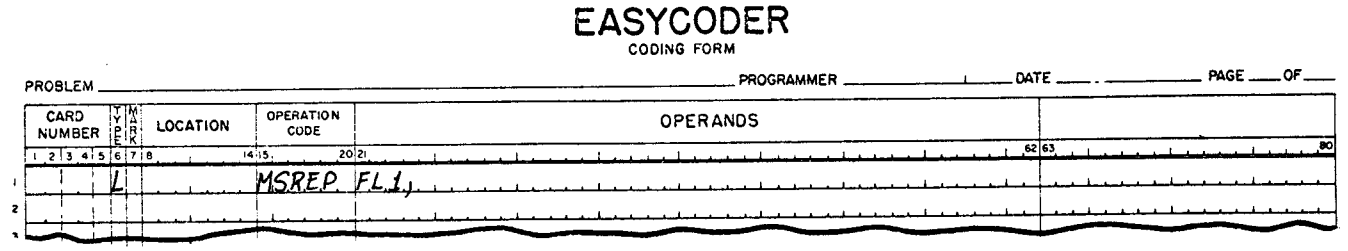
DESCRIPTION:

Operands Field

File-tag: the File-tag is the 1, 2, or 3 character prefix specified in parameter Ø of the appropriate MCA.

EXAMPLE:

The following coding will cause the last item retrieved to be replaced in File 1, FL1.

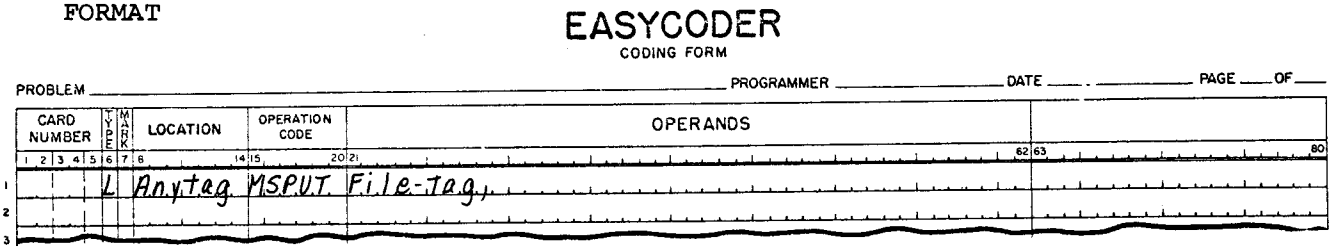


MSPUT	Delivers items sequentially from main memory to mass storage.

FILE ORGANIZATION: Sequential Files.

PROCESSING MODES: OUTPUT ONLY mode.

FORMAT



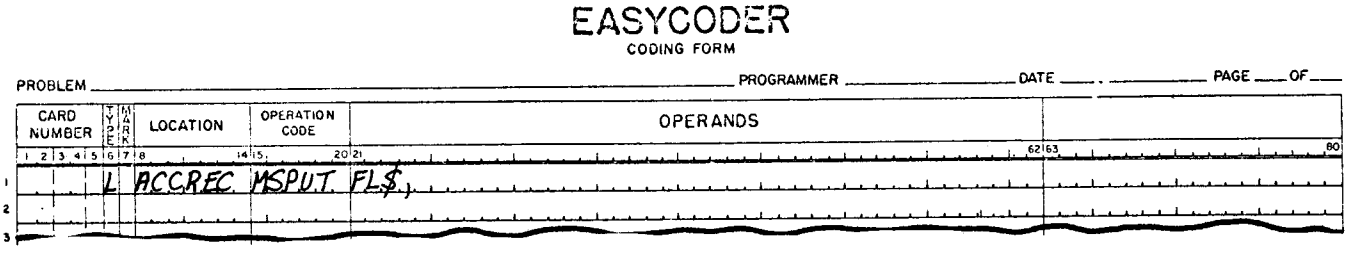
DESCRIPTION:

Operands Field

File-tag: the File-tag is the 1, 2, or 3 character prefix specified in parameter Ø of the appropriate MCA.

EXAMPLE:

The following coding will cause the next sequential item to be delivered from main memory to File $ (FL$) at the location indicated by the tag ACCREC (accounts receivable).



SETM	Begins processing of a specified member in the desired mode.

FILE ORGANIZATION: Partitioned Sequential Files.

PROCESSING MODES: All modes.

FORMAT

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

CARD NUMBER	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8		14 15	20 21	62 63 80
	L	Any tag	SETM	File-tag, Member-name-tag, {IN/OUT / IN / OUT}, ?

DESCRIPTION:

Operands Field

File-tag: the File-tag is the 1, 2, or 3 character prefix specified in parameter Ø of the appropriate MCA.

Member-name-tag: the Member-name-tag points to the address of a user defined field that contains the name of the member desired. This parameter does not have to be specified if parameter 42 of the appropriate MCA was specified.

IN/OUT: this parameter specifies the processing mode as INPUT/OUTPUT.

IN: this parameter specifies the processing mode as INPUT ONLY.

OUT: this parameter specifies the processing mode as OUTPUT ONLY.

EXAMPLES:

The following coding will cause the beginning of processing of Member G (MEMG) of File W (FLW) in the OUTPUT ONLY mode.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8 ... 14 15 ... 20 21 ... 62 63 ... 80				
1	L		SETM	FLW,MEMG,OUT,
2				
3				

The following coding will cause the beginning of processing of ACCREC (accounts receivable) of File 1 (FL1), tagged BILING (billing) in the INPUT/OUTPUT mode.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8 ... 14 15 ... 20 21 ... 62 63 ... 80				
1	L	BILING	SETM	FL1,ACCREC,IN/OUT,
2				
3				

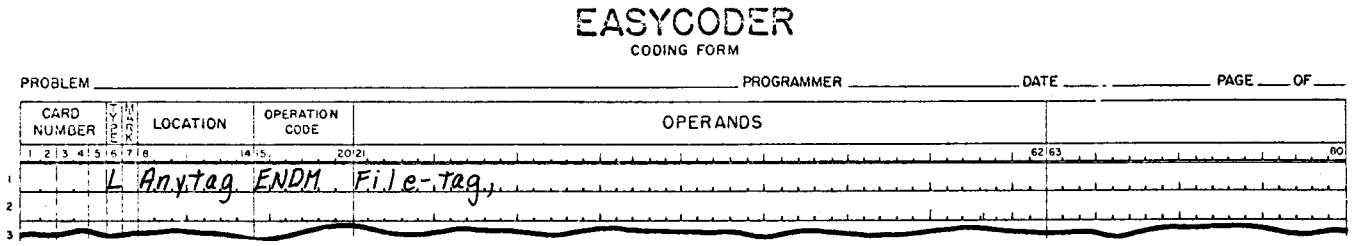
NOTE: The SETM macro must be preceded by a MSOPEN macro.

ENDM	Stops processing of the current member of a Partitioned Sequential File.

FILE ORGANIZATION: Partitioned Sequential File.

PROCESSING MODES: The processing mode is not significant to the ENDM macro.

FORMAT



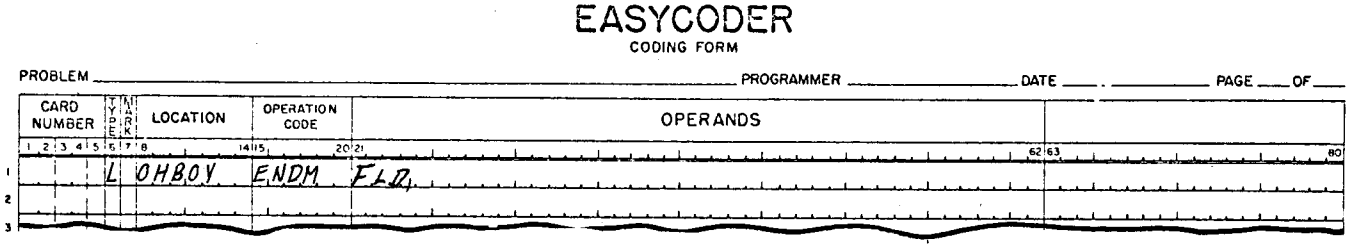
DESCRIPTION:

Operands Field

File-tag; the File-tag is the 1, 2, or 3 character prefix specified in parameter Ø of the appropriate MCA.

EXAMPLE:

The following coding will cause processing of the current member of File D (FLD), tagged OHBOY, to stop.



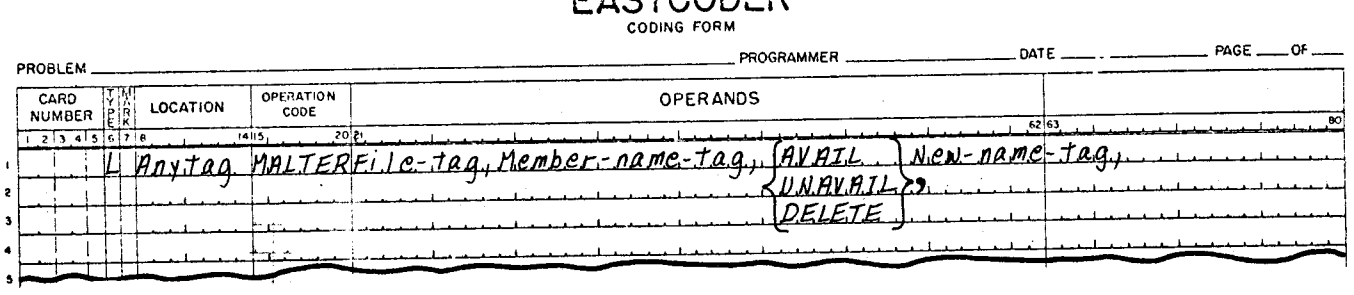
NOTE: The ENDM macro must precede an MSCLOS macro to close the file.

MALTER	Changes the specified member of a file as directed.

FILE ORGANIZATION: Partitioned Sequential Files.

PROCESSING MODES: The processing mode is not significant to the MALTER macro.

FORMAT



DESCRIPTION:

Operands Field

File-tag: the File-tag is the 1, 2, or 3 character prefix specified in parameter Ø of the appropriate MCA.

Member-name-tag: the Member-name-tag points to the address of a user supplied field that contains the name of the desired member.

AVAIL: this parameter changes the member's status to AVAILABLE for OUTPUT ONLY processing.

UNAVAIL: this parameter changes the member's status to UNAVAILABLE for OUTPUT ONLY processing.

DELETE: this parameter changes the member's status to DELETED if the member was available for OUTPUT ONLY processing.

New-name-tag: this tag points to the address of a user supplied field that contains the new name for the member.

NOTE: Either one of the status changing parameters AVAIL, UNAVAIL, or

DELETE or the New-name-tag parameter must be specified. The New-name-tag parameter along with one of the status changing parameters also may be specified.

EXAMPLES:

The following coding will cause the member tagged PDQ of File ABC, tagged FILE1, to become available for output only processing.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7 8	14 15	20 21	62 63 80
1	L		FILE1	MALTER	ABC,PDQ,AVAIL,
2					
3					

The following coding will cause the member tagged O.K of File DIP, tagged BEGIN, to have its name changed to RUN.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7 8	14 15	20 21	62 63 80
1	L		BEGIN	MALTER	DIP,O.K,RUN,
2					
3					

The following coding will cause the member tagged XYZ of File 1 (FL1) to become unavailable for output only processing and have its name changed to HERO.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7 8	14 15	20 21	62 63 80
1	L			MALTER	FL1,XYZ,UNAVAIL,HERO,
2					
3					

MSREL	Used to free up the area occupied by a Partitioned Sequential File.

FILE ORGANIZATION: Partitioned Sequential File.

PROCESSING MODES: The mode in which the file is processed is not significant to the MSREL macro.

FORMAT

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	NAME	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7 8 ... 14 15 ... 20			21 ... 62 63 ... 80
1	L		Anytag	MSREL	File-tag,
2					

DESCRIPTION:

Operands Field

File-tag: the File-tag is the 1, 2, or 3 character prefix specified in parameter Ø of the appropriate MCA.

EXAMPLE:

The following coding will cause the release of the Partitioned Sequential File (FL6) so that no members exist and the complete data area of this file becomes available for use.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	NAME	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7 8 ... 14 15 ... 20			21 ... 62 63 ... 80
1	L			MSREL	FL6,
2					

MSINS	Inserts an item into a Direct Access File.

FILE ORGANIZATION: Direct Access File.

PROCESSING MODES: INPUT/OUTPUT mode.

FORMAT

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	CONT	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 ... 14 15 ... 20	21 ... 62	63 ... 80	
1		L	Anytag	MSINS	File-tag,Bucket-tag,	
2						

DESCRIPTION:

Operands Field

File-tag: the File-tag is the 1, 2, or 3 character prefix specified in parameter Ø of the appropriate MCA.

Bucket-tag: the Bucket-tag points to a user defined bucket address field. This parameter is optional and can be omitted if desired.

EXAMPLES:

The following coding will cause an item to be inserted into an inventory file (INV), tagged ATOPRT (automobile parts). The bucket address for this item is contained in the field tagged WHEELS.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	NAME	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7 8 ... 14	15 ... 20	21 ... 62	63 ... 80	
1	L	ATOPRT	MSINS	INV,WHEELS,		
2						

The following coding will cause an item to be inserted into File X (FLX), tagged ACCREC (accounts receivable).

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	NAME	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7 8 ... 14	15 ... 20	21 ... 62	63 ... 80	
1	L	ACCREC	MSINS	FLX,		
2						

MSDEL	Deletes the last item retrieved from a Direct Access File.

FILE ORGANIZATION: Direct Access File.

PROCESSING MODES: INPUT/OUTPUT mode.

FORMAT

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	INDEX	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7 8	14 15	20 21	62 63 80
1	L		Anytag	MSDEL	File-tag,
2					
3					

DESCRIPTION:

Operands Field

File-tag: the File-tag is the 1, 2, or 3 character prefix specified in parameter Ø of the appropriate MCA.

EXAMPLE:

The following coding will cause the last item retrieved to be deleted from the Direct Access File MON.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	INDEX	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7 8	14 15	20 21	62 63 80
1	L			MSDEL	MON,
2					
3					

Writing A Macro Call

The programmer writes a macro call at the point in his program where a macro routine is to be incorporated. The Type Field contains a C when all the parameter values for a particular macro routine do not fit on one line and require continuation lines to follow; otherwise, the Type Field contains an L. The Location Field may contain a symbolic tag which, when written, is always interpreted as the value of parameter Ø. The Operation Code Field contains the name of the desired macro routine (which is also the name on the PROG line of the routine). The Operands Field contains the parameter values, written in order of parameter number, starting with the value of parameter 1.

CONTINUATION LINES

A continuation line is used where a macro call cannot contain all the parameters for a particular macro routine on a single line. Although the first line of a multiple-line call is not a continuation line, it indicates, with a C in the Type Field, that a continuation line follows. The last continuation line contains an L in the Type Field.

OMISSION OF PARAMETERS

A parameter value may contain any character except the comma. The comma is used to follow each parameter value, including the last. The comma also serves as a method of omitting a parameter value from the macro call. Each missing parameter value is indicated by its comma. However, any number of values may be omitted without their terminating commas if no further values are needed. For example, if a macro routine has 10 parameters (1 - 10) and the programmer wishes the omit values 3, 5, 6, and 8 - 10, he may code the call as follows:

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1	L		TAG	NAME	VAL1,VAL2,VAL4,,,,VAL7,
2					
3					

An alternative method of omitting parameter values is convenient for omitting several consecutive values when more values are to follow. Write the number of the next parameter not to be omitted in columns 15 and 16 of the next continuation line. Then write the actual value of this parameter in the Operands Field and continue as usual. To omit the first n values, do not write any values in the macro call line, and write the number of the first parameter whose value is not to be omitted in columns 15 and 16 of the first continuation line. For example, if a macro routine has parameters 1 - 10 and values 2 and 6 - 9 are to be omitted, the programmer may write the following:

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1	C		TAG	NAME	VAL1,VAL3,VAL4,VAL5,
2				10	VAL10,
3					

To omit values 1 - 4, 6, 7 and 10, the programmer may write the following:

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1	C		TAG	NAME	
2	C			05	VAL5,
3	L			08	VAL8,VAL9,
4					
5					

The following example summarizes the complete relationship of the macro call, the generalized macro routine, and the macro routine after it has been specialized and incorporated into the main program. The macro routine is shown first in its generalized form.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___.___ PAGE ___ OF ___

CARD NUMBER	TYPE	TAG	LOCATION	OPERATION CODE	OPERANDS
1				PROG	PROB
2				SETP	@
3			@@@	SCR	@@5ZEX+3,7@
4			@@5ZAD	A	@@1+X@@2,(@@4)
5				C	(@@4),@@5ZIM
6				BCT	@@5ZCN,45
7			@@5ZEX	B	@@
8			@@5ZCN	MCW	(@@4),@@3+15
9					
10					
11					
12					
13			@@5ZIM	DCW	@@6
14				END	
15					
16					

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___.___ PAGE ___ OF ___

CARD NUMBER	TYPE	TAG	LOCATION	OPERATION CODE	OPERANDS	
1			MAIN	PROGRAM		
2						
3						
4						
5						
6		L	TAGER	PROB	AUG,4,PRINT,SUM,M1,+1@@	Macro Call.
7			TAGER	SCR	M1ZEX+3,7@	Specialized version of the same routine. This is inserted directly after the macro call in the output produced.
8			M1ZAD	A	AUG+X4,(SUM)	
9				C	(SUM),M1ZIM	
10				BCT	M1ZCN,45	
11			M1ZEX	B	@@	
12			M1ZEX	MCW	(SUM),PRINT+15	
13						
14						
15						
16						
17						
18			M1ZLM	DCW	+1@@	
19						
20			MAIN	PROGRAM	(until next call.)	
21						
22						

Writing A Macro Routine

Some routines are Honeywell supplied (e.g. Input/Output Control routines) while others may be written by the user. This section explains how the user should write a generalized macro routine for inclusion in the Library File.

PARAMETER DESIGNATORS

Parameter designators have the form pxy, where p is any alphanumeric character chosen by the programmer, and x and y form the decimal parameter number from ØØ to 63. Although there is no restriction on the characters that are assigned to p, it is the responsibility of the programmer to ensure that the resulting parameter designators do not duplicate the form of any other language element, such as a symbolic tag. A control instruction, Set Parameter Designator (SETP), is used to assign a value to p. SETP is written in the Operation Code Field, and the desired value for p is written in column 21. The value of p may be changed at any time by writing another SETP instruction. If the SETP instruction is not used, the value of p is assumed to be octal 35, which prints as %. The following example illustrates possible assignment of p.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7 8	14 15	20 21	62 63 80
1				SETP	@
2				SETP	*
3					

Parameters are indicated by writing the currently assigned value of p, followed by a parameter number (xy) which the programmer assigns consecutively. When the routine is specialized, the parameter designator is replaced by the explicit value supplied in the macro call. For example, assume that parameter Ø3 is an index register number. An

indexed address using that register with an augment of 1 would appear within the macro routine as 1+XpØ3. When the routine is specialized, the parameter value (e.g. 5) replaces the designator pØ3, creating the address 1+X5. Parameter ØØ is always used to indicate the tag, if any, written in the Location Field of the macro call.

SELECTIVE OMISSION OF CODING

The programmer may desire that certain lines of coding be omitted from the macro routine. The zone portion of y in the parameter designator may be overpunched with R (+) or with X (-). An R (+) overpunch indicates that if the value for parameter xy is blank or omitted in the macro call, this line of coding is omitted from the routine. An X (-) overpunch indicates that if the parameter value is included in the macro call, this line of coding is omitted from the routine.

Suppose, for example, that a macro routine computes a hash total of a particular field on an optional basis. The field to be totaled is parameter Ø1, and the field to contain the total is parameter Ø2. The instruction in the routine to update the total would be coded as follows:

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MAR	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				A	PØ1,PØB	
2						

Note that B is the result of overpunching 2 with R (+).

Conditional Statements

Conditional (COND) control statements may also be used to omit lines of coding from the macro routine. The conditional statement may have the following formats:

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1				COND	nnnnn,pxy,v,c
2				OR	
3				COND	fffffff,pxy,v,c
4					
5					

nnnnn - card number of the next statement not to be omitted when the condition is true.

fffffff - the Location Field of the next statement not to be omitted when the condition is true.

The condition is coded as follows:

C	Condition
ø	Never true.
1	True if value of pxy > v.
2	True if value of pxy = v.
3	True if value of pxy ≥ v.
4	True if value of pxy < v.
5	True if value of pxy ≠ v.
6	True if value of pxy ≤ v.
7	Always true.

The condition is tested by a binary compare of the value of parameter xy (A address) against v (B address), followed by a branch on condition test with a variant character of 4c (where c is interpreted as shown in the preceding list). If c is true, all statements of the macro routine (including additional COND and SETP statements, if any) from this point up to but not including, the designated card number or Location Field are omitted. All rules of the compare instruction apply to the condition function. A void parameter produces an equal result when compared to a field with up to 40 blanks.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1				COND	ØØØ14,PØ2,KRAM,1
2					
3					

When the conditional control statement in the preceding example is processed, the value assigned to parameter Ø2 is compared with the literal value KRAM. If the parameter value is greater than KRAM (since c = 1), the following statements are omitted from the routine, up to statement ØØØ14, which is included in the routine. If the parameter value is less than or equal to KRAM, no statements are omitted from the routine at this point.

Had the conditional statement been coded as follows, statements up to but not including the statement whose Location Field is JUMPΔΔΔ are omitted when parameter Ø2 is greater than KRAM.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___.___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62 / 63 ... 80
1				COND	JUMPΔΔΔ,PØ2,KRAM,1
2					

TAG PREFIXES

Duplication of tags between the calling program and any of the macro routines called for (or between two of the macro routines called) must be avoided. To avoid duplication, it is recommended that each macro routine be assigned a particular prefix and that each of its tags be preceded by this prefix. The length of the tag and its prefix must never exceed 6 characters. If the same macro routine is to be called more than once by a single main program, the tag prefix should be designated by means of a parameter to avoid duplication. Thus, the tag prefix will be different for each insertion of the routine.

ADDING A MACRO ROUTINE TO THE LIBRARY FILE

A generalized macro routine is prepared in the same manner as any other program, except for the presence of parameter designators

to indicate the massing values. It is submitted to the Library File Update program complete with its own PROG and END statements, for addition to the Library File. The Library File Update is described in detail in Section 4 of this manual.

I/O CONTROL PROGRAMMER'S PREPARATION INFORMATION

Program Organization

The routines making up the I/O Control facilities are designed to take a minimum amount of memory locations in any given situation. This is accomplished first by generating only the required coding for processing a given program's files, and secondly by segmenting the coding for those functions that are required on an infrequent basis during program execution. Thus, while the coding to open or close a file is required in any program, this coding is loaded into memory only when the programmer issues an action call for one of these functions. A multi-phase program further reduces the I/O memory requirements by specializing separate MIOC macros with different processing capabilities for each phase. In multi-phase programs, tag uniqueness is insured because a unique character for all tags of each MIOC can be specified by the user. The unique tag capability allows any other macro in the operating system to be specialized into the same program. Each MIOC called into a given program must originate at the same memory location. This is the only restriction when multi-phase programs are being executed.

MIOC SEGMENTATION

In certain cases, the user may wish to have MIOC assembled into his program as a single segment. In most case, however, the user will take advantage of the option to segment seldom used functions. This is accomplished by assigning any letter from A to Z as parameter 1Ø of the MIOC macro call.

When segmentation is desired, the program using the I/O Control facilities must specify segment names to assembly. Then, during assembly of the segment that contains the MIOC macro call, the I/O takes control of assembly segmentation until all the coding for the requested resident and non-resident functions has been generated. The coding for the resident functions is generated in the same segment of the program that contains the MIOC macro call. The coding for each non-resident function requested is generated in an individual segment. Of this non-resident coding, the first segment is xA, where x is equal to the letter assigned as parameter 1Ø of the MIOC macro call. The second segment is xB, the third xC and so forth until all the non-resident function coding is generated. The last segment generated, always xZ, consists of any user coding that followed the call for MIOC in the segment that contained MIOC. Segment xZ will appear regardless of whether or not user coding followed the MIOC macro call in its respective segment. This means that if the segment containing the MIOC macro call contains coding after the MIOC call, this coding will be assembled in a segment different than the original.

When the call to the Supervisor to load the segment containing the MIOC macro call is made in the Normal Start mode, loading proceeds to the end of the resident MIOC coding. At the end of the resident MIOC coding, MIOC will generate an Execute Statement at assembly time. This statement causes the Supervisor to load the last MIOC segment, xZ, without altering any Supervisor Communications Area fields other than the Segment Name field. When the Supervisor completes this loading, control is returned to the location specified in the user's Execute or END Statement in the segment containing the MIOC macro call. In this case, the user <u>cannot</u> assume that his original segment name (i.e., the name of the segment containing the MIOC call) will be preserved in the Supervisor Communications Area.

When the call to Supervisor to load the segment containing the MIOC macro call is made in the Return or Special Start mode, coding following the MIOC call is not loaded. When coding does follow the MIOC call in the segment containing the MIOC call, it is the users responsiblity to load that coding. This is accomplished by a request to load Segment xZ.

For a description of the Supervisor's Normal, Return and Special Start modes, see Section 2 of this manual.

Figure 3-6 illustrates the principles of program segment loading by the Supervisor. In the Normal Start mode, Segment Ø1 would be loaded, followed by Segment BZ. In the Special or Return Start mode, only Segment Ø1 would be loaded. Note that B is assumed to be the value assigned to parameter 1Ø of MIOC and that the user supplied segment containing the MIOC macro call is defined as Segment Ø1.

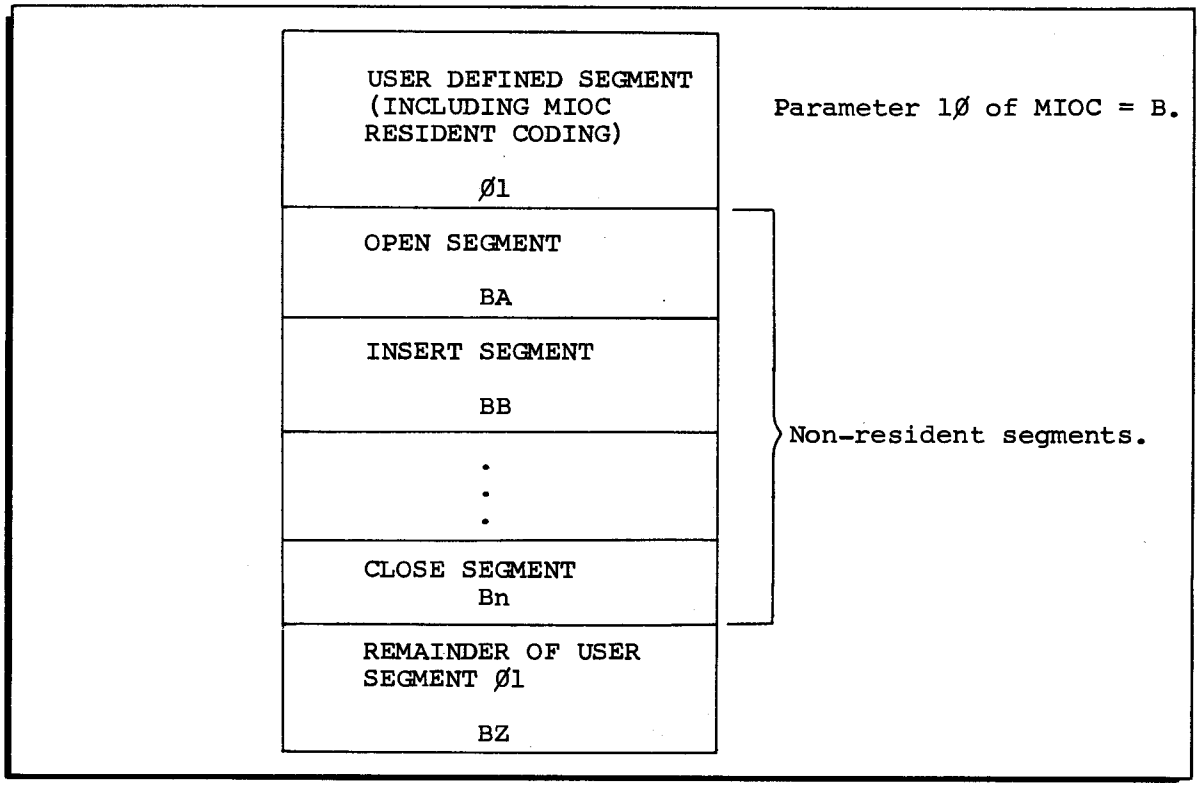


Figure 3-6. Program Segment Loading

SUPERVISOR RESTRICTIONS

To accomplish segment loading, MIOC must utilize certain fields of the Supervisor Communications Area and make certain assumptions about other fields.

The following fields of the Supervisor Communications Area are altered during the loading of non-resident functions. These fields are restored to their original values, however, as soon as a particular loading sequence is completed.

1. Locations 74 and 75 (decimal) are altered to contain the segment name of the currently needed segment.

2. Location 1Ø6 (decimal) is altered to ensure that searching for non-resident functions is in the most efficient direction. This is done to ensure compatibility with the Series 200 Operating System - Mod 1 (Tape Resident).

3. Location 112 (decimal) is altered to Return Start mode.

4. When the program's search mode includes visibility, the I/O will always search by program and segment name and visibility. When visibility is not included, the I/O will always search program and segment name. This is accomplished by preserving the left-most bit of Location 111 (decimal) and altering the five right-hand bits to indicate $2Ø_8$.

Any time a non-resident function is requested, it is assumed by the I/O that Locations 68 through 73 (decimal) of the Supervisor Communications Area contains the program name that contains the current MIOC macro call.

CARD LOADING AND SEGMENTATION

When programs are being loaded from cards, those programs utiliz-

ing segmentation must observe certain segmentation limitations. To facilitate segmented program loading from cards, non-resident functions are placed on binary run files in the following sequence:

1. MSOPEN (When parameter 15 of MIOC is set to COMBINE, the MSOPEN coding is one segment, otherwise, it is as many segments as are necessary to achieve maximum memory usage.)

2. SETM (When parameter 12 of MIOC is set to COMBINE, SETM and ENDM coding becomes one segment.)

3. MSINS (When parameter 11 of MIOC is set to SEGMENT.)

4. ENDM (When parameter 12 of MIOC is Δ .)

5. MALTER

6. MSREL

7. MSCLOS

Note that the segment containing MIOC (or segments containing MIOCs in a multi-phase program) must be loaded by the Card Loader Monitor in the Return or Special Start mode. This is because, in the Normal Start mode, MIOC searches through the non-resident coding segments for segment xZ, the last MIOC segment.

Also note that in the segment containing the MIOC macro call, coding cannot be written after this call.

To summarize, when loading segmented programs from cards, each non-resident function can be called into the common overlay area only once. Therefore, all action macros utilizing that function must be executed before another non-resident segment is requested. Because of this, functions must be requested in the order listed previously.

MIOC - PHYSICAL I/O RELATIONSHIPS

MIOC does not issue PDT or PCB instructions. MIOC does, however, interface with the mass storage Physical I/O (described in Appendix B of this manual) which does issue such instructions. Normally, the user requests that MIOC call and utilize Physical I/O Control (MPIOC). This request is made through parameters 5Ø through 55 of the MIOC macro call. In some cases, however, the user may want to call MPIOC himself. This is done by assigning the value PRESENT to parameter 5Ø of the MIOC macro call.

MCA - PHYSICAL I/O RELATIONSHIPS

The user is required to have one MCA for every file he intends to process in a given program. Each MCA macro automatically generates a Physical I/O Communication Area (MPCA). The user may desire to interrogate some of the fields in the MPCA and does this by writing a MUCA macro call (described in Appendix C of this manual). Because the MCA macro uses the MPCA exclusively, the user should never attempt to alter the contents of any of its fields.

Read/Write Channel Utilization

There are two data transfer rates applicable to the mass storage devices. Data transfer rates for the Type 258 and 259 Devices can only be accomplished by interlocking one and one-half channels (such as 1 A and 3 or 4A and 6). For the Type 259A Device, any single interlocked channel is sufficient. When parameter 55 of MIOC is not specified, channels 2 and 3 or 5 and 6 (depending on the I/O sector of the control unit specified - or implied - in parameter 52 of MIOC) is used for mass storage operations. Current location counters in control memory are not referenced.

When the user needs a slower data transfer rate, he must specify a different read/write channel combination to MIOC, and, when applicable,

to MPIOC. This is done through parameter 55 of MIOC and parameter Ø5 of MPIOC.

Address Mode

The address mode for all Logical I/O macros must be the same. Also, each time the user enters the I/O through a macro call or the I/O returns to the user (normally or through an exit) from a macro routine, the address mode must be the same as that of the macro call.

Index Registers

MIOC, together with Physical I/O, uses and restores index registers X3, X4, X5 and X6. These registers are restored to their initial values whenever a return from the I/O is made to the users coding. It does not matter whether the coding is in the main line of the program or in an exit routine. Index registers X3 and X4 are restored at the last possible moment before the return is made. Hence, they should not be used as a linkage parameter to MCA. Index registers X5 and X6, however, may be used as linkage parameters, since they are restored earlier.

Index resisters are saved and restored with MCW's. The MCW's are from or to each respective register to or from DSA fields in MIOC. The length of the DSA fields is consistent with the current addressing mode. MIOC sets its own index register values with the LCA instructions. Because of this, the user should always punctuate the registers in the normal manner. Namely, word marks should be placed in locations 1Ø, 14, 18 and 22 in the three character addressing mode and at locations 9, 13, 17 and 21 in the four character addressing mode. The permanence of any other punctuation is not guaranteed.

Direct Access Addressing

Direct Access bucket addresses can be relative or actual. A relative bucket address is one in which the bucket's address is the

same as its numeric position from the beginning of the file. In this case, the first bucket in the file is number ØØØ and each bucket following increments this number by one. An actual bucket address is one that is the exact mass storage address of the beginning of the bucket. When a bucket address is not included in an Action Macro call, the address of the bucket which the last Action Macro call used is used again.

The user must generate a field in which bucket addresses are stored. Bucket addresses then are delivered to the I/O from this field, whose right-most location is specified by parameter Ø2 of the Action Macro call. This field can have either of the following octal formats.

1. Relative Address Field. This field must have four character positions and the left-most of these must be word marked. This field will contain the exact sequence number of the bucket within the file. The sequence number of the bucket will be in binary.

2. Actual Address Field. This field must have eight character positions and the left-most of these must be word marked. This field will contain the address of the first record in the desired bucket. The record address is in the form DMCCTTRR; where D = device number, M = magazine number ($ØØ_8$), CC = cylinder number, TT = track number and RR = record number.

Direct Access Item Key Specification

For a GET macro to retrieve an item in direct access processing, the item must contain an identifying key. This key is specified by the user. The length and location within the item are specified when the Direct Access file is allocated. This information is placed in the file description portion, *VOLDESCR*, of the Volume Directory. When a GET action is issued, the I/O retrieves these fields from *VOLDESCR*.

The address of the right-most location of a field that contains the desired key value is specified by parameter Ø3 of the GET Action Macro. When items are to be retrieved by searching for the correct item key, parameter Ø3 of the GET Action Macro must be specified. The field that contains the key value is set up by the user and must contain a word mark in its left-most location. The corresponding field within the item need not contain a word mark, but, if desired, the left-most character of the item key field may contain a word mark. The word mark set up by the user in the key value field terminates the operation when the key value field and the item key field are compared.

Exits And Halts

There are five exits associated with MCA. Each exit pertains to a specific area of I/O processing. These exits are specified in parameters 4Ø through 44 of MCA (see Tables 3-8 through 3-12) and are as follows:

1. Parameter 4Ø - Volume Directory Exit (see Table 3-8).

2. Parameter 41 - Index Exit (see Table 3-9).

3. Parameter 42 - Every Index Entry Exit (see Table 3-10).

4. Parameter 43 - Data Exit (see Table 3-11).

5. Parameter 44 - Device Exit (see Table 3-12).

The exit codes associated with each type of exit identify the reason for the I/O taking the exit. These are contained in individual lists following this description. Whenever any of the MCA parameters 4Ø through 44 are specified, the user must provide coding by which he can interrogate the exit code. The coding provided by the user normally is tagged with the name of the exit type, i.e., if parameter

43 is specified the coding would be tagged DATEX for Data Exit. This coding must be preceded by a DCW in the location immediately before it. The exit code is moved into this DCW whenever an exit for the specified parameter is taken. Also the return code must be moved into this DCW when the user has completed the interrogation.

For example, suppose that a user wants to specify a Device Exit (parameter 44 of MCA) only to re-attempt to correct read and write errors. The exit code for the read error is Ø6 and for the write error 1Ø (this is an unsuccessful write verify). The user can specify one of three return codes to the I/O. A 21 return code means that the I/O is to automatically re-attempt to correct the error. A 52 return code means that the I/O is to ignore the error and continue processing if possible and a 7Ø return code means halt. The following coding illustrates this example.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	CONTINUATION MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
	*		CALL TO MCA			
	C		FL1	MCA	PØ1,PØ2,PØ3,...	
	L			44	DEXIT,	
	*		USER EXIT ROUTINE			
	*		WHEN THIS ROUTINE IS ENTERED, THE FOLLOWING DCW WILL CONTAIN THE			
	*		EXIT CODE.			
	*		WHEN RETURN TO THE I/O IS MADE, THE SAME DCW WILL CONTAIN A CODE			
	*		SPECIFYING THE USER'S DESIRED ACTION.			
			DCI	DCW	#10B,	
			DEXIT	SCR	MYRET,7Ø,	SAVE RETURN
				BCE	RDER,DCI,Ø6	READ ERROR
				BCE	WTER,DCI,1Ø	WRITE ERROR
				MCW	#1C7Ø,DCI	HAVE I/O HALT
			MYRET	B	0	RETURN TO I/O
			RDER	EQU	*	
			WTER	MCW	#1C21,DCI	REQUEST RE-ATTEMPT
				B	MYRET-LA	LA IS THE LENGTH OF AN ADDRESS
	*			NOP		

Table 3-8. MCA Parameter 4Ø - Volume Directory Exit

EXIT CODE	REASON FOR EXIT	RETURN CODE	RETURN CODE MEANING
Ø1	The Volume Directory description, *VOLDESCR*, for the file has been read into memory and can now be interrogated by the use of a MUCA macro. An ADP points to the left end of the entry. For a description of ADP and the MUCA macro, see Appendix C of this manual.	1Ø	Continue processing.
		2Ø	Re-open the file.
11	At the end of file processing - after MSCLOS reads *VOLDESCR* in memory and before writing it back to mass storage, *VOLDESCR* can be interrogated by the use of a MUCA macro. An ADP points to the end of the entry. For a description of the MUCA macro and an ADP, see Appendix C of this manual.	1Ø	Continue processing.
Ø3	The file name specified in the MSOPEN macro cannot be located in *VOLDESCR*.	4Ø	Halt.
		21	Re-open the file.
Ø4	The units of allocation table set up by the user is not large enough to hold all the units of allocation for this file.	4Ø	Halt.
		21	Re-open the file.
14	When this file was allocated, a password was placed in the Volume Directory. The password in this MCA is present but not correct.	4Ø	Halt.
		21	Re-open the file.
24	When this file was allocated, a password was placed in the Volume Directory and there is no password in this MCA.	4Ø	Halt.
		21	Re-open the file
Ø5	A discrepancy exists between the units of allocation specified in the MCA and that recorded in *VOLALLOC* for this file.	4Ø	Halt.
		21	Re-open the file.

Table 3-9. MCA Parameter 41 - Index Exit

EXIT CODE	REASON FOR EXIT	RETURN CODE	RETURN CODE MEANING
Ø3	The SETM macro cannot locate the specified member in the Member Index	4Ø	Halt.
		△	Issue a new action to continue processing.
13	The MALTER macro cannot locate the specified member in this file.	4Ø	Halt.
		△	Issue a new action to continue processing.
Ø4	The SETM macro has been requested to create a new member but there is no room in the Member Index for another entry.	4Ø	Halt.
		△	Issue a new action to continue processing.
14	The SETM macro has been requested to set the processing mode of an existing member to the Output Only mode but the status of that member makes it unavailable for Output Only processing.	4Ø	Halt.
		△	Issue a new action to continue processing.
24	The MALTER macro has been requested to delete a member whose status makes it unavailable for Output Only processing.	4Ø	Halt.
		△	Issue a new action to continue processing.

Table 3-10. MCA Parameter 42 - Every Index Entry Exit

EXIT CODE	REASON FOR EXIT	RETURN CODE	RETURN CODE MEANING
Ø1	The SETM action is the current function and a member index entry is available for interrogation by using a MUCA macro. An ADP points to the left end of the entry. For a description of the MUCA macro and an ADP, see Appendix C of this manual.	3Ø	The SETM macro will control the interrogation and either accept or reject this entry.
		11	This member associated with this entry should not be processed and another entry should be delivered.
		52	The member associated with this entry should be processed.

Table 3-11. MCA Parameter 42 - Data Exit[1]

EXIT CODE	REASON FOR EXIT	RETURN CODE	RETURN CODE MEANING
Ø1	The GET macro has been issued and the end-of-data item has been detected.	4Ø	Halt.
		△	Issue a new action to continue processing.
11	The PUT macro has been issued and there is no more room for more data in the file.	4Ø	Halt.
		△	Issue a new action to continue processing.
34	The SETM macro has been requested to create a new member and there is no more room in the file for a new member.	4Ø	Halt.
		△	Issue a new action to continue processing.
Ø3	The GET macro cannot locate the specified direct access item key.	4Ø	Halt.
		△	Issue a new action to continue processing.
13	The INSERT macro cannot locate an available item position in the Direct Access file.	4Ø	Halt
		△	Issue a new action to continue processing.
Ø4	An invalid bucket address has been specified to the current direct access function.	4Ø	Halt.
		△	Issue a new action to continue processing.

NOTE: 1. This exit must be specified whenever the I/O will never reach an end-of-data situation.

Table 3-12. MCA Parameters 44 - Device Exit[1]

EXIT CODE	REASON FOR EXIT	RETURN CODE	RETURN CODE MEANING
Ø1	Device inoperable		
Ø2	Protection violation.		
Ø3	Device error (after five attempts to position the device).		
Ø4	Formatting error.		
Ø5	The addressed record cannot be located (after five attempts).		
Ø6	Uncorrectable read error. The data, however, has been transferred after 1Ø attempts.		
Ø7	Uncorrectable read error. The data, however, has not been transferred after 1Ø attempts. (The Header may contain a read error.)		
1Ø	Uncorrectable write error. The last write could not be verified after 1Ø attempts.		
11	A track linking record has been read into memory.		
12	The attempt to track link to the next track in this file has not been completed after 1Ø tries.		
		21	Re-attempt the operation that caused this error.
		52	Ignore the error and continue processing if possible.
		7Ø	Halt.

NOTE: 1. When one of these exits is taken, a device error exists. Any Return Codes listed are applicable to all device error exits.

FILE SUPPORT

File Support consists of a set of routines to perform frequently desired functions on files stored on mass storage. These functions are allocate/deallocate, load/unload, and map. The allocate function is used to assign areas of the volume to files as requested by the user and to update the Volume Directory accordingly. This routine also causes formatting and initialization of newly allocated files. The deallocate function removes from the Volume Directory all entries for a named file. This makes available for future allocation all areas used by this file. The load function is used to load files from cards, tape, or another mass storage volume. The unload function unloads files from a mass storage volume onto cards, tape, printer, or another mass storage volume. The map function is used to obtain a printed listing of the contents of the Volume Directory.

All the File Support routines specialize themselves at execution time based on parameters supplied by the user in the job control statements. It is not necessary to perform an assembly operation to specialize these routines to perform operations on a particular file.

Because of the structure of the system, a single File Support run can proceed from file allocation through file loading without operator intervention. This single run may perform operations on many files. File support also includes the capability of execution time inclusion of user's own coding routines for such purposes as randomizing of keys and end-of-file exits. These routines must reside in the same file as the file support routines; namely, the System File.

For one or more file support functions to be executed, job control statements must be input from the card reader for each desired function. These statements must be punched according to the general format of the job control statements described later in

this section. The user may request one function or many functions within one execution of File Support. Within any function one operation or many operations may be performed. For example, a single execution of File Support might include the following functions; Deallocate, Allocate, Load, and Map. Within the Deallocate function there might be four File Statements indicating four files to be deleted. Functions will be performed in the order of the requests. Thus a request for Allocation of a given file should always precede a request to load that file. Also, a request to load File-A, File-B, and File-C will load them precisely in that order. There must be one indication of end of job control statements per execution of File Support. This indication must be either on or following the last job control statement of the last File Support function requested. End of job control statements are indicated by an E in column 7.

Allocate Function

DESCRIPTION

The allocate function is used to assign space to a file on mass storage. Every file on mass storage must be allocated before it can be loaded. The allocate function checks the areas specified for the file, to ensure that no other file occupies any of the area, and updates the Volume Directory to reflect the new file.

The user must supply the name of the file, the file organization, and the units of allocation for the file. A maximum of six units of allocation per file per volume can be specified.

The allocate function is requested by a Function Statement whose first parameter is ALLOCATE. This statement may be followed by FILE, SIZE, UNITS and MEMBER Statements. These can appear in any order after

the Function Statement. More than one file may be allocated by a single Allocate Function Statement. This Function Statement is followed by as many sets of File, Size, Units, and Member Statements as there are files to be allocated.

ALLOCATE FUNCTION JOB CONTROL STATEMENT

Format

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	INAR	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7 8		14 15 20	21 62	63 80
				FUNCT	ALLOCATE,	
				FILE	NAME=file-name,	
					ORG={SEQ / PART / DIR},	
					GENOV={NO / YES},	Optional
					KEY=position,length,	Optional
					PW=password,	Optional
					EXP=yyddd,	Optional
					PROT={A / B / AB / NO},	Optional
				SIZE	ITEM=item-length,	The Size State-
					REC=record-length,	ment is optional.
					BLOCK=items-per-block,	
					INDEX=blocks-in-index,	
					CYLOV=number-of-tracks,	
				UNITS	NAME=volume-name,	
					DEVADD=(pcu,drive),	Optional
					FROM=c,t,	
					TO=c,t,	
					FROM=c,t,	Optional
					TO=c,t,	Optional
				MEMBER	NAME=member-name,	
					LENGTH=number-of-blocks,	

Description

FUNCTION STATEMENT: The Function Statement contains the Operation Code FUNCT and the Operand ALLOCATE which specifies to the system what function to perform.

FILE STATEMENT: The file to be allocated is identified by the File Statement, whose first parameter is NAME. The File Statement is required.

Name Parameter: The Name Parameter (NAME) gives the name of the file to be allocated. This is a required parameter and can be up to 1Ø characters long. When it is less than 1Ø characters long, trailing spaces are automatically added. File and member names cannot begin with a character whose octal value is 2Ø (+), 4Ø (-), 6Ø (△ or △), or 77 (△ or ¢). All numeric values in the parameters, following the keywords, are in decimal format.

Organization Parameter: The Organization Parameter (ORG) specifies the organization of the file to be allocated as sequential (SEQ), Partitioned Sequential (PART), or as Direct Access (DIR). This is a required parameter.

General Overflow Parameter: The General Overflow Parameter (GENOV) applies only to Direct Access Files. It specifies whether the file is to contain a general overflow area or not. The assumed condition, when this parameter is not specified, is that there will be a general overflow area.

Key Parameter: The Key Parameter (KEY) is required for Direct Access Files and cannot be used with Sequential Files. This parameter specifies both the position and length of the Key Field. The position part of the parameter indicates the position in each item of the high order end of the Key Field. The first character of the item is character ØØØ1. The length part of the parameter indicates the length in characters of the Key Field.

Password Parameter: The Password Parameter (PW) specifies the password to be placed in the Volume Directory entry for the file. The password can be up to 8 characters long. However, when it is less than 8 characters, trailing spaces automatically are added. When the password parameter is omitted, (it is not a required parameter) the password field in the Volume Directory entry is set to spaces and no password checking is performed.

Expired Parameter: The Expired Parameter (EXP) specifies the year and day that the file being allocated expires. The yy portion of the parameter gives the tens and units digits of the year of expiration. The ddd portion of the parameter gives the day of expiration. The day of expiration is determined by counting from January 1 as day ØØ1. When this parameter is not specified, the assumed value is ØØØØØ.

Protection Parameter: The Protection Parameter (PROT) gives the type of write protection to be assigned to the file. The significance of the values of the parameter is as follows:

A	The file is to be assigned A-File write protection.
B	The file is to be assigned B-file write protection.
AB	The file is to be assigned both A- and B-File write protection.
NO	No write protection is to be assigned to the file.

For a discussion of the types of write protection available in the operating system, refer to Appendix F. When this parameter is not specified no write protection will be assigned to the file.

SIZE STATEMENT: The Size Statement specifies parameters related to the size of the various units of the file. When the Size Statement is omitted (it is not a required statement), all its parameters are

assigned their standard values. The standard values are as follows:

Item	25Ø characters
Record	25Ø characters
Block	When item size is greater than the record size, 1 item/ block. When item size is less than the record size, X items/ block (where X = as many items as in a record).

Item Parameter: The Item Parameter (ITEM) specifies the number of characters in each item. When not specified, the standard size will be 25Ø characters. When allocating Direct Access Files, the status character must be included in this parameter.

Record Parameter: The Record Parameter (REC) specifies the number of characters in each record. When not specified, the standard size will be 25Ø characters per record.

Block Parameter: The Block Parameter (BLOCK) specifies the number of items per block. When not specified, the standard number will be 1 item per block.

Bucket Parameter: The Bucket Parameter (BUCKET) applies only to Direct Access Files and specifies the number of blocks per bucket. When notspecified, the assumed value of 1 block per bucket is used.

Index Parameter: The Index Parameter (INDEX) specifies the number of blocks in the Member Index for a Partitioned Sequential File. When not specified, the assumed value is 1 block for the Member Index.

Cylinder Overflow Parameter: The Cylinder Overflow Parameter (CYLOV) specifies the number of tracks in the cylinder overflow area for a Direct Access File. When not specified, the allocate function does not access any cylinder overflow area.

UNITS STATEMENT: The Units Statement specifies the units of allocation for the file. There must be only one Units Statement and this statement must include at least one pair of From and To parameters.

Name Parameter: The Name Parameter (NAME) specifies the volume serial number of the volume to be used for this set of units of allocation. The name is 6 characters long. This parameter is not required.

Device Address Parameter: The Device Address Parameter (DEVADD) specifies the peripheral control unit number and the drive number of the device containing the volume for this Units Statement. The peripheral control unit number is written as two octal digits and all bits except the I/O bit must be specified. The drive number is written as one octal digit. When this parameter is not specified, the assumed values of Ø4 for the peripheral control unit and Ø for the drive number are used.

From Parameter: The From Parameter (FROM) gives the low cylinder and track address of a single unit of allocation. This must be followed immediately by a To Parameter which specifies the high cylinder and track address of the same unit of allocation. The cylinder address of the From Parameter must be less than or equal to the cylinder address of the corresponding To Parameter. The same is true for the track address.

To Parameter: The To Parameter (TO) specifies the high cylinder and track address of a single unit of allocation. It is paired with the immediately preceding From Parameter. Note that if a file consists of more than one unit of allocation, the number of tracks assigned per cylinder must be constant for all units of allocation.

MEMBER STATEMENT: The Member Statement enables the user to reserve space for members of a Partitioned Sequential File. This statement is required only when it is desired to reserve space for a specified member by the allocation process. There must be one Member Statement for each member which is being reserved. The parameters of the Member Statement are described in the following paragraphs.

Name Parameter: The Name Parameter (NAME) gives the name of the member, which can be up to 14 characters long.

Length Parameter: The Length Parameter (LENGTH) specifies the number of blocks to be reserved for this member. File and member names cannot begin with a character whose octal value is 2Ø (+), 4Ø (-), 6Ø (Δ or Δ), 77 (Δ or ¢). All numeric values in the parameters, following the keywords, are in decimal format.

ALLOCATE FUNCTION JOB CONTROL LANGUAGE EXAMPLE

The following job control statements request the allocation of a Sequential File named FILE-A. This file has an item length of 1ØØ characters and is to have 5 items per block. Two units of allocation are requested, the first from cylinder 5 track Ø to cylinder 9 track 9; and the second from cylinder 2Ø track Ø to cylinder 24 track 9. The standard assumptions are no password checking, no expiration date checking, the record size is 25Ø characters, the device address is pcu Ø4 drive Ø, and no write protection.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	INSTR	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7 8	14	15 20	21 62	63 80
1				FUNCT	ALLOCATE,	
2				FILE	NAME=FILE-A,ORG=SEQ,	
3				SIZE	ITEM=1ØØ,	
4					BLOCK=5,	
5				UNITS	FROM=(5,Ø),TO=(9,9),	
6		E			FROM=(2Ø,Ø),TO=(24,9),	

Table 3-13 contains a summary of the allocate function job control statements.

Table 3-13. Allocate Function Job Control Statements

JOB CONTROL STATEMENT	PARAMETER	PARAMETER VALUE	DESCRIPTION	REQUIREMENTS
FUNCTION STATEMENT	ALLOCATE	ALLOCATE	Specified the file support function to be performed.	Required.
FILE STATEMENT	NAME	File-name	Names the file to be allocated.	Required.
	ORG	SEQ	Specifies the organization of the file to be allocated as sequential.	Required.
		PART	Specifies the organization of the file to be allocated as partitioned sequential.	
		DIR	Specifies the organization of the file to be allocated as direct access sequential.	
	GENOV	NO	The direct access file being allocated does not require a general overflow area.	Optional. Note that this parameter only applies to direct access files.
		YES	The direct access file being allocated requires a general overflow area.	
	KEY	Position	Indicates the position in the item of the high order end of the key field.	Optional. Note that this parameter only applies to direct access files.
		Length	Indicates the length in character of the key field.	
	PW	Password	Specifies the password to be placed in the volume directory entry for this file.	Optional
	EXP	yyddd	Specifies the year and day the file being allocated expires.	Optional

Table 3-13 (cont.) Allocate Function Job Control Statements

JOB CONTROL STATEMENT	PARAMETER	PARAMETER VALUE	DESCRIPTION	REQUIREMENTS
FILE STATEMENT	PROT	A	The file being allocated is to be assigned A-file write protection.	Optional.
		B	The file being allocated is to be assigned B-file write protection.	
		AB	The file being allocated is to be assigned both A- and B-file write protection.	
		NO	No Write protection is to be assigned to the file being allocated.	
SIZE STATEMENT	ITEM	Item-length	Specifies the number of characters in each item. When not specified each item will have 250 characters.	The size statement is optional. Note, however, that when allocating direct access files and cylinder overflow is desired. The size statement with the CYLOV parameter must be included.
	REC	Record-length	Specifies the number of characters in each record. When not specified each record will have 25Ø characters.	
	BLOCK	Items/ Blocks	Specifies the number of items in each block. When not specified each block will contain one item.	
	INDEX	Blocks/ Index	Specifies the number of blocks set aside for the member index of a partitioned sequential file. When not specifies one block will be allocated to the member index.	
	CYLOV	Number-of-tracks	Specifies the number of tracks in the cylinder overflow area for a direct access file. When not specified no cylinder overflow area is generated.	

Table 3-13 (cont.) Allocate Function Job Control Statements

JOB CONTROL STATEMENT	PARAMETER	PARAMETER VALUE	DESCRIPTION	REQUIREMENTS
UNITS STATEMENT	NAME	Volume-name	Specifies the volume serial number.	Required.
	DEVADD	PCU	Specifies the PCU number in two octal digits.	Optional.
		Drive	Specified the drive number in one octal digit.	
	FROM	c	Specifies the low cylinder address of the unit of allocation.	Required.
		t	Specifies the low track address of the unit of allocation.	
	TO	c	Specifies the high cylinder address of the unit of allocation.	Required.
		t	Specifies the high track address of the unit of allocation.	
MEMBER STATEMENT	NAME	Member-name	Specifies the name of the member for which space is being allocated.	One per member being allocated is required for partitioned sequential files.
	LENGTH	Number-of-blocks	Specifies the number of blocks to be reserved for this member.	

Deallocate Function

DESCRIPTION

The Deallocate Function is used to delete files from mass storage. File deallocation is the only means by which space may be freed for other files. Before a file is deallocated, checks are made on the file's expiration date and on its password to ensure that a protected file is not removed inadvertently.

DEALLOCATE FUNCTION JOB CONTROL STATEMENT

Format

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1				FUNCT.	DEALLOCATE,	
2				VOLUME	NAME=volume-name,	Volume statement
3					DEVADD=(pcu,drive),	is optional.
4				FILE	NAME=file-name,	
5					FXP={NO},	Optional
6					{YES}	
7					PW=password,	Optional
8		E		DAY	yyddd,	Optional
9						
10						

Description

The Deallocate Function is requested by a Function Statement whose first parameter is DEALLOCATE. This statement may be followed by a Volume Statement and must be followed by at least one File Statement. The File and Volume Statements can appear in any order.

FUNCTION STATEMENT: The Function Statement contains the Operation Code FUNCT and the Operand DEALLOCATE, which specifies to the system what function to perform.

VOLUME STATEMENT: The Volume Statement specifies parameters pertaining to the volume containing the file to be deallocated. There may be only one Volume Statement per Deallocate Function. This statement is not required and when not specified the parameters assume the standard values.

The parameters and standard values for the Volume Statement are described in the following paragraphs.

Name Parameter: The Name Parameter (NAME) specifies the serial number of the volume containing the file to be deallocated.

Device Address Parameter: The Device Address Parameter (DEVADD) specifies the physical device address of the mass storage volume. Specifically, it gives the peripheral control unit number written as two octal digits in which all bits except the I/O bit must be specified. Also, it specifies in one octal digit the drive number. When the parameter is not specified, the assumed values of Ø4 and Ø for the pcu and drive respectively are used.

FILE STATEMENT: Each file to be deallocated is specified by a File Statement whose first parameter is NAME. This is a required statement. To deallocate more than one file with a single Function Statement, there must be a File Statement for each file. The parameters of the File Statement are described in the following paragraphs.

Name Parameter: The Name Parameter (NAME) gives the name of the file to be deallocated. It must be exactly as the name appears in the Volume Directory.

Expiration Parameter: The Expiration Parameter (EXP) specifies whether or not the expiration date of the file to be deallocated is to be checked. When this parameter is not specified, the expiration date automatically is checked.

Password Parameter: The Password Parameter (PW) gives the password of the file to be checked against the Password Field in the Volume Directory. The Password Parameter may be omitted only if the file being deallocated is not protected by a password.

When the Password Field in the Volume Directory is not all spaces (as it is when no password is assigned) and the password check is made without this parameter being specified, the result is that the file is not deallocated.

DAY STATEMENT: The Day Statement specifies the day against which the expiration date of the file is checked. If no Day Statement is submitted, the Supervisor Current Date Field is used.

DEALLOCATE FUNCTION JOB CONTROL LANGUAGE EXAMPLE

The following job control statements cause the deallocation of two files on the volume whose serial number is AØØØØØ. The first file to be deallocated is FILE-E and its expiration date is checked (automatically) and its password is DEPT.1ØØ. The second file to be deallocated is FILE-C. Its expiration date is not checked and the password is not checked (it is assumed that this file was not protected by a password).

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . _____ PAGE ___ OF ___

CARD NUMBER	TYPE CHECK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6 7 8	14 15	20 21	62 63	80
1			FUNCT.	DEALLOCATE,	
2			VOLUME	NAME=AØØØØØ,	
3			FILE	NAME=FILE-E, PW=DEPT.1ØØ,	
4			FILE	NAME=FILE-C,	
5	E			EXP=NO.	

Table 3-14 contains a summary of the deallocate function job control statements.

Load/Unload Function

DESCRIPTION

The Mass Storage File Support Subsystem has the facility to load data from and unload data to one-half inch magnetic tape or punched cards. All standard fixed length formats are allowed. Appendix D of this manual summarizes these formats and points out any features that are extensions of previous Honeywell Series 200 Software.

Table 3-14. Deallocate Function Job Control Statements

JOB CONTROL STATEMENT	PARAMETER	PARAMETER VALUE	DESCRIPTION	REQUIREMENTS
FUNCTION STATEMENT	DEALLOCATE	DEALLOCATE	Specifies what file support function to perform.	Required.
VOLUME STATEMENT	NAME	VOLUME NAME	Specifies the serial number of the volume being allocated.	Optional.
	DEVADD	PCU	Specifies the PCU number of the device in two octal digits.	
		Drive	Specifies the drive number of the device in one octal digit.	
FILE STATEMENT	NAME	File name	Specifies the name of the file to be deallocated.	Required.
	EXP	NO	Specifies that the expiration date of the file to be deallocated should not be checked.	Optional.
		YES	Specifies that the expiration date of the file to be deallocated should be checked.	
	PW	Password	This is the password to be checked against the password field in the volume directory entry for this file.	Optional.
DAY STATEMENT	yyddd	yyddd	This is the year and day against which the expiration date of the file being deallocated is to be checked.	Optional.

LOAD/UNLOAD FUNCTION JOB CONTROL STATEMENT

Format

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	NAME	LOCATION	OPERATION CODE	OPERANDS	
				FUNCT	{LOAD UNLOAD},	
				FILE	{IN OUT},	
					NAME=file-name,	Optional
					DEVTYPE=device-type,	Optional
					DEVADD=(pcu,drive),	Optional
					ITEM=item-length,	Optional
					REC=record-length,	Optional
					BAN={YES NO banner},	Optional
					PAD=padding,	Optional
					PAR={ODD EVEN},	Optional
					MODE={SPEC STAND},	Optional
					PW=password,	Optional
					BUCKET={REL ABS},	Optional
					PROT={A B AB NO};	Optional
				MEMBER	NAME=member-name,	Optional Statement
				EXITS	PROG=program-name,	EXITS STATEMENT IS
	E				LMA=low-memory-address,	optional.

Description

A Load or an Unload Function is requested by a Function Statement whose first parameter is either LOAD or UNLOAD. The File Statements specify whether the operation is to or from mass storage. The Function Statement is followed by two File Statements, one for the input file and one for the output file. Each File Statement may be followed by an associated Member Statement. The Member Statement associated with the first File Statement must appear after that File Statement and before any subsequent File Statment. There may also be one Exits Statement.

FUNCTION STATEMENT: The Function Statement contains the Operation Code FUNCT and the Operand LOAD or the Operand UNLOAD. This specifies to the system what function to perform.

FILE STATEMENTS: Both the Input and the Output File Statements are described here since they are essentially equivalent in form. The input file for a Load/Unload Function is identified by a File Statement whose first parameter is IN. The Output file is identified by a File Statement whose first parameter is OUT.

In/Out Parameter: The In/Out Parameter (IN) or (OUT) specifies whether the File Statement applies to the input file or to the output file.

Name Parameter: The Name Parameter (NAME) must be specified when the file is a mass storage file. With other device types (cards or tape), the Name Parameter may be omitted. When this is the case, label checking is not done. When specified, the Name Parameter must be identical to that appearing on the data file.

Device Type Parameter: The Device Type Parameter (DEVTYPE) specifies the storage medium used for the file as well as the type of peripheral device used to access the file. The type number of the device used to access the file is given and this number may be any one of the following:

227	Card Reader - Card Punch
224-1	Card Reader/Punch
223	Card Reader
224-2	Card Reader/Punch
214-1	Card Punch
214-2	Card Reader/Punch
204B	One-half Inch Magnetic Tape
222	Printer
206	Printer

When this parameter is not specified, the standard assumption is that the device type is mass storage.

Device Address Parameter: The Device Address Parameter (DEVADD) allows changes to be made to the standard assignment of the peripheral device used to access the file. The peripheral control unit number is given as two octal digits and all bits must be specified. The drive number is given as one octal digit. When this parameter is not specified, the standard values used are as follows:

Punched Card Input	pcu 41
Punched Card Output	pcu Ø1
Magnetic Tape Input	pcu 4Ø drive 1
Magnetic Tape Output	pcu Ø4 drive 1
Mass Storage Input	pcu 44 drive Ø
Mass Storage Output	pcu Ø4 drive Ø
Printer Output	pcu Ø2

Item Parameter: The Item Parameter (ITEM) gives the length in characters of each item in the file. When the file is on mass storage this parameter must be omitted as the item length is obtained from the Volume Directory entry for the file. When the file is not on mass storage, this parameter may be omitted and the item length will be equal to the item length in the mass storage file.

Record Length Parameter: The Record Length Parameter (REC) gives the number of characters in each record of the file. When the file is stored on mass storage this parameter must be omitted as the record length will be obtained from the Volume Directory entry for the file. When the file is not stored on mass storage, this parameter may be omitted and the record length will be assumed to be equal to the block length in the mass storage file.

Banner Character Parameter: The Banner Character Parameter (BAN) only applies to a magnetic tape file and gives the banner character of each data record in two octal digits. When omitted the file is assumed to be unbannered. When the value of the parameter

equals YES for an output tape, a standard banner character of 41_8 is used. On an input tape, YES indicates the presence of a banner but its value is not checked. When the value of the parameter equals NO, the file is assumed to be unbannered.

Padding Character Parameter: The Padding Character Parameter (PAD) only applies to a magnetic tape file and gives the padding character for the file in two octal digits. When omitted, and the file is using odd parity, the standard value is 77_8. When omitted, and the file is using even parity, the standard value is 11_8.

Parity Parameter: The Parity Parameter (PAR) only applies to a magnetic tape file and gives the parity of the recording as odd or even. When omitted, the standard value is odd parity.

Mode Parameter: The Mode Parameter (MODE) applies only to a punched card file and specifies the reading or punching mode as standard (STAND) or special (SPEC). When not specified, the standard assumption for this parameter is that the special mode is to be used.

Password Parameter: The Password Parameter (PW) applies only to a mass storage file and gives the password to be checked against the Password Field of the Volume Directory entry for this file. When the Password Field of the Volume Directory entry for the file is not all spaces, the password check is made regardless of whether or not this parameter has been omitted.

Bucket Parameter: The Bucket Parameter (BUCKET) applies only to a Direct Access File stored on mass storage. It gives the type of bucket addressing as relative (REL) or as absolute (ABS). When not specified, the standard assumption for this parameter

is that the absolute bucket addressing mode is to be used.

Protection Parameter: The Protection Parameter (PROT) indicates the write protection that was assigned to the file when the file was allocated. The significance of the values of this parameter is as follows:

A	The file was allocated with A-File write protection
B	The file was allocated with B-File write protection
AB	The file was allocated with A- and B-File write protection
NO	The file was allocated with no write protection assigned.

When this parameter is omitted, the standard assumption is that no write protection was assigned to the file when it was allocated. When a file has been allocated with a Protection Parameter other than NO, the same value that was used during allocation must be used when describing the file for the Load/Unload Function.

This parameter is specified only for an Output mass storage file.

MEMBER STATEMENT: The File Statement may be followed by one or more Member Statements. These statements specify that one or more members of a Partitioned Sequential File are to be processed. When the entire file is to be processed, these statements are omitted. The Member Statement has only one parameter, the Name Parameter (NAME). The Name Parameter specifies the name of the member to be processed. It can be up to 14 characters long, and when it is less trailing spaces automatically are added.

EXITS STATEMENT: The Exits Statement enables the exit to a user supplied routine just after accessing an input item and just before writing an output item. The Exits Statement is required for Direct Access mass storage output files to compute the bucket address for the output items. The Exits Statement describes the user supplied routine.

Program Name Parameter: The Program Name Parameter (PROG) gives the program name of the user's routine. This routine is entered at

Table 3-15. Load/Unload Function Job Control Statements

JOB CONTROL STATEMENT	PARAMETER	PARAMETER VALUE	DESCRIPTION	REQUIREMENTS
FUNCTION STATEMENT	LOAD	LOAD	Specifies whether the file support function is load or not.	Required.
	UNLOAD	UNLOAD	Specifies whether or not the file support function is unload.	
FILE STATEMENT	IN	IN	Specifies that this file statement applies to the input file.	One input and one output file statement is required for either function.
	OUT	OUT	Specifies that this file statement applied to the output file.	
	NAME	File name	Specifies the name of the file being loaded or unloaded.	Optional when file is not a mass storage file.
	DEVTYPE	Device type	Specifies the storage medium used for the file as well as the type of device used to access the file.	Optional.
	DEVADD	PCU	Specifies the PCU number in two octal digits.	Optional.
		Drive	Specifies the drive number in one octal digit.	
	ITEM	Item length	Specifies the length of the items in the file in number of characters.	Optional.
	REC	Record length	Specifies the length of the records in the file in number of characters.	Optional.

Table 3-15 (cont). Load/Unload Job Control Statements

JOB CONTROL STATEMENT	PARAMETER	PARAMETER VALUE	DESCRIPTION	REQUIREMENTS
FILE STATEMENT (cont)	BAN	YES	Specifies the banner character for an output tape file as 41_8.	Optional. Applies only to magnetic tape files.
		NO	Specifies that the tape file is unbannered.	
		Banner	Specifies the banner for an output tape file in two octal digits.	
	PAD	Padding	Specifies the padding character for a magnetic tape file in two octal digits.	Optional.
	PAR	ODD	Specifies the parity of the recording as odd for magnetic tape files.	Optional.
		EVEN	Specifies the parity of the recording as even for magnetic tape files.	
	MODE	SPEC	Specifies the reading or punching mode for a card file as special.	Optional.
		STAND	Specifies the reading or punching mode for a card file as standard.	
	PW	Password	This is the password to be checked against the password field of the volume directory entry for this file.	Optional. Applies only to mass storage files.
	BUCKET	REL	Specifies the type of bucket addressing for a direct access file as relative.	Optional.
		ABS	Specifies the type of bucket addressing for a direct access file as absolute.	

Table 3-15 (cont). Load/Unload Job Control Statements

JOB CONTROL STATEMENT	PARAMETER	VALUE	DESCRIPTION	REQUIREMENTS
FILE STATEMENT (cont)	PROT	A	Specifies that this file was allocated with A-file write protection.	Optional.
		B	Specifies that this file was allocated with B-file write protection.	
		AB	Specifies that this file was allocated with both A- and B-file write protection.	
		NO	Specifies that this file was allocated without write protection.	
MEMBER STATEMENTS	NAME	Member name	Specifies the name of the member to be processed.	Optional.
EXITS STATEMENT	PROG	Program name	Specifies the name of the user's program.	Optional.
	LMA	Low memory address	Specifies the lowest memory address used by the user's program.	Must be specified.

the location specified by the Easycoder END Statement and is the starting address of the routine. The return to the Load/Unload Function from the user's routine is made by branching to the address that was in the B Address Register (BAR) at the time the routine was entered.

Low Memory Address Parameter: The Low Memory Address Parameter (LMA) gives the lowest memory address used by the user's routine and must be specified. This parameter is given in decimal.

LOAD/UNLOAD FUNCTION JOB CONTROL LANGUAGE EXAMPLE

The following job control statements request a magnetic tape file to be loaded into mass storage. The input magnetic tape file is named FILE-X. Its item length is 125 characters and its record length is 5Ø1 characters. This file is bannered. Standard value assumptions are that parity is odd and padding is 77_8. The output mass storage file is named FILE-X also and has an item length of 125 characters. The record size of this file has the standard size record, 25Ø characters. Also, the output file is not protected by a password or by write protection.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62 / 63 ... 80
				FUNCT.	LOAD,
				FILE	IN,
					NAME=FILE-X,
					DEVTYPE=204B,
					ITEM=125,REC=501,
					BAN=YES,
				FILE	OUT,
		E			NAME=FILE-X,

Table 3-15 contains a summary of the load/unload function job control statements.

Map Function

DESCRIPTION

The Map function produces selected information about a mass storage

volume from the Volume Directory. This routine has three separate actions: production of description of a file, mapping expired files and mapping unused areas of the volume.

Description Of A File

The description of the structure and other information about mass storage files can be listed. One or several files may be listed, or all files on the volume may be listed. The information listed is taken from the contents of the Volume Directory.

Expired Files

A listing of all files that have expired, as indicated by their expiration dates, can be produced. The user can request that all files whose expiration date is less than a specified date be listed.

Unused Areas

A listing of all unused areas on a mass storage volume can be produced. The installation can use this listing to determine units of allocation for new files.

MAP FUNCTION JOB CONTROL STATEMENTS

Format

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___.___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				FUNCT	MAP {DESCR EXPIRED UNUSED},	
			VOLUME	NAME=volume-name,		The Volume Statement is optional.
				DEVADD=(pcu,drive),		
			FILE	NAME=file-name,		
			FILE	NAME=file-name,		Optional.
		E	DAY	yyddd,		

Description

The Map Function is requested by a Function Statement whose first parameter is MAP. The second parameter of the Function Statement gives

the type of mapping desired. The Function Statement may be followed by a Volume Statement, one or more File Statements, and a Day Statement in any order.

FUNCTION STATEMENT: The Function Statement contains the Operation Code FUNCT and either the operand MAP,DESCR - MAP,EXPIRED - or MAP, UNUSED. This statement specifies to the system what function to perform.

Map Volume Description Parameter: The Map Volume Description Parameter (MAP,DESCR) requests a printed listing of the contents of the Volume Directory for the files specified in the File Statements, or of the entire Volume Directory when no File Statements whose first parameter is NAME is specified.

Map Expired Parameter: The Map Expired Parameter (MAP, EXPIRED) requests a listing of all files whose expiration date is less than the date specified.

Map Unused Parameter: The Map Unused Parameter (MAP,UNUSED) requests a listing of all unused space on the mass storage volume.

VOLUME STATEMENT: The Volume Statement contains parameters that pertain to the volume to be mapped. This statement is not required, and when omitted its parameters are assigned standard values.

Name Parameter: The Name Parameter (NAME) gives the serial number of the volume to be mapped.

Device Address Parameter: The Device Address Parameter (DEVADD) gives the device address of the volume to be mapped. The peripheral control unit number is given in 2 octal digits. All bits except the I/O bit must be specified. The device drive number is given in 1 octal digit. When this parameter is not specified, the pcu address is Ø4 and the drive number is Ø.

FILE STATEMENT: When a listing of Volume Directory information for a selected file is desired, a File Statement whose first parameter is NAME is required. For each file for which a description is desired, a single File Statement is required. When the Volume Directory information for all files on the volume is desired, the File Statement may be omitted. The File Statement is not relevant to the listing of obsolete files or of unused areas. The Name Parameter of the File Statement names the file whose Volume Directory information is to be listed.

DAY STATEMENT: The Day Statement contains a date against which file expiration dates are to be checked when a listing of expired files is being produced. When no Day Statement is given, the Current Date Field of the Supervisor is used. The parameter of the Day Statement is yyddd. The year is specified by yy and the day of the year by ddd (counting from January 1 as day ØØ1).

MAP FUNCTION JOB CONTROL LANGUAGE EXAMPLES

The following Map Function job control statements requests a listing of the unused areas of a volume mounted on drive 1 of peripheral control unit Ø4.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___.___.___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1				FUNCT.	MAP,UNUSED,
2		E		VOLUME	DEVADD=(Ø4,1),
3					

These Map Function job control statements request a listing of the Volume Directory information for files named FILE-F and FILE-G. The standard conditions are that the volume containing these files is at device address pcu Ø4, drive Ø.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___.___.___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1				FUNCT.	MAP,DESCR,
2				FILE	NAME=FILE-F,
3		E		FILE	NAME=FILE-G,

Table 3-16 contains a summary of the map function job control statements.

Table 3-16. Map Function Job Control Statements

JOB CONTROL STATEMENT	PARAMETER	PARAMETER VALUE	DESCRIPTION	REQUIREMENTS
FUNCTION STATEMENT	MAP	DESCR	Specifies that a printed listing of the contents of the volume directory for the files specified in the file statement be produced.	Required
		Expired	Specifies that a printed listing of all files whose expiration date is less than the date specified be produced.	
		Unused	Specifies that a printed listing of all unused space in the volume be produced.	
VOLUME STATEMENT	NAME	Volume name	Specifies the serial number of the volume to be mapped.	Optional.
	DEVADD	PCU	Specifies the PCU number of the volume in two octal digits.	
		Drive	Specifies the drive number of the volume in one octal digit.	
FILE STATEMENTS	NAME	File Name	Specifies the names of the files whose volume directory contents is to be listed.	Required for map, description function
DAY STATEMENT	yyddd	yyddd	This is the date against which file expiration dates are to be checked.	Required for map, expired function.

FILE SUPPORT PROGRAMMER'S PREPARATION INFORMATION

Considerations for Direct Access Files

When specifying a Direct Access file, the item length as contained in the Volume Directory is interpreted as including the status character (right-most character) of the item. The value of this character is set to "inactive" for all items of the file during file allocation. For any item loaded, the value is set to "active" during the load process. The possible values of this character are as follows:

LAST BLOCK OF FILE	ALL OTHER BLOCKS	MEANING
76	77	Inactive item.
ØØ	Ø1	Active item.

LOADING A DIRECT ACCESS FILE

When loading a Direct Accessfile on mass storage, the EXITS statement must always be specified since the user must supply the bucket address (in binary) for each item in the file via an own code routine.

UNLOADING A DIRECT ACCESS FILE

Direct Access files are unloaded in a sequential manner in the physical order in which the active items are encountered on the file. Only active items are unloaded. The user is never requested to supply a bucket address but he may, however, specify an own-code routine to modify, delete or examine the items being processed.

Considerations For Sequential Files

A Sequential file is always loaded and unloaded in a sequential manner. An own-code routine may be used as described for unloading a Direct Access file.

Considerations For Partitioned Sequential Files

Each member of this file type is processed individually. Within each member, the items are processed in a sequential manner in the physical

order in which they are encountered.

UNLOADING A PARTITIONED SEQUENTIAL FILE

To unload a Partitioned Sequential file, no member names are specified in the Job Control File; only the file name is specified. All active members of the partitioned file are unloaded in the order that their names appear in the Member Index for that file.

To unload selected members of a partitioned sequential file the desired member names are specified in the Job Control File. These are unloaded in the order in which the names appear in the Job Control File.

LOADING A PARTITIONED SEQUENTIAL FILE

Loading By File

The user may load an entire Partitioned Sequential File by either of the following means:

1. Specify no member names in the Job Control File. In this case the member names are taken from the Input File.

2. Specify in the Job Control File the member names of all members which comprise the output mass storage file.

Loading Selected Members

The user may load selected members of an output mass storage Partitioned Sequential file by specifying the desired member names in the Job Control File.

Processing By Member Names

When loading an output mass storage file the Load function takes the output member names from the Job Control File, if specified, or, when not specified, from the Input File.

Whether loading by file or member name, if the name under which the

member is to be loaded already exists in the Member Index of the output mass storage file, and if the member can be processed in the Output Only mode, the input data will replace that members data on the output mass storage file. If the member name does not already exist, the input member and its data will be added as a new member to the output mass storage file.

When member names are specified, and if the output member names in the Job Control File are exhausted before all indicated input members have been processed, loading is terminated and control transferred to the next routine. Member names are specified only for a file which is on mass storage; not for card or tape files.

Own Coding

During a load or unload function the user may execute an own-coding routine for further item processing. In the case of Direct Access files which are being loaded onto mass storage, an own-code routine is required. In all other cases this own-coding routine is optional. The user may examine, modify or delete items at this time. File Support branches to own-coding once for each active item.

STRUCTURE OF OWN-CODING

The own-coding routine must be written and assembled as a single segment program. This program should originate where it occupies the memory area immediately below the floating portion of the Supervisor. The File Support program will load the own-coding only from the same storage medium (and, if stored on mass storage, the same Executable Program File) as the File Support program itself.

OWN-CODE COMMUNICATION WITH THE LOAD/UNLOAD FUNCTION

In the EXITS Statement of the Load/Unload function the user is required to specify the lowest memory address of the own-coding. One character should be reserved at that address (LMA) for communication with

the File Support program. When File Support gives a new item to the user, the communication character is set to zero. More detailed use of this character is given in subsequent paragraphs. The branch to own-coding will occur at the next character location (LMA+1).

Address communication is made through Index Registers 1 and 5.

Index Register 1: This register is set by File Support to the left-most character of the current item.

Index Register 5: This register is set by the own-code routine to the right-most character of a user supplied field into which the user will place his bucket address when loading a Direct Access file. The field is four characters if relative bucket addressing was specified and eight characters (in the form DMCCTTRR) if actual bucket addressing was specified. The left-most character of the field must contain a word mark.

Return to File Support is made via the B address register setting, stored at the time own-coding was entered.

Deletion Of Items

As mentioned previously, the communication character is set to zero (ØØ) when the item is given to the user. If the item is to be written out to the output file, the communication character remains zero. If the user desired to delete the item, the communication character would be set to one (Ø1) prior to return to File Support.

Invalid Bucket Addresses

If the branch to own-coding shows a one in the communication character, then the last bucket address supplied to File Support was an invalid address. When this is the case, the user may do either of the following:

1. Return to File Support with a communication character of zero to have that item bypassed.

2. Return to File Support with a communication character of one to terminate the loading of this file. In this case, processing will proceed to the next File Support function.

Insufficient Space

If the branch to own-coding shows two (Ø2)in the communication character, there was no room left in the bucket or overflow area(s) for the last item given to the load function. In this case the user may do either of the following:

1. Return to File Support with a communication character of zero to have the item bypassed.

2. Return to File Support with a communication character of one to terminate the loading. In this case, processing will proceed to the next File Support function.

NOTE: A complete description of the card and tape files processed by File Support is contained in Appendix D of this manual.

SECTION IV

PROGRAM DEVELOPMENT SUBSYSTEM

The Program Development Subsystem enables the user to translate source language programs, establish and maintain libraries of programs, and test programs. The Program Development Subsystem has several features of importance to the user. These features are discussed in the following paragraphs.

FEATURES OF THE PROGRAM DEVELOPMENT SUBSYSTEM

Some of the more important features of the Program Development Subsystem, such as the independent operation for each programmer, unbatched operation, and automatic operation, are discussed in the following paragraphs.

Independent Operation For Each Programmer

The Program Development Subsystem is designed so that the programmer makes independent requests for a related series of operations on his one program or library routine. He does this independently of other programmers, who are operating with programs that may be completely unrelated to this program. All information necessary to the operations for the programmer is submitted by the programmer; he is not required to submit any information that is not related to his operations and his program. For example, he does not have to worry about batching his program with other, unrelated programs; he does not concern himself with the names of the system programs to be run; he is not concerned with the equipment assignments because the system uses the standard values defined for each installation.

Unbatched Operation

The operations of the Program Development Subsystem are performed on one program or library routine at a time. The programmer submits

a separate request for each routine. This type of operation is called "unbatched" and provides a shorter turn-around time because the programmer does not have to wait for the completion of operations on other unrelated programs that have been batched with his.

Automatic Operation

The Program Development Subsystem automatically controls the sequencing of the several processing routines required to perform the operations requested by the programmer. All the functions of the Program Development Subsystem are performed under this control.

ELEMENTS OF THE PROGRAM DEVELOPMENT SUBSYSTEM

There are four basic elements to the Program Development Subsystem. These are language translators, program library file maintenance, program test facilities, and Easycoder source language analysis. Program test facilities and Easycoder source language analysis are not included in the first software release. Each of these is discussed briefly in this paragraph. Fully detailed discussions of the Easycoder Assembly, Library File Update, and Executable File Update features follow in subsequent paragraphs.

Language Translators

The operating system provides languages that enable the programmer to express programs in forms that can easily be learned and can readily be used. The Program Development Subsystem provides translators that convert programs written in these languages into machine executable form. All of the language translators in the Program Development Subsystem produce the same format of machine-executable code. Their outputs may be stored in a common file. The language translators of the operation system are:

1. Easycoder Assembly. Easycoder Assembly is a symbolic machine oriented assembly language with facilities for the inclusion of macro routines. The language level is compatible with Easycoder D of the Operating System - Mod 1 (Tape Resident).

2. COBOL. COBOL is a business data processing language that is close to normal English language usage. The language level is comparable to COBOL B of the Basic Programming System. A COBOL compiler that operates under the Program Development Subsystem is not included in the first software release.

3. FORTRAN. FORTRAN is a scientific language similar to usual mathematical notation. The language level is comparable to FORTRAN Compiler D of the Operation System-Mod 1 (Tape Resident). A FORTRAN compiler that operates under the Program Development Subsystem is not included in the first software release.

Program Library File Maintenance

The Program Development Subsystem provides routines to maintain program libraries on mass storage. Two types of program libraries can exist in the operating system. These are a Library of Macro Routines and an Executable Program File.

LIBRARY OF MACRO ROUTINES

A file of macro routines, written in the Easycoder symbolic language, can be maintained on mass storage. Routines from this library may be specialized and included in an Easycoder Assembly language program. This is the macro facility of the Easycoder Assembler. Routines may be added to, deleted from, or replaced in a

source language library. Individual statements in a routine also may be corrected.

EXECUTABLE PROGRAM FILE

A file of executable programs can be maintained on mass storage. Such a file is updated with the output of the language translators; all translators in the Program Development Subsystem produce the same format of machine-executable code. Routines may be added to, deleted from, or replaced in an executable program file.

Program Test Facilities

A translated program may be executed for testing immediately after translation or stored in the executable program file to be executed later.

Facilities will be available for dumping selected areas of main memory and mass storage. Program test facilities are not included in the first software release.

Easycoder Source Language Analysis

An analyzer routine will provide a symbolic cross-reference listing for an Easycoder Assembly language program. The analyzer routine is not included in the first software release.

EASYCODER ASSEMBLY

The following paragraphs provide a general description of the Easycoder Assembly and its functions along with the Easycoder Assembly language and the Job Control Language for the Easycoder Assembly.

General Description

Easycoder Assembly C translates Easycoder source language programs into machine-executable code. The source language is

compatible with Easycoder D of the Series 200/Operating System - Mod 1 (Tape Resident). Only minor differences exist between the languages of Mass Storage Assembly C and Easycoder Assembly D of the Mod 1 system. However, operating procedures for the two programs are different. The term "language", as used in this paragraph, refers to the ordinary statements expressing a program. It does not include the PROG Statement, the ECD Statement, or any other source of information to control the operation of the assembler.

The two functions of assembly and macro specialization are parts of the assembly function in the Program Development Subsystem.

The library file of macro routines on mass storage consists of symbolic card images only, without any machine language information. Updating this file is a function of the Program Development Subsystem and can be performed before assembly.

Easycoder Assembly Functions

Easycoder Assembly performs two distinct functions: assembly and macro inclusion and specialization.

1. Assembly. The Easycoder Assembly functions are the same as those of Easycoder Assembly D of the Operating System Mod 1 (Tape Resident), with the following exceptions:
   a. Correction Listing - The assembler does not print a listing of symbolic corrections.
   b. Source Language Library Directory - The assembler does not print a directory of the source language library.
   c. Literal Processing - All literals, both pooled and non-pooled, appear in the listing and the machine-executable code immediately before the appropriate EX or END statement or after the appropriate LITORG statement.

d. Machine Executable Output - The available outputs are: Binary Load Deck (BLD) - a card deck suitable for loading with the Mod 1 Card Loader-Monitor B and a Binary Run File (BRF) - a work file on mass storage that may be used for updating the Executable Program File.

2. Macro Inclusion and Specialization. The macro inclusion and specialization function of the Easycoder Assembly is the same as that of Easycoder Assembly D of the Operating System - Mod 1 (Tape Resident), with the following exceptions:

a. Relationship to Assembly - The macro inclusion and specialization process (Library Processor) is an integral part of the assembler. It cannot be run as a separate program to punch source language statements from the source language library.

b. Macro Library File - The source of macro routines is a source language library file on mass storage. It is organized as a partitioned sequential file and each macro is one member. A macro library update function is provided as part of the Program Development Subsystem.

c. Option to leave Macros Unspecialized - The option to leave selected macros unspecialized after assembly is not available. All macros are specialized, unless the user specifies that no macro specialization is to be performed for the program.

Easycoder Assembly Language

The Easycoder Assembly language is essentially the same as that

of Easycoder Assembly D of the Operating System Mod 1 (Tape Resident), with the following exceptions:

1. Assembly Language.
   a. Line Number Generation - The assembler does not generate line numbers. This is a function of the source language update.
   b. SETLIN Statement - The assembler does not process SETLIN statements. This is a function of the source language update.
   c. Temporary Remarks Statement - The assembler does not process temporary remarks statements. (T in the type field, column 6.)
   d. Data Statements - The assembler does not process data statements (D in the type field). A card image data file may be placed on a mass storage volume through the standard File Support routines described in Section 3 of this manual.
   e. Macro Routine Limiters - Macro routine limiters (M and N type statements) are not used for the deletion of old specialized macro coding for the source language library because the assembler does not perform an update. They are used, however, to cause temporary suspension of line number sequence checking.

2. Macro Inclusion and Specialization. The assembler does not generate line numbers for statements included in a program as a result of macro processing. The listing shows the same line number as the corresponding unspecialized statement in the macro library file.

Easycoder Assembly Statements

Table 4-1 is a comprehensive list of all source language statements acceptable as input for translation to Mass Storage Easycoder Assembly C. The function performed by each statement and the method for writing each in a source language program are fully described in Honeywell Series 200 Programmers' Reference Manual (Models 200/1200/2200), Order Number 139. There are three addendums to this manual, numbered #1, #2 and #3 that update or amplify the explanation of some of the Easycoder statements. It should be noted, however, that a statement not included in Table 4-1 will not be processed by Mass Storage Easycoder Assembly C. For example, the HSM statement in the reference manual and the SETLIN statement in Addendum #2 are not processed by the assembler.

TABLE 4-1. Easycoder Assembly Statements

STATEMENT	TYPE	MNEMONIC OPERATION CODE	FUNCTION
INSTRUCTIONS	ARITHMETIC	A	Decimal Add
		S	Decimal Sub.
		BA	Binary Add
		BS	Binary Sub.
		ZA	Zero & Add
		ZS	Zero & Sub.
		M	Multiply
		D	Divide
	LOGIC	EXT	Extract
		HA	Half Add
		SST	Substitute
		C	Compare
		B	Branch
		BCT	Branch On Condition Test
		BCC	Branch On Character Condition
		BCE	Branch If Character Equal
		BBE	Branch On Bit Equal
	CONTROL	SW	Set Word Mark
		SI	Set Item Mark
		CW	Clear Word Mark
		CI	Clear Item Mark
		H	Halt
		NOP	No Operation
		MCW	Move Characters To Word Markd
		LCA	Load Characters To A-Field Word Mark
		SCR	Store Control Registers
		LCR	Load Control Registers
		CAM	Change Addressing Mode
		EXM	Extended Move
		MAT	Move & Translate
		MIT	Move Item & Translate
		LIB	Load Index/ Barricade Indicator
		SIB	Store Index/ Barricade Indicator
	INTERRUPT CONTROL	SVI	Store Variant & Indicators
		RVI	Restore Variant & Indicators
		MC	Monitor Call
		RNM	Resume Normal Mode
	EDITING	MCE	Move Characters & Edit

Table 4-1 (cont). Easycoder Assembly Statements

STATEMENT	TYPE	MNEMONIC OPERATION CODE	FUNCTION
INSTRUC-TIONS (cont)	INPUT/OUTPUT	PDT	Peripheral Data Transfer
		PCB	Peripheral Con-Trol & Branch
ASSEMBLY CONTROL		PROG	Program Header
		SEG	Segment Header
		EX	Execute
		ORG	Origin
		MORG	Modular Origin
		LITORG	Literal Origin
		ADMODE	Admode
		EQU	Equals
		CEQU	Control Equals
		SKIP	Skip
		SFX	Suffix
		REP	Repeat
		GEN	Generate
		CLEAR	Clear
		END	End
DATA FORMAT-TING		DCW	Define Constant With Word Mark
		DC	Define Constant Without Word MARK
		RESV	Reserve Area
		DSA	Define Symbol Address
		DA	Define Area

Easycoder Assembly Function Job Control Statements

FORMAT

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

CARD NUMBER	TYPE CODE	MACRO INSTR	LOCATION	OPERATION CODE	OPERANDS	
1				FUNCT	EASYCODER,	
2					MACRO=NO,	Optional
3					LIST=NO,	Optional
4					GO={BLD / BRF / NO},	Optional
5						
6						
7		E		DATE	date-field,	Optional State
8						ment
9						

DESCRIPTION

Easycoder Assembly Function job control language consists of a Function Statement and a Date Statement.

Function Statement

The Function Statement identifies the function to be performed as Easycoder Assembly. If the function is to be performed under completely standard conditions - that is, if all the parameter standard assumptions are acceptable - no other job control information is required. When there are exceptions to the standard conditions, additional parameters must be included with the Function Statement.

MACRO PARAMETER: The Macro Parameter (MACRO=NO) is entered when there are no macros or, if there are macros and the programmer wishes to leave them unspecialized. Normally, Easycoder Assembly specializes all macros appearing in the input file.

LIST PARAMETER: A listing showing source language and machine language coding normally is produced. When the listing is to be omitted, the List Parameter (LIST=NO) is entered.

GO PARAMETER: Easycoder Assembly can produce two forms of machine-executable output: a Binary Load Deck (BLD) on punched cards or a Binary Run File (BRF) on mass storage. It is possible to request one of these, both of these, or neither of these. When the parameter is omitted, the BRF output is produced. When the parameter is written GO=NO, no machine-executable output is produced; when written GO=BLD, a Binary Load Deck is produced; and when written as GO=BRF, a Binary Run File is produced. When both types of output are desired, the parameter is entered twice; once as GO=BLD and once as GO=BRF.

Date Statement

A date to be printed on the listing is submitted through the Date Statement. The date-field consists of 8 characters in any form. The date is printed without change on the listing. When a Date Statement is used, it must follow the Function Statement.

Easycoder Assembly Function Job Control Language Examples

The following job control statements cause the program TEST-A to be assembled. Macros are specialized, a listing is produced, and the machine-executable file is on mass storage.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

	CARD NUMBER	TYPE	MACRO	LOCATION	OPERATION CODE	OPERANDS
	1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62 63 ... 80
1			E		FUNCT	EASYCODER,
2					PROG	TEST-A
3				Easycoder	Statements	
4					END	
5						

The following job control statements cause the program TEST-A to be assembled. Macros are specialized, no listing is produced, and the machine-executable output file is on punched cards.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ____ . ____ PAGE ____ OF ____

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14 15	20 21	62 63	80
1				FUNCT	EASYCODER,GO=BLD,	
2		E			LIST=NO,	
3				PROG	TEST-A	
4			Easycoder		Statements	
5				END		

Table 4-2 contains a summary of Easycoder assembly function job control statements.

Table 4-2.
Easycoder Assembly Function Job Control Statements

JOB CONTROL STATEMENT	PARAMETER	VALUE	DESCRIPTION	REQUIREMENTS
FUNCTION STATEMENT	EASYCODER	EASYCODER	SPECIFIES WHAT PROGRAM DEVELOPMENT FUNCTION IS TO BE PERFORMED	REQUIRED
	MACRO	MACRO=NO	SPECIFIES THAT MACROS, IF INCLUDED IN PROGRAM, ARE NOT TO BE SPECIALIZED	OPTIONAL
	LIST	LIST=NO	SPECIFIES THAT A LISTING OF SOURCE AND MACHINE LANGUAGE CODING SHOULD NOT BE PRODUCED	
	GO	GO=BLD	SPECIFIES THAT MACHINE EXECUTABLE OUTPUT OF ASSEMBLY SHOULD BE A BINARY LOAD DECK.	
		GO=BRF	SPECIFIES THAT MACHINE EXECUTABLE OUTPUT OF ASSEMBLY SHOULD BE A BINARY RUN FILE ON MASS STORAGE.	
		GO=NO	SPECIFIES THAT NO MACHINE EXECUTABLE OUTPUT SHOULD BE PRODUCED BY ASSEMBLY.	
DATE STATEMENT	DATE FIELD	8 CHARACTERS	THIS IS THE DATE THAT SHOULD BE PRINTED ON THE LISTING PRODUCED.	OPTIONAL

LIBRARY FILE UPDATE

The following paragraphs provide a general description of the Library File Update, followed by description of the update functions, files, and job control language.

General Description

The Mass Storage Library File Update creates and maintains a library file of Easycoder source language macro routines in unspecialized form. These routines may be specialized and included into an Easycoder source language program by the Mass Storage Easycoder Assembler (see page 4-5). The source language library is maintained on Mass Storage.

The Libaray File Update can add a macro routine to or delete one from the library; or it can correct individual statements in a macro routine in the library.

As a part of the Program Development Subsystem, the Library File Update operates in an unbatched mode. That is, one macro routine at a time is updated; each macro routine is operated upon independently of the preceding routine.

Library File Update Functions

The Library File Update functions described in the following paragraphs can be performed on macros in the Library File. In all cases where a macro is marked deleted the space occupied by the macro does not become available. To make the space available, the Library File must be reorganized. The normal File Support routines are used for this purpose.

1. Add Function. A macro can be added to the Library. The source language statements for the macro appear in the Input File. The name of the macro to be added must not duplicate

the name of a macro already in the Library. When a duplication is found, the Library File Update produces an error message on the Listing File and does not add the macro routine to the Library.

The user has no control over the physical placement of a macro to be added to the Library.

2. Delete Function. A macro can be deleted from the Library. The space occupied by the macro is marked, so that the macro can no longer be accessed.

3. Replace Function. A macro can be replaced in the Library. The existing macro of the same name is marked as deleted, and the new macro is added to the Library. The source language statements for the new macro appear in the Input File. The new macro may be assigned a name different from the macro being replaced.

4. Correct Function. A macro can be corrected in the Library. The corrections appear in the Input File, in correct line number sequence. The macro is located in the Library and is copied to a new area in the Library File with the corrections specified in the Input File. The original macro then is marked as deleted. The corrected macro need not have the same name as the original macro.

   Corrections are applied by line number. They may be delections (a single line number or a range of line numbers) replacements, or additions.

5. Creating a New Version Function. A new version of a macro may be created in the Library. Corrections, which are applied only to the new version, may appear in the Input

File. The macro is located in the Library and is copied to a new area along with the corrections specified in the Input File.

The new version must be given a name different from that of the old version.

Library File Update Input and Output Files

The following paragraphs describe the Library File, the Input File, and the Output (listing) File.

1. Library File. The Source Language Library File is a partitioned sequential file on mass storage. Each macro is one member of the file. The Library is a card image file; no additional information is kept. The standard Honeywell physical record format is used. The sequence of routines in the Library File is not under control of the user. Routines may be accessed, both for updating and for inclusion in an Easycoder program, only by their six-character program names. It is not possible, therefore, to have several macros with the same name in one Library File.

2. Input File. The control statements and the card image input appear in the Input File, which must be a card reader. The control statements state the update action to be performed and any options the user desires. The card image input is either an entire macro routine to be added to the Library, an entire macro routine to replace an existing macro in the Library, or corrections to an existing macro routine in the Library.

3. Output (Listing) File. The Output (Listing) file is a listing of the input statements. Both the control statements and the updating input normally are listed. The listing always shows control statements and diagnostic messages. The user may request that the listing of the updating input be omitted. The listing must be on a printer.

Library File Update Function Job Control Statements

FORMAT

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___.___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				FUNCT	SOURCE,	
					ACTION={ADD / COR / REP / DEL / NEW},	Optional
					NEWPROG=new-program-name,	Optional
					PROG=program-name,	Optional
					LIST={NO / YES},	Optional
		E		DATE	date-field,	Optional

DESCRIPTION

Function Statement

The Function Statement identifies the function to be performed as the Library File Update. If the function is to be performed under completely standard conditions - that is, if all the parameter standard assumptions are acceptable - no other job control information is required. When there are exceptions to the standard assumptions, additional parameters must be included.

ACTION PARAMETER: The Action Parameter (ACTION=action-name) specifies the update action to be performed. Values for this parameter may be any of the following.

ADD	Add the program to the Library File
COR	Correct the program
REP	Replace the program
DEL	Delete the program
NEW	Create a new version of the program.

When the update action is not specified, the ADD action is assumed.

NEW PROGRAM NAME PARAMETER: The New Program Name Parameter (NEWPROG) specifies the name to be applied to the program. It is a 6 character value. This parameter is required for the NEW action. It is optional for the COR and REP actions.

PROGRAM NAME PARAMETER: The Program Name Parameter (PROG) specifies the name of the program to be deleted from the Library File. It is required for the DEL action and does not apply to any other update action.

LIST PARAMETER: The List Parameter (LIST) is entered as LIST=NO when it is desired to omit the listing. The listing of the input data, whether a complete program or corrections to an existing program, may be omitted. A listing of the job control statements, including the update action, is always provided. It is never necessary to specify YES.

Date Statement

The Date Statement specifies the date to be printed on the listing. This statement, when used, must follow the Function Statement. The date-field is an 8 character field of any form. It is printed without change on the listing.

Library File Update Function Job Control Language Examples

The following statements cause the program MACRO1 to be added to the Library File.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___.___.___ PAGE ___ OF ___

CARD NUMBER	SYSTEM MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6 7 8	14 15	20 21	62 63 80
1	E		FUNCT	SOURCE,
2			PROG	MACRO1
3		Easycoder	Statements	
4			END	
5				

The following statements cause the program TEST-B to be inserted into the Library File as a replacement for a program of the same name. The date Ø6/15/66 is to be printed on the listing.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1				FUNCT	SOURCE,ACTION=REP,
2		E		DATE	Ø6/15/66,
3				PROG	TEST-B
4			Easycoder	Statements	
5				END	
6					

The following statements cause the program TEST-C to be deleted from the Library File.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1		E		FUNCT	SOURCE,ACTION=DEL,PROG=TEST-C,
2					
3					

The following statements cause the program TEST-D, which is already in the Library File, to be corrected. The Easycoder statements are the corrections to be applied to TEST-D.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1		E		FUNCT	SOURCE,ACTION=COR,
2				PROG	TEST-D
3			Easycoder	Statements with line numbers.	
4				END	
5					
6					

Table 4-3 contains a summary of the library file update job control statements.

Table 4-3.
Library File Update Job Control Statements

JOB CONTROL STATEMENT	PARAMETER	VALUE	DESCRIPTION	REQUIREMENTS
FUNCTION STATEMENT	SOURCE	SOURCE	SPECIFIES THAT THE LIBRARY FILE UPDATE FUNCTION OF PROGRAM DEVELOPMENT IS TO BE PERFORMED.	REQUIRED
	ACTION	ACTION=ADD	SPECIFIES THAT A PROGRAM IS TO BE ADDED TO THE LIBRARY FILE.	OPTIONAL
		ACTION=COR	SPECIFIES THAT A PROGRAM NAMED IS TO BE CORRECTED.	
		ACTION=REP	SPECIFIES THAT A PROGRAM IN THE FILE IS TO BE REPLACED.	
		ACTION=DEL	SPECIFIES THAT THE NAMED PROGRAM IS TO BE DELETED.	
		ACTION=NEW	SPECIFIES THAT A NEW VERSION OF AN EXISTING PROGRAM IN THE FILE IS TO BE CREATED.	
	NEW PROG	6 CHARACTERS	GIVES THE NAME OF THE NEW PROGRAM BEING CREATED ON THE FILE.	REQUIRED FOR ACTION=NEW
	PROG	PROGRAM NAME	GIVES THE NAME OF THE PROGRAM TO BE DELETED FROM THE FILE.	REQUIRED FOR ACTION=DEL. DOES NOT APPLY TO OTHER UPDATE ACTIONS
	LIST	LIST=NO	SPECIFIES THAT NO LISTING OF INPUT DATA IS TO BE PRODUCED.	OPTIONAL
		LIST=YES	SPECIFIES THAT A LISTING OF INPUT DATA IS TO BE PRODUCED.	
DATE STATEMENT	DATE FIELD	8 CHARACTERS	THIS IS THE DATE TO BE PRINTED ON THE LISTING.	OPTIONAL

EXECUTABLE PROGRAM FILE UPDATE

The following paragraphs provide a general description of the Executable Program File Update, followed by descriptions of the functions, the use of visibility, and job control language.

General Description

The Executable Program File Update (also called the Binary Run File, or BRF, Update) creates and maintains on Mass Storage a file of machine-executable programs in the proper format to be accessed and loaded by the Operating System Supervisor. A file of machine-executable programs is called an Executable Program File. The Master Executable Program File is referenced here as the "Master File."

The BRF Update can add a program or segment to the Master File, or delete one from the Master File. It can replace a program or segment in the Master File. Additions to and replacements for the Master File can come either from magnetic tape or Mass Storage. The BRF Update can change the name of a program or segment in the Master File.

As a part of the Program Development Subsystem, the BRF Update operates in the unbatched mode. That is, one program or segment at a time is updated; each program or segment is operated upon independently of those preceding it.

Executable Program File Update Functions

This paragraph provides the necessary definitions of the Input and Output files used by the Executable Program File Update, along with definitions of Update Units and Keys. Following the definitions are descriptions of the Update functions.

1. Input and Output Files (Definitions). The following paragraphs define the Input and Output files of the BRF Update.

   a. Master File - The master file for the update operation, also called the "Master BRF", is a partitioned sequential file on mass storage. The standard Honeywell physical record format is used. Each program segment in the file is one member. Normally, the master file is the resident file (the file containing the Supervisor, the system programs of the operating system, and the user's programs). The sequence of segments in the Master BRF is not under the user's control. Segments are identified by two keys: program name (6 characters) and segment name (2 characters). Segments may be accessed for updating by any one of several combinations of these keys. These key combinations are defined in Paragraph 2. It is not possible to have two segments with the same two key values in one Master BRF except when visibility is used. Visibility, a third key that may be used, is discussed in this section.

   b. Input - The input to the BRF Update consists of control statements and transaction input. These are described in the following paragraphs:

      (1) Control Statements - The control statements specify the update action to be performed and any options used. Control Statements must appear in the Card Reader. These are more fully described in Section 6 of this manual.

(2) Transaction Input - The transaction input to the update operation consists of one or more segments, in machine-executable format, to be inserted into the Master BRF. These can be either additions or replacements. Transactions input can be either on tape (Transaction BRT) or on mass storage (Transaction BRF). When the transaction input is on tape, the specified programs or segments are located by searching from the beginning of the tape. When the transaction input is on mass storage, the specified programs are located through the member index of the Transaction BRF. The structure of a Transaction BRF is the same as that of a Master BRF. They differ only in file name. The Transaction BRF normally is the output of the translators of the Program Development Subsystem and is called the Go File.

2. Update Units and Keys (Definitions). The functions of the BRF Update are defined in terms of update units. An update unit may be either an entire program or a single segment. The updating key used to specify an entire program is a 6 character program name. The updating key used to specify a single segment is a 6 character program name and a 2 character segment name.

3. Update Actions. The update functions listed in the following paragraphs can be performed on update units in the Master BRF. In all cases where a segment is marked deleted, the space occupied by the segment does not become available.

To make the space available, the Master BRF must be reorganized by using the normal File Support routines described in Section III of this manual.

a. Add - An update unit can be added to the Master BRF. The transaction input for this unit appears either in a Transaction BRF or in a Transaction BRT. The specified keys of a unit to be added must not duplicate the values of the same keys for any unit in the Master BRF. For example, if a program unit is to be added. the Master BRF must not contain any other program of the same name. If a program unit is specified, the segment names are obtained from the transaction input. The user has no control over the physical placement of a unit to be added to the Master BRF. The user accesses a unit by name, not by relative position

b. Delete - An update unit can be deleted from the Master BRF. The space occupied by a deleted unit is marked so that the unit can no longer be accessed. In determining what segments in the Master BRF should be deleted, only the specified keys are considered. For example, if a program name is specified, all segments of the named program will be deleted.

c. Replace - An update unit can be replaced in the Master BRF by a like unit of the same name (key). The transaction input for this unit appears either in a Transaction BRT or on a Transaction BRF. The existing unit is marked as deleted and the new unit is added to the Master BRF. The replacement unit is always assigned the same name (keys) as the unit replaced. When a different name is desired, the Replace action may be

followed by a Rename action. When a program unit is being replaced, the new unit does not have to correspond to the old unit in the number and names of its segments.

d. Rename - An update unit can be renamed. The unit to be renamed can be located by any permissible combination of keys. When a program unit is renamed, the program name may be changed but segment names remain the same. When a segment unit is to be renamed, the segment name may be changed but the program name remains the same.

Visibility

Visibility may be used as a key, in addition to the program name and the segment name. Thus, 1, 2, or 3 keys may be specified. The visibility key is 6 characters. The following key combinations are permitted:

Key Combinations	Update Unit
Program Name	Program
Program Name, Visibility	Program
Program Name, Segment Name	Segment
Program Name, Segment Name, Visibility	Segment

Any keys not specified in the identification of a unit are assumed to be of no significance. For example, when a program is identified only by program name, the BRF Update does not check visibility in locating the program in the Master BRF. Thus, when a program is deleted and visibility is not specified, all segments with the specified program name are deleted.

When visibility is specified and the update unit is a program, there may be two or more programs with the same program name in the Master BRF, but they must differ in visibility.

When visibility is specified and the update unit is a segment, there may be two or more segments of the same program and segment names, but they must differ in visibility.

If visibility is not specified in an Add action, the unit is added under a standard visibility. In a replace operation, no part of the key (including visibility) may be changed.

In the Rename action, the unit to be renamed can be located by any permissible combination of keys. The visibility may be changed, whether the unit is a program or a segment.

Executable Program File Update Function Job Control Statement

FORMAT

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___.___.___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				FUNCT	EXECUTABLE,	
2					ACTION={ADD / REP / DEL / REN},	Optional.
3						
4						
5						
6					GO={BRT / BRF},	Optional.
7						
8					PROG=program-name,	
9					SEG=segment-name,	Optional.
10					VIS=visibilities,	Optional.
11					NEWPROG=new-program-name,	Optional.
12					NEWSEG=new-segment-name,	Optional.
13		E			NEWVIS=new-visibilities,	Optional.
14						
15						

DESCRIPTION

Function Statement

The Function Statement identifies the function to be performed as the Executable Program File Update. If the function is to be performed under completely standard conditions - that is, if all the parameter standard assumptions are acceptable - certain job control parameters may be omitted. When there are to be exceptions to the standard assumptions, additional parameters must be included.

ACTION PARAMETER: The Action Parameter (ACTION) specifies the update action to be performed. When it is omitted the ADD action is performed. The acceptable values of this parameter are as follows:

ADD	Add the program to the Executable Program File
REP	Replace the program
DEL	Delete the program
REN	Rename the program

GO PARAMETER: The update can accept two forms of machine-executable input: a Binary Run Tape (BRT) on type 2Ø4B magnetic tape, or a Binary Run File (BRF) on mass storage. When this parameter is omitted, a BRF input is assumed.

UPDATE UNIT KEY PARAMETERS: It is necessary to specify the keys by which a unit to be added, replaced, deleted, or renamed can be identified. Each of three possible keys is considered as a single parameter. The permissible combinations of keys are listed in this section of the manual. The Program Name Parameter must always be specified but the Segment Name and Visibilities Parameters need not be stated unless they are desired.

The Program Name Parameter is a 6 character value identifying a program. The Segment Name Parameter is a 2 character value identifying a segment within a program.

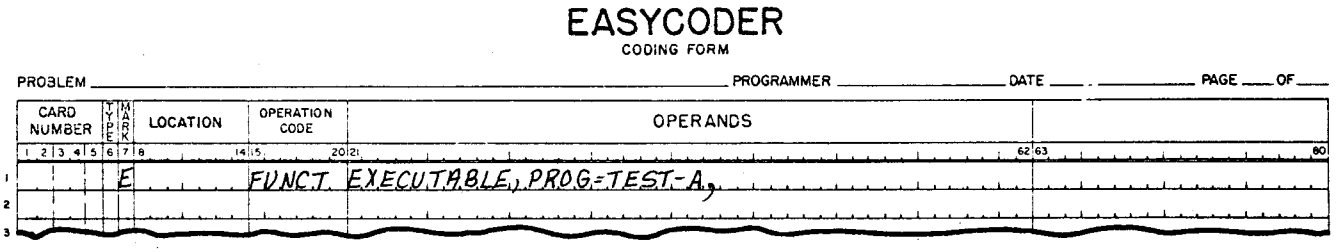
The Visibilities Parameter is a value used to distinguish between units otherwise similarly named. As a parameter of the Function Statement, it has a variable length of from 1 to 36 characters. The characters can be chosen from the letters (A - Z) and the digits (0 - 9). The parameter value may also have one of the two special values:

Δ	All visibility bits will be zero
*	All visibility bits will be 1

NEW UPDATE UNIT KEYS PARAMETERS: These parameters apply only to the Rename Action and at least the new program name must be specified. The new segment name and the new visibilities need not be specified unless they are desired. These parameters are exactly the same as described for the Update Unit Key Parameters described previously in this paragraph.

Executable Program File Update Function Job Control Language Examples

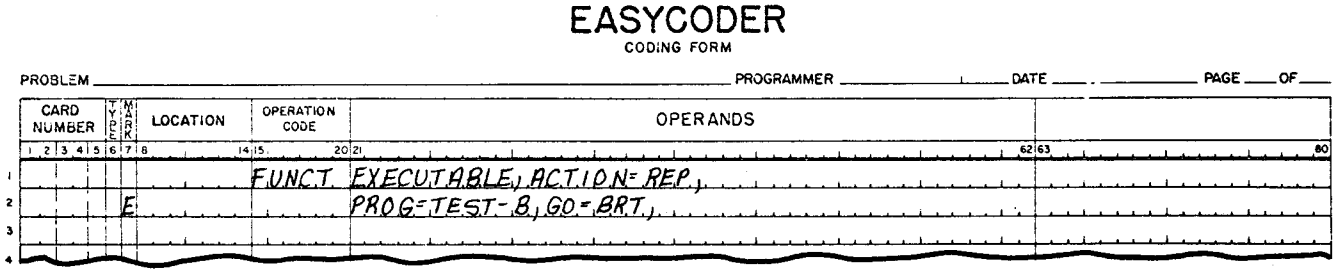
The following statements cause the program TEST-A to be added to the master BRF. The source of TEST-A is the mass storage GO File.


EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5 6 7 8			14 15	20 21	62 63	80
1		E		FUNCT. EXECUTABLE, PROG=TEST-A,		
2						
3						

The following statements cause the program TEST-B on the Master BRF to be replaced by a program of the same name, found on the Transaction BRT.


EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5 6 7 8			14 15	20 21	62 63	80
1				FUNCT. EXECUTABLE, ACTION=REP,		
2		E		PROG=TEST-B, GO=BRT,		
3						
4						

The following statements cause the program TEST-C whose visibility is F to be deleted from the Master BRF.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5 6	7	8	14 15	20 21	62 63	80
1				FUNCT	EXECUTABLE,ACTION=DEL,PROG=TEST-C,	
2		E			VIS=F,	
3						
4						

The following statements cause the segment Ø6 of program TEST-B on the Master BRF to be replaced by the identically named program segment on the Transaction BRT.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5 6	7	8	14 15	20 21	62 63	80
1				FUNCT	EXECUTABLE,ACTION=REP,	
2					PROG=TEST-B,SEG=Ø6,	
3		E			GO=BRT,	
4						
5						

The following statements cause the segment S1 of program TEST-C on the Master BRF to be renamed as Segment SØ.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5 6	7	8	14 15	20 21	62 63	80
1				FUNCT	EXECUTABLE,ACTION=REN,	
2		E			PROG=TEST-C,SEG=S1,NEWSEG=SØ	
3						
4						

Table 4-4 contains a summary of job control statements for the executable program file update function.

Table 4-4.
Executable Program File Update Function Job Control Statements

JOB CONTROL STATEMENT	PARAMETER	VALUE	DESCRIPTION	REQUIREMENTS
FUNCTION STATEMENT	EXECUTABLE	EXECUT-ABLE	SPECIFIES THAT THE EXECUTABLE PROGRAM FILE UPDATE FUNCTION OF PROGRAM DEVELOP-MENT IS TO BE PER-FORMED.	REQUIRED.
	ACTION	ACTION= ADD	SPECIFIES THAT THE NAMED PROGRAM IS TO BE ADDED TO THE FILE.	OPTIONAL
		ACTION= REP	SPECIFIES THAT THE NAMED PROGRAM IS TO BE REPLACED IN THE FILE.	
		ACTION= DEL	SPECIFIES THAT THE NAMED PROGRAM IS TO BE DELETED FROM THE FILE.	
		ACTION= REN	SPECIFIES THAT THE NAMED PROGRAM IS TO BE RENAMED IN THE FILE.	
	GO	GO=BRT	THE INPUT TO THE UPDATE FUNCTION IS ON A BRT.	OPTIONAL
		GO=BRF	THE INPUT TO THE UPDATE IS ON THE BRF.	
	PROG	6 CHAR-ACTER PROGRAM NAME	NAMES THE PROGRAM ON WHICH THE UP-DATE ACTION IS TO BE PERFORMED.	THESE ARE THE UPDATE UNIT KEY PARA-METERS. PROG MUST BE SPECIFIED BUT SEG AND VIS ARE OPTIONAL
	SEG	2 CHAR-ACTER SEGMENT NAME	THIS IS THE NAME OF THE SEGMENT IN THE PROGRAM TO BE UPDATED.	
	VIS	A - Z OR Δ OR *	THIS DISTINGUISHES BETWEEN UNITS WITH SIMILAR NAMES.	

Table 4-4.
Executable Program File Update Function Job Control Statements
(continued)

JOB CONTROL STATEMENT	PARAMETER	VALUE	DESCRIPTION	REQUIREMENTS
FUNCTION STATEMENT (cont.)	NEWPROG	6 CHARACTER PROGRAM NAME	SEE PROG ABOVE	THESE ARE THE NEW UPDATE UNIT KEYS PARAMETERS AND ONLY APPLY TO THE RENAME ACTION. NEWPROG IS REQUIRED, NEWSEG & NEWVIS ARE OPTIONAL.
	NEWSEG	2 CHARACTER SEGMENT NAME.	SEE SEG ABOVE	
	NEWVIS	A - Z OR Δ OR *	SEE VIS ABOVE	

PROGRAM DEVELOPMENT PROGRAMMER'S PREPARATION INFORMATION

Allocation Of Files To Use Program Development

To run the entire Program Development Subsystem, the following five files must be allocated: System Residence File, Go File, Library File, Assembly Work File 1 and Assembly Work File 2. Descriptions of and the functions of these files are included in the following paragraphs.

SYSTEM RESIDENCE FILE

FUNCTION: This is the file from which the Supervisor loads programs, both systems programs and object programs. It is created and updated by the Mass Storage Executable Update.

FILE NAME: *DRS1RES

FILE TYPE: Partitioned sequential

BLOCK SIZE: 1 item

RECORD SIZE: 25Ø characters

ITEM SIZE: 25Ø characters

NO. OF DATA BLOCKS: D = 5T

where D = number of data blocks

T = number of machine language characters in all programs to be placed in this file, expressed in thousands of characters.

For example, if the file is to contain 4Ø programs, and these programs contain an average of 6ØØØ machine language characters each, then a minimum of 12ØØ blocks should be allocated for the loading data.

In practice, the file should be made larger, to allow for updating. The Executable Update Program does not make available the space occupied by deleted or replaced programs. This space can be made available only by reorganizing the file, through the use of the appropriate File Support functions. The frequency of reorganization can be reduced by allocating a larger number of blocks to the system residence file. The optimum size for a given installation depends on frequency of updates, as well as the needs of other files on the same disk pack.

The number of blocks required for the software in the first release will be specified later.

NO. OF MEMBER INDEX BLOCKS:

$$M = \frac{S+2}{10}$$

where M = number of blocks required for the member index
S = number of segments in the file.

For example, if the file is to contain 4Ø programs and these programs contain 2 segments each, then a minimum of 9 blocks should be allocated for the member index.

GO FILE

FUNCTION: This is a work file containing the machine language output of Mass Storage Easycoder Assembly. It contains only one program at a time. It is a transaction input file to the Executable Update.

FILE NAME: *DRS1GO

FILE TYPE: Partitioned sequential

BLOCK SIZE: 1 item

RECORD SIZE: 25Ø characters

ITEM SIZE: 25Ø characters

NO. OF DATA BLOCKS: D = 5T,

where D = number of data blocks
T = number of machine language characters in the largest single program to be assembled.

NO. OF MEMBER INDEX BLOCKS:

$$M = \frac{S+2}{10}$$

where M = number of blocks required for the member index
S = largest number of segments in any single program to be assembled.

LIBRARY FILE

FUNCTION: This file contains unspecialized macro routines. It is updated by the Mass Storage Library Update. It is accessed, as the source of macro routines by the library processor portion of Mass Storage Easycoder Assembly.

FILE NAME: *DRS1LIB

FILE TYPE: Partitioned sequential

BLOCK SIZE: 3 items

RECORD SIZE: 25Ø characters

ITEM SIZE: 8Ø characters

NO. OF DATA BLOCKS: $D = \frac{C}{3}$,

where D = required number of data blocks
C = total number of card images in all macro routines to be placed in the file

NO. OF MEMBER INDEX BLOCKS:

$$M = \frac{R+2}{10} ,$$

where M = number of blocks required for the member index
R = total number of macro routines to be placed in the file.

In order to allow room for updating, more space than the required minimum should be allocated.

ASSEMBLY WORK FILE 1

FUNCTION: This file is used whenever the library processor function of assembly is exercised. It contains the specialized symbolic card images for a single program, including the non-macro portion of the program.

FILE NAME: *DRS1WORK1

FILE TYPE: Partitioned sequential

BLOCK SIZE: 3 items

RECORD SIZE: 25Ø characters

ITEM SIZE: 8Ø characters

NO. OF DATA BLOCKS: $D = \frac{C}{3}$,

where D = required number of data blocks
C = number of card images/after macro specialization, in the largest single program to be assembled.

NO. OF MEMBER INDEX BLOCKS

$$M = \frac{2L+2}{10}$$,

where M = number of blocks required for the member index
L = largest number of macro calls in any single program to be assembled. Nested macros are counted the same as first level macros.

ASSEMBLY WORK FILE 2

FUNCTION: This file contains the intermediate results passed from one phase of Easycoder Assembly to another.

FILE NAME: *DRS1WORK2

FILE TYPE: Sequential

BLOCK SIZE: 1 item

RECORD SIZE: 250 characters

ITEM SIZE: 750 characters

NO. OF DATA BLOCKS: $D = \frac{C}{6}$,

where D = required number of data blocks
C = number of card images, after macro specialization, in the largest single program to be assembled.

SECTION V

SERVICE ROUTINES

This section of the manual describes several program packages which are part of the Mass Storage Operating System. These three programs are grouped together in this section for convenience of presentation. The three program packages are:

1. Volume Preparation
2. Mass Storage Sort
3. Mass Storage Edit

VOLUME PREPARATION

The Volume Preparation program prepares a mass storage volume for use under the data management conventions of the operating system. Volume preparation must be performed at least once for every mass storage volume (at the time the volume is first entered into the Mass Storage Operating System).

Functional Description

FUNCTIONS

The Volume Preparation program performs the following functions:

1. Formats all tracks of the volume with standard records.
2. Checks for bad surface areas.
3. Writes the volume label.
4. Creates the volume directory.

TRACK FORMAT

The standard track format written by the Volume Preparation program is Honeywell standard size records plus a track linking record. The track linking record links to the next physically consecutive track.

BAD SURFACE AREAS

After each track is formatted, it is read back in the "verify" mode. If a read error occurs that cannot be corrected by reformatting, a listing is produced giving the bad cylinder and track address and the preparation operation is terminated.

Volume Preparation Function Job Control Statements

FORMAT

Honeywell
ELECTRONIC DATA PROCESSING

EASYCODER
CODING FORM

PROJECT _____ Dept. _____ PROGRAMMER _____ DATE / / PAGE ___ OF ___

CARD NUMBER	TYPE	WORK	LOCATION	OPERATION CODE	OPERANDS	
1				VOLUMENAME=volume-serial-number,		
2					MAXF=maximum-number-of-files,	Optional.
3					DEVADD=(pcu,drive),	Optional.
4		E		DAY	yyddd,	This Statement is optional.
5						
6						
7						
8						
9						
10						

DESCRIPTION

Volume Statement

The Volume Statement gives parameters pertaining to the volume to be prepared. Note that the Volume Preparation Function can also be operated in the MOD - 1 Tape Resident Operating System under the Floating Tape Loader/Monitor C. In that case, the Execute Statement is replaced by the console call card of the tape resident system.

NAME PARAMETER: The Name Parameter (NAME) gives the volume serial number to be written in the Volume Label Record on the mass storage volume. The volume serial number is 6 characters long.

MAXIMUM NUMBER OF FILES PARAMETER: The Maximum Number Of Files Parameter (MAXF) gives the maximum number of files expected to be stored on this volume. This is specified as a number in decimal in the range from 1 to 86. When this parameter is omitted, a value of 26_{10} is used.

Sufficient space is allowed in the Volume Directory to accommodate the number of files specified. The user should ensure that this value is adequate, since the only means for increasing the number of files allowed is to unload all files, run the volume preparation function with a larger maximum specified, and reload all files.

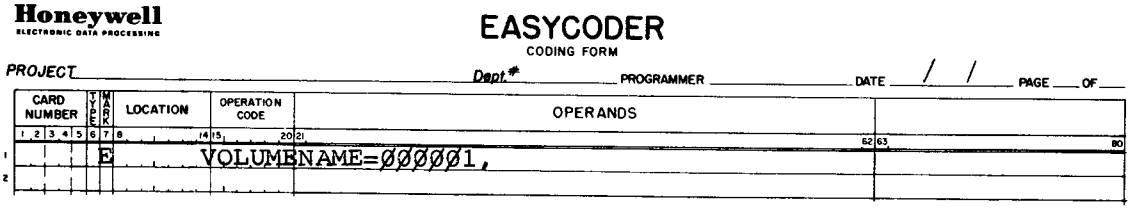
DEVICE ADDRESS PARAMETER: The Device Address Parameter (DEVADD) gives the peripheral control unit number in two octal digits. The drive number is given in one octal digit. When this parameter is omitted, the values of Ø4 for the pcu and Ø for the drive are used.

Day Statement

The Day Statement specifies the creation date of the Volume Directory. When the Day Statement is omitted, the creation date is taken from the Current Date Field of the Supervisor. The yy portion of the value gives the year of the creation in decimal digits and the ddd portion the day of the creation (counting from January 1 as day ØØ1).

Volume Preparation Function Job Control Language Example

The following job control statement will cause the Volume Label Record to be written as ØØØØØ1. The maximum number of files to be stored on this volume is 26 and the device address is the standard address, pcu Ø4 - drive Ø. The day of creation of the Volume Label is the same as the Current Date Field of the Supervisor.


Honeywell
ELECTRONIC DATA PROCESSING

EASYCODER
CODING FORM

PROJECT Dept.# PROGRAMMER DATE / / PAGE OF

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62 63 ... 80
1		E		VOLUMENAME=ØØØØØ1,	
2					

MASS STORAGE SORT

The Mass Storage Sort uses and obeys the data management system. Only files created and maintained as part of that system may be input to the sort.

Functional Description

GLOSSARY OF TERMS

Before describing this package and giving the language elements, a few terms used in the text are defined:

Sort-Keys. Those fields of an item that determine the order of the sort output and are part of the sort output.

Extract-Fields. Those fields of an item that may be selected to be part of the sort output. Extract and sort-key fields must be distinct.

Sort-Item. The internal item of the sort that has been derived from a source item. It contains sort-key fields and may contain extract-fields and a source item address.

Residue. That data of an item that is not contained in a sort-key field.

Fetch. Fetch is an input macro that facilitates the processing of the mass storage edit as well as the input file from which the sort file was generated. It is described in detail in the following paragraphs.

USE OF MASS STORAGE SORT

The sort is primarily a "key" sort with certain optional capabilities. A key[1] sort is such that its end result is a file which contains only an ordered collection of keys which have appended to them the address of the source item. If the desired output of the sort is an ordered collection of the original source items, a "fetch" may be used to retrieve each source item in the sort ordered sequence. The Mass Storage Sort offers such a key-sort, but one of the principal extensions it offers to this concept is that it allows for an output item that contains data from the source item in addition to the keys. This type of sort can be described as an "extract" sort. Specifically, a user may select up to 11 non-key fields, or the residue of the item where the keys are excluded. These non-key fields appear with the ordered keys as the sort output with the corollary that if the residue of the item is specified, the output item is a source item in its original format. Optionally, the address of the source item may be dropped. The use of the "extract" sort allows those applications which do not require access to the entire item to eliminate those elements of the "fetch" which retrieve the source item, thus implying a considerable saving in time.

[1]Keys in this specific context means the fields of an item which are used to obtain the required ordering.

SUMMARY OF CAPABILITIES

The Mass Storage Sort performs the following functions:

1. Sort fixed length items.
2. Sort according to sort-key fields up to a maximum of 10 in ascending, descending, or mixed sequence.
3. Allows up to 11 extract fields or the residue of an item to be an element of the sort-item.
4. Permits the selection of only those items for the sort that have a field with a specific content.
5. Permits deletion from the sort of those items that have a field with a specific content.
6. Allows user's own coding on an input item-by-item basis.
7. Allows user's own coding on an output sort item-by-item basis.
8. Accepts input from a mass storage device.
9. Provides output in a work-file area on the mass storage device.
10. Preserves the original input file.

FUNCTION BY PROGRAM

The Mass Storage Sort processes a file on the mass storage device. The file must be organized according to data management conventions. From each item that is a valid entrant to the sort, a sort-item is developed which contains as a unit the sort-key fields. An address of the source item may be appended to that sort-key unit, and the extract or residue may be prefixed to that same unit.

The sort finally produces one string or file of logically sequenced sort-items in a work area. Ordering in the logical sequence is a function of the Honeywell Collating Sequence

(binary ØØØØØØ to 111111) as well as the sort-key fields. A user requiring some other collating sequence is able to achieve this through own-coding. At the completion of the sort process, the sort-items may then be made available to the user through a specialized "fetch" which may also execute the retrieval of the associated source item.

FUNCTIONS BY SEGMENTS

The sort may be considered as consisting of two logical segments, presort and merge, where the latter is subdivided into two phases, ONE CYLINDER (1), and MULTI-CYLINDER.

Presort

The presort develops a sort-item from each item that is acceptable to the particular sort application. These sort-items are arranged into ordered strings which have a length that is determined by the memory available to the sort and the requirement that an efficient covering is made of the available area of a cylinder.

Own coding can be entered during the presort. It permits the inspection, modification, deletion, and addition of items.

Merge One-Cylinder

The merge one cylinder segment merges the strings that exist on one cylinder until they are reduced to a string of sort-items. Memory available to the MERGE-ONE segment is a factor in the determining of the way of the merge.

Merge Multi-Cylinder

The merge multi-cylinder employs two cylinders merging until the final string is produced. It employs a sliding buffer technique that ensures optimum activity on a cylinder once it has been accessed.

This segment utilizes the memory that is available by creating as many buffers as it can.

A cylinder is defined as the number of tracks made available to the read/write heads by the execution of a single "seek" instruction.

When the final string is being created, any reorganization of the sort item that might be required is effected. If Merge own-coding is requested, it becomes active during this final phase and permits access to the sort-item after it has been subjected to any necessary reorganization.

THE FETCH MACRO

Fetch is an input macro that facilitates the processing of the Mass Storage Sort output-file as well as in the input file from which the Sort file was generated. It makes available the sort-item and its associated source-item in an item-by-item mode. Hence access can be made to the items of the input-file in the logical sequence promoted by any sort application. The user may exercise options that restrict his access to the sort-item only or just to the original input items.

Fetch Exits

Linkage between the user's program and the specialized version of Fetch that is embedded in that program is effected through Fetch exits. These exits are addresses in the user's program to which Fetch branches when it executes the function associated with a particular exit. (The addresses are specified through the parameters of the Fetch macro call card.) The two principal exits are named:

Examine sort-item

Examine source-item

Examine Sort Item. When Fetch branches to the "sort-item" exit, a sort-item has been brought into main memory and the address of the left-hand end of that item is available to the user through an index register that he has selected. The user may now process that sort-item and then return to Fetch for the execution of his next request. Two returns are available, normal return and delete return.

Normal Return. If the user wishes to have access to the source-item that is associated with the current sort-item, he executes a "normal return". This is done by branching to a location whose address was to be found in the contents of the B-address register when the "sort-item" exit was made. Fetch will then return to the user's program through the source-item exit with the main memory address of the requested item stored in a user-designated index register. However, if the source-item exit was not specified in the Fetch macro call card, the next exit from Fetch will be through the sort-item exit but with the main memory address of the next logically sequential sort-item.

Delete Return. This return has meaning only if a source-item exit has been specified. When this return is used, it is assumed that the user is not interested in the source-item associated with the current sort-item. Therefore, the next exit from Fetch will be through the "sort-item" exit with the main memory address of the next logically sequential sort-item.

The address of the "delete return" differs from that of the normal by a constant equal to the length of the address mode plus one.

Examine Source-Item. Fetch branches to the source-item exit when an input item is available in main memory. The main memory address of that item is given through a user-specified index register. Input items are at the disposal of the user and are presented in the sequential order of a particular sort application. After the user has processed an item, the return to Fetch is made by using the "normal" return as described in paragraph "Normal Return" above.

Specialization of Fetch

The specialization of Fetch is principally achieved through ascribing values to the parameters of the Fetch macro call statement. However, certain parameters required for the full specialization can be ascertained only after the Sort has produced its sort-item file; therefore, the values for these parameters are written by the Sort on a mass storage device. The block containing these values is accessed by Fetch and the final specialization is achieved. A user may modify this final version by requesting an exit before any file processing begins. Control is returned to Fetch by using the "normal return".

Initiation of Fetch

The user initiates Fetch by branching to the tagged location specified in the location fields of the Fetch macro call. Fetch exercises control until it can meet the demand associated with a specified exit. When that exit is made, control lies with the user's program until a Fetch return is made.

Summary of Fetch Exits

There are two sets of exits, normal processing and error promoted.

1. Normal Processing.

   a. File open (optional), normal return.

   b. Examine sort-item (optional) a) normal return, b) delete return - inhibits the user's access to the associated source-item where the "examine source-item" is active.

   c. Examine source-item (optional), normal return.

   d. Close file (mandatory), no return.

Although the "examine sort-item" and "examine source-item" exits are classified as optional, Fetch requires that at least one of them is active.

2. Error Promoted.

   These two further exits are both optional.

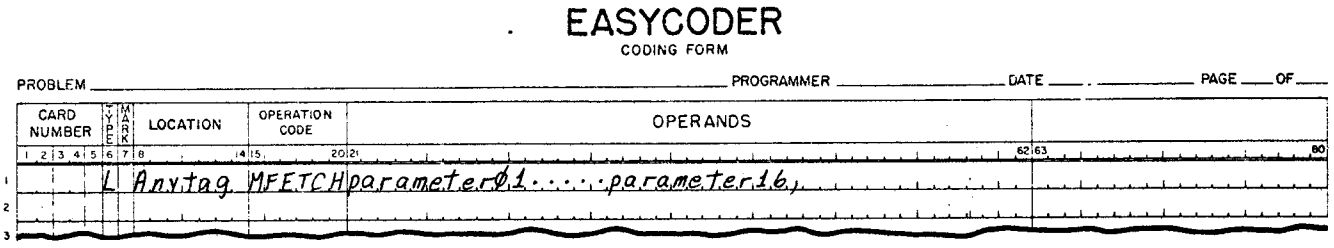
   a. Uncorrectable read error on the sort-item file.

   b. Uncorrectable read error on the source-item file.

Fetch Macro

The Fetch Macro has the name MFETCH. One macro call is required per program using the Fetch Function (Sorting).

FORMAT



DESCRIPTION

Parameter Number	Value
01	Two-character prefix applied to all symbols in the Fetch macro. This parameter is required.
02	Address for own-code exit when the Fetch has been fully specialized; symbolic tag or decimal address. This parameter is optional.
03	Address for own-code exit for sort-item examination; a symbolic tag or decimal address. This parameter is optional.
04	Address for own-code exit for source-item examination; symbolic tag or decimal address. This parameter is optional; however, at least one of the exits specified by parameters 03 and 04 must be specified. If this exit is to be utilized, then the sort-item format specified to the preceding sort must include the source-item address.
05	Address for own-code exit when the end of the sort-item file is reached; symbolic tag or decimal address. This parameter is required.
06	Address of own-code exit for uncorrectable read error on sort-item file; symbolic tag or decimal address. This parameter is optional.
07	Address of own-code exit for uncorrectable read error on source-item file; symbolic tag or decimal address. This parameter is optional.
08	Buffer size required for reading blocks from the sort-item file; up to 4 decimal characters.
09	Buffering mode for reading sort-item file: SINGLE - Single buffering. DOUBLE - Double buffering. The default value is DOUBLE.
10	Index register used for sort-item file; one decimal character in the range 1 to 4. This index register will contain the address of the high-order character of the sort-item at the own-code exit for each sort-item. This parameter is required because at least one of the two examination exits must be specified and this parameter is used as a default value if parameter 13 is not written.
11	Buffer size required for reading blocks from the item file; up to 4 decimal characters.

Parameter Number	Value
12	Buffering mode for reading source-item file: SINGLE - Single buffering. DOUBLE - Double buffering. The default value is DOUBLE.
13	Index register used for source-item file; one decimal character in the range 1 to 4. This index register will contain the address of the high order character of the source-item at the own-code exit for each source-item. The default value is the same index register as that specified by parameter 10.
14	Device type on which the input files to the Fetch will be found. The exact form of this parameter and its default value will be specified in a later revision.
15	The address of a constant (DCW) in the user's program containing the name of the work file which holds the sort-item file input to the Fetch; symbolic tag or decimal address. The DCW must be 10 characters long. When two work files are specified to the preceding sort, the name defined by the DCW must be that of the second file. The default value is that the work file name is *SORTWORK.
16	The address of a constant (DCW) in the user's program containing the device address of the device on which the work file named in parameter 15 can be found; symbolic tag or decimal address. The DCW must be 2 characters long and contain the device address in exactly the form it would appear in a PDT instruction. The default value is peripheral control unit 04, drive 0.

Sort Function Job Control Statements

FORMAT

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

CARD NUMBER	TYPE	MASK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7 8	14	15 20	62 63	80
				SORT	HMA={high-address / nnM},	PMA is an
						optional parameter
					SEQ={A / D},	Optional.
					ITADD={NO / YES},	Optional.
				FILE	IN,	
					NAME=file-name,	
					DEVADD=(pcu,drive),	Optional.
					PW=password,	Optional.
				FILE	WORK1,	Optional State-
					NAME=file-name,	ment.
					DEVADD=(pcu,drive),	
				FILE	WORK2,	Optional State-
					NAME=file-name,	ment.
					DEVADD=(pcu,drive),	
				FILE	INFO,	Optional State-
					DEVADD=(pcu),	ment.
				FIELDS	KEYS=((position,length<,R>)	
					<,(position,length,R)...>),	
					EXTR=((position,length).....),	Optional.
					SEL=(position,VAL=aa...a),	Optional.
					DEL=(position,VAL=aa...a),	Optional.
				EXITS	PSOPEN=address,	Optional State-
					PSIEM=address,	ment.
					MERGE=address,	
					PROG=program-segment-name,	
					VIS=visibility-mask,	

DESCRIPTION

Sort Statement

A Sort Statement is required as the first statement (except for the Execute Statement) in the job control file. Even if the standard values are usable for all parameters, there must still be a Sort Statement to confirm that the parameters are intended for a sort operation.

HIGH MEMORY ADDRESS PARAMETER: The High Memory Address Parameter (HMA) defines the highest memory address available to the sort. However, if an own-code program is resident during a phase of the sort and the base address of that program is lower than that specified by the HMA parameter but is higher than the base address of the sort itself, then the base address of that program less one is taken as being the highest memory address available to that phase.

The high-address value of the parameter is written as a 6 character decimal number and represents the highest address available. Optionally, the number of 16K memory modules available (nnM) can be written. The highest address available then will be computed by the sort, and will be one less than nnM times 4Ø96.

The standard assumption is that the HMA value is to be taken from the Supervisor Communication Area Field that specifies the base address of object program memory. When HMA is specified but has a higher value than the address in the Supervisor, the Supervisor value is used.

SEQUENCE PARAMETER: The Sequence Parameter (SEQ) describes the primary sorting sequence. When the A value is used, the sorting sequence is ascending. When D is used, the sequence is descending. When the parameter is omitted, the ascending sequence is used.

ITEM ADDRESS PARAMETER: The Item Address Parameter (ITADD) indicates whether the mass storage address of the item is to be appended to the sort item or not. When YES is the specified value of the parameter, the address will be included with the sort item. When NO is specified, the item address is not appended to the sort item. The standard assumption is that the item address will be appended to the sort item. The use of NO as the parameter value is only meaningful if the application does not require accessing the input file after completion of the sort. See the description of the Fetch Macro on pages 12-13 of this section.

File Statements

The File Statements specify information about the input and output files used by the sort. The I/O function name, which is the first parameter of each File Statement, specifies to which file the statement refers.

INPUT FILE STATEMENT: The Input File is the file of input data to the sort. The parameter IN identifies the File Statement as refering to the input file. The File Statement for the input file must be present; there is no standard assumptions about this file.

Name Parameter: The Name Parameter (NAME) specifies the file name of the input file to the sort. The file name can be up to 1Ø characters. Trailing spaces automatically are added.

Device Address Parameter: The Device Address Parameter (DEVADD) specifies the physical device address of the volume containing the input file. The peripheral control unit number (pcu) is specified in 2 octal digits. The I/O bit is not required, but all other bits must be specified. The drive number is specified in 1 octal digit. When this parameter is omitted, the standard device address is pcu Ø4 and drive Ø.

Password Parameter: The Password Parameter (PW) defines the password of the sort input file. The password can be up to 8 characters long. When the parameter is omitted, no password checking is done.

WORK FILES STATEMENTS: The sort uses work area on mass storage. The work area can be composed of more than one file, but the total number of units of allocation of all work files cannot exceed 5.

The parameter WORKn (where n is 1 or 2) specifies that the File Statement applies to the sort work files. When the parameter of the work File Statement is written as WORK1, the File Statement applies to Work File 1. When written as WORK2, the File Statement applies to Work File 2. When the File Statements for the Work Files are omitted, the standard assumption is that there is only one Work File, named *SORTWORK, on pcu Ø4, drive Ø. When only a WORK2 File Statement is used, the standard assumption for WORK1 is used.

Name Parameter: The Name Parameter (NAME) specifies the name of the sort Work File. It can be up to 1Ø characters long. When the Name parameter is omitted, the standard name *SORTWORK is used.

Device Address Parameter: The Device Address Parameter (DEVADD) specifies the physical device address of the volume containing the Work File. The peripheral control unit number (pcu) is written as 2 octal digits. The I/O bit is not required, but all other bits must be specified. The drive number is written as 1 octal digit. When this parameter is not specified, the standard device address of pcu Ø4 drive Ø is used.

INFORMATION FILE STATEMENT: The Information File is a printed program history. The parameter INFO identifies the File Statement as referring to the Information File. When the Information File Statement is omitted, no program history will be printed.

Device Address Parameter: The Device Address Parameter (DEVADD) specifies the physical device address of the Information File. The Information File must be on a Printer, so no drive number is required. The peripheral control unit number (pcu) is written as 2 octal digits. All 6 bits are required. When the Device Address Parameter is omitted, the standard pcu number of Ø2 is used.

Fields Statement

Parameters belonging to the Fields Statement define the functions of various fields of the input item in relation to the sort operation.

KEYS PARAMETER: The Keys Parameter (KEYS) specifies the sort-key data and is followed by a list. Up to 1Ø sort-key fields may be specified. Their order in the list indicates their sort significance. The order is in decreasing importance. Keys are assumed to be sorted in the order defined by the primary sequence of the sort unless a "reverse" key is specified. A "reverse" key will be sorted in the reverse sequence from the primary order.

The Postition Value of the Keys Parameter is written in 4 decimal digits and gives the position of the left-most (high order) character of the Key Field in the item. The first character in the item is considered to be in position ØØØ1.

The Length Value of the Keys Parameter is written in 2 decimal digits and gives the number of characters in the Key Field.

When the parameter value R is specified, the key with which it is associated is a "reverse" key.

There are no standard assumptions about the Key Parameter. If a single key is specified, the keyword parameter KEY is accepted.

EXTRACT FIELDS PARAMETER: The Extract Field Parameter (EXTR) specifies the Extract Field information. Extract Fields are fields of an input item that may be included in the sort-item but have no significance in determining the final sort order. Up to 11 Extract Fields may be specified and the order in the list determines their relative positions in the sort-item. When the user wishes the Extract Fields

to be all those fields of the item that have not been specified as keys, then the value given to the EXTR Parameter is ITEM. The specification of ITEM implies that the output of the sort will be a string of items in the same format as the original input item.

Key Fields and Extract Fields are mutually exclusive.

The Position Value of the Extract Parameter is written in 4 decimal digits and gives the position of the left-most (high order) character of the Extract Field in the item. The first character in the item is considered to be in position ØØØ1.

The Length Value of the Extract Parameter is written in 2 decimal digits and gives the number of characters in the Extract Field.

When the EXTR Parameter is omitted, no Extract Fields are used.

SELECT PARAMETER: The Select Parameter (SEL) indicates that the sort application allows only certain items of the input files to be accepted as input to the sort. The value of the SEL Parameter is a list consisting of 2 parameters. The first defines the position in the item of the field on which the selection is based. The second, which is prefixed by a keyword, defines the contents of the select field. Only 1 SEL parameter may be specified.

The Position Value of the SEL Parameter is written in 4 decimal digits and defines the left-most (high order) character of the field on which the selection is based. The first character of the item is defined as ØØØ1.

The Value Portion of the SEL Parameter (aaa...a) identifies the contents of the select field. Blanks are significant. The maximum number of characters that the VAL Parameter can define is 3Ø.

When this parameter is omitted, the standard assumption is that no select function is required.

DELETE PARAMETER: The Delete Parameter (DEL) defines the field that allows an application of the sort to bypass those items that have a specified value in one field. An item that meet the DEL specification is not deleted from the resident input file. The format of the DEL value is the same as that described for the SEL value. Only one DEL parameter may be specified. When this parameter is omitted, the standard assumption is that the delete function is not required.

Exits Statement

The Exits Statement specifies own-coding exits from the sort to user procedures. When the Exits Statement is omitted, there are no own-coding exits.

PRESORT OPEN PARAMETER: The Presort Open Parameter (PSOPEN) specifies an own-coding exit that permits the user to have access to the input file label. The standard assumption when this parameter is omitted is that there is not a Presort Open Exit.

The Address Value of the PSOPEN Parameter is written in 6 decimal digits and gives the address to which the sort will branch.

PRESORT ITEM PARAMETER: The Presort Item Parameter (PSITEM) specifies an own-coding exit that occurs for each accepted (selected) input item. Also, there is an exit after the presort has been specialized but before the first input item is accessed. When the Presort Item Parameter is omitted, the standard assumption is that there is no Presort Item Exit.

MERGE PARAMETER: The Merge Parameter (MERGE) specifies an own-coding exit that occurs for each output item when the final output of the sort is being created. The merge own-coding may be linked with the sort during the final phase of the merge, by loading it at that time. When the Merge Parameter is omitted, the standard assumption is that there is no Merge Exit.

The Address Value of the Merge Parameter is written in 6 decimal digits and gives the address to which the sort will branch.

PROGRAM PARAMETER: The Program Parameter (PROG) specifies that the named program is to be loaded for the own-coding merge exit. When the Program Parameter is omitted, the standard assumption is that the merge own-coding was resident in main memory before the final merge phase.

The Program Segment Name Value of the Program Parameter gives the program segment name of the program to be loaded.

VISIBILITY PARAMETER: The Visibility Parameter (VIS) specifies the visibility mask of the merge own-coding. When the Visibility Parameter is omitted, the standard assumption is that visibility is not used when loading the merge own-code program.

The Visibility Mask Value of the Visibility Parameter gives the visibility mask to be used when loading the merge own-coding program.

Sort Function Job Control Language Examples

Example 1.

The input file to the sort is named TRANSACT and the device address is the installation standard address. One work-file is available under the name *SORTWORK and its device address is the installation standard address. The highest memory address is to be taken from the Supervisor

Communication Area and the output sequence is to be ascending. The sort item will consist of one key with the hardware address of the associated input item appended.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	CONT	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6 7 8	14 15	20 21	62 63 80
			SORT	
			FILE	IN,NAME=TRANSACT
		E	FIELD	KEY=(2Ø,1Ø),

Example 2.

A sort is to be executed on the file whose name is TRANSACT and whose volume resides on a mass storage device with a peripheral address which is $(Ø7)_8$ for the control unit and Ø for the device. There are two work files with names and addresses as given. The high memory address is 9Ø45 and the output is to be in descending sequence. The final output will be a string of input items that have the characters LOBSTERΔPOT starting in postion 4Ø.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	CONT	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6 7 8	14 15	20 21	62 63 80
			SORT	HMA=9Ø45,SEQ=D,ITADD=NO,
			FILE	IN,NAME=TRANSACT,DEVADD=(Ø7,Ø),
			FILE	WORK1,NAME=SPARE1,DEVADD=(Ø4,1),
			FILE	WORK2,NAME=SPARE2,DEVADD=(Ø4,2),
			FIELD	KEYS=((3Ø,1Ø),(7Ø,6,R)),EXTR=ITEM,
		F		SEL=(AØ,VAL=LOBSTERΔPOT),

Example 3.

Sort the file TRANSACT but bypass as input to the sort those items that the characters ZZZZ begin in position 1Ø. Two work files are available, the primary work file is named *SORTWORK and the secondary

EXWORK. Both are on the installation standard device address. The sort-item consists of 2 Extract Fields followed by 3 key fields and the item address is appended. Merge own-coding is required and the sort is to call the own-code program. The highest memory address is 8(4Ø96)-1 = 32767.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ____ PAGE ____ OF ____

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1				FIELD	IN,NAME=TRANSACT,PW=ONLY ONE,
2				FILE	WORK2,NAME=EXWORK,
3				FIELD	KEYS=((3Ø,2),(1ØØ,5),(2Ø,4)),EXTR=((4Ø,1Ø),
4					(9Ø,8)),DEL=(1Ø,VAL=ZZZZ),
5				SORT	HMA=8M,
6				EXITS	MERGE=29ØØØ,PROG=MERGOCØ1,
7					
8					

SORT FUNCTION PROGRAMMER'S PREPARATION INFORMATION

Work Files

The sort can utilize two work files. If two work files are provided for the sort-work area, they may exist on the same volume or they may each be on a different volume. The device addresses associated with the two work files must have the same control unit. A work file must have been properly allocated through File Support as a sequential file with 25Ø character records.

UNITS OF ALLOCATION

The sort work area may be defined by five or less units of allocation. If the total number of units of allocation associated with the work files exceeds five, the extra units will be ignored apart from the sort using the last record of the last unit of allocation for internal control information. This record is also used to store information for Fetch.

When the number of units of allocation is five or less, the last record of the last unit of allocation is similarly used.

RELATIONSHIPS BETWEEN UNITS OF ALLOCATION AND SORT EFFICIENCY

There are no restrictions on the units of allocation, but, if they are to achieve optimum efficiency they should have the following characteristics:

1. The width of a unit of allocation should be a full cylinder (10 tracks).
2. The number of units of allocation should be as small as possible.
3. If an input file exists on one device and a work file is available on a second device, the work file should be specified as WORK1 to the sort.
4. When the work file and the input file are on the same device, the work file should be as physically near the input file as possible. The cylinder range of a unit of allocation for a work file may overlap the cylinder range of a unit of allocation for an input file, provided they do not use common tracks.

Calculation Of Sort-Item Block Size

Parameter Ø8 of the Fetch macro call specifies the buffer size required for reading blocks from the sort-item file. If the value given to this parameter is less than that required, the Fetch process cannot be executed. If, however, the assigned value of the parameter exceeds the requirement some memory locations will be wasted, but the Fetch process will be executed. Calculation of the sort-item block size is necessary also for an accurate computation of the work area required to execute the sort.

The calculation of the sort-item block size is divided into a number of steps; each with its own heading. Certain other information is derived as a result of these calculations, such as the maximum sort-item size acceptable to the sort.

CALCULATION OF THE HIGHEST MEMORY LOCATION AVAILABLE TO THE PRESORT (HMPS)

No Own-Coding Present

HMPS = HMA where HMA represents the highest memory address available to the sort. Note that HMPS is not always equal to the value specified by the user for highest memory location.

Own-Coding Present

When own-coding is present, the highest memory address available to any logical segment of the sort (i.e. Presort, Merge-One-Cylinder or Merge-Multi-Cylinder) may be determined from the base address of the own-code program that is resident during the execution of that logical segment.

OWN-CODING OUTSIDE THE SORT AREA: If all own-code values are greater than the highest memory address parameter or lie below the base of the Sort, then:

HMPS = HMA

Note that own-coding can only lie below the base of the Sort if the Sort has been relocated from its present base value of $2ØØ_{1Ø}$.

OWN-CODING WITHIN THE SORT AREA: If there is no merge-own-coding or the merge-own-code program is to be called by the Sort, then:

1. PSOPEN is the value of the presort open exit.
2. PSITEM is the value of the presort item exit.
3. HMPS = PSOPEN-1 if PSOPEN is less than PSITEM; otherwise HMPS = PSITEM-1.

When merge-own-coding is loaded prior to the execution of the Sort, then:

1. MERGE is the value of the merge-own-code exit.
2. HMPS is the least of the three values PSOPEN-1, PSITEM-1 or MERGE-1.

HIGHEST MEMORY LOCATION AVAILABLE TO THE MERGE

Merge-own-code is only active during the merge-multi-cylinder segment and, therefore, the merge-one-cylinder is not affected if the merge-own-code is resident at the beginning of Sort execution. Calculation of the HMMC is only significant in terms of the merge-multi-cylinder segment.

No Merge-Own-Coding Present

HMMC = HMA

Merge-Own-Coding Present

OWNCODING OUTSIDE THE SORT AREA: If the merge-own-coding value is greater than the highest memory address parameter or lies below the base of the Sort, then:

HMMC = HMA

OWN-CODING WITHIN THE SORT AREA: HMMC = (MERGE-1)

CALCULATION OF SORT-ITEM SIZE (SIS)

KL = total number of key characters.

EL = total number of extract characters.

AS = Ø if no item address appended, 11 if item address is appended.

SIS = KL + EL + AS

If an item sort is requested and IL represents the input item length, then:

SIS = IL + AS

In general, for an item sort it is expected that AS = Ø.

CALCULATION OF SPACE AVAILABLE TO MERGE (PM)

PM = HMMC - 55ØØ - SUP - 2 (SIS)

MAXIMUM SORT-ITEM SIZE ACCEPTABLE

The maximum sort-item size acceptable to the sort is the lesser of the two values derived by the following calculations:

1. $\frac{PS - INB - 25}{2}$ where INB is the size of the input block.

2. $\frac{PM - 34}{3}$

A further restriction on the size of the sort-item block is that it cannot exceed 3741 characters in length.

SINGLE AND DOUBLE BUFFERING

The determination as to whether or not to use double buffering is made at execution time of the presort. This decision is significant in determining the sort-item block size.

Table 5-1. Disk Table

COLUMN A	COLUMN B
Data Characters Per Block	Records Per Block
241	1
741	3
1241	5
3741	15

From Table 5-1, find the least value in column A that is greater than SIS. Let that value be represented by PSB (Physical Block Size).

1. Calculate $NI_1 = \left\lceil \frac{PBS}{SIS} \right\rceil$ where NI_1 is a temporary value for the sort-item block factor.

2. Calculate $IS_1 = 4 \left\lceil NI_1(SIS+12) \right\rceil$ where IS_1 is a temporary item storage factor.

3. Calculate $OS_1 = NI_1 \left\lceil SIS \right\rceil$ where OS_1 is a temporary value for the block size.

The presort will operate with double buffering if the two following conditions are met. Namely:

1. $PS \geq 2(INB)+IS_1+2(OS_1)+26$
2. $PM \geq 3(OS_1)+26$

CALCULATION OF SORT-ITEM BLOCK SIZE FOR SINGLE BUFFER MODE

1. $NI_2 = \left\lfloor \frac{PS-INB-13}{2\ (SIS+6)} \right\rfloor$ (rounded down to the next lower integer)
2. Calculate $NI_3 = \left\lfloor \frac{PM-34}{SIS} \right\rfloor$ (rounded down to the next lower integer)

NI is the least of the three values NI_1, NI_2 or NI_3 where NI represents the sort-item blocking factor (NI_1 having been calculated in the previous paragraph). The sort-item block size for single buffering, then, is: OS = NI(SIS)+9.

CALCULATION OF THE SORT-ITEM BLOCK SIZE FOR THE DOUBLE BUFFER MODE

When the presort find it is possible to double buffer, it attempts to increase the sort-item block size; provided that a string of items can be maintained that holds at least four times the number of items that can be contained in a sort-item block. A further factor in the computation is that there should be an efficient covering of the work file area.

1. Calculate PMD = PM-2(INB)-26 where PMD is the space available to the presort after an allowance has been made for the two input buffers and output buffer control.

2. Calculate ISB = 6(SIS)+48 where ISB is defined as an item capacity unit.

3. Calculate $NI_1 = \frac{PMD}{ISB}$ (rounded down to the next lowest integer).

4. Calculate $OS_1 = NI_1(SIS)$.

5. Using Table 5-1, find in column A the least value greater than OS_1. Let that value be PBS_h. If there is a member of the table below PBS_h, represent it as PBS_L. This rule has two exceptions, as follows:

   (a) If PBS_h = 241, go to step 8 (c).

   (b) If OS_1 is greater than 3741, take PBS_h as 3741.

6. Calculate the following:

   (a) $NI_1 = \frac{PBS_h}{SIS}$ (rounded down to the next lowest integer)

   (b) $W_h = PBS_h - NI_2(SIS)$ where W_h is the number of physical records represented by PBS_h that would not be occupied by a sort-item block.

7. Calculate the following:

   (a) $NI_2 = \frac{PBS_L}{SIS}$ (rounded down to the next lowest integer)

   (b) $W_L = PBS_L - NI_2(SIS)$ where W_L is the number of characters of the group of physical records that would not be occupied by the sort-item block

8. Calculate the following:

   (a) If $W_h < W_L$ then $NI = NI_1$

   (b) If $W_h \geq W_L$ then $NI = NI_2$

   (c) When $PBS_h = 241$, $NI = \frac{241}{SIS}$ (rounded down to the next lowest integer)

9. The sort-item block size = OS = NI(SIS) +9

CALCULATION OF SORT WORK AREA REQUIRED

The sort will handle up to five units of allocation that may be concentrated within one work file or split across two work files. If the total number of units of allocation ascribed to the file(s) is five or less, the last record of the last unit of allocation is used as storage for sort internal control information. It is also used to pass information to the Fetch function.

Calculation Of The Number Of Sort-Items That Might Be Contained In A Unit Of Allocation

If a unit of allocation is defined as $C_1T_1C_2T_2$ the number of 250-character physical records on a cylinder is given by: $15(T_2-T_1+1) = NRT$

1. Calculate the number of physical records required to contain a sort-item block (SIB)
   SIB = NI(SIS)+9. Using Table 5-1, find in column A the least value greater than SIB. Then, the number of physical records required to contain a sort-item block is given by the corresponding value in column B. Let this value be represented by PRC. For example, if SIB = 730 a search of Table 5-1 would show the value 741 in column A therefore PRC = 3 (the corresponding value in column B).

2. Calculate the number of sort-items contained within a cylinder (NC). $NC = \left[\frac{NRT}{PRC}\right] (NI)$

3. Calculate the number of sort-items that may be contained within a unit of allocation.

   (a) When the unit of allocation is not the last unit and the total number of units of allocation common to the specified work file(s) exceeds five:
   Number of sort-items = $NC \left[C_2-C_1+1\right]$

   (b) When the unit of allocation is the last one to be used and the total number of units for the work file(s) is five or less: Number of sort-items = $NC \left[C_2-C_1+1\right] -2-NI$

   (c) When the unit of allocation is the last one to be used and the total number of units of allocation common to the specified work file(s) exceeds five:
   Number of sort-items = $NC \left[C_2-C_1+1\right] -NI$

Calculation Of The Number Of Cylinders Required For The Input

If the difference between the highest and lowest tracks (T_2-T_1) is the same for all units of allocation, the number of cylinders (CR) required to handle the input when there are five or less units of allocation is given by:

1. $CR = \frac{(I+2NI)\ PRC}{15(T_2-T_1+1)NI}$ (rounded up to the next highest integer)

where I is the total number of input items to the sort.

If the work files have more than five units of allocation the formula becomes the following:

1. $CR = \frac{I(PRC)\ (I+NI)}{15(T_2-T_1+1)(NI)}$ (rounded up to the next highest integer)

Parameters Resident In Memory

When the parameters to the Sort are in main memory prior to the Sort being called, no default conditions are allowed. The starting location of the parameter area is 21Ø (decimal).

MERGE OWN-CODING PROGRAM

If there is to be merge own-coding and it is not already in memory, the Sort will call in the program via the Supervisor according to the parameters shown in the summary.

SORT KEY-FIELDS

Up to ten key-fields may be specified, in decreasing order of importance. Specification of each field requires seven characters. Four of these are used to indicate the position of the leftmost (high-order) character in the field, and two for the number of characters in that field (both in decimal with leading zeros). The seventh character is defined as the reverse key parameter and its use permits the key with

which it is associated to be in the reverse sequence from that specified by the parameter in character 121. If less than ten key fields are used, the remaining characters in the key-field parameter area must be blank.

EXTRACT FIELDS

Extract fields are fields of the item that may be included in the sort-item but have no significance in determining the final sort order. Up to eleven extract fields may be specified and the order of specification determines their position in the sort-item. Specification of each extract field requires six characters: four for the position of the leftmost (high-order) character of the field, and two for the number of characters in that field. If less than eleven extract fields are used, the remaining characters in the extract field parameter must be blanks.

The first character of an item is considered to be position ØØØ1. When the user wishes the extract field to be all those fields of an item which have not been specified as keys, then characters 193 - 196 are specified as ITEM and the remaining extract parameter area is set to blanks. When ITEM is specified, the final output sort-item which is input to the Fetch function, is a string of items in the same format as the original input item. Key fields and extract fields are mutually exclusive.

SELECT OPTION

Only one select field may be specified. When such a field is specified, only those input items that have the same value as the specified select field will be accepted as input to the Sort.

DELETE OPTION

Only one delete field may be specified. Items that have the field identified by the delete parameters are bypassed as input to the Sort. They are not deleted from the original input file.

SUMMARY OF SORT PARAMETERS RESIDENT IN MAIN MEMORY

Table 5-2 lists the sort parameters resident in main memory.

TABLE 5-2. SORT PARAMETERS RESIDENT IN MAIN MEMORY

PARAMETER NAME	CHARACTERS	VALUE	DESCRIPTION
FILE INFORMATION	1 - 1Ø	Input File Name	Input File Name.
	11 - 18	Password	Password.
	19 - 21	XX_8 $ØX_8$ $ØØ_8$	Address of primary Input file control unit. Address of primary input file device. Address of primary input file magazine number.
	22 - 5Ø	Δ	Reserved for use of the operating system. Must be set to blanks.
	51 - 6Ø	Name of Primary Work File	Name of primary work file.
	61 - 63	XX_8 $ØX_8$ $ØØ_8$	Address of primary work file control unit. Address of primary work file device. Address of primary work file magazine.
	64 - 73	Name of Second Work File	Name of second work file. Blank if a second work file is not used.
	74 - 76	XX_8 $ØX_8$ $ØØ_8$	Address of second work file control unit. Address of second work file device. Address of second work file magazine. } Blank if a second work file is not used.
	77 - 8Ø	Δ	Reserved for use of the operating system. Must be set to blanks.
	81	XX_8	Printer control unit device address. Must be blank if not used. When this parameter is specified the program history is printed and the block that has given rise to an uncorrectable read error will be printed.

Table 5-2 (cont). SORT PARAMETERS RESIDENT IN MAIN MEMORY

PARAMETER NAME	CHARACTERS	VALUE	DESCRIPTION
HIGHEST MEMORY ADDRESS	82 - 87		This is the highest memory address available to the sort and may be expressed in any of the following three ways: 1. In decimal with leading zeros. 2. As the number of 4K modules, with leading zeros. 3. As a binary address, right justified in the parameter field and with leading spaces. If the value is greater than 32K, then 4 characters are required.
	88	△	Reserved for the use of the operating system. Must be set to a blank.
PRESORT OWN-CODING	89 - 94		The address, in decimal, with leading zeros, to which the presort branches when an input file has been opened. Blank if the option is not used.
	95 - 1ØØ		The address, in decimal, with leading zeros, to which the presort will branch (1) after the presort has been specialized and (2) just prior to processing each item. Blank if the option is not used. Alternatively, the address may be specified as a binary value, right justified with leading blanks. If the address lies above 32K, 4 characters are required.
PRESORT ITEM OWN-CODING ADDRESS	1Ø1 - 1Ø6		The address, in decimal, with leading zeros, to which the final merge phase will branch when an item has been processed. The merge is said to be in its final phase when it is producing the single string output. Blank if this option is not used. Alternatively, the address may be specified as a binary value, right justified with leading blanks. If the address lies above 32K, 4 characters are required.
MERGE OWN-CODING PROGRAM	1Ø7 - 112		Merge own-coding program name.
	113 - 114		Merge own-coding segment name.
	115 - 12Ø		Visibility mask associated with merge own-coding program. Blank if not used.

Table 5-2 (cont). SORT PARAMETERS RESIDENT IN MAIN MEMORY

PARAMETER NAME	CHARACTERS	VALUE	DESCRIPTION
ASCENDING OR DESCENDING OUTPUT	121	△	Ascending sequence. When a mixed sequence is required, this is expressed through the reverse-key parameter.
		D	Descending sequence.
ITEM ADDRESS APPENDAGE	122	△	Item address is to be appended to the sort-item.
		N	Item address is not appended to the sort-item.
SORT-KEY FIELDS	123 - 126	dddd	Position of the primary sort-key
	127 - 128	dd	Number of characters in primary key.
	129	△	Key is to be sorted in the sequence defined by character 121.
		R	Key is to be sorted in sequence opposite that defined by character 121.
	13Ø - 136		Second key parameters. Same as characters 123 - 129.
	137 - 143		Third key parameters. Same as characters 123 - 129.
	144 - 15Ø		Fourth key parameters. Same as characters 123 - 129.
	151 - 157		Fifth key parameters. Same as characters 123 - 129.
	158 - 164		Sixth key parameters. Same as characters 123 - 129.
	165 - 171		Seventh key parameters. Same as characters 123 - 129.
	172 - 178		Eighth key parameters. Same as characters 123 - 129.

Table 5-2 (cont). SORT PARAMETERS RESIDENT IN MAIN MEMORY

PARAMETER NAME	CHARACTERS	VALUE	DESCRIPTION
SORT-KEY FIELDS	179 - 185		Ninth key parameters. Same as characters 123 - 129.
	186 - 192		Tenth key parameters. Same as characters 123 - 129.
	193 - 196	dddd	First extract field position.
		ITEM	Extract fields are the residue of the item.
	197 - 198	dd	Number of characters in the first extract field.
	199 - 2Ø4		Second extract field. Same as characters 193 - 198.
	2Ø5 - 21Ø		Third extract field. Same as characters 193 - 198.
EXTRACT FIELDS	211 - 216		Fourth extract field. Same as characters 193 - 198.
	217 - 222		Fifth extract field. Same as characters 193 - 198.
	223 - 228		Sixth extract field. Same as characters 193 - 198.
	229 - 234		Seventh extract field. Same as characters 193 - 198.
	235 - 24Ø		Eighth extract field. Same as characters 193 - 198.
	241 - 246		Ninth extract field. Same as characters 193 - 198.
	247 - 252		Tenth extract field. Same as characters 193 - 198.

Table 5-2 (cont). SORT PARAMETERS RESIDENT IN MAIN MEMORY

PARAMETER NAME	CHARACTERS	VALUE	DESCRIPTION
EXTRACT FIELDS	253 - 258		Eleventh extract field. Same as characters 193 - 198.
SELECT OPTION	259 - 262	dddd	Position of the leftmost (high-order) character in the item of the field on which the sort is to be selected. Expressed in decimal with leading zeros.
		ΔΔΔΔ	Select option not used.
	263 - 293		These are the characters of the select field. Definition of the field must be determined by a comma. Imbedded blanks are significant. A comma may not be a character of the select field. The maximum size of a select field is 3Ø characters.
DELETE OPTION	294 - 297	dddd	The position of the left-most (high-order) character of the delete field of the item. Expressed in decimal with leading zeros.
			Delete option not used.
	298 - 328		These are the characters of the delete field. Definition of the delete field must be terminated by a comma. Embedded blanks are significant. A comma may not be a character of the delete field. The maximum size of the delete field is 3Ø characters.
	329 - 4ØØ		Reserved for the use of the operating system. Must be set to blanks.

Own-Coding

PRESORT OPEN

The presort branches to the location specified by the value of the presort open parameter after it has read the file name in *VOLDESCR*. Index register 1 (X1) contains the left-hand (high-order) address of that item. When the file is protected by a password, the item is not made available until the password check is successfully completed. When the item is made available for inspection, it may be moved to the own-code program area but no punctuation is present in the sort area it will occupy. When the own-code program sets punctuation in the sort area, it is responsible for clearing the punctuation before returning control to the Sort. The presort is re-entered at the location defined by the contents of the B address register when the presort branched to the own-code program. An SCR of the B address register should be the first instruction of the presort own-code program.

PRESORT ITEM-BY-ITEM

Definition

1. Normal Return A return of the control to the Sort from an own-code program made by branching to the location specified by the contents of the B address register when control is given to the own-code program.

2. Branchspace Constant The number of characters required for a branch operation code and one address field. If the Sort is operating in the 3 character mode, the value of this constant is 4, in 4 character mode the value of the constant is 5.

Processing An Item

The presort will branch to the location specified by the value of the item-by-item parameter prior to processing each input item, however, if that item is to be deleted it will not be delivered to the user. Index register 1 (X1) contains the address of the left-hand (high-order) character of the item as it is in the input buffer. The item may have its contents modified, but its length can never be changed. Punctuation will be present in the item: a word mark will appear on the high-order character of every key field pertinent to the particular sort application. Any change in the status of the item's punctuation will give unspecified results. Control is returned to the Sort by making a normal return. Figure 5-1 illustrates the punctuation of an input item.

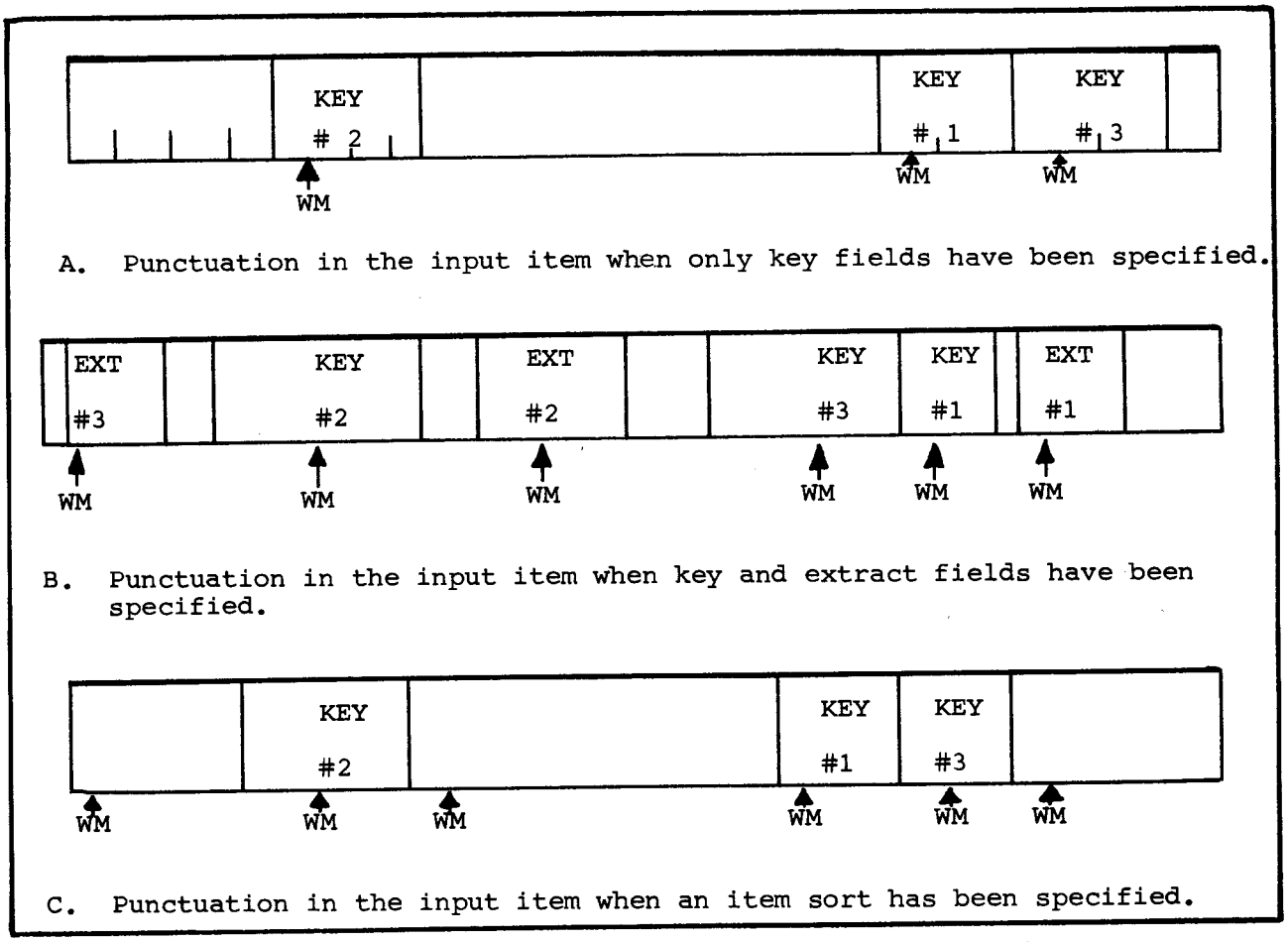


Figure 5-1. Illustrations Of Input Item Punctuation

Adding An Item

An item, or those fields relevant to the sort application, may be added by placing the address of the left-hand (high-order) character of the item in index register 1 (X1). The item must have the same format as the specified input, i.e., item length, key fields, etc. Word marks must appear in the item in the following locations:

1. On the first character of each key field.
2. On the first character of each extract field.

If an item sort has been specified, every non-key field of the item is treated as an extract and so a word mark is required on the first character of all such fields. This is illustrated in Figure 5-1.

When the sort application specified that an item address was to be appended to the sort-item, the own-code program must supply this address to the Sort. This is done by placing the address of the high-order character, of an eleven character field, in index register 2 (X2). The contents of this eleven character field may be an item hardware address, but the presence of a single character (77_8) in the high-order position of that field indicates to the Fetch program that the item does not exist on the on-line mass storage device. When the address is a valid one, the format is as follows:

1. Device Address --- Character 1: pcu with the high-order bit as zero.
   Character 2: drive number.
   Character 3: magazine number.
2. Block Address --- Characters 4 and 5: Cylinder number.
   Characters 6 and 7: Track number.
   Characters 8 and 9: Number of the record at the beginning of the block containing the item.

Characters 1Ø and 11: Relative item number in the block. The number for the first item in the block is Ø.

No punctuation may be present in the eleven character field, except for an optional word mark on the high-order character. This field is illustrated in Figure 5-2.

Control is returned to the Sort by branching to a location whose address is formed by adding a branchspace constant to the normal return address.

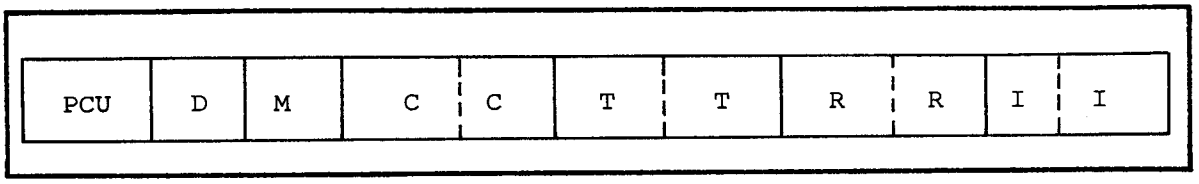


Figure 5-2. Contents Of The Item Address That May Be Appended To The Sort-Item

Where: PCU = peripheral control unit address

D = drive number

M = magazine

CC = cylinder

TT = track

RR = number of the record at the beginning of the block containing the item

II = relative item number in the block counting the first item in the block as Ø.

Deleting An Item

When an item is to be deleted from the Sort, after inspection of that item by own-coding, the presort is re-entered by branching to a location with an address formed by adding two branchspace constants to the normal return address.

TERMINATING OWN-CODING

When all own-code processing is completed, the own-coding routine must branch to a location whose address is obtained by adding three branchspace constants to the normal return address. Presort own-coding may be terminated before or after the presort has processed all its input. However, the own-coding routine must be terminated. An item mark set on the location specified by the own-coding address indicates that the presort has processed all its input. If more items are to be added, the own-code program follows the procedure for adding an item. The presort will continue to exit to the own-code program until the terminate own-code return is made.

MERGE OWN-CODE

Only when the merge is creating its final one-string output will it branch to the location specified by the merge own-coding parameter. The address of the high-order character of the sort-item, as it is in the output buffer, will be found in index register 1 (X1), when the branch occurs. The item may be inspected, modified, or moved to the own-code area, but when control is returned to the Sort, all punctuation of the item must be cleared to its original state. Punctuation and format of the sort-item, when the sort-item is made available to the merge own-code, are shown in Figure 5-3. When the merge has processed all the

sort-items (and just prior to its indication through the Console that the sort is completed), an item mark will be set on the merge own-code location and the own-code branch is taken. It is not necessary that the program return control to the sort after this last exit is taken. Control is returned to the Sort by making a normal return.

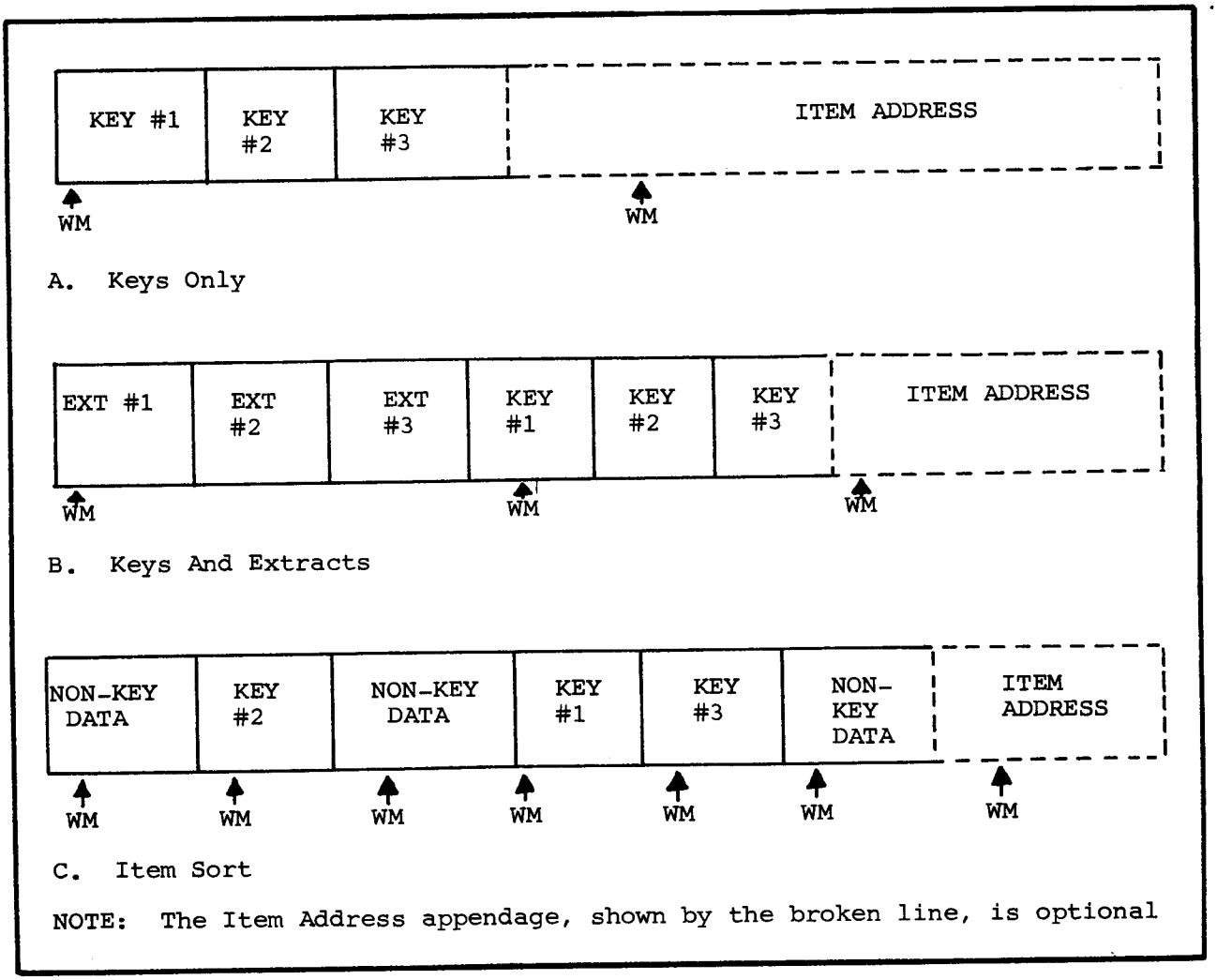


Figure 5-3. Punctuation And Format Of The Sort-Item When It Is Made Available To A Merge Own-Code Program

Considerations For Using The Fetch

The maximum memory requirements for the Fetch, including buffer space and the Supervisor, is 55ØØ locations. This requirement is reduced if not all the options of the Fetch are used; such as, when no source-item examination exit is specified. The buffer space requirements depend on certain parameter values.

EXAMINE SORT-ITEM ONLY

If parameter Ø4 is blank, only the examine sort-item exit is specified.

Single Buffering

Single buffering is specified if parameter Ø9 is SINGLE. In this case, buffer space = OS + 3. Where OS is the sort-item block size.

Double Buffering

Double buffering is specified if parameter Ø9 is DOUBLE or if it is blank. In this case, buffer space = 2 (OS + 3).

EXAMINE SOURCE-ITEM ONLY

If parameter Ø3 is blank, only the examine source-item exit is specified.

Single Buffering

Single buffering is specified if parameter 12 is SINGLE. In this case, buffer space = OS + INB + 6 where INB is the source-item block size.

Double Buffering

Double buffering is specified if parameter 12 is DOUBLE or if the parameter value is left blank. In this case, buffer space = 2(OS) + 2(INB) + 12.

BOTH SOURCE-ITEM AND SORT-ITEM EXITS SPECIFIED

When both the source-and sort-item exits are specified, either file may be associated with single or double buffering. As a general rule, if there is a memory restriction it is more advantageous to use single buffering for the sort-item and double buffering for the source-item.

1. If SINGLE is specified for both source and sort-item files, the buffer size required is OS+INB+6.
2. If DOUBLE is specified for both, the buffer size required is 2(OS+INB+6).
3. If SINGLE, parameter Ø9, is specified for the sort-item file and DOUBLE, parameter 12, for the source-item file, the buffer size required is OS + 2INB + 9.
4. If DOUBLE, parameter Ø9, is specified for the sort-item file and SINGLE, parameter 12, for the source-item file, the buffer size required is 2OS + INB + 9.

INITIATION OF FETCH

The Fetch program, MFETCH, is initiated by branching to the tagged location specified in the Location Field of the MFETCH macro call.

USE OF PHYSICAL I/O

The Fetch macro always calls MPIOC. If the user wishes to have MPIOC in his program and he is using the MFETCH macro, he should use that MPIOC called by MFETCH. The unique suffix associated with the MPIOC called in by MFETCH is % (+,8,5 keypunch). File tables generated by MFETCH utilize the value assigned to parameter Ø1 of the MFETCH macro call.

MASS STORAGE EDIT

The Mass Storage Edit routine requires that the area to be edited has been formatted by the Volume Preparation Routine or the File Support Allocate function.

Functional Description

FUNCTIONS

The Mass Storage Edit provides a printed representation of the physical data stored on any Mass Storage device. Parameters to the edit routine specify the area to be edited and the device address. The parameters are entered through the job control file or through either the card reader or a console keyboard. The area edited is bounded below and above by the track values of the starting and terminating address parameters.

The edit routine does not respect the data management conventions. Therefore, a user wishing to edit a specific file should use the file support routine "unload" to printed paper.

FEATURES OF MASS STORAGE EDIT

Header Line

The header line of the Mass Storage Edit contains the following information printed at the top of each page:

1. Device Number (pos. 1-12)
2. Title MSEDIT (pos. 46-52)
3. Page Number (Pos. 1ØØ-11Ø). This number is automatically incremented for each page (right-justified)

Header Line Record

This line contains the following information:

1. Cylinder and track number in decimal (Pos. 1-8)
2. Header record information in decimal, except for flag character (Pos. 21-33)

Data Portion Line

This line contains the following information:

1. Read error information (Pos. 1-3)
   (On first line of record only)
2. Character Line numbers (Pos. 8-10)
   Indicates position in record of first character in this line.
3. Data characters of record (Pos. 14-133)
   This line type is repeated as necessary to exhaust the record.

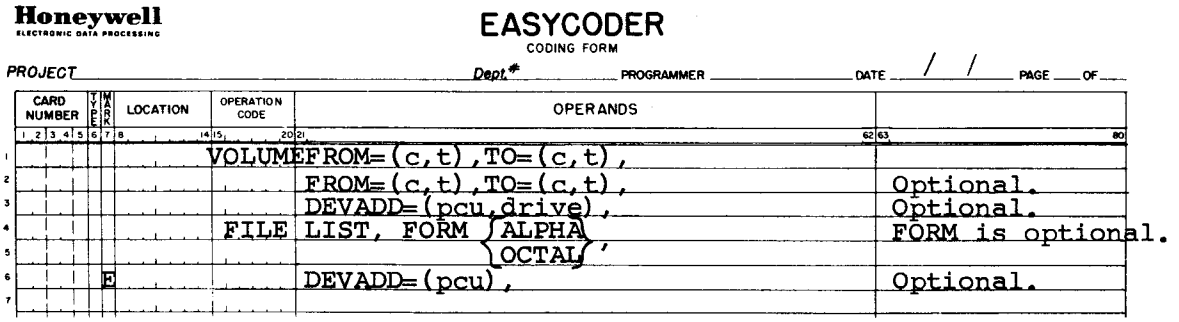
End-of-Job Line

This line contains the following information:

1. END Mass Storage EDIT (Pos. 1-12)
2. Number of Tracks processed (Pos. 20-23)
3. Number of Records processed (Pos. 42-53)
4. Number of Errors (Pos. 60-75)

Edit Function Job Control Statements

FORMAT



Honeywell
ELECTRONIC DATA PROCESSING

EASYCODER
CODING FORM

PROJECT _____ Dept.# _____ PROGRAMMER _____ DATE / / PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1				VOLUME	FROM=(c,t),TO=(c,t),	
2					FROM=(c,t),TO=(c,t),	Optional.
3					DEVADD=(pcu,drive),	Optional.
4				FILE	LIST, FORM {ALPHA / OCTAL},	FORM is optional.
5						
6		E			DEVADD=(pcu),	Optional.
7						

DESCRIPTION

Volume Statement

The Volume Statement gives the parameters of the edit function. One Volume Statement may describe several areas to be edited, as long as they are all on the same mass storage device.

FROM AND TO PARAMETERS: One pair of From and To Parameters describes one mass storage area to be edited. There must be at least one such pair. However, one Volume Statement may contain several From and To pairs; describing several areas to be edited.

The c value of the parameter is the cylinder address in decimal. The t value of the parameter is the track address.

The area edited for a given pair of From and To parameters is all tracks from the track stated in the From parameter to the track stated in the To parameter (inclusive) on each cylinder from the cylinder stated in the From parameter to the cylinder stated in the To parameter (inclusive). All records on each track are edited.

DEVICE ADDRESS PARAMETER: The Device Address Parameter (DEVADD) specifies the physical device address of the mass storage device being edited. The peripheral control unit number (pcu) of the mass storage control unit is written in 2 octal digits. The I/O bit is not required, but all other bits must be specified including the sector bits. The drive number is written in 1 octal digit. When the Device Address Parameter is omitted, the standard assumption is that the device address is pcu Ø4 and drive Ø.

File Statement

The File Statement has the I/O function name LIST as the first parameter of the statement.

FORM PARAMETER: The Form Parameter specifies the format of the editing listing. When written as FORM=OCTAL, an octal listing is produced. When written as FORM=ALPHA, an alphabetic listing is produced. When the Form parameter is not specified (omitted), the standard listing is alphabetic.

DEVICE ADDRESS PARAMETER: The Device Address Parameter (DEVADD) specifies the physical address of the printer. The peripheral control unit number (pcu) is written in 2 octal characters. When this parameter is omitted, the standard assumption is that the pcu is Ø2.

APPENDIX A

FILE REASSIGNMENT

INTRODUCTION

For each system program, every system file is given a functional name and a specific assignment. The assignment, consisting of a file name, a physical device type, and a peripheral control unit and drive number, normally is made by Honeywell when the system program is delivered to the installation. In certain cases, however, the installation may change these assignments. Whether or not the installation changes these assignments, the assignments are called "Installation Standard Assignments".

System Files are those files created or used by the system programs of the mass storage operating system. These files are stored on external media such as punched cards, mass storage, magnetic tape, console keyboard/typewriters, printers, etc. System Files are referred to either by a functional name or by a file name.

The functional name for a System File is in terms of the function the file serves in a given systems program. The functional name of a System File is used in a File Statement to define which file of the system program is to be reassigned. The System Files, their functional names, and their definitions are included in Table A-1.

Table A-1. Function and Definition of System Files

SYSTEM FILE	FUNCTIONAL NAME	DEFINITION
Residence	RES	The Executable Program File containing the programs to be run (Supervisor, System Programs, and user programs).
Job Control	JOB	The control information identifying a job and defining its operating requirements. Parameters submitted at execution time are included in this file.
Operation Control	OP	The operation control information either from the system to the operator or from the operator to the system that affects the operation of the current job.
Information	INFO	Listed information from the system to the operator. This information may affect the later operation of related jobs, but not the operation of the current job. This file is always produced online (printer, console/keyboard, etc.).
Input	IN	Input card image data to a system program, such as a source language program to be translated.
List	LIST	Output line image data from a system program for listing. This file normally does not affect the operation of the current job or of later jobs. This file, therefore, is normally produced off-line (mass storage, magnetic tape, etc.).
Go	GO	Translated (compiled or assembled) programs either for immediate execution or for updating a permanent executable program library, such as the Residence File.
Library	LIB	Easycoder Assembly (source) language routines, suitable for specialization and inclusion in the assembly language programs.
Work N	WORKN	Temporary file, used by system program during its operation. A work file is released when the program using it has finished. The N represents a decimal number, written without leading zeros, for example, WORK1 WORK10, etc.

The file name (used in the assignment) is a name that identifies a particular file and distinguishes it from other files. The file name is stored with the file, in accordance with the data management conventions of the medium on which the file is stored. The Volume Directory of a mass storage volume, for example, contains a field for the file name for each file on the volume.

File names for system programs consist of 10 characters, the first of which is always an asterisk (*). The remaining nine characters are letters and numbers. No special characters are used. Spaces can appear only as the rightmost characters of a name.

GENERAL DESCRIPTION OF FILE REASSIGNMENT JOB CONTROL LANGUAGE

The Job control language for file reassignment consists of a File Statement and several parameters. These are described in the following paragraphs.

Format

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	CONTINUATION	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6	7	8	14 15	20 21	62 63 80
				FILE	FUNCTIONAL-NAME,
					OTHER=new-functional-name,
					NAME=new-file-name,
					DEVTYPE=device-type,
					DEVADD=(pcu,drive),

Description

FILE STATEMENT

The File Statement gives the functional name of the file being reassigned. This identifies for the system program which file is referred to by the File Statement. The functional names for the files are listed in the preceding paragraph of this appendix.

OTHER SYSTEM FILE PARAMETER

This parameter, when used, must appear immediately after the File Statement. The other parameter gives the functional name of a file whose Installation Standard Assignment is to be used.

FILE NAME PARAMETER

This parameter gives the file name to be used in place of the Installation Standard Assignment of file name.

DEVICE TYPE PARAMETER

This parameter gives the type of device to be used in place of the Installation Standard Assignment of device type.

DEVICE ADDRESS PARAMETER

The Device Address Parameter specifies in two octal characters the peripheral control unit (pcu) number. The input/output bit (bit one of the first character) is not used when the control unit requires two control unit addresses. Two control unit addresses are required when the unit is an input/output unit.

The Device Address Parameter specifies the drive number in one octal character. The drive number may be omitted. When omitted, the drive number is 0 if it is relevant to the particular device type.

FILE REASSIGNMENT JOB CONTROL STATEMENTS

The following paragraphs discuss the job control statements for those portions of the mass storage operating system whose installation assignments can be changed at execution time. Only the job control statements available to the user at the time of the initial release of the system are discussed here. The Supervisor's Execute statement calls in the appropriate portion of the Operating System, for example, Program Development.

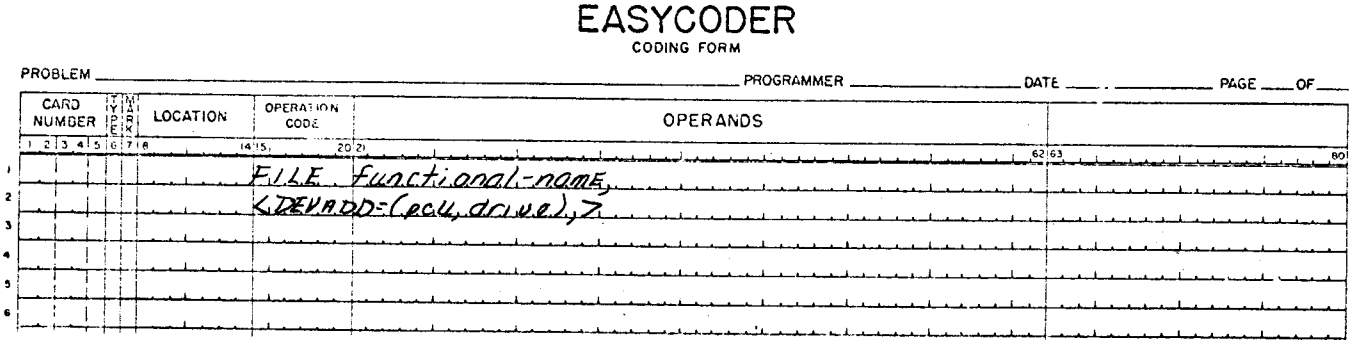
FILE STATEMENTS FOR PROGRAM DEVELOPMENT

Library Update

None of the files used by the Library Update may be assigned to an alternative device type.

The Library File and the List File may be assigned to alternate device addresses. However, it is seldom necessary to do so because the standard installation addresses are taken as default assumptions.

Format


EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MAR IX	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7 8	14 15	20 21	62 63	80
1				FILE functional-name,		
2				<DEVADD=(pcu,drive),>		
3						
4						
5						
6						

Description

Functional-name is the functional name of the file to be reassigned. Values for this parameter may be:

LIST - List file

LIB - Library file

PCU is the non-standard peripheral control unit number, written as two octal characters. All bits except the I/O bit (bit one) must be specified.

Drive is the non-standard drive number, written as one octal character.

EXECUTABLE PROGRAM FILE UPDATE

If any of the files are to be assigned to non-standard device addresses, the additional control statements must follow the function statement.

None of the files used by the Executable Program File Update may be assigned to an alternate device type, with the exception of the transaction input file. This file is reassigned by means of the GO parameter described in the Executable Program File Update section of Program Development.

The Master File and the Transaction Input File may be assigned to non-standard device addresses. However, it is seldom necessary to do so because the standard installation addresses are taken as default assumptions.

Format

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62	63 ... 80
1				FILE	Functional-name,	
2					DEVADD=(pcu, drive),	
3						
4						

Description

Functional-name identifies the file to be reassigned. Values for this parameter may be:

RES - Master File (Residence)

GO - Transaction Input File (either BRF or BRT)

PCU is the peripheral device control unit number, written as two octal characters. All bits except the I/O bit (bit one) must be specified.

Drive is the drive number, written as one octal character.

EASYCODER ASSEMBLY

None of the files used by Easycoder Assembly can be assigned to a different device type, with the exception of the transaction output (GO) file. The method of doing so is described in the Easycoder Assembly section of Program Development.

Several of the files may be assigned to non-standard device addresses. However, it is seldom necessary to do so because the standard installation addresses are taken as default assumptions.

Format

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

CARD NUMBER	VPER	MISK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				FILE	functional-name,	
					DEVADD=(pcu,drive),	

Description

Functional-name identifies the file to be reassigned. Values for this parameter may be:

LIST - List file.

LIB - Library file, the source of macro routines.

WORK1 - First work file.

WORK2 - Second work file.

GOBRF - Machine-executable output file on mass storage.

GOBLD - Machine-executable output file on the card punch.

PCU is the non-standard peripheral control unit number, written as two octal characters. All bits except the I/O bit (bit one) must be written.

Drive is the non-standard drive number, written as one octal character.

FILE STATEMENT FOR MASS STORAGE SORT

The "FILE" statement has the I/O function name LIST as the first parameter of the statement.

Format

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYP	CNT	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 ... 14 15	20 21	... 62 63	80
1				FILE	LIST,<FORM=(ALPHA),>	
2					(OCTAL)	
3					<DEVADD=(pcu),>	
4						

FORM PARAMETER

The form parameter specifies the format of the editing listing.

Description

OCTAL - An octal listing is produced.

ALPHA - An alphabetic listing is produced.

Comment

The default assumption is ALPHA

DEVICE ADDRESS PARAMETER

The device address parameter specifies the physical address of the printer.

Description

pcu - The peripheral control unit of the printer, two octal characters.

Comment

The default assumption is pcu 02.

APPENDIX B

PHYSICAL INPUT/OUTPUT CONTROL

INTRODUCTION

Physical Input/Output Control functions as the interface between a program and the mass storage hardware. This eliminates the need for the user to reference the hardware directly. In most cases, a program utilizes an I/O routine (as described in section 3 of this manual) which, in turn, references Physical Input/Output Control whenever access to mass storage hardware is required. However, a program may reference the Physical Input/Output Control directly.

Mass Storage Physical Input/Output Control consists of a set of macro routines that provide a simple means of processing data stored on mass storage peripheral devices. These routines fall into three categories: control, communication, and action. To access an area of the storage device, the user issues the appropriate action macro. The action macro references the file table set up by the communication macro and links to the control macro. The control macro, in turn, initiates the required action according to the current contents of the file table.

MASS STORAGE PHYSICAL INPUT/OUTPUT CONTROL (MPIOC) MACRO

The MPIOC macro controls the actions on mass storage devices associated with a single Type 250 Control Unit. When more than one control unit is being used, a separate MPIOC macro must be called for each.

MPIOC Format

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1	C		ANYTAG	MPIOC	PARAMETER1,,,,,,,,,	
2	L			Ø5	PARAMETER5,	
3						

MPIOC Description

TYPE FIELD

The Type Field of the line containing the MPIOC command must have a C when more than one line is used for the call. All lines except the last must have a C in the Type Field. The last line of a MPIOC macro call must have an L in the Type Field.

LOCATION FIELD

The Location Field can contain any acceptable assembly tag. This tag is equated to the lowest memory location occupied by MPIOC. The Location Field entry is parameter Ø of the MPIOC macro.

OPERATION CODE FIELD

The Command Field contains the Operation Code MPIOC, which specifies to the system what function to perform.

OPERANDS FIELD

The Operands Field contains the parameters required for MPIOC. These are described in table B-1.

Table B-1. MPIOC Parameters

NUMBER	NAME	VALUE	DESCRIPTION
1	Suffix	X	A single alphanumeric character that will serve as a unique suffix to all tags in MPIOC. This parameter must be specified.
2	Peripheral Control Unit Address	XX_8	The peripheral control unit address to which the Type 250 Control Unit applicable to this MPIOC is attached.
		△	The Honeywell recommended address assignment for the Type 250 Control Unit ($Ø4_8$).
3	Write Verification	V	The automatic Verify coding will be included.
		△	The automatic Verify coding will not be included.
4	Control of More Than One P.C.U.	M	The peripheral control unit number will be specialized from the File Table at execution time each time the peripheral control unit number changes.
		△	The peripheral control unit number will be specialized at assembly time and cannot be changed without re-assembling.
5	R/W C Definition	XX_8	Interlocked starting R/W counter to be used for all data transfers. Cannot be changed without re-assembling and must correspond to sector of pcu statement.
		VAR	RWC will be specialized from the File Table for each action call.
		△	Automatic specialization at assembly time depending on parameter 2: 56_8 if P2 $\leq$ 7, 76_8 if P2 $\geq$ 7.

MASS STORAGE PHYSICAL COMMUNICATIONS AREA (MPCA) MACRO

The MPCA macro provides a communications area to contain a series of fields. These fields are available to the user. The user may change or interrogate the values of these fields as required. To accomplish this, the MUCA and MLCA macros are used. These macros are described in Appendix D of this manual.

MPCA Format

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7 8 ... 14		15 ... 20	21 ... 62 63 ... 80
1	C		TAG,	MPCA	PARAMETER1,
2	L			Ø7	PARAMETER7,
3					

MPCA Description

TYPE FIELD

The requirements for the Type Field of the MPCA macro are the same as those described for the MPIOC macro.

LOCATION FIELD

This field contains a tag that can be up to 3 characters long. This tag is used as a prefix to all MPCA entries. The tag in the Location Field is considered as parameter Ø of the MPCA macro.

OPERATION CODE FIELD

The Command Field contains the Operation Code MPCA, which specifies to the system what function to perform.

OPERANDS FIELD

The Operands Field contains the parameters required for MPCA. These are described in table B-2.

Table B-2. MPCA Parameters

NUMBER	NAME	VALUE	DESCRIPTION
1	Suffix	X_{10}	Suffix relating to the MPIOC with which this MPCA is to initially associate. This parameter must be specified.
2	Buffer Address	Address	Location of the left-most character of an area to or from which data transfer will begin. Any legal address form. This parameter must be specified.
3	Error Exit	Tag	Address of a user provided routine to which MPIOC should branch in case of an uncorrectable error condition. This parameter must be specified.
4	C_3 Variant	XX_8	An octal value defining the type of data transfer to be executed by MPIOC. Permissible values are: (Ø4) = Load/Unload Address Register (Ø2) = Search and Read/Write (22) = Extended Search and Read/Write (Ø3) = Search and Read/Write Next (ØØ) = Read Initial. The high order bit of C2 must be 1. (2Ø) = Extended Read Initial. The high order bit of C2 must be 1. (Ø1) = Read. The high order bit of C2 must be 1. (21) = Extended Read. The high order bit of C2 must be 1. The value of C3 is $(Ø4_8)$.
5	Protection Bits	XX_8	An octal value indicating the Protection bits to be loaded into the address register. The significance of these bits from left to right is as follows: Bit B Ø Must be Ø Bit A Ø Must be Ø Bit 8 1 Permit B-File Write Bit 4 1 Permit A-File Write Bit 2 1 Permit Data Write Bit 1 1 Permit Format Write
		△	The initial value will be ØØ

Table B-2 (cont). MPCA Parameters

NUMBER	NAME	VALUE	DESCRIPTION
6	Variable RWC	XX_8	A starting RWC number for initial data transfers to this file. Stored in a single character field.
		△	Specializes field at assembly time to $\emptyset\emptyset_8$.
7	Variable PCU Number	XX_8	A peripheral trunk number to be used for initial data transfers to this file. Stored in a single character field.
		△	Specializes field at assembly time to $\emptyset 4_8$.

MASS STORAGE PHYSICAL I/O ACTION MACROS

Action macros provide the user with the ability to initiate the following actions: Read, Write, Check, Restore, and Verify. Each time one of these action macros is entered, the corresponding function will be initiated or executed by MPIOC. MPIOC will reference the indicated communication area as required for that purpose. Each of these macros is described in the following paragraphs. Their Call descriptions are at the end of these paragraphs.

Read Action Macro

The call for this macro is inserted in the user's program wherever a data transfer from the mass storage device to the main memory is required. MLCA functions may also be accomplished by this action (see Appendix C).

READ ACTION MACRO FORMAT

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62	63 ... 80
1	C		ANYTAG	READ	PARAMETER1,....,	
2	L			11	PARAMETER11,	
3						

READ ACTION MACRO DESCRIPTION

Type Field

The requirements for the Type Field are the same as those described for the Type Field of the MPIOC macro.

Location Field

The Location Field can contain a tag that will be assembled as the tag of the first instruction in the generated coding. The Location Field does not have to contain a tag if the user does not desire to have one. This field is considered as parameter Ø of the macro.

Operations Code Field

The Command Field contains the Operation Code READ, which specifies to the system the function to perform.

Operands Field

The Operands Field contains the parameters necessary for the READ macro. These are described in the following paragraphs.

PARAMETER 1: - Parameter 1 is a three character MPCA prefix tag which must be the same as parameter Ø of the appropriate MPCA.

PARAMETERS 2, 4, 6, 8, and 10: - These parameters are the same as Parameters 2, 4, 6, 8, and 10 of the MLCA macro described in Appendix C of this manual. These parameters contain the address of a user provided

field which contains the data required to alter the MPCA table.

PARAMETERS 3, 5, 7, 9, and 11: - These parameters are the same as Parameters 3, 5, 7, 9, and 11 of the MLCA macro. These parameters contain specific mnemonics indicating the right-hand end of the actual area within the appropriate MPCA table which is to be altered.

WRITE Action Macro

The call for this macro is inserted in the user's program wherever a data transfer from main memory to the mass storage device is required. MLCA functions also may be accomplished by this action (see Appendix C).

WRITE ACTION MACRO FORMAT

The WRITE action macro format is the same as described for the READ action macro.

WRITE ACTION MACRO DESCRIPTION

The WRITE action macro description is the same as described for the READ action macro.

WAIT Action Macro

The call for this macro is inserted in the user's program wherever the user desired to wait for the completion of and check for error on the last data function for the file specified. If the normal return to the user occurs following entrance to this macro, the user is guaranteed that the data transfer was completed successfully.

WAIT ACTION MACRO FORMAT

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . ___ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62	63 ... 80
1	L		ANYTAG	WAIT	PARAMETER1,,	
2						

WAIT ACTION MACRO DESCRIPTION

Type Field

The requirements for the Type Field are the same as described for the Read action macro.

Operation Code Field

The requirements for the Command Field are the same as described for the Read action macro.

Operands Field

The requirements for the Operands Field are the same as described for parameter 1 of the Read action macro.

RESTORE Action Macro

The call for this macro is inserted in the user's program wherever the user desires to restore a specific device to its initial state.

RESTORE ACTION MACRO FORMAT

The Restore action macro format is the same as described for the Wait action macro.

RESTORE ACTION MACRO DESCRIPTION

The Restore action macro description is the same as that for the Wait action macro.

VERIFY Action Macro

The call for this macro is inserted in the user's program wherever the user desires to read, without a data transfer (in the verify mode), the area last written on the mass storage device. The WRITE macro and the VERIFY macro may be separated by user's coding if the mass storage device is not referrenced in this interval.

VERIFY ACTION MACRO FORMAT

The Verify action macro format is the same as described for the Wait action macro.

VERIFY ACTION MACRO DESCRIPTION

The Verify action macro description is the same as described for the Wait action macro.

MASS STORAGE PHYSICAL I/O PROGRAMMERS PREPARATION INFORMATION

General Information

ADDRESS MODE

The coding generated as a result of a Physical I/O macro call is in address mode 3, since this is the only address mode useable with the operating system at this time.

SPECIAL CONSIDERATIONS FOR SPECIFYING PARAMETERS

Use Of Index Registers

Physical I/O uses index registers X5 and X6 but restores them to their original values before returning to the user's program regardless of whether the return is in the normal mode or is an uncorrectable error exit. These index registers, X5 and X6, therefore, may be employed by the user's written program and can be used in conjunction with the MLCA and MUCA macro functions.

Specifying A Variable PCU Number

Normally, Physical I/O controls a single PCU that, in turn, controls a single speed device. This means that the PCU number and the R/W Channel assignment can remain constant throughout an execution of a program. More than one PCU can be controlled by a single specialization of Physical I/O, however, which means that more than one device speed is possible. Parameters Ø4 and Ø5 of MPIOC and Ø6 and Ø7 of MPCA are significant here.

When the user intends to vary the PCU number during execution, he must give the value M to parameter Ø4 of MPIOC. Then, during execution, the PCU number can be altered by using the mnemonic PCU in an MLCA macro call. The PCU number specified in parameter Ø2 of MPIOC is not affected by assigning M as the value of parameter Ø4, just as the PCU number specified in parameter Ø7 of MPCA is not affected, and the Honeywell standard value for the PCU number may still be used simply by not assigning a value to either parameter Ø2 of MPIOC or Ø7 of MPCA.

Whenever the PCU Number is variable during the execution of a program, the possibility exists that the R/W Channel assignment may also have to be variable. When the PCU Number is variable, but all the PCUs controlled by a single MPIOC are on the same I/O Sector, a fixed R/W Channel assignment

is acceptable as long as all the devices to be accessed have the same data transfer rate. In this case, parameter Ø6 of MPCA must not have a value assigned to it, but parameter Ø5 of MPIOC can be assigned a two digit octal number for the R/W Channel or it can be blank. Parameter Ø5 of MPIOC must not, however, be assigned the value VAR.

The value VAR is assigned to parameter Ø5 of MPIOC only when a change in the PCU Number during execution of a program will result in a change in the I/O Sector or will involve accessing a device with a different data transfer rate. In this case, parameter Ø6 of MPCA may be blank or it may be assigned a two digit octal number for R/W Channel.

When a change in the PCU assignment necessitates a change in the R/W Channel assignment, the user must issue a WAIT action macro call before changing the PCU number through an MLCA macro. This will ensure that all actions are completed before the new PCU, with its R/W Channel, is used. The user, however, must ensure that the new R/W Channel assignment is consistant with the I/O Sector of the new PCU and with the speed of the new device.

Considerations For MPIOC Parameter Specification

SUFFIX CHARACTER

The Suffix Character specified in parameter Ø1 is used as the last character of all tags in MPIOC. For this reason, any tag written in the program by the user should not end with this character. When a program contains more than one MPIOC, each must be assigned an individual Suffix Character.

PCU ASSIGNMENT

When the PCU Number is specified by the user, he must express it as an output assignment (ØØ through Ø7 or 2Ø through 27). When the user intends

to change the PCU assignment during the execution of the program, parameter Ø4 of MPIOC must be assigned the value M. For considerations related to variable PCU assignments, see the preceding paragraph.

R/W CHANNEL DEFINITION

Parameter Ø5 of MPIOC is used to specify the R/W Channel. When no value is assigned to parameter Ø5, MPIOC automatically makes the R/W Channel assignment, based on the PCU Number specification of parameter Ø2 of MPIOC. Parameter Ø2 can be blank, in which case, the Honeywell recommended PCU Number assignment is used, or it can be assigned a two digit octal number as the address assignment of the PCU. In either of these cases, when parameter Ø5 is left blank, MPIOC makes the R/W Channel assignment. This assignment is 56 for PCU I/O Sector 1 and 76 for Sector 2. The 56 assignment specifies that R/W Channels 2 and 3, for Sector 1, are interlocked, while the 76 assignment specifies that R/W Channels 4 and 5, for Sector 2, are interlocked. These are high speed interlocks available on all Series 200 Central Processors except that 201-0 and 201-1.

When the programmer intends to use some other R/W Channel assignment, he must assign a two digit octal number as the value of MPIOC parameter Ø5. This number then is used as the variant character in PDT and PCB instructions generated by Physical I/O.

Should the programmer intend to have a variable R/W Channel assignment, for reasons given in preceding paragraphs, he must assign the value VAR to MPIOC parameter Ø5.

Considerations For MPCA

An area in memory, specialized in a fixed format, is provided by MPCA for communications. The area is specialized according to the values assigned the MPCA parameters. A separate MPCA macro call must be in the program for each set of data, such as a file, for which separate communication

values are to be established. Each Physical I/O action macro call is linked through the file prefix parameter of MCA to a specific MPCA, and by the MPIOC suffix character to that MPIOC. The MPIOC performs the action requested by the Physical I/O action macro, obtaining the required values from the related MPCA.

FILE PREFIX

The file prefix is established by assigning 1, 2, or 3 characters to parameter ØØ of MPCA and is used to identify the tags used by the MPCA from all other tags in the program. When the program contains more than one MPCA, each file prefix value must be different. Also, each character used as a file prefix must be valid in a tag according to the Easycoder Assembly rules.

SUFFIX OF RELATED MPIOC

Because it is possible for a program to contain more than one MPIOC, the value assigned the suffix parameter, MPCA parameter Ø1, must be the same character assigned as the value of MPIOC parameter Ø1 to which the MPCA is linked. This ensures that the Physical I/O action macros will link to the appropriately specialized MPIOC.

BUFFER ADDRESS (AAD)

An address constant (DSA) is generated by the buffer address parameter, MPCA parameter Ø2. Except for indexing with index registers X5 and X6, any form of addressing can be used. Also, the value of the DSA may be changed prior to any Physical I/O action macro except VERIFY.

USER'S UNCORRECTABLE ERROR ROUTINE ENTRANCE (EAD)

Parameter Ø3 of MPCA contains the symbolic address (tag) of the user supplied uncorrectable error routine. Any form of addressing can be used, except for indexing with index registers X5 and X6. This address can be

changed at any time.

TYPE OF READ OR WRITE (TRW)

When a Read or a Write action is initiated, MPIOC interrogates the value assigned to parameter Ø4 of MPCA to determine the type of Read or Write desired. This value is changed frequently during the execution of the program through the Read, Write or MLCA macro. Therefore, frequently no value is assigned to parameter Ø4. This enables the loading and unloading of the address register. The values that can be assigned to parameter Ø4 are two digit octal numbers. These are contained in the following list along with the type of Read or Write action that will be performed.

VALUE OF PARAMETER	TYPE OF READ/WRITE PERFORMED
△ or Ø4	Load/Unload address register
Ø2	Search and Read/Write
22	Extended Search and Read/Write
Ø3	Search and Read/Write next
23	Extended search and Read/Write next
ØØ	Read initial
2Ø	Extended Read initial
Ø1	Read
21	Extended Read

NOTE: In assigning 2Ø, 21, 22, or 23 the automatic verify action is enabled. To enable the automatic verify action for any of the other numbers, the most significant zero should be changed to a one. For example, if a Search and Read next was desired, it would be enabled by assigning the value Ø3 to parameter Ø4. If the successful completion of this action was to be verified automatically, the number 13 would be assigned to parameter Ø4.

CONTROL UNIT CURRENT ADDRESS AND STATUS

In each MPCA there is a 10 character field, word marked at its left end. This field contains the current control unit address and its status for the actions being issued through that MPCA. The field is not an exact image of the PCU Address Register, particularly when more than one MPCA is included in the program. The field does indicate the status of the one set of operations being issued that MPCA.

The field is shown in Figure B-1 and the three mnemonics shown, CYL, CAD, or PRT, can be included in a Read, Write or MLCA macro call to change the contents of the applicable portion of the field. The contents of each character in the field is in binary form.

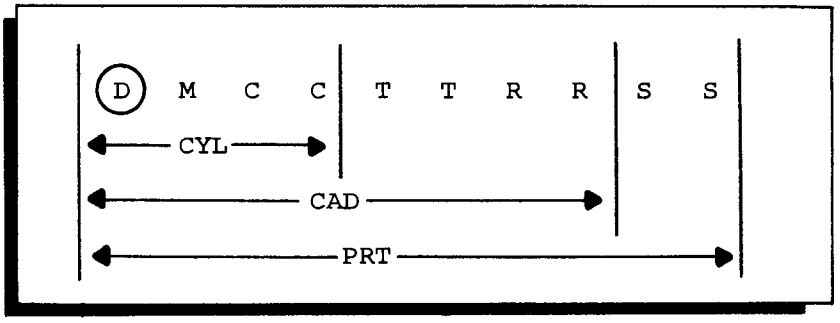


Figure B-1. MPCA Ten Character Field

The significance of the characters in the field is as follows:

D = Device number
M = Magazine number, must be zero
CC = Cylinder number
TT = Tract number
RR = Record number
SS = Status

The right-most two characters of the field contain the status and error condition in the most significant character position and the type of file protection in the least significant character position. Whenever the mnemonic PRT is used in an MLCA macro to load the 10 character field,

the left-most character position of the status portion of the field must be zero. The file protection character can be set to any of the following binary numbers:

```
000001 = Permit format write
000010 = Permit data write
000011 = Permit format and data write
000100 = Permit A-File write
000101 = Permit format and A-File write
000110 = Permit data and A-File write
000111 = Permit format, data and A-File write
001000 = Permit B-File write
001001 = Permit format and B-File write
001010 = Permit data and B-File write
001011 = Permit format, data and B-File write
001100 = Permit A- and B-File write
001101 = Permit format, A- and B-File write
001110 = Permit data, A- and B-File write
001111 = Permit all writes
```

Notice that the value of a given bit must be zero to protect against the corresponding type of write and must be one to permit that type of write. A write operation cannot be performed if the corresponding switch is not set at the control unit. Also, the data write permit bit must be one to allow any type of write.

Considerations For Action Macros

Normally, return from an action macro is to the location following the generated coding. When an uncorrectable error was caused by the action, however, return is made as the address specified by the EAD field of the associated MPCA. In the action macro call, parameter ØØ, written in the location field on the coding form, may be used as a tag referrencing the first (high-order) character of the generated coding. Parameter Ø1 of the action macro call, starting in column 21 on the coding form, must be assigned the same value as the unique prefix specified as parameter ØØ of the related

MPIOC.

READ ACTION MACRO

The Read macro always performs a data transfer from mass storage to main memory, and may perform MLCA macro functions. The type of read operation performed depends on the TRW field of the associated MPCA (previously described).

WRITE ACTION MACRO

The Write macro operates like the read macro except that the data transfer is in the opposite direction, i.e., from main memory to mass storage. Note, however, that if the verify bit is set in the TRW, no data transfer occurs; only the readability of the data is checked.

VERIFY ACTION MACRO

The Verify macro is used to read the data recorded by the last write action. There is no data transfer associated with the verify operation, but this is not the same as specifying a read action with the TRW bit set to verify in the MPCA. When desired, the verify call must be issued after a write to be checked and before any other action call is issued.

WAIT ACTION MACRO

Whenever the programmer intends to check the last action initiated by MPIOC, via the appropriate MPCA, for error free completion, he issues a WAIT action macro call. If the MPCA indicated in the call was not the last MPCA to be active, there is no guarantee that any other action initiated by MPIOC was completed successfully. If the action was completed successfully, a normal return to the user is made. If the action was not completed successfully, the error detection and correction action is performed automatically. If the error is corrected, a normal return to the user is made; if not, the user's uncorrectable error routine is entered.

RESTORE ACTION MACRO

Whenever the programmer intends to restore a device to its initial state, he issues a restore action macro call, the address register is checked for error indications, the restore action is entered, and a normal return to the user's coding is made. When this return is made, there is no guarantee that the restore operation was completed successfully.

Considerations For The User's Uncorrectable Error Routine

When MPIOC returns control to the user at the address specified in EAD of the appropriate MPCA, the user's program must direct the actions to be taken because of the condition that has occurred. The MPCA involved in the condition contains information that enables the user's program to determine which path to follow at this point. The user can do one of three things at this point: he can re-enter MPIOC and re-initiate the action, he can cause the instruction in the action coding that caused the error to be bypassed, or he can issue a different action macro call.

RE-EXECUTION OF THE CORRECTION PROCEDURE

At the time of the return to the user's coding, the B-address register contains the address at which MPIOC may be re-entered to try the instruction sequence in error again. This return is especially valuable if the ERI contains the value Ø1. Ø2, Ø3, or Ø4, since these types of errors may possibly be corrected by manual action.

Suggested manual operations to be performed in these cases are listed below.

VALUE OF ERI	CONDITION	SUGGESTED CAUSE AND MANUAL ACTION
Ø1	Device inoperable	A. Device may not be turned on. B. Device may be cycled down. Stop the device (if necessary) and cycle it up.
Ø2	Protection violation	Manual protection switches may not be set correctly. Set the switches correctly.

VALUE OF ERI	CONDITION	SUGGESTED CAUSE AND MANUAL ACTION
Ø3	Device error	Clear the error condition at the device.
Ø4	Format violation or overflow	Same as protection violation.

All other conditions except that represented by an ERI of 11 may possibly be corrected by re-execution. Note that MPIOC has already made a number of attempts to correct the condition.

BYPASS THE ERROR CONDITION

If the programmer intends to accept the last execution of an operation as correct, he can re-enter MPIOC to bypass the error condition. This may have some value in certain cases such as a read error, but could be a dangerous situation if the operation in error was a seek operation. To bypass the error condition, the programmer must add one plus the current address mode to the value stored in the B-Address Register. For example, if Physical I/O is being executed in the three character address mode, then in order to bypass the error condition, a four must be added to the B-Address Register value.

ISSUING A NEW MACRO CALL

When the programmer intends to discontinue the current path of processing because of an error condition, he can issue a different action macro call. Any action call can be issued, but some will not be very effective because of the type of error encountered. For example, if the error condition is that the device is inoperable, any new action will not be completed successfully until the error condition is corrected.

APPENDIX C

I/O COMMUNICATIONS AREA SERVICE MACROS

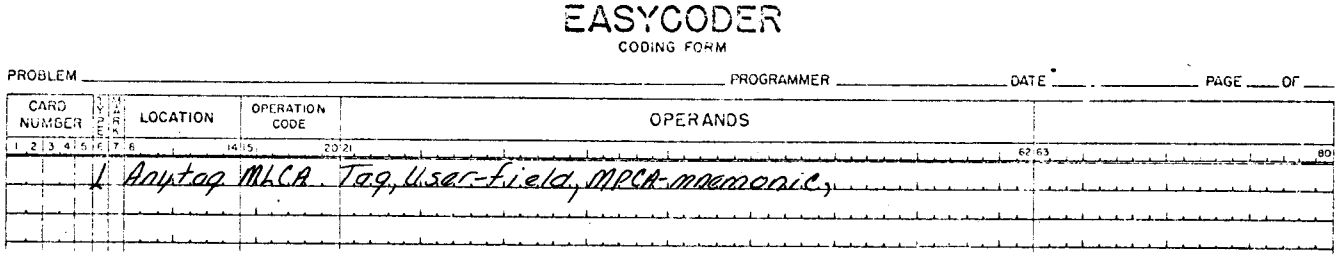
INTRODUCTION

There are two I/O Communications Area Service Macros. The Mass Storage Load the Communication Area (MLCA) macro and the Mass Storage Unload the Communications Area (MUCA) are used to store or alter fields in the I/O Communications Area (MLCA) and to access the values of certain fields in the I/O Communications Area (MUCA). Any address specified as a parameter of these macros may be of any type. But, Index Registers 5 and 6 must not be referenced.

MLCA MACRO

The MLCA macro provides the user with the ability to update the contents of fields in the I/O Communications Area. Each of these fields has a specific mnemonic designator. To alter the contents of a specific field, the user must associate its designator with a main memory address in which the value to be placed in the I/O Communications Area is stored. At execution time, the MLCA macro will set the indicated area to its proper status in the I/O Communications Area. As many of these pairs (value and main memory address) as are required may be specified in a single MLCA macro.

MLCA Macro Format



MLCA Macro Description

TYPE FIELD

The last line of the MLCA call must always contain an L. All other lines of the MLCA call must contain a C in the Type Field.

LOCATION FIELD

The Location Field is considered as parameter Ø of the MLCA macro. This field can contain any acceptable assembly tag, but does not have to be specified.

OPERATION CODE FIELD

The Command Field contains the Operation Code MLCA, which informs the system of what function to perform.

OPERANDS FIELD

The Operands Field contains the parameters for the MLCA macro.

Parameter 1

Parameter 1 of the MLCA macro is a tag that can be up to three characters long. This tag is used as a prefix to all MPCA macro tags and must be the same as parameter Ø of the MPCA macro.

NOTE: Subsequent parameters of the MLCA macro are treated as pairs. The first parameter of each pair is the main memory address containing the value to be placed in the I/O Communications Area, and the second parameter of each pair is the mnemonic designator of the field to be updated. The first omitted (blank) main memory address parameter or mnemonic designator will terminate the action. The order in which the pairs are specified is not significant unless one field will overlay another.

Parameter 2

This parameter contains the right-hand end address of a user supplied field that contains the data required to alter the I/O Communications Area

table. The field must contain a word mark to stop the right to left transfer of the value.

Parameter 3

This parameter contains a mnemonic indicating the right-hand end of the actual area within the appropriate MPCA table which is to be altered. If the low-order characters of the designated field are not to be updated, address arithmetic may be used in specifying the mnemonic designator (i.e. CAD-1).

Table C-1 lists each acceptable mnemonic designator and provides a description of each.

Table C-1. MLCA Mnemonic Designators for MLCA and MUCA

MNEMONIC	DESCRIPTION
CYL	This designator refers to a 4 character field containing the device, magazine, and cylinder number, in binary, for any future action related to the File Table referenced by parameter Ø1. The left-most character of this field must contain a word mark.
CAD	This designator refers to an 8 character field containing the field CYL and its high-order characters. The remaining four characters are the 2 character track and the 2 character record numbers in binary and in that order. Note that CAD and CYL can overlay each other. The final value of this field will be loaded into the address register by MPIOC whenever required. The left-most character of this field must contain a word mark.
PRT	This designator refers to the right-hand end of a 1Ø character field whose high-order 8 characters are defined by CAD. The character to the right of CAD must be Ø. The tenth character corresponds to parameter Ø5 of MPCA. MPIOC will load the address register of the control unit with the current value of these 1Ø characters. Note that the 1Ø characters include CAD, which, in turn, includes CYL.
TRW	This designator refers to a single character corresponding to parameter Ø4 of MPCA (C3 variant).
AAD	This designator corresponds to the buffer address as defined for parameter Ø2 of the MPCA macro.
EAD	This designator refers to the entrance address to a user's error routine. Refer to parameter Ø3 of the MPCA macro.
RWC	This designator refers to the read/write counter value and is significant only if parameter Ø5 of the MPIOC is VAR.
CPU	This designator refers to the peripheral trunk and is significant only if parameter Ø4 of the MPIOC macro is M.

Parameters 4, 6, and 8

These parameters are the same as parameter 2 described proviously.

Parameters 5, 7, and 9

These parameters are the same as parameter 3 described previously.

MUCA MACRO

This macro provides the user with the ability to access the contents of fields of the I/O Communications Area. The use of this macro corresponds to that of MLCA except that the transfer of values is in the opposite direction. That is, information is transferred from the I/O Communications Area to the main memory location indicated. Each such transfer moves from right to left (data bits only) as many characters as there are in the designated field. If right-most characters of the field are not desired, address arithmetic may be used with the mnemonic designator. For example, EDF-4.

MUCA Macro Format

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE ___ . _____ PAGE ___ OF ___

CARD NUMBER	TYPE	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6 7	8 14	15 20	21 62	63 80
1	L	Anytag MUCA	Tag,User-field,MLCA-mnemonic,...,...		
2					
3					

MUCA Macro Description

TYPE FIELD

The requirements for the Type Field of MUCA are the same as described for MLCA.

LOCATION FIELD

The requirements for the Location Field are the same as described for MLCA.

OPERATION CODE FIELD

The requirements for the Command Field of MUCA are the same as described for MLCA except that the Operation Code MUCA must appear in this field.

OPERANDS FIELD

The Operands Field contains the parameters required for MUCA.

Parameter 1

Parameter 1 of MUCA is the same as described for MLCA.

Parameter 2

Parameter 2 of MUCA is the same as described for MLCA.

Parameter 3

Parameter 3 of MUCA is the same as described above for MLCA in Table C-1.

The following mnemonic designators in Table C-2 can also be used with the MUCA macro.

Table C-2. Additional Mnemonic Designators For MUCA

MNEMONIC	DESCRIPTION
LAD	This designator refers to an 8 character field designating the address of the last record involved in the previous data transfer initiated via the related I/O Communications Area. Its format is DMCCTTRR (Device, Magazine, Cylinder, Track, and Record numbers).
ECT	This designator refers to a 1 character field containing a binary count of the number of re-reads or re-writes executed by MPIOC in attempting to correct read and write errors detected in exectuions for the designated MPCA Table.
ERI	This designator refers to a single character field containing an indication of the type of uncorrectable error condition which was last encountered in executions for the designated MPCA Table.
EDF	This designator refers to a 14 character field containing the contents of the address register at the time the last error condition was detected in executions for the designated MPCA File Table.
LRT	This is a field containing the address of the last return, to the user, from the initiation of an action referencing the designated MPCA Table.

Parameters 4, 6, and 8

These parameters are the same as parameter 2 of the MUCA macro.

Parameter 5, 7, and 9

These parameters are the same as parameter 3 of the MUCA macro.

PROGRAMMER'S PREPARATION INFORMATION

General Description Of MLCA And MUCA Macros

MLCA MACRO

The MLCA macro is used at execution time to alter the values of certain fields of a specified Physical I/O Communications Area (MPCA). Each MLCA macro call, therefore, must identify (via parameter Ø1 of the macro call) the appropriate MPCA, the areas in the user's program where the new values are located (via the even numbered parameters Ø2, Ø4, Ø6 etc. of the macro call) and the specific fields of the MPCA to be changed (via the odd numbered parameters Ø3, Ø5, Ø7 etc. of the macro call). The odd numbered parameters Ø3, Ø5, Ø7 etc. are assigned a mnemonic from the listings on pages C 4 and C 7. Each set of even and odd parameters, such as the set Ø2 and Ø3 or the set Ø4 and Ø5, are treated as pairs. As many as 31 pairs can therefore effect as many as 31 changes to the MPCA in a single MLCA macro call.

At execution time the change in the values of the specified MPCA fields is accomplished by an Extended Move (EXM) instruction in the MLCA coding. This instruction moves only the data bits in the user's fields (from right to left) from the user's fields to the specified fields of the MPCA. The move is terminated by a word mark at the left-hand (high-order) end of the user's field. Because of this, the user must ensure that his fields are the same length as the fields of the MPCA that are being changed.

Any acceptable form of addressing, including referrencing index registers X5 and X6, may be specified as values for the user's area addresses (even numbered parameters).

MUCA MACRO

The MUCA macro is used at execution time to access the values of certain fields within a specified MPCA. This macro operates in the same manner as the MLCA macro, so its parameters are specified in the same manner. The notable differences between these macros is that while the data bits are moved from right to left in the MUCA macro (as in the MLCA macro) the data is transferred from the MPCA to the user's fields. This last is just opposite to the MLCA macro. Also, the moves are terminated by a word mark in the MPCA fields rather than in the user's fields.

MLCA And MUCA Parameters

The information following is a supplement to the information in the preceding portion of this appendix.

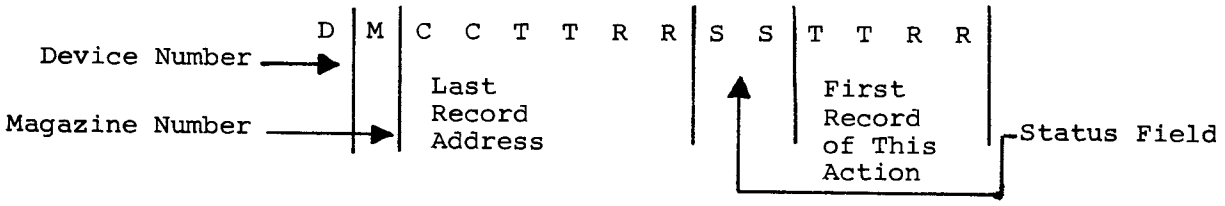
ERROR TYPE INDICATOR (ERI) MNEMONIC DESIGNATOR

The designator ERI refers to a single-character field indicating the type of the last error encountered while processing with this MPCA. The possible values of this field and their meanings are contained in the following list.

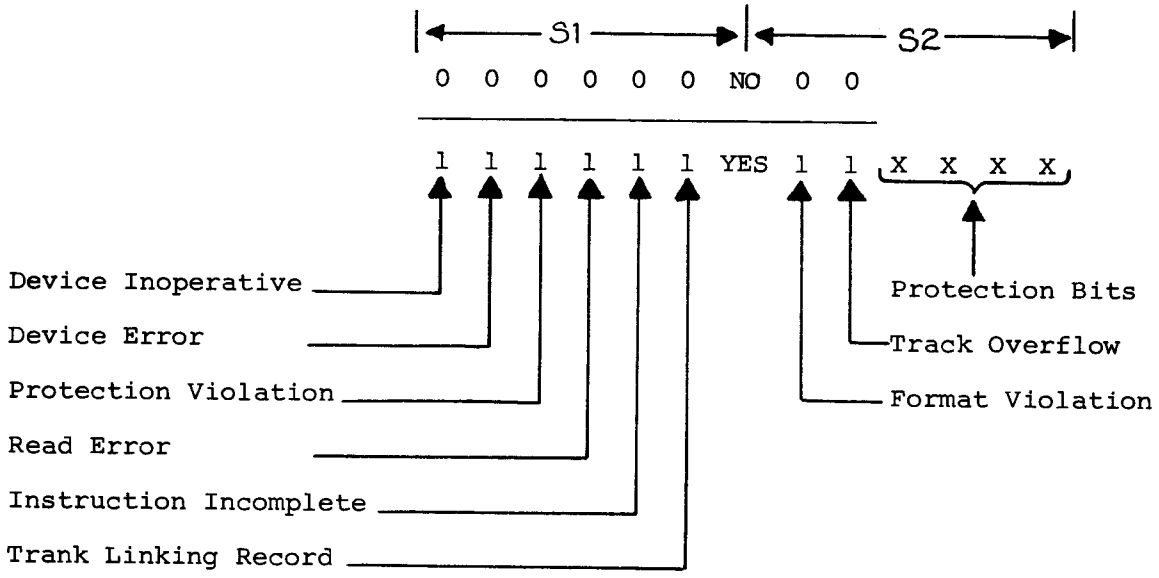
OCTAL VALUE	MEANING
ØØ	No errors.
Ø1	Device Inoperative.
Ø2	Protection violation.
Ø3	Device error (after five attempts at positioning).
Ø4	Formatting error (format violation or track overflow).
Ø5	Addressed record not located (after five attempts).
Ø6	Uncorrectable read error, data transfer was completed.
Ø7	Uncorrectable read error, data transfer was not completed.
1Ø	Automatic verification failed (after ten re-verify attempts).
11	Track linking record was read into core memory or re-written from core memory. (This is not necessarily an error.)
12	Read error in track linking record while attempting to link. Contents of current CCTTRR are invalid (after ten attempts at re-reading).

ADDRESS REGISTER CONTENTS AT TIME OF ERROR EXIT (EDF)

The designator EDF refers to a 14 character field that reflects the contents of the pcu address register at the time the last error condition was recognized by MPIOC for this MPCA. The format of this field is as follows.



The format of the status field is as follows.



APPENDIX D

TAPE AND CARD FORMATS USED IN FILE SUPPORT
LOAD/UNLOAD FUNCTION

INTRODUCTION

The Mass Storage File Support Subsystem has the facility to load data from and unload data to one-half inch magnetic tape or punched cards. All standard fixed length formats are allowed. This appendix summarizes these formats and points out any features that are extensions of previous Honeywell Series 200 software.

ONE-HALF INCH TAPE FORMATS

Header Label

The Header Label is 8Ø characters in length and must be the first record of a file. It consists of the following fields:

FIELD	CHARACTER POSITIONS	CONTENTS
1.	1 - 5	1HDRΔ
2.	6 - 10	Tape Serial Number
3.	11 - 15	File Serial Number
4.	16	- (minus sign)
5.	17 - 19	Reel Sequence Number
6.	20	Blank
7.	21 - 30	File Name
8.	31 - 35	Creation Date
9.	36	- (minus sign)
10.	37 - 39	Retention Cycle
11.	40	Blank
12.	41 - 80	Unused

The File Support Subsystem uses only fields 1, 2, and 7 with the exception of a Partitioned Sequential File.

When a Partitioned Sequential File exists on tape, each member is treated as one file and a multi-file reel. To identify the member on tape, the header includes an additional field in characters 51 through 64 giving the member name. The Load/Unload Function assumes that tapes are properly positioned. No searching for the file name or member name is performed. The Load/Unload Function operates according to the following rules.

When performing LOAD to a Partitioned Sequential File and the member name is specified in the job request, the member name field is not required in the label. The member name is taken from the job request, and the tape file currently positioned will be loaded as that member.

When performing LOAD to a Partitioned Sequential File and the member is not specified in the job request, the member name field in the label is required. All files on that reel of tape are loaded as members until a 1ERI record is encountered on that tape. The member name is assumed to be correct and is entered into the mass storage file.

When performing UNLOAD of a Partitioned Sequential File, the member name field in the tape label is always filled in by File Support.

Data Records

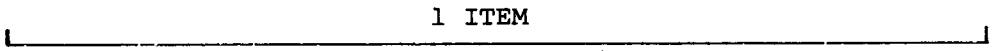
Data records processed by File Support must be fixed length (blocked or unblocked) and must use one of the following combinations of parity and bannering:

PARITY	ODD	ODD	EVEN	EVEN
BANNER[1]	YES	NO	YES	NO
PADDING	77_8	77_8	11_8	11_8

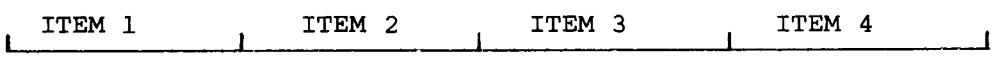
[1] It should be remembered that when banner is specified, one additional character should be provided in the REC = parameter.

The four types of data record blocking, bannering, and padding can be illustrated as follows:

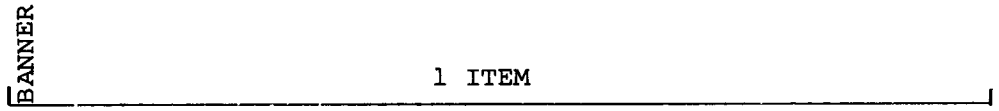
1. Unblocked, Unbannered



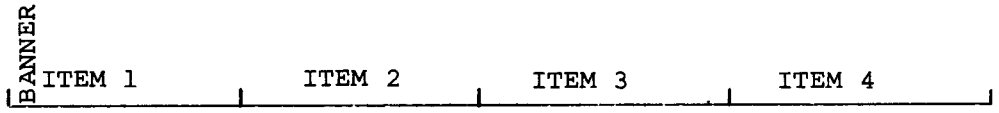
2. Blocked, Unbannered



3. Unblocked, Bannered



4. Blocked, Bannered



For those installations trying to decide which type of file to use, the odd parity, bannered file is the Honeywell recommended standard.

Trailer Labels

The trailer label is 8Ø characters in length and must be the first record following the last data record of a file. Only two fields of that record are used by File Support.

FIELD	CHARACTER POSITIONS	CONTENTS
1	1 - 5	Must be 1EOF△
2	6 - 10	Is not checked on input; on output (UNLOAD), the tape record count (decimal) is entered.

In the normal situation, this record is followed by an 8Ø character record containing 1ERIΔ (End of Record Information) in the first five characters. However, in Partitioned Sequential, each 1EOF record is followed by the next 1HDR record until all members are accounted for. Only the last 1EOF record is followed by 1ERI.

Tape Marks

Tape marks on an input tape are ignored. On output files, tape marks are not created.

CARD FILE FORMATS

Header Label

Each card file must have a label card with the 1HDRΔ in columns 1 - 5 and optionally the file name in columns 21 - 30. Partitioned Sequential Files are handled in exactly the same way as in the one-half inch tape files described in this appendix.

Data Records

Card records are always unblocked. The record consists of the minimum number of cards which can handle one item. Any character positions left over are ignored. Each item is assumed to start in column 1.

Trailer Labels

Trailer labels for cards are the same as for one-half inch tape as described in this appendix.

APPENDIX E

PARTITIONING

INTRODUCTION

When the partitioning option is used, there are several additional advantages to the sequential file organization. With this option, the sequential file is broken into any number of sub-files (called members). Each member of a partitioned sequential file must have identical properties such as item size, record size, etc. A member index is maintained to enable direct access to the beginning of any member. The number of blocks required to store the member index is specified by the user. The member index begins with the first block in the file and continues through the number of blocks specified. The record size and block size of the member index are identical to those of the data area of the file. The item size of the member index contains the name of the member, its address, the number of blocks in the member, and the status of the member.

MEMBER INDEX

The name of the member identifies the member. A member name is 14 characters in length. The address of the member is the address of the first record in the member. The address is of the form

CCTTRR

This identifies the cylinder, track, and record of the first item of the member. The block count simply records the number of blocks in the member. The status of a member may be one of the following:

1. Deleted
2. Able to be processed as input or input/output only
3. Able to be processed as input/output, input only, or output only.

The processing modes are fully described in the paragraphs that discuss the input/output control functions. An example of a member index for a given partitioned sequential file is shown in Figure E-1.

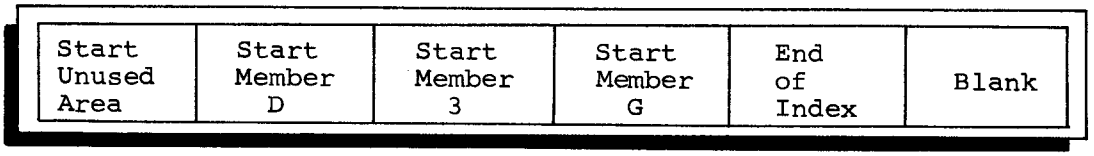


Figure E-1. Member Index for a Partitioned Sequential File

The first item in the index always contains the address of the first record in the file that is available for the addition of a new member. When the partitioned sequential file is created, that is, when it is formatted and space is allocated and before data is recorded in it, the member index contains items indicating the unused area and the end of index. When a member is deleted, its data area is not reusable until the file has been reorganized. Figure E-2 shows a sequentially organized file using the partitioning option.

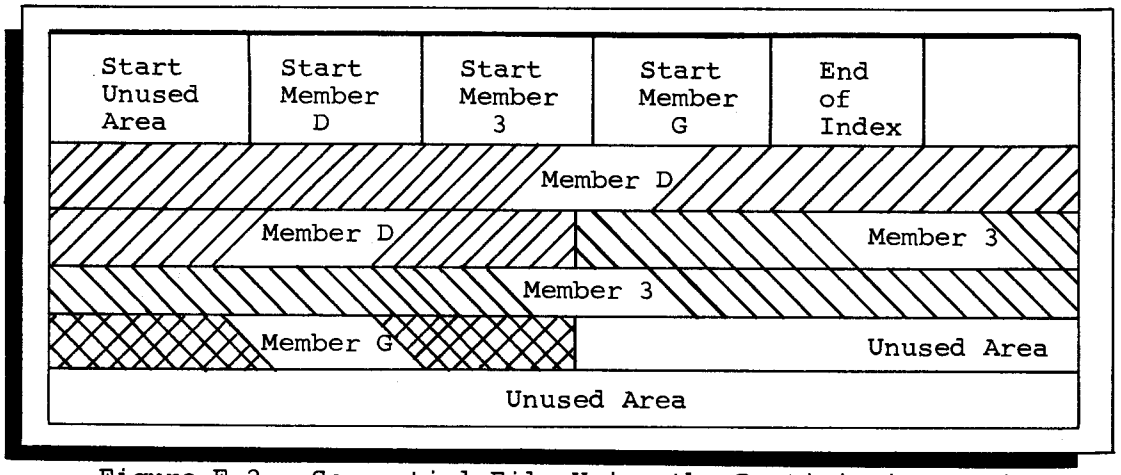


Figure E-2. Sequential File Using the Partitioning Option

NOTE: If N blocks are reserved for a member index, the number of members which this index can hold is

$$\frac{N \text{ Block Size}}{25} - 2.$$

APPENDIX F

MASS STORAGE FILE PROTECTION

FILE PROTECTION

The introduction of mass storage devices into data processing brings additional considerations into the area of data file protection. In magnetic tape processing, several methods of protection against inadvertent destruction have been in use for some time. With the Honeywell 204B tape drives, a user may put any drive in protect by using a manual switch on that drive. He may also remove the plastic ring on the back of the tape itself. Finally, in common practice, each file is contained on a separate reel of tape. These three methods of protection are generally adequate. In addition, if a particular tape file is confidential, it's owner (for example, a payroll department) can keep that reel in its own restricted area of storage. This guarantees that no unauthorized persons have access to this file.

On mass storage, however, it is common for more than one file to exist on a single volume. When this is true, the old "tape oriented" methods of protection are not adequate. To provide the user with maximum data protection, the Honeywell Mass Storage Operating System offers two types of protection: 1. A hardware/software protection against inadvertent data destruction; 2. A software protection against unauthorized access to a confidential file.

These two features are explained in detail below.

WRITE PROTECTION

There are four classes of write protection offered through a combination of hardware and software features. These classes are:

1. Format Write Protection
2. Data Write Protection

3. "A-File" Write Protection

4. "B-File" Write Protection

Corresponding to these four classes of write protection are four hardware switches.

For example, to do any formatting, the Format Write Switch must be in PERMIT. In terms of this operating system, formatting would occur during any run of Volume Preparation or a run of File Support which is doing allocation.

Any program which is doing any form of writing (e.g. an update as assembly or a sort) requires that the control unit to which the write is being done must have the Data Write Switch in PERMIT. In addition, if it is a user's program, parameter 31 in MIOC of Logical I/O must be Ø2 (i.e., permit Data Write).

The use of these two switches is not optional. When formatting is in progress, the Format Write Switch must be in PERMIT. When writing is in progress, the Data Write Switch must be in PERMIT.

The use of "A-File" and "B-File" protection, however, is optional. For example, if there is a master file which may only be written on by a limited, well-defined number of programs, it may be desirable to give this file further protection. To illustrate, let us suppose that FILE-X is a payroll master file which may be updated by only one program. In addition to the payroll file, however, there may be from time to time one or two other files on the same volume. To protect FILE-X from inadvertent destruction, it is decided to give this file "B-File" protection.

When allocating FILE-S, the parameter PROT = B is used. If the file is being loaded by the File Support Load function, the PROT = B parameter must be used again. In addition, the "B-File" Write Switch and Data Write Switch must both be in PERMIT during this load process.

The program written to update this file must include in parameter 31 of Logical I/O's MIOC macro the value 12 (permit "B-File" Write Switch and the Data Write Switch must both be in PERMIT.

The possible combinations of file write protection are:

Combination of Protection	Switches in Permit When Writing
NONE	Data Write
"A-File"	Data Write & "A-File"
"B-File"	Data Write & "B-File"
"A-File" and "B-File"	Data Write, "A-File", & "B-File".

PASSWORD PROTECTION

In addition to guaranteeing that a file will not be improperly destroyed, it is often important to guarantee that a file is not read by unauthorized personnel. Thus, in the preceding example, it is important to be able to know that the other users of the volume containing FILE-X, the payroll master file, cannot open, read, or write FILE-X. To effect this type of protection, this operating system provides the use of a password.

Thus, in the above example, a password of PAY66164 might be used. During Allocation, the parameter PW = PAY66164 is entered. If using the Load function, PW = PAY66164 is used again. Any program accessing FILE-X must have as parameter 21 of Logical I/O's MCA macro a tag pointing to a field in memory which contains PAY66164.

For example:

```
        C       MCA     .....
        C       21      PWORD
PWORD   DCW     @PAY66164@
```

APPENDIX G

SPACE ALLOCATION FOR SEQUENTIAL FILES

The unit of allocation is of the form $C_1T_1C_2T_2$. To determine how much space is required for a given Sequential File, the following process is used:

1. The following figures must be known and are represented symbolically as follows:
   A. Block (or Buffer) Size = BK
   B. Total number of items in the file = I
   C. Tracks per Cylinder = T[1]
   D. Items per Block = IB

2. Using Table G-1, locate the correct values for Number of Records per Track (RT) and number of Records per Block (RB). This is accomplished by going down the left-hand column to locate the Block Size (BK) and then taking the corresponding values for Records per Track (RT) and Records per Block (RB).

3. Compute Blocks per Cylinder (BC) as follows:
   $BC = \frac{RT \times T}{RB}$ (Ignore any remainder.)

4. Compute Items per Cylinder (IC) as follows:
   IC = BC x IB

5. Compute the number of Cylinders (C) required for this file as follows:
   $C = \frac{I}{IC}$ (Round up to the next higher integer.)

[1] Normally, Tracks per Cylinder (T) will be 1Ø for the 258 or 259 Disc. The user may, however, use any smaller number of tracks.

EXAMPLE

Assume the following:

Block Size (BK) = 1218

Total Items (I) = 55ØØ

Tracks per Cylinder (T) = 1Ø

Items per Block (IB) = 6

There is to be one unit of allocation starting on Cylinder 2Ø.

1. From Table G-1, Records per Track (RT) = 1Ø and Records per Block (RB) = 3.

2. Blocks per Cylinder (BC) = $\frac{1Ø \times 1Ø}{3}$ = 33. (The remainder is dropped.)

3. Items per Cylinder (IC) = 33 x 6 = 198

4. Cylinders for the file (C) = $\frac{55ØØ}{198}$ = 27 (plus a remainder of 154.) This is rounded up to 28.

5. Therefore, the unit of allocation for this file in the form $C_1 T_1 C_2 T_2$ would be: 2Ø - Ø - 47 - 9.

Table G-1. Optimum Record Size

CHARACTERS PER BLOCK (BK)	RECORD SIZE	NUMBER OF REC/TRACK (RT)	NUMBER OF REC/BLOCK (RB)
79 - 82	Same as block	32	1
83 - 87	Same as block	31	1
88 - 92	Same as block	30	1
93 - 97	Same as block	29	1
98 - 104	Same as block	28	1
105 - 110	Same as block	27	1
111 - 116	Same as block	26	1
117 - 124	Same as block	25	1

Table G-1 (cont). Optimum Record Size

CHARACTERS PER BLOCK (BK)	RECORD SIZE	NUMBER OF REC/TRACK (RT)	NUMBER OF REC/BLOCK (RB)
125 - 132	Same as block	24	1
133 - 140	Same as block	23	1
141 - 150	Same as block	22	1
151 - 160	Same as block	21	1
161 - 171	Same as block	20	1
172 - 184	Same as block	19	1
185 - 197	Same as block	18	1
198 - 213	Same as block	17	1
214 - 230	Same as block	16	1
231 - 250	Same as block	15	1
251 - 271	Same as block	14	1
272 - 297	Same as block	13	1
298 - 327	Same as block	12	1
328 - 363	Same as block	11	1
364 - 406	Same as block	10	1
407 - 459	Same as block	9	1
460 - 523	Same as block	8	1
524 - 608	Same as block	7	1
609 - 721	Same as block	6	1
722 - 726	Block/2	11	2
727 - 877	Same as block	5	1
878 - 918	Block/2	9	2
919 - 1112	Same as block	4	1
1113 - 1216	Block/2	7	2
1217 - 1218	Block/3	10	3
1219 - 1506	Same as block	3	1
1507 - 1569	Block/3	8	3

Table G-1 (cont). Optimum Record Size

CHARACTERS PER BLOCK (BK)	RECORD SIZE	NUMBER OF REC/TRACK (RT)	NUMBER OF REC/BLOCK (RB)
1570 - 1754	Block/2	5	2
1755 - 1824	Block/3	7	3
1825 - 1836	Block/4	9	4
1837 - 2289	Same as block	2	1
2290 - 2295	Block/4	9	4
2296 - 2432	Block/3	7	3
2433 - 2631	Block/3	5	3
2632 - 3012	Block/2	3	2
3013 - 3040	Block/5	7	5
3041 - 3336	Block/3	4	3
3337 - 3508	Block/4	5	4
3509 - 3605	Block/5	6	5
3606 - 3648	Block/6	7	6
3649 - 3661	Block/7	8	7
3662 - 3672	Block/8	9	8
3673 - 4578	Block/2	2	2

EXAMPLE FOR USING THE TABLE

To provide for a 36ØØ character block, go down the left-hand column until 36ØØ is bracketed. Reading the appropriate line shows:

35Ø9 - 36Ø5 Block/5 6 5

Thus, the record size would be $\frac{36ØØ}{5}$ = 72Ø.

There would be six 72Ø-character record per track and five records per block. This makes 1 1/5 blocks per track.

APPENDIX H

ALLOCATION AND ADDRESSING FOR DIRECT ACCESS FILES

To be properly utilized, a direct access file requires careful planning. There are two essential inputs the user himself must calculate: (1) Proper arrangement and formatting of storage space, and (2) Addresses for every item, assigned in such a way that items are dispersed as evenly as possible throughout the space allocated to the file. The following paragraphs provide some general guidelines for arranging storage space and assigning addresses in direct access files.

SPACE ALLOCATION

Any method of calculation used to translate item keys into storage addresses generally produces a number of duplicate addresses (synonyms). These synonyms are handled by two formatting methods: (1) Buckets are made large enough to handle several items, and (2) Overflow areas are provided to handle the bucket overflow due to uneven distribution. If every bucket contained the same number of synonyms, there would be no overflow. But since some buckets contain more and some less, some will overflow.

The amount of overflow that occurs is directly related to two factors: (1) Bucket Size, and (2) Storage Density. The more items there are in a bucket, the lower the probability for any item that it will overflow. (But also, since increasing the number of blocks in a bucket increases the average time required to access an item, the average access time to an item may be much higher.) Storage density also affects bucket overflow. A file with space for 1000 items will have more bucket overflow when it contains 800 items

Table H-1. Overflow Probabilities

ITEMS PER BUCKET1	Storage Density and Number of Items/Allocated Space											
Bucket Size	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2
1	4.84	9.37	13.61	17.58	21.31	24.80	28.08	31.17	34.06	36.79	39.35	41.77
2	0.60	2.19	4.49	7.27	10.36	13.65	17.03	20.43	23.79	27.07	30.24	33.30
3	0.09	0.63	1.80	3.61	5.99	8.82	11.99	15.37	18.87	22.40	25.91	29.33
4	0.02	0.20	0.79	1.96	3.76	6.15	9.05	12.32	15.86	19.54	23.25	26.93
5	0.00	0.07	0.37	1.12	2.48	4.49	7.11	10.26	13.78	17.55	21.42	25.30
6	0.00	0.02	0.18	0.67	1.69	3.38	5.75	8.75	12.24	16.06	20.06	24.11
7	0.00	0.01	0.09	0.41	1.18	2.60	4.74	7.60	11.04	14.00	19.00	23.19
8	0.00	0.00	0.05	0.25	0.84	2.03	3.97	6.68	10.07	13.96	18.15	22.46
9	0.00	0.00	0.02	0.16	0.61	1.61	3.36	5.94	9.27	13.18	17.44	21.86
10	0.00	0.00	0.01	0.10	0.44	1.29	2.88	5.32	8.59	12.51	16.85	21.36
11	0.00	0.00	0.01	0.07	0.33	1.04	2.48	4.80	8.01	11.94	16.34	20.94
12	0.00	0.00	0.00	0.04	0.24	0.85	2.15	4.36	7.51	11.44	15.89	20.58
14	0.00	0.00	0.00	0.02	0.14	0.57	1.65	3.64	6.67	10.60	15.15	19.99
16	0.00	0.00	0.00	0.01	0.08	0.39	1.28	3.09	6.00	9.92	14.56	19.53
18	0.00	0.00	0.00	0.00	0.05	0.28	1.01	2.65	5.45	9.36	14.07	19.16
20	0.00	0.00	0.00	0.00	0.03	0.20	0.81	2.30	4.99	8.88	13.66	18.86
25	0.00	0.00	0.00	0.00	0.01	0.09	0.48	1.65	4.10	7.95	12.87	18.31
30	0.00	0.00	0.00	0.00	0.00	0.04	0.29	1.23	3.47	7.26	12.31	17.93
35	0.00	0.00	0.00	0.00	0.00	0.02	0.18	0.94	2.98	6.73	11.87	17.66
40	0.00	0.00	0.00	0.00	0.00	0.01	0.12	0.73	2.60	6.29	11.53	17.47
50	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.45	2.01	5.63	11.03	17.20
60	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.30	1.65	5.14	10.68	17.03
70	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.20	1.37	4.76	10.41	16.93
80	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.13	1.14	4.46	10.21	16.86
90	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.97	4.20	10.05	16.80
100	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.83	3.00	9.92	16.77

NOTES:
1. These probabilities are given as percentages.
2. This table assumes random distribution. In actual practice, perfect random distribution is seldom, if ever, obtained. The actual probability of overflow, therefore, will usually be higher.

(storage density = 0.8) than when it contains 500 items (storage density = 0.5). In allocating a direct access file, these two factors must be weighed against each other to achieve a desirable compromise.

Table H-1 summarizes the overflow probabilities for any item, assuming a random distribution.

ALLOCATION PROCEDURES

To illustrate the procedure for allocation of a Direct Access file we shall use two examples: one to show how to optimize speed; the other to show how to optimize storage density.

The following figures are fixed for both cases:

Size of file	10,000 items
Characters per item	200
Characters per block	800
Storage Density	0.8
Tracks per cylinder	10

Example 1:

For this example, we shall make the bucket equal to one block. Thus, each bucket has a capacity of four items. Using the probabilities chart (Table H-1), we see that the likelihood of any one item overflowing its bucket is 12.3%. If we allow one track for cylinder overflow, we would have 1/9 or 11.1% of the cylinder set aside for overflow.

If the important consideration for this file is average access time, then one track per cylinder for overflow would probably be sufficient (along with the general overflow). However, if it is important that no access exceed a certain time limit, then two tracks could be used to gain 2/8 or 25% overflow provision.

Using the one track for overflow, the cylinders required for allocation would be computed as follows:

$$\text{Item Space required} = \frac{10{,}000}{.80} = 12{,}500$$

$$\text{Buckets required} = \frac{12{,}500 \text{ items}}{4 \text{ item/bucket}} = 3125 \text{ buckets}$$

From Table G-1 in Appendix G, we see that we should have 5 blocks per track, hence, 5 buckets per track.

$$\text{Buckets per cylinder} = \frac{5 \text{ buckets}}{\text{track}} \times \frac{9 \text{ tracks}}{\text{cylinder}} = \frac{45 \text{ buckets}}{\text{cylinder}}$$

$$\text{Cylinders per file} = \frac{3125 \text{ buckets}}{45 \text{ buckets/cylinder}} = 69.4 \text{ or } 70$$

plus one cylinder per unit of allocation for general overflow. Assuming that there is one unit of allocation, there would be 71 cylinders required for this file.

If we were to use two tracks for overflow, then the following calculations change.

$$\text{Buckets per cylinder} = \frac{5 \text{ buckets}}{\text{tracks}} \times \frac{8 \text{ tracks}}{\text{cylinder}} = \frac{40 \text{ buckets}}{\text{cylinder}}$$

$$\text{Cylinders per file} = \frac{3125 \text{ buckets}}{40 \text{ buckets/cylinder}} = 78.1 \text{ or } 79$$

Again, if general overflow is desired, it is added in accordingly.

Example 2:

In the second example, we shall strive to make more efficient use of storage, and sacrifice a little speed. So we will plan to have only two buckets per cylinder. If we set aside one track for overflow, we would have:

$$\frac{9 \text{ tracks}}{\text{cylinder}} \times \frac{5 \text{ blocks}}{\text{track}} = \frac{45 \text{ blocks}}{\text{cylinder}}$$

$$\frac{45 \text{ blocks per cylinder}}{2 \text{ buckets per cylinder}} = 22.5 \text{ blocks per bucket.}$$

Since there cannot be partial blocks in a bucket, there are 22 blocks per bucket. (With 4 items per block, this gives 88 items per bucket.)

Looking at the probability chart (Table H-1), we see that the likelihood of overflow in this case is about 0.1%. This is so small that it would be considerably more efficient to have no cylinder overflow, but rather use only general overflow. In this case, we use 10 tracks and compute bucket size as follows:

$$10 \frac{\text{tracks}}{\text{cylinder}} \times 5 \frac{\text{blocks}}{\text{track}} = 50 \frac{\text{blocks}}{\text{cylinder}}$$

$\frac{50}{2}$ = 25 blocks per bucket (100 items per bucket).

To compute the cylinders required for this file, we already know that there are 2 buckets per cylinder.

$$\text{Buckets required} = \frac{12{,}500 \text{ items}}{100 \text{ items/bucket}} = 125 \text{ buckets}$$

$$\text{Cylinders per file} = \frac{125 \text{ buckets}}{2 \text{ buckets/cylinders}} = 62.5 \text{ or } 63 \text{ cylinders}$$

One cylinder per unit of allocation must be added for general overflow. Assuming that there is one unit of allocation, there would be 51 cylinders required for this file.

Note that in the first case, if we were using relative addressing, we would require addresses distributed between Ø and 3124. In the second case, we would reguire addresses between Ø and 125.

APPENDIX I

RANDOMIZING TECHNIQUES

RANDOMIZING ADDRESSING

Randomizing is the process of transforming the key of an item into a valid storage address. This actually consists of obtaining a relative bucket address (e.g., the 204th bucket of the file), which is then converted by the I/O into a valid mass-storage bucket address (i.e., cylinder, track, record). This enables the user to retain his present item numbering system, and yet have full on-line processing capability.

There are many randomizing methods, each one being somewhat better suited to one particular application than another. All have the same objective--to produce a valid storage address for each item from its control field (key) in such a way that the items are evenly distributed throughout the file. Depending on the randomizing technique employed, storage utilization can reach between 80 and 90 percent efficiency. But a file packed that densely requires an average of 1.2 to 1.5 accesses to retrieve each item, due to the duplicate storage addresses (synonyms) that occur. Generally speaking, a technique that achieves a high storage utilization generates more synonyms and so increases access time. So the randomizing technique chosen will depend on the relative importance of storage utilization and access time. But the structure of the file is also important and will affect the choice. For instance, if several items were blocked into a large multi-item record, then more synonyms would not adversely affect access time.

Once a randomizing technique has been selected for possible use, the technique should be evaluated with a sample selection of actual item keys. This evaluation should provide information on the efficiency of storage utilization, the frequency and distribution of synonyms, the processing time required for the calculation, and how evenly the generated storage addresses are distributed. The results will enable the user to select the

technique most suited to his particular requirements and data pattern.

The following paragraphs outline a few of the most commonly used key transformation methods. They have the advantage of being economical in processing time and core memory requirements. There are many possible variations of these, in addition to far more complicated methods not covered in this bulletin.

Prime Number Division

Division of the item key field by a prime number (a number divisible only by itself of unity) is a widely accepted method of transforming a key into a mass storage address. The prime-number divisor should be slightly less than the number of item locations allocated to the data area of the file. But it should be as large as possible. The larger the prime-number divisor, the smaller the chance of generating synonyms.

This method consists of dividing the item key by the selected prime-number divisor, discarding the quotient, and using the remainder as the basis for the mass-storage address.

EXAMPLE:

A file of 5,000 items on a Model 259 stores five items per bucket, one bucket per track, with item keys ranging from 000,000,000 to 999,999,999. Space is allocated to this file for 1,000 buckets. The file is to start on Cylinder 50, Track 0. The prime-number divisor chosen is 997, which will leave three buckets unused from the 1,000 allocated.

Now suppose that 777,775,925 is the key of the item to be accessed. Then $\frac{777,775,925}{997}$ = 780,116 with remainder 268

Thus, this item is to be placed in the 268^{th} bucket from the beginning of the file.

The I/O will then, using this relative bucket address, compute that the actual address is 26 cylinders and 8 tracks after the starting location of the file (Cylinder 50, Track 0), which in this case would be Cylinder 76, Track 8.

It has been assumed here that a unit of allocation is made up of a whole cylinder (10 tracks) and there are no cylinder overflow tracks.

In cases where purely alphabetic or mixed alphabetic/numeric item keys are concerned, the item key can be treated as a binary field to be binary divided by the binary form of the prime number. The final calculations will also be in binary so that the mass-storage address will be produced in a usable binary form.

Table I-1 is a list of prime numbers. It is divided into two sections: the first section contains every third prime between 2 and 2939, the second section contains every fifth prime between 2593 and 8039.

Table I-1. Prime Numbers

PRIMES (EVERY THIRD PRIME 2-2939)													
5	137	307	487	677	883	1093	1303	1543	1753	1999	2239	2447	2707
13	151	317	503	701	911	1109	1321	1559	1783	2017	2267	2473	2719
23	167	347	523	727	937	1129	1367	1579	1801	2039	2281	2531	2741
37	181	359	557	733	953	1163	1399	1601	1831	2069	2297	2549	2767
47	197	379	571	761	977	1187	1427	1613	1867	2087	2333	2579	2791
61	223	397	593	787	997	1213	1439	1627	1877	2111	2347	2609	2803
73	233	419	607	811	1019	1229	1453	1663	1901	2131	2371	2633	2837
89	251	433	619	827	1033	1249	1481	1693	1931	2143	2383	2659	2857
103	269	449	643	853	1051	1279	1489	1709	1951	2179	2399	2677	2887
113	281	463	659	863	1069	1291	1511	1733	1987	2213	2423	2689	2909

Table I-1 (cont). Prime Numbers

ADDITIONAL PRIMES (EVERY FIFTH PRIME - 2953-8039)												
2957	3343	3697	4073	4457	4861	5233	5641	6029	6373	6803	7211	7603
3001	3373	3733	4111	4507	4909	5281	5659	6067	6427	6841	7243	7649
3041	3433	3779	4153	4547	4943	5333	5701	6101	6481	6883	7307	7691
3083	3467	3823	4211	4591	4973	5393	5743	6143	6551	6947	7349	7727
3137	3517	3863	4241	4639	5009	5419	5801	6199	6577	6971	7417	7789
3187	3541	3911	4271	4663	5051	5449	5839	6229	6637	7001	7477	7841
3221	3581	3931	4327	4721	5099	5501	5861	6271	6679	7043	7507	7879
3259	3617	4001	4363	4759	5147	5527	5897	6311	6709	7109	7541	7927
3313	3659	4021	4421	4799	5189	5573	5953	6343	6763	7159	7573	7963

Square Enfold And Extract

The item key field is squared, the result is split in half, and the two halves are added together. Then the required number of digits needed for a mass-storage address are extracted from the middle of the result. Normally the two low-order characters are ignored and the extraction is made from the third low-order character and above.

EXAMPLE 1:

File of 10,000 items; item keys of 9 digits; 10 items per bucket; 1 bucket per track. Therefore, there are 1,000 buckets.

Control number: 493,725,816

Squared: 243, 765, 181, 384, 865, 856

Enfolded:
243,765,181
384,865,856
628,631,037

Extracted Result: 310 Relative bucket address.

Logical I/O Computes: $\frac{310}{10}$ = Cylinder 31 Track 0 (Model 259)

Since the field extracted will range over some power of 10, depending on the number of digits extracted, so unless the number of buckets available is some whole multiple of 10, the result of this calculation will not be suitable. The extracted number can be compressed by multiplying the

result by a percentage. If a 3-digit field is extracted, this gives a range of 1,000 numbers, which may be multiplied by 70% if there are only 700 buckets available.

EXAMPLE 2:

If the file consisted of 600 buckets instead of 1,000 buckets with the same control number range:

Control number:	569,183,582
Squared:	323,969,950,018,350,724
Enfolded:	323,969,950
	018,350,724
	342,320,674
Extracted Result:	206

60%: 123.60 (.60 discarded)

This gives a relative bucket address of 123.

Radix Conversion

When this method is applied to purely numeric item keys, each decimal digit is interpreted as if it were a radix-11 digit instead of the actual radix-10. When applied to alphabetic or alphanumeric item keys, where each character is treated as 2-octal digits which are edited into 2-decimal digits (see non-numeric item keys). Now each digit is interpreted as if it were a radix-9 digit instead of the actual radix-8. In this case, the numbers can only range from 0 to 7, whereas in the numeric case, the numbers could range from 0 to 9.

The normal procedure, then, is to truncate the result by discarding high-order digits until a field of the desired length is obtained. Note that compression of the resultant number can be done by multiplying it by a percentage as in the Square, Enfold, and Extract method.

EXAMPLE OF COMPRESSION

Item key 301,283

Radix - 11 = $(3 \times 11^5) + (0 \times 11^4) + (1 \times 11^3) + 2 \times 11^2) + (8 \times 11^1) + (3 \times 11^0)$

= 483,153 + 0 + 1,33, + 242 + 88 + 3

= 484,817, leaving 817 as the truncated address

(Cylinder 81, Track 7 on Model 259)

Radix conversion is a better method than truncation alone since it tends to disperse troublesome runs of keys differing in the numeric case by some power of 10 (e.g., 02309 and 12309) or in the alphanumeric case by some power of 8 (e.g., 0247_8 and 1247_8). The main advantage of this method is the simplicity of calculation. The conversion from radix-11 to radix-10 or from radix-9 to radix-8 may be accomplished without multiplication. It can be done simply by a series of decimal additions and shifts, or binary additions and shifts. Radix conversion does tend, however, to produce more synonums than prime number division.

EXAMPLE 1:

Item key 301,283 can be reduced to:

((((3x11+0)x11+1)x11+2)x11+8)x11+3)

```
        3
+      30
+     _0
       33
+    330
+     _1
      364
+   3640
+    _2
     4006
+  40060
+   _8
    44074
+ 440740
+    _3
   484817
   ______10
```

EXAMPLE 2:

Item key 247_9 can be reduced to:

(2x9+4)9+7

$$\begin{array}{r} 2 \\ 20 \\ \underline{\quad 4} \\ 26 \\ 260 \\ \underline{\quad 7} \\ 315_8 \end{array}$$

Non-Numeric Item Keys

Where item key fields comprise purely alphabetic or special characters, or a mixture of alphanumeric, one method is to treat the field as a binary number and perform binary arithmetic on it. This has the advantage of retaining zone bits and therefore avoiding unnecessary synonyms.

Another method is to consider each 6-bit character as 2-octal characters which are extracted to form 2-decimal digits in the range 0 to 7 by means of binary addition and extraction. The resultant key is then mainpulated by decimal arithmetic according to the particular method employed. This method is useful where binary arithmetic is impracticable, but it does result in doubling the length of the control fields.

EXAMPLE:

Key	Decimalized octal
810	10010000000000000000000000
8246Y2-951-7	10020406700240110501400700
8415RST	10040105516263000000000000
84X113-177-16	10046701010340010707400106
(13 characters)	(26 characters)

NOTE: One common misconception is that in converting alphabetic keys, the zone bits should be dropped before converting. This, however, immediately produces three groups of synonyms:

G, H, I P, Q, R X, Y, Z

Zone suppression, with the consequent advantages of decimal arithmetic, may be an acceptable method, however, for cases where the item keys are largely numeric with only a few non-numeric characters scattered through them.

Multi-Field Keys

Up to this point only item keys with a single field have been considered, where the range of key numbers is broadly sequential no matter whether continuous or not. It is, however, fairly common for control keys to be divided into definite fields where each field has a range which is quite independent of the other fields. To treat such keys as a single field may be wasteful unless each field has a maximum value such that the entire key forms a continuous series.

e.g.

00	0	000	00
to	to	to	to
77	9	999	99

Apart from cases like the above example, it is generally desirable to manipulate each field independently. Otherwise, an unduly large number of synonyms would be generated. Unless a weighting factor is applied to the most significant keys, most of the methods previously discussed would generate too many synonymous storage addresses. One such technique has been developed by Honeywell for a customer. It has the advantage of being readily adaptable to other multi-field key applications, and it generates no synonyms.

Suppose the file contains 30,000 items, each of 100 characters, which are to be blocked 6 items to a block, 1 block per bucket on a Model 259.

Each item has a 6-digit item key comprising 3 fields:

Division No.	Page No.	Line No.
1 char.	3 chars.	2 chars.
Range 1 to 5	1 to 120	1 to 50

The calculation stages are as follows:

1. Subtract 1 from Division number.
2. Multiply the resulting Division number by the sum of the maximum number of Pages multiplied by the maximum number of lines, i.e., 120 x 50 = 6,000, and place the result in Final Result X.
3. Subtract 1 from Page number.
4. Multiply the resulting Page number by the maximum number of Lines, i.e., 50, and add the result into Final Result X.
5. Subtract 1 from Line number.
6. Add the resulting Line number (1) into Final Result X.
7. Divide Final Result X by the number of items per bucket. The quotient will be the relative bucket number.

This method will convert each 6-digit Key field into an unique number in the range 0 to 29,999. If the field numbers ranged from zero instead of one, the subtractions in stages 1, 3 and 5 would be omitted since their only function is to convert each field to a range commencing with zero.

EXAMPLE:

Division 5, Page 120, Line 50

1. 5-1 = 4
2. 4x120x50 = 24,000
3. 120-1 = 119
4. 119x50 = 5,950+24,000 = 29,950
5. 50-1 = 49
6. 49+29,950 = 29,999
7. $\frac{29,999}{6}$ = 4999 remainder 5.

The remainder is discarded, giving a relative bucket address of 4999.

Frequency Analysis

This method consists of analyzing the keys of all the items in the file to determine the frequency that any digit appears in any one position of the item key. For each digit position of the item key, go through all the items to determine the number of times any one digit (0 through 9) appears. (For instance, if there were 16,045 items in the file, a 0 might occur in the fifth key position for 5168 different items, a 1 might occur in the fifth key position for 5138 different items, a 2 might occur in that position for 4958 items, a 3 might occur for 281 items, and the numbers 4 through 9 might not occur in this position for any item.) This count gives the actual distribution of digits occuring in each key position. If the distribution were perfectly even, each of the ten digits would occur the same number of times as any other digit--so each digit would occur 1/10th of the time. With 16,045 items, each digit should occur approximately 1605 times in any one key position.

To determine the devience from this ideal distribution, you take the difference between the actual number of times a digit occurs in the key position and the ideal number of times it should occur--in this case 1605. (Thus, if 0 actually occurs in the fifth key position of 5168 different items, the devience would be 5168 minus 1605 = 3563.) You do this for each digit that appears in that key position and then sum all the results to find the total devience for that key position. This then could be expressed as a percentage of the total number of items. The lower the sum, the more even is the distribution. The pattern of distribution indicates which positions are best to use when truncating or extracting storage addresses from the item keys.

EXAMPLE 1:

16,045 Items

Variance factor 1605, i.e. 10% of number of items.

Digit	KEY POSITION NUMBER						
	1	2	3	4	5	6	7
0	16045	0	0	1852	5168	1807	1738
1	0	0	4408	3147	5638	2120	1748
2	0	2198	3792	1174	4958	1745	1743
3	0	576	2231	2724	281	1684	1610
4	0	1195	2459	1194	0	1378	1617
5	0	12076	3155	1267	0	1647	1688
6	0	0	0	1243	0	1560	1606
7	0	0	0	1228	0	1329	1450
8	0	0	0	1227	0	1415	1411
9	0	0	0	989	0	1360	1434
Total Variance	28885	22133	16045	5821	21903	1961	1035
% file	180	138	100	36	137	12	6

One method of utilizing these results to convert a mass-storage address is to express each digit count in an item key field position as a percentage of the number of file items. A cumulative total is formed for each digit to which is added half of the actual percentage for that digit to give an adjusted constant for each digit in every item key position. The constants for every digit in an item key are accumulated and the total (excluding the whole number carry) multiplied by the number of storage locations allocated. The whole number product is then converted to a cylinder and Track address in the normal manner.

EXAMPLE 2:

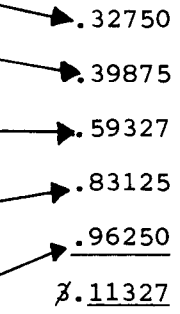
File of 20,000 items. Storage allocated - 25,000 locations.

Key position 1 illustrated.

Digit	Count	Percentage	Cumulative Total	Adjusted Constant
0	6,400	.32000	.00000	.16000
1	300	.01500	.32000	.32750
2	1,300	.06500	.33500	.36750
3	800	.04000	.40000	.42000
4	1,200	.06000	.44000	.47000
5	0	.00000	-	-
6	0	.00000	-	-
7	4,800	.24000	.50000	.62000
8	1,200	.06000	.74000	.77000
9	4,000	.20000	.80000	.90000

The above process is repeated for every key position and a table of adjusted constants is built up as follows, illustrating just the constants required for Item 13689.

Digit	ITEM KEY POSITION				
	1	2	3	4	5
0					
1	.32750				
2					
3		.39875			
4					
5					
6			.59327		
7					
8				.83125	
9					.96250



25,000 x .11327 = 2831.75000

02831 = Relative bucket address

The table of adjusted constants has to be set up initially, but the actual key transformation is fairly quick. Such a table would have to be recalculated when sufficient changes had occurred to affect materially the frequency distribution. The table itself will require 50 locations for every item key field position, i.e., 250 locations for a 5-digit control key.

HONEYWELL EDP TECHNICAL PUBLICATIONS

USERS' REMARKS FORM[1]

TITLE: SERIES 200/OPERATING SYSTEM - MOD 1 (MASS STORAGE RESIDENT)

DATED: DECEMBER, 1966

FILE NO: 123.0005.131C.0-427

ERRORS NOTED:

Fold

Cut Along Line

SUGGESTIONS FOR IMPROVEMENT:

Fold

FROM: NAME _____ DATE _____

COMPANY _____

TITLE _____

ADDRESS _____

_____

[1] Please restrict remarks to the publication itself. Comments concerning hardware/software difficulties and improvement requests should be submitted through the channels established for that purpose.

FIRST CLASS
PERMIT NO. 39531
WELLESLEY HILLS
MASS.

BUSINESS REPLY MAIL

No postage stamp necessary if mailed in the United States

POSTAGE WILL BE PAID BY

HONEYWELL
ELECTRONIC DATA PROCESSING DIVISION
60 WALNUT STREET
WELLESLEY HILLS, MASS. 02181

ATT'N: TECHNICAL COMMUNICATIONS DEPARTMENT

Cut Along Line

Honeywell
ELECTRONIC DATA PROCESSING