File Number S360-50
Order No. GC24-5073-2

DOS

IBM

Systems Reference Library

DOS System Programmer's Guide

This reference publication is intended primarily for the system programmer who is involved in making decisions relating to the components of the installation's supervisor, file organization and program design. To form a single publication, this manual brings together and expands upon information from many sources. Major topics discussed are:

1. Supervisor Planning Concepts
2. Data Management
3. Program Design
4. Debugging Aids

For each major section, the Preface lists the most closely related publications. For a complete list of available publications, see the <u>IBM System/360 and System/370 Bibliography</u>, GA22-6822.

Third Edition (September 1971)

This publication was formerly titled IBM System/360 Disk Operating System: System Programmer's Guide. Although titles of some DOS publications (including this one) have been simplified, the change does not affect the contents of the publications.

This edition replaces and obsoletes GC24-5073-1. Changes are continually made to the specifications herein; before using this publication in connection with the operation of IBM systems, consult the latest System/360 and System/370 SRL Newsletter, GN20-0360, for the editions that are applicable and current.

Summary of Amendments

For a list of changes made in this edition, see page 3.

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

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

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Laboratory, Publications Dept., P.O. Box 24, Uithoorn, Netherlands.

Summary of Amendments
for GC24-5073-2

NEW FUNCTIONS

This edition documents support for these system control and service functions:

- Independent Directory Read-In Area (IDRA)
- On-Line Test Executive Programs (OLTEP)
- Job Accounting Interface
- Data Set Security
- ISAM Track Hold
- Private Core Image Library (PCIL)
- Label Cylinder Display (LSERV)
- Recovery Management Support (RMS) for System/370, which consists of Machine Check Analysis and Recording (MCAR) and Channel Check Handler (CCH)
- Reliability Data Extractor (RDE) function for System/370
- RETAIN/370 for System/370

PROGRAM ENHANCEMENTS

This edition also documents the enhancements to these programs:

- Error Statistics by Tape Volume (ESTV)
- Environmental Recording, Editing, and Printing (EREP)
- DOS Stand-Alone Dump Generator (DUMPGEN)
- Problem Determination Serviceability Aids (PDAID)
- Forced End-of-Volume for Disk macro (FEOVD)
- Directory Service Display (DSERV)
- Linkage Editor (LNKEDT)

NEW DEVICES

This edition also documents support for these new devices:

- IBM 1255 Magnetic Character Reader
- IBM 2319 Disk Storage
- IBM 3210 Console Printer-Keyboard
- IBM 3211 Printer
- IBM 3215 Console Printer-Keyboard
- IBM 3420 Magnetic Tape Unit

MISCELLANEOUS CHANGES

Rewritten sections: The section on hard waits is expanded and rewritten. The section on COBOL ANS replaces the COBOL D section. A glossary is included.

ORGANIZATION OF PUBLICATION

The manual has been reorganized and reformatted. Information in Appendixes A - L in the previous edition has been incorporated into the appropriate sections in this edition. The total publication has a General Contents, a Figure list, and an Index. Each section has a Section Outline and a separate figure list. Figure numbers throughout the publication are in the form: Figure 1.3, where 1 is the section number and 3 is the figure number within that section. A tab in the upper right-hand corner of the beginning of each section identifies the section by number.

Preface

This publication is divided into the following four major sections:

1. Supervisor Planning Concepts
2. Data Management
3. Program Design
4. Debugging Aids

The Supervisor Planning Concepts section describes system residence organization, some supervisor tables, optional supervisor facilities at a conceptual level, and presents guidelines for their implementation. The Data Management section explains the concepts of file organization and data manipulation at the physical and logical IOCS levels. The Program Design section contains suggestions for efficient programming. The topics discussed are link editing functions, overlay structure, self-relocating programs, checkpoint/restart facility, the 3211 Printer support, and macro writing. The Debugging Aids section contains aids for debugging problem programs written in assembler language, PL/I, FORTRAN, COBOL, and RPG.

Multitasking and link editing examples, and physical IOCS, STXIT support, self-relocating, and language translator sample programs are included.

Closely related publications by section follow.

Note: Although titles of some DOS publications have been simplified, the change does not affect the contents of the publications.

Supervisor Planning Concepts

DOS System Generation, GC24-5033.

DOS System Control and Service, GC24-5036.

DOS Operating Guide, GC24-5022.

DOS Messages, GC24-5074.

Data Management

DOS Data Management Concepts, GC24-3427.

DOS Supervisor and I/O Macros, GC24-5037.

DOS DASD Labels, GC24-5072.

Tape Labels for BPS, BOS, TOS and DOS, GC24-5070.

IBM System/360 Principles of Operation, GA22-6821.

IBM System/370 Principles of Operation, GA22-7000.

Program Design

DOS System Control and Service, GC24-5036.

DOS Supervisor and I/O Macros, GC24-5037.

IBM System/360 Disk and Tape Operating Systems, Assembler Language, GC24-3414.

Debugging Aids

DOS OLTEP, GC24-5086.

IBM System/360 Disk and Tape Operating Systems, COBOL Programmer's Guide, GC24-5025.

IBM System/360 Disk and Tape Operating Systems, PL/I Programmer's Guide, GC24-9005.

IBM System/360 Disk Operating System, FORTRAN IV Programmer's Guide, GC28-6397.

IBM System/360 Disk and Tape Operating Systems, Report Program Generator Specifications, GC26-3570.

IBM System/360 Disk and Tape Operating Systems, Assembler Language, GC24-3414.

DOS Messages, GC24-5074.

For further information concerning terms referenced in this publication, see the IBM Data Processing Glossary, GC20-1699.

Contents

Figures

Introduction

As a system programmer, you make decisions involving the components of your installation's supervisor, file organization, program design, and so forth. From time to time, you may be called upon to advise other programmers concerning DOS. In order to assist you in this task, this publication brings together and expands upon information from many sources. It is divided into four major sections: Supervisor Planning Concepts, Data Management, Program Design, and Debugging Aids.

The Supervisor Planning Concepts section is of interest to the person(s) responsible for tailoring the IBM-supplied Disk Operating System to meet the needs of the installation. This section describes in detail optional supervisor facilities available under DOS. Using the information presented in this section together with the information on main storage requirements and implementation procedures found in the DOS System Generation listed in the Preface, the system programmer can decide whether or not to include a particular facility within the installation's supervisor. Guidelines for implementing these facilities at system generation time are also presented.

The Data Management section is of interest to the person(s) responsible for choosing the type of file organization best suited for an application. This section discusses data management concepts, the advantages and disadvantages of each type of file organization (sequential, direct access and indexed sequential) and criteria for choosing the best file organization and retrieval method.

In addition, data manipulation is described at both the physical and logical IOCS levels. Detailed information for coding at the physical IOCS level is included. This section also defines the macros for implementing logical IOCS and describes the interrelationships of the DTF and logic module generation macros.

The Program Design section is of interest to the person(s) responsible for program design and implementation. The Disk Operating System offers the programmer a great deal of flexibility in the generation of his system and in its operation. This section discusses effective use of the linkage editor and the checkpoint/restart facility. In addition, system programming techniques such as macro coding, overlay structure and self-relocating programs are discussed. The IBM 3211 Printer support is also discussed.

The Debugging Aids section is of interest to both the application and system programmer. This section describes system action on a cancel condition, gives register conventions for following program flow, describes the types of documentation to be gathered for debugging purposes and the action to be taken when a hard wait or program loop is encountered. Problem determination aids are briefly described. Aids for debugging problem programs written in assembler language, COBOL, FORTRAN, PL/I and RPG are given.

Note: In case of difference between the conventions given in this manual for control program functions and those appearing in IBM-supplied DOS component publications (such as guides for language translators, sorts, utilities, specifications manuals, etc), observe the specific restrictions of the component.

Section 1: Supervisor Planning Concepts

Section Outline

Section Figures

To assist you in planning a supervisor tailored to meet the needs of your installation, this section presents the organization of a disk resident system as received from Program Information Department (PID), describes optional supervisor facilities at a conceptual level, and presents guidelines for their implementation. Those optional facilities requiring special consideration are described in the order in which they appear as parameters within the supervisor generation macros. For information on the other optional supervisor facilities, see the DOS System Generation listed in the Preface.

System Residence Organization

Figure 1.1 describes the organization of the DOS disk resident system. The disk resident system may be on an IBM 2311 Disk Storage Drive, an IBM 2314 Direct Access Storage Facility, or an IBM 2319 Disk Storage Facility. The organization of the disk pack is as follows:

IPL (Cylinder 0, Track 0, Records 1 and 2)

This area contains the IPL bootstrap program.

Volume Label (Cylinder 0, Track 0, Record 3)

The volume label contains the address of the Volume Table of Contents (VTOC) established when the pack was initialized.

System Directory (Cylinder 0, Track 1)

This directory consists of five records that make up the system master directory, Records 1 through 4 are 80 bytes in length.

Record 1 contains information describing the core image library and directory. Records 2 and 3 contain the starting address of the relocatable library directory and the source statement library directory, respectively. Record 4 is not used. Record 5 is the IPL loader program ($$A$IPL2).

System Work Area (Cylinder 0, Tracks 2, 3 and 4)

This 3-track area is reserved as a work area for the librarian programs and linkage editor. The format of the records in the librarian area depends on the program using the area at a specific time.

Transient Directory (Cylinder 0, Track 5)

This single track directory contains entries for the A- and B-transient routines that are located in the core image library. The entries in this directory are taken from the core image library directory.

The core image library phases referenced in this directory have phase names prefixed by $$A (A-transients) or $$B (B-transients). This directory has a maximum capacity of 144 entries for the 2311, or 270 entries for the 2314/2319. Track format is identical to the core image library directory.

Open Directory (Cylinder 0, Track 6)

This single track directory contains entries for the LIOCS open phases located in the core image library. The entries in this directory are taken from the core image library directory. The core image library phases referenced in this directory have phase names prefixed by the characters $$BO. This directory has a maximum capacity of 144 entries for the 2311, or 270 entries for the 2314/2319.

Library Routine Directory (Cylinder 0, Track 7)

This single track directory contains entries for frequently used core image library phases, such as job control, linkage editor, and so forth. The entries in this directory are taken from the core image library directory. The core image library phases that are placed in this directory have phase names prefixed by a $ (for example, $LNKEDT). This entry has a maximum capacity of 144 entries for the 2311, or 270 entries for the 2314/2319.

Foreground Program Directory (Cylinder 0, Track 8)

This single track directory contains entries for the foreground program phases located in the core image library. The entries in this directory are taken from the core image library directory. The core image library phases referenced in this directory have phase names prefixed by the characters FGP. This directory has a maximum capacity of 144 entries for the 2311, or 270 entries for the 2314/2319.

Phase Directory (Cylinder 0, Track 9)

This single track directory contains entries for the phases of the current BG problem program. The entries in this directory are constructed by job control before each job step is executed in the background partition only. They are taken from the core image library directory.

The phase naming conventions that permit the use of the phase directory are:

1. All program names must be unique in the first four characters.
2. The first four characters of the name of each phase of a program must be identical to the first four characters of the program name. All eight characters of the first phase name must be identical to the program name.

   Example: WXVZPROG
   WXVZPROG - phase 1
   WXVZPH1 - phase 2
   WXVZPH2 - phase 3

The maximum capacity of this directory is 144 entries for the 2311 or 270 entries for the 2314/2319.

Core Image Library Directory

This directory consists of one or more tracks, depending on the allocation specified by the user. It contains one entry for each of the phases in the core image library.

Note: A phase is an overlay of a multiphase program or a complete program if not multiphase.

Each directory entry describes one phase in the core image library and contains:

- Phase name
- Loading address
- Number of blocks
- Entry point
- Starting disk address in the core image library
- Length of last block

Core Image Library

The core image library consists of five or more tracks, depending on the allocation specified by the user. For the 2311, each track contains two blocks with a maximum capacity of 1728 bytes. For the 2314/2319, each track contains four blocks with a maximum capacity of 1688 bytes. The number of programs (phases) and the size of each program to be contained in the core image library dictates the number of cylinders that must be allocated. Each program starts with a new block and only the last block of a program can contain less than 1728 bytes of data for the 2311 or 1688 bytes of data for the 2314/2319.

Note: A phase is an overlay of a multiphase program or a complete program if not multiphase.

Relocatable Library Directory

This directory consists of one or more tracks, depending on the allocation specified by the user. It contains two types of information:

1. System directory information for the relocatable directory and library. This information occupies the first five entries of the first record in the relocatable directory.
2. An entry that describes each module in the relocatable library and contains:
   a. Module name
   b. Total number of text-record blocks required to contain this module
   c. Starting disk address of the first text-record of this module
   d. Change level identification.

NO.	COMPONENT		STARTING DISK ADDRESS				NUMBER OF TRACKS (Allocation)	R=REQUIRED O=OPTIONAL
			BB	CC	HH	R		
1	IPL Bootstrap Record 1 (AIPL1)		00	00	00	1	1	R
	IPL Bootstrap Record 2 (AIPLA)		00	00	00	2		R
	Volume Label		00	00	00	3		R
	User Volume Label		00	00	00	4		O
2	System Directory	Record 1	00	00	01	1	1	R
		Record 2	00	00	01	2		R
		Record 3	00	00	01	3		R
		Record 4	00	00	01	4		R
	IPL Retrieval Program ($$A$IPL2)		00	00	01	5		R
3	System Work Area (Librarian Area)		00	00	02	1	3	R
4	Transient Directory ($$A and $$B Transients)		00	00	05	1	1	R
5	Open Directory ($$B0)		00	00	06	1	1	R
6	Library Routine Directory ($ Phasenames)		00	00	07	1	1	R
7	Foreground Program Directory (FGP)		00	00	08	1	1	R
8	Phase Directory (For Problem Program Phases)		00	00	09	1	1	R
9	Core Image Library Directory		00	01 for 2311 00 for 2314/2319	00 for 2311 10 for 2314/2319	1	*	R
10	Core Image Library		00	End of CI Directory X	Y +1	1	*	R
11	Relocatable Library Directory		00	End of CI Library Z+1	00	1	*	O
12	Relocatable Library		00	End of RL Directory X	Y+1	1	*	O
13	Source Statement Library Directory		00	End of RL Library Z+1	00	1	*	O
14	Source Statement Library		00	End of SS Directory X	Y+1	1	*	O
15	Volume Area File Definition Storage Area			End of SS Library Z+1	00	1	2311:10 2314/2319:20	R
16	User Area			End of Volume Area Z+2	00	1	*	O

*Allocation Dependent On User Requirements
X=Ending CC of the Preceding Directory
Y=Ending HH of the Preceding Directory
Z=Ending CC of the Preceding Library

Figure 1.1. System Residence Organization

Relocatable Library

The relocatable library consists of five or more tracks, depending on the allocation specified by the user. The number of modules and the size of each module to be contained in this library dictates the number of tracks that must be allocated. Each allocated track contains 9 blocks (2311) or 16 blocks (2314/2319), and each block has a fixed length of 322 bytes. Each module starts with a new block but not necessarily a new track.

Source Statement Library Directory

This directory consists of one or more tracks, depending on the allocation specified by the user. It contains two types of information:

1. System directory information for the source statement directory and library. This information occupies the first five entries of the first record in the source statement directory.
2. An entry that describes each book (see Note 1) in the source statement library and contains:
   a. A sublibrary prefix: any alphameric character, $, #, or @, except A and C (see Note 2).
   b. Book name.
   c. Starting disk address of the first block of this book.
   d. Total number of blocks required to contain this book in the source statement library.
   e. Change level identification.

Note 1: A book is a sequence of source language statements, in compressed card image format, accessed by a single name.

Note 2: A and C are reserved for assembler and COBOL, respectively.

Source Statement Library

The source statement library consists of five or more tracks, depending on the allocation specified by the user. The number of books and the size of each book to be contained in this library dictates the number of tracks that must be allocated. Each track contains 16 blocks (2311) or 27 blocks (2314/2319), and each block has a fixed length of 160 bytes. Each book starts with a new block but not necessarily a new track. Each book in the source statement library contains compressed card images of the source language input to the assembler or language translators. A compressed card image can overflow from one block to another.

Label Information Cylinder

The label information cylinder (10 tracks for 2311 or 20 tracks for 2314/2319) contains background and foreground user and standard label information.

SUPVR Macro

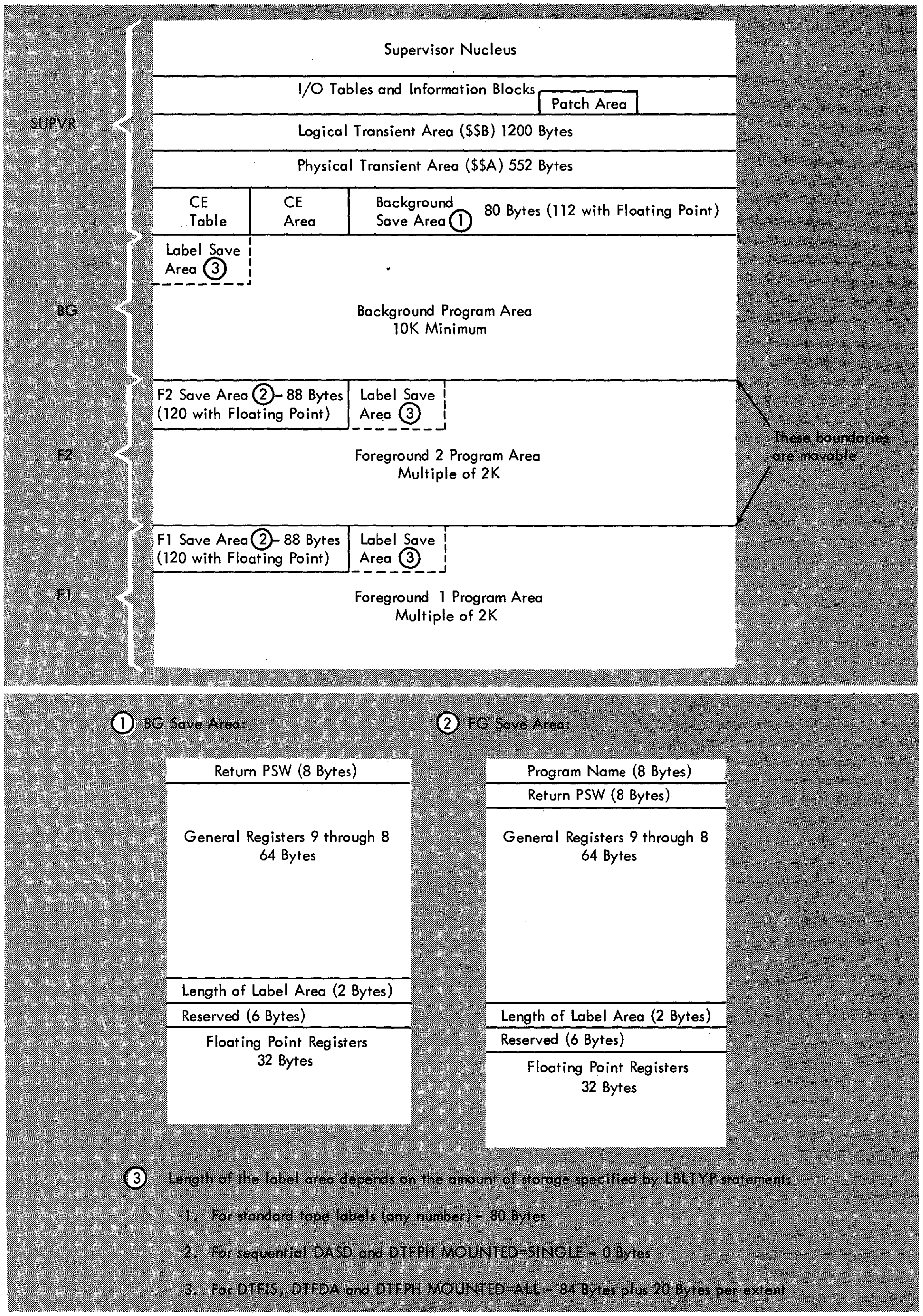
MULTIPROGRAMMING SUPPORT (MPS)

Multiprogramming is the ability to run multiple programs concurrently, provision for which must be included in the DOS supervisor at system generation time. Each program resides in a different area of main storage called a partition. The three problem program partitions are designated background (BG), foreground 1 (F1), and foreground 2 (F2).

The background partition must be at least 10K because job control runs in the background partition and requires 10K bytes of main storage. However, 14K allows faster assemblies and linkage editing.

The remaining main storage is divided between the two foreground partitions. To satisfy the requirements for the storage protect special feature, these partitions must begin and end on 2K boundaries. Because the MPS supervisor requires a minimum of 8K and the background partition requires a minimum of 10K, MPS will not function on systems with less than 24K of main storage (see Figure 1.2).

MPS operates under the principle that in most commercial installations, the CPU is heavily I/O bound. Much of the CPU running time is spent waiting for a printer, a reader or a punch to complete a previous operation before the subsequent one can be started. With MPS, when a partition becomes I/O bound (that is, it cannot continue until the completion of some I/O operation), a task selection routine in the MPS supervisor attempts to give CPU control to the next partition that is ready to run.



③ Length of the label area depends on the amount of storage specified by LBLTYP statement:

1. For standard tape labels (any number) – 80 Bytes
2. For sequential DASD and DTFPH MOUNTED=SINGLE – 0 Bytes
3. For DTFIS, DTFDA and DTFPH MOUNTED=ALL – 84 Bytes plus 20 Bytes per extent

Figure 1.2. MPS Storage Map

The partition to which control is given is determined by a priority system. F1 has highest priority followed by F2 and BG. A partition cannot be interrupted by one of lower priority. When an F1 program is ready to run, the task selection routine seizes control from a program of lower priority. One of the disadvantages in such an arrangement is the possibility of a high priority program never relinquishing control to other partitions. This can happen if the high priority program has few I/O requests. In general, the program with the largest number of I/O operations or wait time should reside in the highest priority partition. A compute (CPU) bound program should reside in a low priority partition (for example, BG).

Because the channel scheduler attempts to keep devices and channels busy asynchronously, it is an advantage for all programs to use sufficient I/O areas to keep the channel queue stacked with requests. A lower priority partition is more likely to require multiple I/O areas to achieve this than is a higher priority partition.

Partitioning

Multiprogramming has been defined as a technique whereby two or more programs may operate concurrently, sharing system resources between them. The DOS multiprogramming support requires that such programs be co-resident in main storage. To achieve this, storage is divided into 3 sections termed partitions (BG, F1, and F2). Each partition is capable of holding a distinct problem program. When a partition contains a program that is in course of execution, the partition is said to be active. When a partition does not contain such a program, or is not physically present in the system, then it is said to be inactive.

Each partition is of fixed physical size and is defined by fixed boundaries. For this reason, DOS multiprogramming is termed fixed partition multiprogramming. Partition boundaries may be altered in any manner when all partitions are inactive (minimum background size is 10K). When any partitions are active, boundaries can be altered only if the lower limit of each active partition is unchanged and the size of each active partition is not reduced.

Each partition is allotted its own unique storage protection key; for this reason the size of any partition must be an integral multiple of 2K. Thus, a program contained in any one partition can read but not alter the contents of core locations contained in any other partition.

Control Method

Multiprogramming increases system efficiency by making better use of the available system resources than is possible in a single partition batched job environment.

Multiprogramming support is written in such a way that the central processing unit does not enter the wait state if useful processing can be performed in any partition. Multiprogramming allows the input and output functions of one program to overlap with the processing functions of other programs.

Task Selection

A program yields control by issuing a supervisor call instruction to pass control to the supervisor routines. The SVC instruction contains a code that indicates its purpose. The most numerically significant of these codes in a multiprogramming context are those associated with input and output operations; therefore, only these codes are described. A complete list of DOS supervisor calls is given in Figure 1.3.

Macro Supported	SVC		Function
	Dec.	Hex.	
EXCP	0	0	Execute channel programs.
FETCH	1 2 3	1 2 3	Fetch any phase. Fetch a logical transient (B-transient). Fetch or return from a physical transient (A-transient).
LOAD	4	4	Load any phase.
MVCOM	5	5	Modify supervisor communications region.
CANCEL	6	6	Cancel a problem program or task.
WAIT	7	7	Wait for a CCB or TECB.
	8	8	Transfer control to the problem program from a logical transient (B-transient.)
LBRET	9	9	Return to a logical transient (B-transient) from the problem program after an SVC 8.
SETIME	10*	A	Set timer interval.
	11 12 13	B C D	Return from a logical transient (B-transient). Logical AND (Reset) to second job control byte (displacement 57 in communications region). Logical OR (Set) to second job control byte (displacement 57 in communications region).
EOJ	14	E	Cancel job and go to job control for end of job step.
	15	F	Same as SVC 0 except ignored if CHANQ table is full. (Primarily used by ERP.)
STXIT (PC)	16*	10	Provide supervisor with linkage to user's PC routine for program check interrupts.
EXIT (PC)	17*	11	Return from user's PC routine.
STXIT (IT)	18*	12	Provide supervisor with linkage to user's IT routine for interval timer interrupts.
EXIT (IT)	19*	13	Return from user's IT routine.
STXIT (OC)	20*	14	Provide supervisor with linkage to user's OC routine for external or attention interrupts (operator communications).
EXIT (OC)	21*	15	Return from user's OC routine.
	22* 23*	16 17	The first SVC 22 seizes the system for the issuing program by disabling multiprogram operation. The second SVC 22 releases the system (enables multiprogram operation). Load phase header. Phase load address is stored at user's address.
SETIME	24*	18	Provide supervisor with linkage to user's TECB and set timer interval.
	25* 26* 27*	19 1A 1B	Issue HALT I/O on a teleprocessing device, or HALT I/O on any device if issued by OLTEP. Validate address limits. Special HIO on teleprocessing devices.

* = optional

Figure 1.3. Supervisor Calls (Part 1 of 2)

Macro Supported	SVC		Function
	Dec.	Hex.	
EXIT (MR)	28*	1C	Return from user's stacker select routine (MICR type devices only).
	29*	1D	Provide return from multiple wait macros WAITF and WAITM (except MICR type devices).
QWAIT	30*	1E	Wait for a QTAM element.
QPOST	31*	1F	Post a QTAM element.
	32 33 34	20 21 22	(Reserved) Reserved for internal macro COMRG. Reserved for internal macro GETIME.
HOLD	35*	23	Hold a track for use by the requesting task only.
FREE	36*	24	Free a track held by the task issuing the FREE.
STXIT (AB)	37*	25	Provide supervisor with linkage to user's AB routine for abnormal termination of a task.
ATTACH	38*	26	Initialize a subtask and establish its priority.
DETACH	39*	27	Perform normal termination of a subtask. It includes calling the FREE routine to free any tracks held by the subtask.
POST	40*	28	Inform the system of the termination of an event and ready any waiting tasks.
DEQ	41*	29	Inform the system that a previously enqueued resource is now available.
ENQ	42*	2A	Prevent tasks from simultaneous manipulation of a shared data area (resource).
	43* 44* 45* 46* 47* 48 49 50 51*	2B 2C 2D 2E 2F 30 31 32 33	Provide supervisor support for external creation and updating of SDR records. Provide supervisor support for external creation of OBR records. Provide emulator interface. Provide OLTEP with the facility to operate in supervisory state. Provide return from wait multiple WAITF for MICR type device. (Reserved) (Reserved) Reserved for LIOCS error recovery. Return phase length at OLTEP request.

* = optional

Figure 1.3. Supervisor Calls (Part 2 of 2)

SVC Code 0 (EXCP)

This code requests the supervisor to initiate an input or output operation. The address of a Command Control Block (CCB) located in the requesting program is also passed to the supervisor via register 1. This block contains information that describes the precise nature of the operation to be performed.

When the supervisor receives the EXCP request but is unable to initiate the required operation, it places the request in a queue for later action. The traffic bit in the relevant CCB is then set 0 to whether or not an operation can be started when requested. The supervisor then returns control to the program that requested the operation.

SVC Code 7 (WAIT)

This code informs the supervisor that the program is unable to proceed further until a previously requested operation has been completed, and that the operation is still in progress because the relevant traffic bit is still set to 0. The program is placed in the wait state. Note that the instructions immediately preceding the supervisor call instruction form a test of the traffic bit so that, if the traffic bit is set to 1, the supervisor call is bypassed. When the SVC 7 is recognized, the program return address is reduced so that the instructions generated by the WAIT macro will be issued.

When the supervisor recognizes an SVC 7 interrupt, it records that the program is in the wait state. The supervisor then gives control to the next partition of the highest priority that is ready to run. If such a partition does not exist, the supervisor causes the central processing unit to enter the wait state until an interrupt occurs.

A program loses control to the supervisor whenever an interrupt occurs. Only the input/output interrupts are described in the following text because they are the most significant interrupts in a multiprogramming environment.

When an input/output interrupt occurs, the supervisor identifies the operation in question and checks whether it has been satisfactorily performed. If it has, the traffic bit in the related CCB is set to 1; the owning partition is made ready to run, that is, if it was in a wait state it is removed from that state; and any further input/output operation pending for this channel or device is initiated. The task selection routine in the supervisor then gives control to the next partition of the highest system priority that is ready to run.

The following three figures show the relationship between six different operations while operating under MPS. All three partitions are active.

Figure 1.4 assumes that programs were initiated at some prior point in time in all three partitions. The first part of Figure 1.4 shows that foreground 1 has control of the CPU and is processing. At some point in time, the program in F1 returns control of the CPU to the supervisor by issuing an I/O wait (SVC 7).

The supervisor goes through a task selection process, determines that foreground 2 is ready to run and gives CPU control to it. F2 processes its program until it requires I/O, and then returns control of the CPU to the supervisor by issuing an I/O wait (SVC 7).

The supervisor goes through a task selection process, determines that F1 is not ready to run but the background is ready to run, and gives CPU control to the background. BG processes its program until it requires I/O and then returns control of the CPU to the supervisor by issuing an I/O wait (SVC 7).

The supervisor goes through a task selection process, determines that no partitions are ready to run, and gives control of the CPU to a task called All Bound. This task loads a PSW that puts the system in the wait state with all interrupts enabled. Note that no I/O interrupts have occurred.

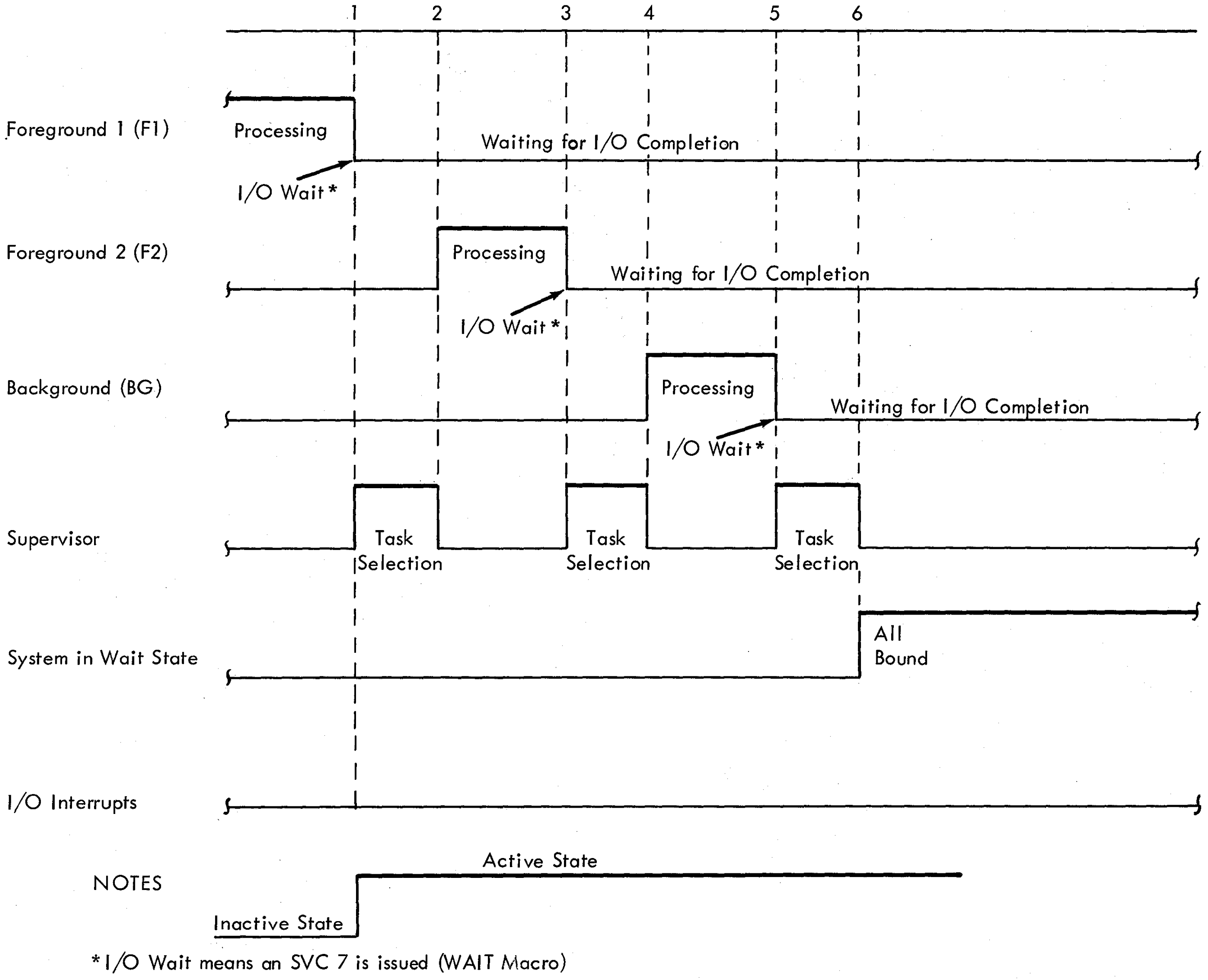


Figure 1.4. Processing and I/O Requests Relationship

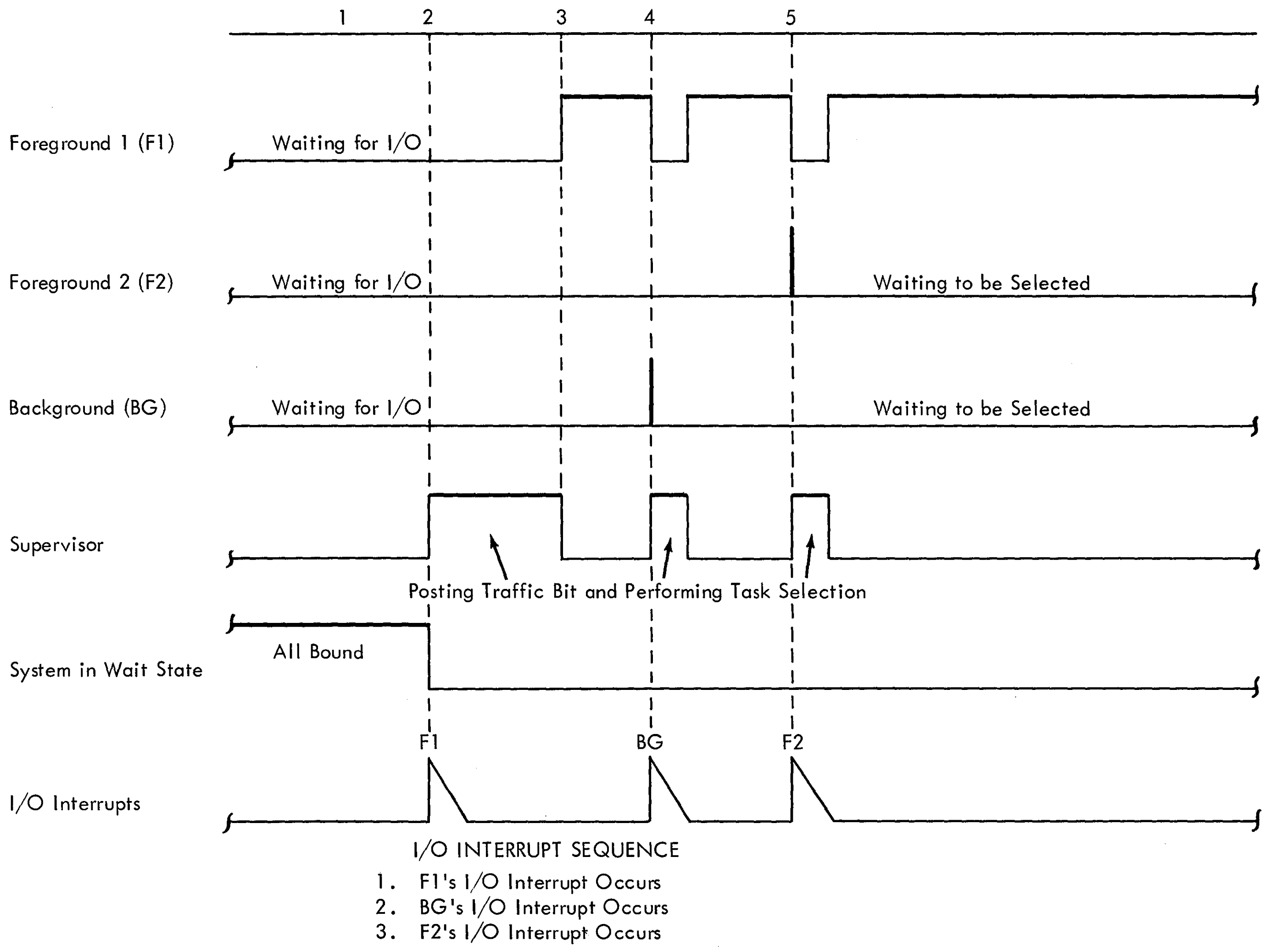


Figure 1.5. First Example of Processing and I/O Requests with I/O Interrupts

Figure 1.5 starts with all partitions waiting for I/O, and the system in the wait state with interrupts enabled.

When the first I/O interrupt occurs, the supervisor I/O interrupt routine gets control of the CPU, and turns on the traffic bit in the CCB associated with the device causing the I/O interrupt (F1 partition). The supervisor task selection routine then gets control, determines that F1 is through waiting, and gives CPU control to it. F1 starts processing its program.

An I/O interrupt occurs and control passes to the supervisor I/O interrupt routine. It turns on the traffic bit in the CCB associated with the device causing the I/O interrupt (BG partition). The supervisor task selection routine gets control, determines that F1 is still the highest priority program ready to run, and gives CPU control to it. Note that BG is still waiting, but now is ready to run because its I/O wait is complete.

While F1 is processing, another I/O interrupt occurs and control passes to the supervisor I/O interrupt routine. It turns on the traffic bit in the CCB associated with the device causing the I/O interrupt (F2 partition). The supervisor task selection routine then gets control of the CPU, determines that F1 is still the highest priority program that is ready to run, and gives CPU control to it. Note that both BG and F2 are now ready to run, but control of the CPU is retained by the highest priority partition (F1) until it issues an SVC or until one of the four other system interrupts occurs.

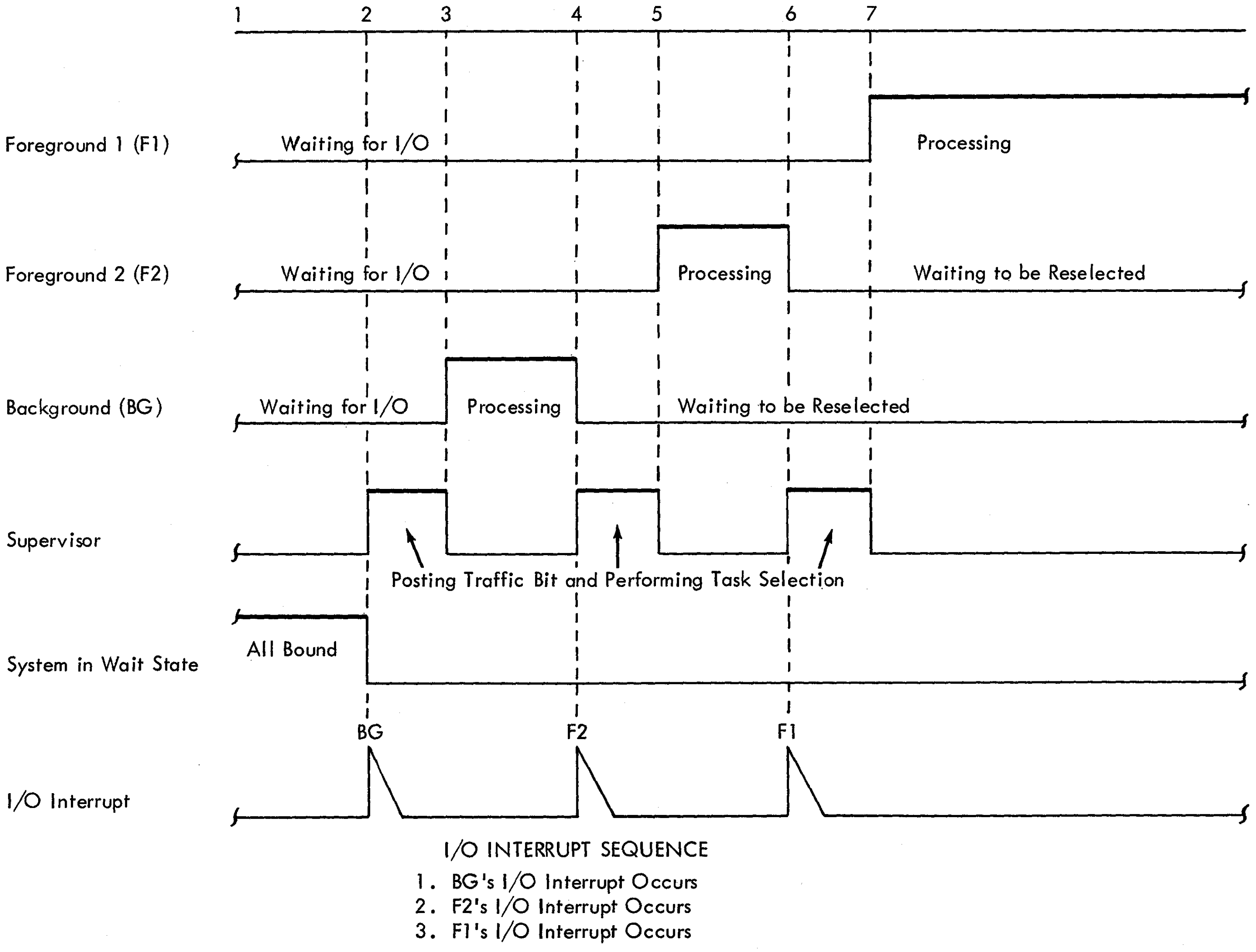


Figure 1.6. Second Example of Processing and I/O Requests with I/O Interrupts

Figures 1.4 and 1.5 show a simplified version of what actually occurs. In actual practice, the process time for F1 would be longer than F2 or BG, and the sequence of I/O interrupts would probably be staggered.

Figure 1.6 starts with all partitions waiting for I/O, and the system in the wait state with interrupts enabled.

When the first I/O interrupt occurs, the supervisor I/O interrupt routine gets control of the CPU, and turns on the traffic bit in the CCB associated with the device causing the I/O interrupt (BG partition). The supervisor task selection routine gets control, determines that the BG partition is the highest priority task ready to run, and gives CPU control to it. The BG partition starts processing with the instruction 6 bytes before the I/O wait (SVC 7).

When the second I/O interrupt occurs, the supervisor I/O interrupt routine gets control of the CPU, and turns on the traffic bit in the CCB associated with the device causing the I/O interrupt (F2 partition). The supervisor task selection routine gets control of the CPU, determines that the F2 partition is now the highest priority partition ready to run, and gives CPU control to it. The F2 partition starts processing with the instruction 6 bytes before the I/O wait (SVC 7).

When the third I/O interrupt occurs, the supervisor I/O interrupt routines gets control of the CPU, and turns on the traffic bit in the CCB associated with the device causing the I/O interrupt (F1 partition). The supervisor task selection routine gets control of the CPU, determines that the F1 partition is now the highest priority partition ready to run, and gives CPU control to it.

The foreground 1 partition remains in control of the CPU until it issues an SVC or until one of the four other system interrupts occurs.

System Considerations

Any program that is process-bound (compute-bound) completely suppresses any event from taking place in any partition of lower priority, regardless of the utilization of channels and devices. Thus, if the program in the foreground 1 partition is process-bound, multiprogramming cannot take place because a partition of lower priority cannot get control. In general, a foreground program should never be process-bound in a multiprogramming environment.

Because the slowest devices are those that are associated with unit record equipment (e.g., card readers, card punches, line printers and terminals), unit record and teleprocessing devices in a multiprogramming environment should be associated with the partitions of highest system priority.

In general, the slowest input/output devices should run with maximum efficiency, because loss of efficiency on such devices is more serious in terms of system throughput than on faster devices.

Note: I/O bound programs with the slowest system devices should be allocated to the highest priority partitions.

The efficiency of a low priority program depends on the quantity and frequency of processing time made available to it. The most advantageous high priority program (in terms of overall system efficiency) combines lengthy input and output operations with a minimum of processing.

Every time an input/output interrupt occurs, it must be interrogated for all partitions of higher priority than the partition to which it relates. Therefore, a reduction in the number of such interrupts (which may be achieved by increasing blocking factors of tape and disk files), promotes greater multiprogramming efficiency because it lowers system overhead. Increased blocking factors, however, result in increased core storage requirements.

Another important system consideration is the use of two input or output areas in connection with one file (double buffering). Double buffering increases computer utilization by allowing the overlap of input/output operations with processing. Multiprogramming has the same purpose. In double buffering, input/output operations are overlapped with processing relating to the same program; in multiprogramming, input/output operations are overlapped with processing relating to a different program. Therefore, the more efficiently double buffering operates in relation to a given foreground program, the less time will be freed for use by the background program.

In a multiprogramming environment, it may prove more efficient in terms of total system throughput to dispense with double buffering for programs operating in foreground partitions. In most cases, main storage can be better applied to the accommodation of greater block sizes than to the accommodation of double input/output areas.

Concurrent Peripheral Operation (CPO)

The CPO (Concurrent Peripheral Operation) concept is the application of multiprogramming techniques to the basic utility operations of card-to-tape, tape-to-punch, tape-to-printer, and their disk equivalents. Such operations are performed in a separate storage partition.

A typical CPO program is a file-to-file utility containing a minimum amount of processing. Blocking and deblocking operations and some data selection can be performed, but this is all. Thus, CPO fits the specifications for an efficient high priority program. A CPO program is normally associated with a unit record device and the greater part of its input/output time is likely to utilize subchannels of the multiplexor channel. Therefore, channel contention with other concurrent programs is greatly reduced. CPO is one of the most efficient practical applications of the multiprogramming technique.

Operational Considerations

Two methods used to initialize and operate programs in the foreground partitions are:

1. Batch Job Foreground (BJF): This method uses the job control program.

The foreground partition(s) essentially operates like the background partition (i.e., batched processing automatic job-to-job transition). The foreground partition(s) must be a minimum of 10K.

2. Single Program Initiator (SPI): This method uses the IBM 1052 Printer-Keyboard for System/360 or the IBM 3210 or 3215 Console Printer-Keyboards for System/370 to initialize single programs in the foreground partition(s). When the single program is terminated (either under its own control by issuing an EOJ, DUMP or CANCEL macro, or through operator action, a program error or certain I/O failures), a terminating message is printed on the console and the foreground partition becomes inactive. To run the next single program in the foreground partition, the operator must again initialize it from the 1052, 3210, or 3215 console. The major advantage of SPI is that the foreground partition size can be a minimum of 2K bytes.

Programming Considerations

The output of the language compilers can be link-edited to run in any partition, provided enough core storage is available.

In a system that supports both batched-job foreground and private core image libraries (see Private Core Image Libraries), the function of compiling can be run in the foreground partitions as well as the background partition, providing enough core storage is available in which to execute the compiler. The linkage editor can execute in any batched-job partition. A private core image library is required when executing the linkage editor in a foreground partition.

System Generation Guidelines

The multiprogramming facility is specified at system generation time by the MPS= keyword parameter in the SUPVR macro.

$$\text{MPS=}\left\{\begin{array}{l}\underline{\text{NO}}\\ \text{YES}\\ \text{BJF}\end{array}\right\}$$

NO	This option automatically creates a background partition consisting of all available main storage above the supervisor and negates any and all multiple partition operation. NO is the assumed value.
YES	When yes is specified, one, two, or three problem program partitions may exist. The foreground partitions may only be initialized using the single program initiator (SPI).
BJF	Under this option, one, two, or three problem program partitions may exist and the foreground partition(s) may be controlled either by job control or the single program initiator (SPI).

The parameter YES or BJF must be specified if the supervisor is to be generated to support QTAM.

AMERICAN NATIONAL STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII) SUPPORT

In addition to processing EBCDIC data files, DOS can process magnetic tape files written in ASCII (American National Standard Code for Information Interchange), a 128-character, 7-bit code. The high-order bit in the System/360 8-bit environment is zero. ASCII tape files may be either unlabeled or labeled according to the specifications of the American National Standards Institute, Inc. (ANSI).

System Considerations

ASCII tape files may be processed in either a foreground or background partition. Because internal processing of ASCII files is performed in EBCDIC, the data is translated at I/O time. Input files containing ASCII data are translated to EBCDIC as soon as the record is read into the I/O area. Output files described as ASCII are translated from EBCDIC to ASCII just prior to writing the record.

Two translate tables (providing for the conversion from ASCII to EBCDIC and from EBCDIC to ASCII) are generated at system generation time by specifying the ASCII=YES parameter in the SUPVR macro. These tables are located immediately before the Seek Address Blocks in the supervisor. The address of the ASCII-EBCDIC translate table is in locations 44-47 (decimal) of the communications region extension. Add 256 (decimal) to this table to get the address of the EBCDIC-ASCII translate table. The address of the communications region extension is in bytes 136-139 (decimal) of the communications region. Figures 1.7 and 1.8 show the system and background communications regions.

System Generation Guidelines

To generate supervisor support for handling ASCII tape files, ASCII=YES must be specified in the SUPVR macro.

$$\text{ASCII=}\left\{\begin{matrix}\underline{\text{NO}}\\ \text{YES}\end{matrix}\right\}$$

NO	When ASCII=NO is specified, or if the parameter is omitted or incorrectly specified, the translate tables are not generated. The system will then process EBCDIC files only.
YES	When YES is specified, the two translate tables are generated in the supervisor. The address of the first table (ASCII-EBCDIC) is inserted in each communication region extension to enable accessibility by any task in any partition.

COMREG*

Displacement hexadecimal	0	8	0A	0C	17	18	20	24	28	2C
Displacement decimal	0	8	10	12	23	24	32	36	40	44
	Date	Address of PPBEG	Address of EOSSP	Problem Program Use	UPSI Byte	Job Name	Highest Storage Address of the Partition	End Address of Last Phase Fetched or Loaded	Address of Uppermost Byte of Phase with Highest Ending Address	Label Area Length
	XXXXXXXX	XX	XX	XXXXXXXXXXX	X	XXXXXXXX	XXXX	XXXX	XXXX	XX

Displacement hexadecimal	2E	30	34	35	36	37	38	39	3A	3B	3C	3E
Displacement decimal	46	48	52	53	54	55	56	57	58	59	60	62
	PIK (PID)	End of Storage Address	Machine Confg. Byte	System Confg. Byte	Standard Language Translator I/O Options	Dump, Log and ASCII Options	Job Control Byte	Linkage Control Byte	Language Translator Control Byte	Job Duration Indicator Byte	Disk Address of Label Cylinder	Address of FOCL
	XX	XXXX	X	X	X	X	X	X	X	X	XX	XX

Job Control Switches

Displacement hexadecimal	40	42	44	46	48	4A	4C	4E	4F	58	5A	5C
Displacement decimal	64	66	68	70	72	74	76	78	79	88	90	92
	Address of PUB	Address of FAVP	Address of JIB	Address of TEB	Address of FICL	Address of NICL	Address of LUB	Line Count for SYSLST	System Date	LIOCS Comm. Bytes	Address of 1st Part of PIB Table	ID Number of Last Checkpoint
	XX	XX	XX	XX	XX	XX	XX	X	XXXXXXXXX	XX	XX	XX

Displacement hexadecimal	5E	60	62	64	66	68	6A	6C	6E
Displacement decimal	94	96	98	100	102	104	106	108	110
	Length of LUB ID Queue = No. of Channel Queue Entries	Address of Disk Information Block (DIB)	Address of Error Recovery Block	Address of PC Option Table less 8 bytes	Address of IT Option Table less 8 bytes	Address of OC Option Table less 8 bytes	Key of Program with Timer Support	Address of the LUBID Queue	Logical Transient Key
	XX	XX	XX	XX	XX	XX	XX	XX	XX

Displacement hexadecimal	70	7C	7E	80	84	86	87	88
Displacement decimal	112	124	126	128	132	134	135	136
	Supervisor Constants	Address of 2nd Part of PIB Table	Address of MICR DTF Table (PDTABB)	Address of QTAM Vector Table	Address of BG Comm. Region	Option Indicator	System Configuration Byte 2	Pointer to Comm. Region Extension
	XXXXXXXXXXXX	XX	XX	XXXX	XX	X	X	XXXX

* The address of the communications region is in fixed location X'14' - X'17'.

Displacement values illustrated can be used to access the listing and/or the key that follows the figure. The key offers more detailed information about each area when necessary.

Figure 1.7. System Communications Region (Part 1 of 5)

Key to Communications Region Displacements:

0 MM/DD/YY or DD/MM/YY obtained from the job control date statement. Format controlled by COMREG + 53 (System Configuration Byte, date convention bit 0).

8 Address of the problem program area.

10 Address of the beginning of the problem program area. Y (EOSSP)=Y (PPBEG) if the storage protection option has not been selected. Y (EOSSP) equals the first main storage location with a storage protection key of 1, if storage protection is supported.

12 User area. If seek separation option is specified, bytes 12 and 13 are used at IPL time for the address of the seek address block.

23 User program switch indicator.

24 Job name set by the job control program from information found in the job statement.

32 Address of the uppermost byte of the problem program area as determined by the IPL program (Clear storage routine determines the address, ENDRD routine of $$A$IPL2 stores it.), or the address of the uppermost byte of the partition as determined during processing of the ALLOC statement.

36 Address of the uppermost byte of the last phase of the problem program fetched or loaded. The initial value (as shown) is overlaid by the first fetch or load to the problem program area.

40 Highest ending main-storage address of the phase among all the phases having the same first four characters as the operand on the EXEC statement. For the background partition only, job control builds a phase directory of these phases. The address value may be incorrect if the program loads any of these phases above its link-edited origin address. If the EXEC statement has no operand, job control places in this location the ending address of the program just link-edited.

44 Length of the problem program label area.

46 Program Interrupt Key - PIK (if asynchronous processing is not supported): Value is equal to the displacement from the start of the PIB table to the PIB for the task.

OR

Partition Identifier - PID (if asynchronous processing is supported): Value is hex 10, 20, or 30 to identify the partition in which a maintask or a subtask is running. (See the communications region extension, displacement 18, for the PIK in an asynchronous processing supervisor.)

First byte - always zero.
Second byte - contains the key of the program that was last enabled for interrupts, or the partition identifier in an AP supervisor.

Task	PIK (PID) Value
*All Bound	X'00'
BG	X'10'
*F2	X'20'
*F1	X'30'
Attn Rtn	X'40'
Quiesce I/O	X'50'
Supervisor	X'60'

*These tasks do not exist in a non-MPS supervisor.

48 Logical end of main storage address.

Figure 1.7. System Communications Region (Part 2 of 5)

Key to Communications Region Displacements:

52 Machine Configuration Byte (Values set at supervisor generation time.)

Bit 0: 1 = Storage protect feature
0 = No storage protect feature
1: 1 = Decimal feature
0 = No decimal feature
2: 1 = Floating-point feature
0 = No floating-point feature
3: 1 = Physical transient overlap option
0 = No physical transient overlap option
4: 1 = Timer feature
0 = No timer feature
5: 1 = Channel switching device
0 = No channel switching device
6: 1 = Burst mode on multiplex channel support
0 = No burst mode on multiplex channel support
7: Reserved

53 System Configuration Byte

Bit 0: 1 = DDMMYY
0 = MMDDYY } (Date convention bit set at generation time by STDJC)
1: 1 = Multiprogramming environment
0 = Batch job environment
2: 1 = DASD file-protect supported
0 = No file-protect support for DASD
3: 1 = DASD SYSIN - SYSOUT
0 = No DASD SYSIN - SYSOUT
4: 1 = Teleprocessing
0 = No teleprocessing
5: 1 = Batch job in foreground
0 = No BJF
6: 1 = Asynchronous processing
0 = No AP
7: 1 = Track Hold
0 = No Track Hold

54 This byte contains the standard language translator I/O options (set by the STDJC macro).

Bit 0:	DECK option	1 = yes, output object modules on SYSPCH
1:	LIST option	1 = yes, output source module listings and diagnostics on SYSLST
2:	LISTX option	1 = yes, output hexadecimal object module listings on SYSLST (compilers only)
3:	SYM option	1 = yes, output symbol tables on SYSLST/SYSPCH
4:	XREF option	1 = yes, output symbolic cross reference list on SYSLST
5:	ERRS option	1 = yes, output diagnostics on SYSLST (compilers only)
6:	CHARSET option	1 = 48, input on SYSIPT is 48 or 60 character set
7:	Reserved	

55 This byte contains the standard supervisor options for abnormal EOJ and control statement display, and the indicator for the presence of the ASCII-EBCDIC and EBCDIC-ASCII translation tables.

Bit 0:	Always on	
1:	DUMP option	1 = yes, dump registers and storage on SYSLST
2:	Reserved	
3:	LOG option	1 = yes, list all control statements on SYSLST
4-6:	Reserved	
7:	ASCII option	1 = yes, ASCII supported

Figure 1.7. System Communications Region (Part 3 of 5)

Key to Communications Region Displacement:

56 Job control byte

Bit 0: 1 = Job Accounting Interface (JA) not supported
0 = Job Accounting Interface (JA) is supported
1: 1 = Return to caller on LIOCS disk open failure
0 = Do not return to caller on LIOCS disk open failure
2: 1 = Job control input from SYSRDR
0 = Job control input from SYSLOG
3: 1 = Job control output on SYSLOG
0 = Job control output not on SYSLOG
4: 1 = Cancel job
0 = Do not cancel job
5: 1 = Pause at end-of-job step
0 = No pause at end-of-job step
6: 1 = SYSLOG is not a 1052
0 = SYSLOG is a 1052
7: 1 = SYSLOG is assigned to the same device as SYSLST
0 = SYSLOG is not assigned to the same device as SYSLST

57 Linkage control byte

Bit 0: 1 = SYSLNK open for output
0 = SYSLNK not open for output
1: 1 = $ or FG program phase deleted, renamed, or cataloged (flag bit for $MAINEOJ)
2: 1 = Allow EXEC
0 = Suppress EXEC
3: 1 = Catalog linkage editor output
0 = Do not catalog linkage editor output
4: 1 = Supervisor has been updated
0 = Supervisor has not been updated
5: 1 = Executing in AUTOTEST mode
0 = Not executing in AUTOTEST mode
6: 1 = Reallocate or condense in progress
7: 1 = Fetch $MAINEOJ at end of job to update system directory
0 = Do not fetch $MAINEOJ at end of job for update

58 Language processor control byte. This is a set of switches used to specify nonstandard language translator options. The switches within the byte are controlled by job control OPTION statements and when set to 1, override standard options. The format of this byte is identical to the stnadard option byte (displacement 54) with one exception: Bit 7 in this byte is used to indicate to LIOCS that the rewind and unload option has been specified.

59 Job duration indicator byte

Bit 0: 1 = Within a job condition
0 = Outside a job condition
1: 1 = Dump on an abnormal end-of-job condition
0 = No dump on abnormal EOJ
2: 1 = Pause at EOJ step } Set by Attention Routine for Job Control
0 = No pause at EOJ }
3: 1 = Job control output on SYSLST
0 = Output not on SYSLST
4: 1 = Job is being run out of sequence with a temporary assignment for SYSRDR
0 = Conditions for 1 setting not met
5: 1 = PCIL is being condensed
0 = PCIL is not being condensed
6: Reserved
7: 1 = Batch command just issued
0 = Condition for 1 setting did not occur

Figure 1.7. System Communications Region (Part 4 of 5)

Key to Communications Region Displacements:

60	Binary disk address of the volume label area (label cylinder).
62 → 76	As illustrated (Figures for information blocks, I/O tables, and pointers begin at Figure 1.14 which refers to more detailed Figures).
78	Set to the value nn specified in the LINES = nn parameter of the STDJC macro.
79	The format of the system date contained within this field is determined by the IPL program from information supplied in the date convention byte (displacement 53). Bytes 85 - 87 contain the day count.
88	Bytes reserved for use by LIOCS. Transient dump programs insert a key to indicate to the LIOCS end-of-volume routine, $$BCMT07, that it was called by a B-transient.
90	Address of the first part of the program information block (PIB) table. (See Figures 4.4, and 4.5.)
92	ID number of the last checkpoint. Temporary indicator of file protected DASD. Used at IPL time, when DASDFP is specified.
94	Length of the LUBID queue (in bytes). This equals the number of channel queue entries. It can also be used to access the REQID, LUBDSP, and TKREQID queues: (See Figure 1.29.)
96	Address of disk I/O position data. This is the starting address of the disk information block (DIB) table (See Figure 1.24).
98	Address of the beginning of the error recovery block. The error recovery block contains addresses of error recovery exits, error recovery queue information that can be used by physical transients routines, and defines storage for the error queue entries
100 → 104	As illustrated (See Figure 1.13).
106	Key of the program (BG, F2, or F1) that has timer support.
108	As illustrated (See Figure 1.29).
110	Logical Transient Key (LTK) contains the same value as the PIK (PID) (Displacement 46) when the logical transient is requested. When the transient area is not in use, LTK is equal to zero. The SVC 2 routine sets the LTK. The SVC 11 routine resets the LTK.
112	Supervisor constants: DOLLARBO (4 bytes) = C'$$BO' SSKADR (5 bytes) = XL5'0' LTAREA (3 bytes) = Adcon of LTSVPT, logical transient save pointer
124	Address of second part of program information block (PIB) table (See Figure 4.6).
126	Address of PDTABB, table of DTF addresses for MICR support (See Figure 1.9).
128	Address of QTAM vector table (IJLQTTAD).
132	Address of background communications region.
134	Option Indicator Byte Bit 0: 1 = MCRR indicated for OBR writer 0 = No MCRR indicated for OBR writer 1: 1 = EU interface active 0 = EU interface not active 2: 1 = Teleprocessing request 0 = No teleprocessing request 3: 1 = Supervisor support for only 9-track tape 0 = Supervisor does not support 9-track tape exclusively 4: Reserved 5: 1 = RETAIN/370 support generated 0 = RETAIN/370 support not generated 6-7: Reserved
135	System Configuration Byte 2 Bit 0: 1 = PCIL supported 0 = PCIL not supported 1 -7: Reserved
136	Pointer to communications region extension (See Figure 1.8).

Figure 1.7. System Communications Region (Part 5 of 5)

BGXTNSN (See Note)

0 (Hexadecimal Displacement)	4	8	0C	10	12	14	18	1C	20
0 (Decimal Displacement)	4	8	12	16	18	20	24	28	32
CE Table Address	Track Hold Table Address (THTABAD)	Difference Between 1st and 2nd Part of PIB Table (PIBDIFF)	AB Termination Table Address -8 (ABPTR)	ID of Task Owning LTA (LID)	ID of Task Running (PIK)	Task Requester ID Table Address (TKIDPTR)	Address Used by QTAM (MVCFLD)	SDR Communications Address (SDRTABLE)	TEBV Table Address (TEBVTAB)
XXXX	XXXX	XXXX	XXXX	XX	XX	XXXX	XXXX	XXXX	XXXX

24 (Hexadecimal Displacement)	28	2C	30	34	38	3C
36 (Decimal Displacement)	40	44	48	52	56	60
OLTEP Linkage Address	RMS Linkage Address (RASLINK)	ASCII-EBCDIC Translation Table Address	(Reserved)	JAI Common Table Address (ACCTCOMN)	JAI Partition Table Address (ACCTxx)	&SYSPARM Field Address
XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX

Key to displacements:

0	CE Table Address.
4	Track Hold Table Address (THTABAD).
8	Difference between addresses of first part of PIB table and second part of PIB table (PIBDIFF).
12	Abnormal Termination Table Address (minus 8) (ABPTR).
16	Identification (LID) of the task owning the Logical Transient Area. Contains same value as PIK (displacement 18) when LTA is in use. Contains zero when LTA is not in use.
18	Program Interrupt Key (PIK) if asynchronous processing is supported. Value is equal to the displacement of the start of the PIB table to the PIB of the main task or subtask being selected (running). First byte - zero Second byte - contains the displacement into the PIB table for a maintask or a subtask. Maintask - PIK value is hex 10, 20, or 30. Subtask - PIK value is hex 70, 80, 90, . . . F0.
20	Task Requester ID Table Address (TKIDPTR).
24	MVCFLD address used by QTAM.
28	Statistical Data Recorder Table Address (SDRTABLE).
32	Tape Error Blocks by Volume Table Address (TEBVTAB).
36	Pointer to OLTEP Linkage Addresses
40	RMS Linkage Area Address (RASLINK)
44	ASCII-EBCDIC Translation Table Address.
48	(Reserved)
52	JAI Common Table Address (ACCTCOMN)
56	JAI Partition Table Address (ACCTxx; where xx = BG, F2, or F1).
60	Address of &SYSPARM Field.

Note: If communications regions are generated for the foreground partitions, the labels in those extensions will be F2XTNSN and F1XTNSN. The extensions, wherever used, are generated by the COMMNEX macro. Following the background extension (and immediately preceding the MCRR Linkage Table) is a six-byte area. The first four bytes are the address of the background save area (BGSAV), and the last two bytes are the value 4,096, used to restore base registers.

Figure 1.8 Background Communications Region Extension (BGXTNSN)

MAGNETIC INK CHARACTER RECOGNITION SUPPORT (MICR)

A Magnetic Character Reader, such as an IBM 1255, 1259, 1412, or 1419, can be attached to a channel for reading magnetically-inscribed data on checks and other banking documents. They differ mainly in document reading rates. The 1255 reads at speeds as high as 500 six-inch-long documents per minute on its Model 1, and 750 documents per minute on its Models 2 and 3; the 1259 reads at speeds as high as 600 six-inch-long documents per minute; the 1412, at speeds as high as 950 documents per minute; the 1419, at speeds as high as 1,600 per minute. Specific speeds depend on document length as well as on the program.

System Considerations

The DOS supervisor support allows operation of Magnetic Ink Character Recognition (MICR) devices in either a foreground or background area. An extension to the DOS supervisor monitors, by means of external interrupts, the reading of documents into a user-supplied I/O area (document buffer area). All MICR documents must be accessed through logical IOCS. Logical IOCS gives you the next sequential document and automatically engages and disengages the devices, as necessary, to provide a continuous stream of input. Detected error conditions and information are passed to you in each document buffer.

The magnetic character readers are unique in that documents must be read at a rate dictated by the device rather than by the program. To ensure time for necessary processing (including determination of pocket selection) a MICR device generates an external interrupt at read completion of each MICR document. The supervisor gives highest priority to external interrupt processing.

In an MPS system with MICR document processing, any partition (background or foreground) can use MICR devices. For programs with one MICR device, GET macro instructions are provided. For multiple MICR processing, READ, CHECK, and WAITF macro instructions allow processing to continue as long as one of the files has documents ready for processing. Figure 1.9 shows the tables for MICR DTF addresses and pointers.

System Generation Guidelines

To specify supervisor support for MICR devices, the MICR= parameter must be included in the SUPVR macro.

1. If both 1412s and 1419s are present, specify MICR=1419. If 1255s or 1259s are to be supported, also specify MICR=1419. MICR=1419D indicates Dual Address Adapter 1419s.
2. 1419 support gives 1255/1259/1270/1275 capability. The 1270/1275 are optical reader/sorters.
3. If 1255/1259/1270/1275/1412/1419s are attached to the multiplexor channel, the PIOCS parameter BMPX=YES is not supported.
4. If MICR support is required on a Model 65, specify MODEL=65 in the CONFG macro.
5. For MICR support on selector channel, specify MRSLCH=YES in the PIOCS macro.

MICR processing requires at least two I/O channels. If MICR devices are attached to the multiplexor channel, no burst mode devices are supported on the multiplexor channel. MICRs should be attached as the highest priority devices on the multiplexor channel. Single addressing 1412s or 1419s are supported on any selector channel, but device performance is maintained only if a selector channel is dedicated to a single MICR device. Also note that the Dual Address 1419 is not attachable to selector channels.

In addition, MICR processing requires either the direct control feature or the external interrupt feature.

The table of DTF addresses (PDTABB) contains six 8-byte entries; one for each external line of the direct control feature on the system.

PDTABB

Byte →	0	1	2	3	4	5	6	7
0	NI	PDSTAT+1,	X'FE'		Ownership Flags	DTF address for MICR: Device on line 7		
8	NI	PDSTAT+1,	X'FD'			Device on line 6		
16	NI	PDSTAT+1,	X'FB'			Device on line 5		
24	NI	PDSTAT+1,	X'F7'			Device on line 4		
32	NI	PDSTAT+1,	X'EF'			Device on line 3		
40	NI	PDSTAT+1,	X'DF'			Device on line 2		

Background = 10
Foreground 2 = 20
Foreground 1 = 30

- Bytes 0-3 -- Contain an 'AND' instruction that is executed in main line coding to turn off the external line status after its detection.

  PDSTAT+1 contains one or more of the following interrupt codes:

PSW Interrupt Code Bit	Interrupt Code (PSW Bits 26-31)*	External Interrupt Cause
31	nnnnnnn1	External signal 7
30	nnnnnn1n	External signal 6
29	nnnnn1nn	External signal 5
28	nnnn1nnn	External signal 4
27	nnn1nnnn	External signal 3
26	nn1nnnnn	External signal 2

- Byte 4-- Contains the flag of the partition containing the DTF.
- Bytes 5-7 -- Contain the address of the DTF table.

Table of pointers (PDTABA) to DTF addresses associated with the external interrupt line. The table contains the status in descending order from Bit 31 to Bit 26 of the external old PSW.

PDTABA

Byte →	0	1	2	3	4	5	6	7
0	00	08	00	10	00	08	00	18
8	00	08	00	10	00	08	00	20
16	00	08	00	10	00	08	00	18
24	00	08	00	10	00	08	00	28
32	00	08	00	10	00	08	00	18
40	00	08	00	10	00	08	00	20
48	00	08	00	10	00	08	00	18
56	00	08	00	10	00	08	00	

*n =other external-interrupt conditions.

Bytes 126 and 127 (X'7E'-'7F') of the communications region contain the address of these tables. Label PDTABB identifies the first byte of the first table.

Figure 1.9. Tables for MICR DTF Addresses and Pointers

ASYNCHRONOUS PROCESSING (AP)

The asynchronous processing function, also known as multitasking, provides greater use of system resources at the partition level. Multitasking provides the ability to execute more than one program in a partition, that is, the ability to do multiprogramming within a partition (or in all three partitions) of the DOS system. Just as multiprogramming between partitions can increase the system throughput, multitasking can increase overlap of I/O activity and computer processing for a given job.

To perform multiprogramming within a partition, the program must consist of a main program (main task) and one or more subprograms (subtasks).

Because multitasking is a logical extension of the current task selection mechanism, a maximum of nine subtasks can exist in the system at any given time. These nine subtasks can all reside in one partition, or can be spread among the three available partitions. A total of 12 tasks (a task can be considered either a main task or a subtask) can be executed concurrently in the system.

The subtasks share the same partition with their associated main task. The main task initiates (attaches) execution of the subtasks. The ability of the main task to attach subtasks minimizes operator intervention. Storage within the partition may be allocated to the main task and its associated subtasks in any way desired by the user. Subtasks have the same storage protect key as the main task.

When subtasks are attached to a given partition, they retain the priority of that partition. Priorities are also established within the partitions. The priority within a partition is determined by the order in which a subtask is initiated. The first subtask to be attached has the highest priority, and as each subsequent subtask is attached, it has the next highest priority, followed by the main task which has the lowest priority. When a subtask is attached, it receives control from the system before control is returned to the main task. See Figure 1.10 for an example describing priority structure in a multitasking environment. If the F1 partition has two attached subtasks, the F2 partition has four attached subtasks, and the BG partition has three attached subtasks (a maximum of nine subtasks), their priority would be as shown (with 1 being the highest priority and 12 the lowest priority).

A subtask can operate independently of its main task and has its own save area for registers. The subtask can communicate with other subtasks and main task via a set of macro instructions (see Intertask Communication).

Partition		Priority
F1	Subtask 1	1
	Subtask 2	2
	Main Task	3
F2	Subtask 1	4
	Subtask 2	5
	Subtask 3	6
	Subtask 4	7
	Main task	8
BG	Subtask 1	9
	Subtask 2	10
	Subtask 3	11
	Main Task	12

Figure 1.10. Example of Multitasking Priorities

When a subtask is no longer required, it can be detached from the system. The subtask can either detach itself or be detached by its main task. When one or more subtasks are detached, subtasks with lower priorities receive the next highest priority. If a detached subtask is later attached, it becomes the lowest priority subtask in the partition, but it still has higher priority than the main task.

System Considerations

Under DOS there are additional optional components and specifications that greatly enhance multitasking operations.

Track Hold: The track hold facility prevents two independent subroutines in the same partition or in two different partitions from simultaneously trying to update the same record or write a new record on the same track when processing DTFDA, DTFIS, and DTFSD files. When this facility is used, a second routine requesting an I/O operation on a track being held must wait for that track to be freed by the first routine. Because track hold is implemented by programming rather than hardware, all routines processing the same DASD files must use this facility to ensure proper protection. This facility can be used without specifying AP=YES and is specified at system generation time. See discussion of TRKHLD parameter under FOPT Macro for more information.

Multiple Wait: Under DOS, a number of independent logical IOCS operations (requiring explicit waiting for completion) can be initiated before waiting for the completion of any particular operation. Once all logical IOCS operations have been initiated, you must determine the sequence in which you will wait for their completion. Once you wait for a particular operation, you no longer have control, even though one of the remaining operations completes before the one on which the wait occurs and useful processing could have been done. This can be avoided at the physical IOCS level, but requires some additional coding effort on your part.

The multiple wait facility allows you to wait asynchronously for any one of a number of I/O operations to complete at either the logical or physical IOCS level for the above situation. This facility provides increased I/O overlap processing and is specified at system generation time. See discussion of WAITM parameter under FOPT Macro for more information.

Abnormal Termination: Under DOS, your program is canceled when certain error conditions occur. In many cases, it is desirable to perform certain termination functions (e.g., close files) to minimize any problems that may occur. The abnormal termination facility allows for these situations via a user exit. This function is specified at system generation time and implemented via the AB operand of the STXIT macro.

When the supervisor determines that the task has been abnormally terminated, control passes to the task's abnormal termination routine. In this routine, you may close data files (such as an indexed sequential ADD) or perform other operations that are necessary to minimize any possible damage. Abnormal termination exits can be established for both main tasks and subtasks or, if desired, subtasks can share the coding of their main task's abnormal termination routine. It is strongly suggested that in the shared abnormal termination routine no I/O be performed. If I/O is attempted and causes cancelation, all tasks in the partition are canceled. Any abnormal termination within the abnormal termination routine causes the task (or job if in the main task) to be canceled without regard to an abnormal termination exit. This facility of sharing an abnormal termination routine can be used even if the multitasking function (AP=YES) is not used. See AB parameter under FOPT Macro for more explanation.

Reentrant Modules: Reentrant modules for CDMOD, DAMOD, ISMOD, MTMOD, PRMOD, SDMOD, and DIMOD allow a module to be shared by the same device type DTFs in a multitasking environment. For example, one PRMOD can support several subtasks using multiple printers within a partition. One DAMOD can support several subtasks within a partition.

Multitasking Macro Usage

Although these four functions (asynchronous processing, track hold, multiple wait and abnormal termination) can be used independently, they are discussed under the heading where they are most frequently used. The multitasking macros are designed to handle three basic situations: subtask initiation and normal termination,, resource protection and intertask communication. See Supervisor and I/O Macros listed in the Preface for a description of the macro formats. Some examples using the multitasking macros are included.

Subtask Initiation and Normal Termination

Subtask initiation can only be performed by a main task that issues an ATTACH macro instruction. Normal subtask termination can be performed by either a main task or a subtask that issues a DETACH macro instruction.

ATTACH Macro Considerations

Only a main task can attach subtasks. A maximum of nine subtasks can be attached in the system at any given time. They can all reside in one partition or be spread among the three partitions in any combination. If a main task attempts to attach a tenth subtask to the system, a supervisor Event Control Block (SPVECB) is unposted (SPVECB+2, bit 0, set to zero), the address of the ECB is stored in general register 1 of the main task, and bit 0 of register 1 is set to 1 giving the register a negative value. The main task can test register 1 for a negative value and, if found, wait on register 1 until one of the nine subtasks is detached. Figure 1.11 illustrates the ECB.

When a subtask is successfully attached, it has a higher priority than its main task. Therefore, control is passed to the subtask before it is returned to the main task. In addition, the registers of the subtask contain the same values as the registers of the main task (both the general registers and floating point registers if specified), with the following two exceptions:

1. Register 1 of the subtask contains the address of the save area for the main task.
2. Register 0 of the main task contains the address of the byte immediately following the save area of the subtask (save area+96 if no floating point registers, or save area+128, if floating point registers).

The passing of the main task's registers to its attached subtask(s) is worth noting, because the subtask(s) can be under control of and use the main task's base register without initializing it. In addition:

1. The subtask ID (a value from X'70' to X'F0') is stored in the subtask's save area (save area+88, if no floating point registers or save area+120, if floating point registers)
2. The address of the subtask's entry point is stored in the save area (save area+13), and
3. Byte 2 bits 0 and 1 of the subtask's ECB are set to 0 (unposted).

You should store the subtask name in the first eight bytes of the save area to be used for subtask identification when messages are printed on SYSLOG.

In certain instances, a routine to be attached may not be in main storage. In this case, the entry point could be the label of a FETCH or LOAD routine that fetches the desired routine into storage.

The following conditions cause cancelation of a main task (or possibly a subtask).

1. A main task has not issued the ATTACH macro (issued by another subtask).
2. The subtask save area is not aligned on a doubleword boundary.
3. The save area of the subtask being attached is not within the partition.
4. The entry point of the subtask itself is not within the partition.
5. The ABSAVE save area, if any, is not within the partition.
6. The Event Control Block (ECB) of the subtask is not within the partition.

If a main task is canceled, all subtasks in that partition are canceled.

DETACH Macro Considerations

A main task can detach any subtask (within its partition), but a subtask can only detach itself. In addition, a subtask can be detached by issuing the CANCEL, EOJ and DUMP macros. If a subtask is detached, all pending I/O operations are completed before the DETACH operation is completed. In addition, any tracks being held by the subtask are freed.

If the subtask has an ECB, the ECB is posted (ECB+2, bit 0 set to one) and any tasks waiting on the ECB are removed from wait state. The task with the highest priority then gains control. The supervisor ECB is also posted (SPVECB+2, bit 0 set to one) and any main task waiting on it is removed from wait state.

Although a main task can detach a subtask, it is generally more desirable for a subtask to detach itself. The entire system could be put into wait state if two (or all three) main tasks attempted to attach more than the nine subtasks allowed by the system. The following two examples show what could happen if the main task is allowed to detach its subtasks.

Example 1: All partitions attempt to attach five subtasks apiece. Each set of subtasks is independent and processing is such that each main task has a chance to attach a subtask before any one main task has attached all its subtasks. The entire system could then be placed in wait state, because the main task is not able to get to the routine to detach a subtask when it has successfully completed (it is waiting to attach another subtask).

Example 2: Placing one or two of the three partitions in wait state is another situation that could occur, if all partitions attempted to attach five subtasks each. The F1 partition may attach all five of the subtasks. F2 partition may attach four of its five subtasks, and the BG partition may not attach any of its subtasks and, therefore, be in wait state. The main task of the F2 partition would also be in wait state because it still has one more subtask to attach. These subtasks remaining to be attached have to wait until one or more of the attached subtasks are detached. In addition, if the five subtasks in the F2 partition are dependent upon each other, the entire F2 partition could also be in wait state. Thus, only the tasks in the F1 partition may be executing.

The following conditions cause cancelation of a main task or a subtask:

1. The main task detaches and does not pass the address of the subtask save area (if a subtask detaches and passes the save area address, it is ignored).

2. The main task detaches and the subtask is already canceling or canceled.

3. The limits of the save area specified in the DETACH macro do not reside in the partition of the main task. If the main task is canceled, all subtasks within the partition are canceled.

4. The subtask ID stored in the save area is not a valid subtask ID (hexadecimal 70-F0).

In the last case, a check cannot be made if the value has been altered to that of another subtask ID. In this case, it would be possible for the wrong subtask to be detached because this is the only way the system can locate the task being detached. (This is also the reason why the main task must specify the save area of the subtask.) In addition, the system sets the invalid ID to binary zeros.

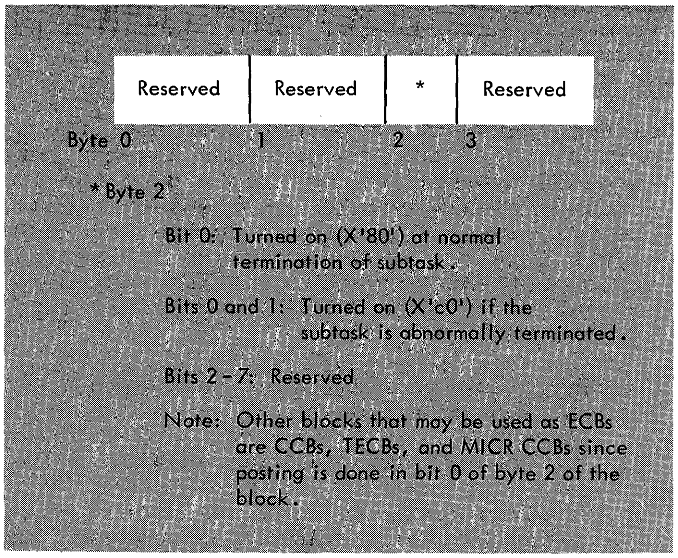
Resource Protection

The resource may be a file, an I/O device, a DTF, a work area or I/O area, or a set of non-reentrant code, etc. In general, it is anything that has the possibility of being shared by two or more tasks. A means of protection has been provided so that two tasks sharing the same resource don't access the resource at the same time.

Resource protection can be accomplished in one of two ways, depending upon the resource to be protected.

1. The first technique applies to the types of resources just stated and requires three macro instructions (RCB, ENQ, DEQ). The RCB macro generates a Resource Control Block (RCB) that is associated with the resource to be protected, but is not necessarily a part of that resource. When a resource is to be protected, an ENQ macro must be issued to enqueue an RCB. This places a hold on the RCB associated with the resource until the enqueuing task releases the RCB by issuing a DEQ macro to dequeue the RCB. These resource protection macros apply only within a partition and not across partition boundaries. That is, a resource protected by an enqueue in one partition is not protected in another partition because an RCB is generated in its own partition and cannot be accessed by other partitions. Figures 1.11 and 1.12 show the ECB and RCB.

2. The second technique can only be applied to DTFDA, DTFIS, and DTFSD DASD files or those files you created using physical IOCS (EXCP/WAIT macros). This is the track hold facility previously discussed. In contrast to the first technique, track hold applies across partitions.



	Reserved	Reserved	*	Reserved
Byte	0	1	2	3

*Byte 2

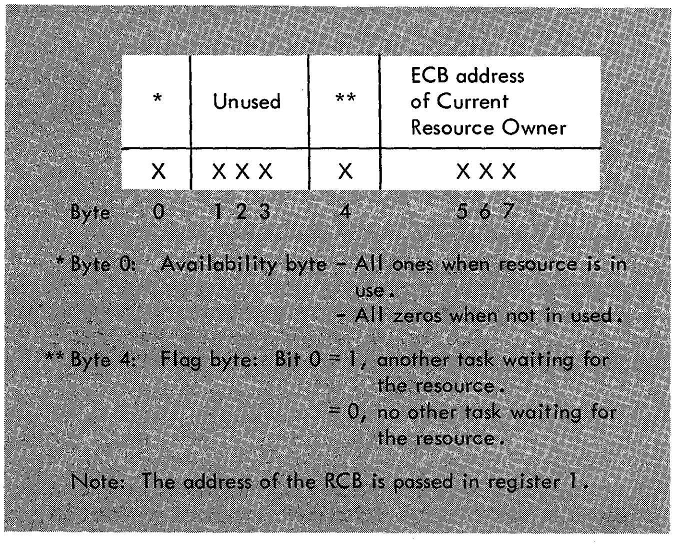
Bit 0: Turned on (X'80') at normal termination of subtask.

Bits 0 and 1: Turned on (X'c0') if the subtask is abnormally terminated.

Bits 2-7: Reserved

Note: Other blocks that may be used as ECBs are CCBs, TECBs, and MICR CCBs since posting is done in bit 0 of byte 2 of the block.

Figure 1.11. Event Control Block (ECB)



	*	Unused	**	ECB address of Current Resource Owner
	X	X X X	X	X X X
Byte	0	1 2 3	4	5 6 7

* Byte 0: Availability byte - All ones when resource is in use.
- All zeros when not in used.

** Byte 4: Flag byte: Bit 0 = 1, another task waiting for the resource.
= 0, no other task waiting for the resource.

Note: The address of the RCB is passed in register 1.

Figure 1.12. Resource Control Block (RCB)

ENQ Macro Considerations

A resource can only be protected within a partition.

Every subtask that enqueues a resource must have an ECB specified in its ATTACH macro (issued by the main task) and that ECB should not be used for any other purpose while a resource is enqueued. The address of the ECB is stored in the RCB. A main task does not require an ECB and has no means of specifying that it has an ECB (the address of which could be stored in the RCB). When a main task enqueues a resource, the ECB address field of the RCB is set to binary zeros which identifies the resource owner as being the main task.

A task requesting use of a resource is either enqueued and executed or put into wait state if the resource has already been enqueued by another task (byte 0 of the RCB contains binary ones). In the latter case, the flag byte is turned on in the RCB (byte 4, bit 0 is set to one) and the ECB address of the current resource owner is stored in general register 1 of the task placed into wait state.

The following conditions can cause cancelation of a task:

1. A subtask does not have an ECB.
2. A previous owner of a resource terminated without having dequeued the resource. (If the main task terminated, the entire partition was terminated.)
3. A task issued two consecutive ENQs for the same RCB without having issued an intervening DEQ.
4. The limits of the RCB specified in the ENQ do not reside in the partition of the enqueuing task.

DEQ Macro Considerations

A resource can only be dequeued within a partition. Only the current owner of a resource can dequeue that resource.

When an RCB is dequeued, byte 0 is reset to binary zeros, and all tasks waiting for that resource are removed from wait state. The task with the highest priority obtains control. If no other tasks are waiting for the RCB, control returns to the dequeuing task.

The following conditions cause cancelation of a task:

1. The RCB has the wait bit on in its flag byte (byte 4) and no waiting task which has been enqueued can be found for the RCB. This could be caused by the flag byte being inadvertently altered while a resource was enqueued.
2. A subtask does not have an ECB.
3. The limits of the RCB specified in the DEQ do not reside in the partition of the dequeuing task.

Intertask Communication

In certain situations, tasks may be dependent on other tasks within a partition. In these cases, macro instructions (POST, WAIT and WAITM) have been provided to permit synchronization of tasks for intertask communication. To use these macros, each task must have an Event Control Block (ECB) associated with it.

When a particular task is performing a function and other tasks are waiting for its completion, it can indicate completion via the ECB by issuing a POST macro instruction. The tasks waiting for the completion may indicate this by issuing the WAIT macro instruction, designating the ECB on which the waiting is to be done. If a task is waiting for completion of a number of events (in which the order of completion is of no importance), it can issue the multiple wait macro instruction (WAITM) designating a number of ECBs.

POST Macro Considerations

The tasks removed from wait state are those placed in wait state by ENQ, WAIT, or WAITM. When the SAVE parameter is omitted in the POST macro instruction, all tasks waiting on the specified ECB are removed from wait state. By specifying a SAVE parameter, only the task identified by the save area is removed from wait state. This parameter can be used for synchronizing the order in which tasks are to receive control. The synchronization technique prevents the priority task within a partition from gaining control.

Be careful with this technique when the ECB to be posted is the ECB specified in the ATTACH macro and ENQ/DEQ macros are used, because DEQ also removes all waiting tasks from wait state. When the posting task dequeues, all tasks waiting for the RCB are removed from wait state.

A similar situation exists if the posting task dequeues before posting. The DEQ removes all tasks waiting for the RCB from wait state. Then, if it issues a POST to a particular task, the POST acts as a NOP because task selection gives control to the highest priority task ready to use the CPU. Although the task being posted is removed from wait state, the posting task is still active. If the posting task issues another POST to the same or another ECB, all other tasks waiting on the posted ECB are removed from wait state. To avoid this situation, use a second ECB when synchronizing tasks. It is your responsibility to reset the wait bit in the second ECB that is to be posted (MVI ECB+2,X'00'), so that tasks testing that ECB can be put in wait state.

If a task associated with the specified save area cannot be found, the post operation is ignored and control passes to the highest priority task that is ready to run.

A task can be canceled if the ECB specified in the POST macro instruction does not reside within the partition.

Summary of Multitasking Considerations

Maximum Number of Tasks: A maximum of 9 subtasks can be attached to the system. They can all reside in one partition or be spread among the three partitions. Thus, a total of 12 tasks can be executed concurrently in the system.

Subtask Priority: Each subtask must be initiated by a main task. A subtask has a higher priority than its main task. Subtask priority within a partition is determined by the order of attachment. The first subtask attachment has the highest priority in the partition, the next subtask has the second highest priority in the partition, etc. Of course, the priority of the partitions remains the same (i.e., F1, F2, and BG). If the highest priority subtask is terminated and later reattached, it will then be the lowest priority subtask within the partition, but still higher in priority than its main task, or any subsequent subtask that may be attached within that partition.

Storage Protection: Because subtasks are subprograms within a partition, they have the same storage protect key as the main task. Therefore, the main task and its subtasks do not have storage protection from each other. The ENQ, DEQ, and RCB macros offer protection of resources, but only if all subtasks enqueue and dequeue before using the resource. They do not protect against inadvertent coding errors. In addition, the user must be careful when using ENQ/DEQ. If a higher priority subtask dequeues an RCB and does not enter wait state before enqueuing the RCB again, a lower priority task does not gain control of that RCB on which it has also enqueued. Thus, the lower priority task has to wait until the higher priority task terminates, before it can enqueue the RCB.

Access To Communications Region: Only one communications region exists in a batched-job partition. Therefore, it is likely that only one task per partition has meaningful access to it.

System Logical Units: Only one set of system logical units exist per partition (SYSLST, SYSRDR, etc). Therefore, interspersed usage by several independent tasks is not practical, although, if either the resource protection facilities or the intertask communication macros are employed, it can be done.

Operator Intervention: While operator intervention is minimized for subtask initiation, SYSLOG will probably be used by all tasks within all partitions. The additional number of messages possible on one SYSLOG could possibly increase the responsibility of the operator and require more careful operation than in the past.

STXIT Macro Usage: Subtasks may only provide their own AB and PC routines via the STXIT macro. IT and OC operations must be performed via the main task. An AB exit is not taken for a task when it is already in its AB routine (prevents looping on abnormal termination condition). The task is canceled.

Checkpoint Consideration: Only main tasks can issue checkpoints.

Track Hold Facility: Files being shared on DASD are not protected unless the HOLD option is specified by the various users (only applies to DTFDA, DTFIS, DTFSD, and DTFPH files).

Register Usage: Although a subtask has the register values (2-15) of its main task upon being attached, the registers cannot be used for passing information between tasks once attached. (It is possible for a task to access the register save areas of other tasks within its partition.)

Process Bound Tasks Considerations: Because subtasks are executed in priority order, a process-bound task can degrade performance of lower priority tasks, or in extreme cases, even prevent execution of lower priority tasks until it has terminated.

Task Synchronization: Task synchronization is normally performed by POST, WAIT, and WAITM macros. This can also be done by ATTACH/DETACH or ENQ/DEQ providing you are careful, particularly when intermixing POST, ENQ/DEQ, and ATTACH/DETACH macros. While POST may be used to free one waiting task, DEQ and DETACH can free all tasks waiting on the ECB posted, if the ECB is the same one specified in the ATTACH.

Resource Protection: The POST, WAIT, and WAITM macros are also used for resource protection, providing you are careful in your synchronization techniques.

Resource Contention: The problem of resource contention cannot be over emphasized in this system. It has already been pointed out that you can interlock two or more tasks, or even put the system into wait state when two or more partitions are concurrently attaching more than 9 subtasks or when two or more tasks (or partitions) are contending for the same sets of tracks while using track hold. In addition, a similar problem can exist when two or more tasks within a partition are enqueuing and dequeuing on the same set of RCBs. For example, if task A enqueues RCB 1 and task B enqueues RCB 2, task A is put in wait state when it attempts to enqueue RCB 2. The same is true when task B attempts to enqueue RCB 1. Neither task is able to get out of wait state to release the resource it has enqueued.

This problem can be avoided by having each task, which shares common resources with other tasks, enqueue on the same resources in order. For example, task A enqueues on RCB 1 and then task B enqueues on RCB 1 (instead of RCB 2 first); task B goes into wait state. Task A can now enqueue on RCB 2 without entering wait state. When task A dequeues RCB 1, task B has the chance of enqueuing RCB 1 (providing task A does not enqueue RCB 1 again, before task B has a chance to reattempt its ENQ).

Another possibility of task interlock is for two tasks to wait on ECBs, with each task assuming that the other task will post the ECB on which it is waiting.

Subtask Cancelation: While the cancelation of a subtask frees tracks being held, and posts the subtask's ECB (as specified in the ATTACH macro) it does not dequeue any RCBs enqueued by that subtask. (Cancelation of a subtask executes the DETACH routines of the supervisor as well as the cancel routines.) Therefore, when the abnormal termination routine is entered, you should dequeue all RCBs that the subtask could have enqueued.

If the subtask issues a DEQ for an RCB on which it has not enqueued, the DEQ is ignored and the supervisor returns to the subtask's abnormal termination routine.

Wait Considerations: Although tasks can wait on ECBs to be posted by other tasks or on the ECBs of other tasks (in their own partition), they cannot wait on the CCB of another task when that task has initiated the I/O operation. This does not mean that two tasks cannot share a CCB as a resource. It only means that the system identifies the CCB with the task doing the I/O operation. Therefore, only that task can be removed from wait state by the system. Any other task waiting on another task's CCB can only be removed from wait state by having the task that started the I/O operation issue a POST to the CCB. In this case, the CCB would function like an ECB. Also, note that the task doing the I/O operation must issue the POST macro after the WAIT macro rather than before the WAIT macro. Otherwise it would never enter the wait state or determine when the I/O operation is completed.

Abnormal Termination: In all abnormal termination conditions where an exit is taken to an abnormal termination routine, the register values are stored in the ABSAVE save area before the appropriate error code is stored in the low-order byte of register 0. To have this value available when looking at a storage dump, you should store (STC or ST) register 0 in another save area upon entry into the abnormal termination routine. You will find that the SVC code shown in the "0S04I ILLEGAL SVC-..." message along with the error codes in register 0 will be helpful in tracing program errors. See Debugging Aids section for additional information on abnormal termination codes.

System Generation Guidelines

The multitasking facility is provided at system generation time by specifying AP=YES in the SUPVR macro. When AP=YES is specified, MPS=YES and WAITM=YES are implied. To implement the other facilities related to multitasking, the following additional specifications are required at system generation time:

- For abnormal termination support, AB=YES must be specified in the FOPT macro. AP=YES in the SUPVR macro is not required to utilize this function.

- For multiple wait support, WAITM=YES must be specified in the FOPT macro. Although the multiple wait function can be used without specifying AP=YES, AP=YES cannot be used without specifying WAITM=YES.

MULTITASKING EXAMPLES

ATTACH Macro Example

The normal procedure for attaching subtasks is as follows:

```
 1.  MAINTASK   BALR     2,0
 2.             USING    *,2
                •
                •
                STXIT    AB,MTABEND,MTSAVE
 3.             MVC      SUB1SAV(8),SUB1NAME    Initialize subtask 1 save area
 4.  ATST1      ATTACH   SUBTASK1,SAVE=ST1SAV,ECB=ST1ECB,ABSAVE=ST1ABSV
 5.             LTR      1,1                    Test if ATTACH is successful
 6.             BNM      ATST1OK                BR if successful
 7.             WAIT     (1)                    WAIT to retry ATTACH
 8.             B        ATST1                  BR to retry
 9.  ATST1OK    BCTR     0,0                    Get end of subtask 1 save area
10.             ST       0,ST1SVEND             Store ending address of subtask 1 save
                                                area
                •
                •
11.  SUBTASK1   BALR     3,0
12.             USING    *,3
13.             ST       1,MTSVAR               Store address of main task save area
14.  MTABEND    STC      0,ABSVCODE             Save ABTERM code
15.             C        1,=A(ST1ABSV)          Test if subtask 1 ABTERM
16.             BE       ST1ABEND               BR if YES
                •
                •
17.  ST1ABEND   EQU      *
                •
                •
18.             DS       0D                     Align on doubleword boundary
19.  ST1SAV     DC       16D'0'                 Subtask 1 save area
                                                with floating point
                                                registers
20.  ST1ABSV    DC       9D'0'                  Subtask 1 AB save area
21.  ST1ECB     DC       F'0'                   Subtask 1 ECB
22.  MTSVAR     DC       F'0'                   Address of main task
                                                save area
23.  ST1SVEND   DC       F'0'                   Ending address of
                                                subtask 1 save area
24.  SUB1NAME   DC       C'SUBTASK1'            Subtask 1 name
25.  ABSVCODE   DC       X'0'
26.  MTSAVE     DS       9D                     Main task save area
                                                used by STXIT
```

Explanation for ATTACH Macro: Statement 3 initializes the subtask save area with the name of the subtask which is used for messages for subtask identification when messages are written on SYSLOG.

Statement 4 is the ATTACH of the subtask. SUBTASK1 is the entry point of the subtask, ST1SAV is the save area for the subtask, ST1ECB is its ECB, and ST1ABSV is the ABTERM save area for the subtask. In this case, the subtask is using the main task's abnormal termination routine.

Statements 5 and 6 test for a successful ATTACH. If the ATTACH was not successful (nine subtasks already attached), the main task waits until another subtask is detached and retries the ATTACH. If the ATTACH was successful, the main task stores the ending address of the subtask's save area for later reference, if necessary. The main task can then continue to do other processing.

Statement 11 is the entry point to the subtask. In this example, the subtask and the main task use different base registers. This may not be necessary, depending on program design. The subtask could have omitted the BALR and USING statements because addressability is available through the main task register (register 2). The values in the main task registers are passed to the task. Therefore, register 2 would still be initialized.

Statement 13 saves the address of the main task's save area for reference by the subtask (if it is necessary for the subtask to name the main task in the POST macro instruction). Statement 14 stores the ABTERM code when the abnormal termination routine is entered. This routine is shared by both the main task and subtask 1. Statements 15 and 16 determine which task abnormally terminated (ABTERM save area of the task in error is stored in register 1). Statement 18 aligns the save areas on a doubleword boundary.

Statement 21 is the user-coded ECB for the subtask.

DETACH Macro Example

A subtask may detach itself or be detached by the main task.

```
 1.  MAINTASK   BALR      2,0
 2.             USING     *,2
                •
                •
 3.  ATST1      ATTACH ST1,SAVE=ST1SAV,ECB=ST1ECB
                •
                •
 4.  ATST2      ATTACH ST2,SAVE=ST2SAV,ECB=ST2ECB
                •
                •
 5.             DETACH SAVE=ST1SAV               Detach subtask 1
                •
                •
 6.  ST1        ST        1,MTSVAR1
                •
                •
 7.             B         ST1+4
 8.  ST2        ST        1,MTSVAR2
                •
                •
 9.  *DETACH Subtask 2
10.             DETACH
```

Explanation for DETACH Macro: The main task attaches two subtasks. When subtask 1 completes processing, it indicates this to the main task. The main task then detaches subtask 1 by issuing a DETACH macro and specifying the save area for subtask 1 (statement 5). When subtask 2 completes its processing, it detaches itself (statement 10). Note that an operand was not specified when subtask 2 detached itself, and that the comment was placed in a comments card (statement 9). The comment would have acted as an operand, resulting in an error.

ENQ/DEQ and RCB Macros Examples

EXAMPLE 1: When two subtasks share the same resource within a partition, they can use the resource protection macros as follows:

```
 1.  MAINTASK   BALR      2,0
 2.             USING     *,2
                •
                •
 3.  SUBTASK1   EQU       *
                •
                •
 4.  SBTASK1A   ENQ       RCB1                     Protect resource.
 5.             BAL       4,WRITEDTA               Write a record.
 6.             DEQ       RCB1                     Release resource.
                •
                •
 7.             B         SBTASK1A
                •
                •
 8.  SUBTASK2   EQU       *
                •
                •
 9.  SBTASK2A   ENQ       RCB1                     Protect resource.
10.             BAL       4,WRITEDTA               Write a record.
11.             DEQ       RCB1                     Release resource.
                •
                •
12.             B         SBTASK2A
                •
                •
13.  RCB1       RCB                                Resource control block for WRITEDTA
```

Explanation for Example 1: Both subtask 1 and subtask 2 are sharing the same file using a common subroutine. The subroutine is not reentrant, and the file cannot use track hold. Therefore, it is necessary for each subtask to enqueue on the RCB associated with the resource and dequeue when the resource can be released for a waiting subtask.

EXAMPLE 2: In the following example, two subtasks are sharing a common processing routine defined in the first subtask.

```
 1.  MTASK     START 0
               •
               •
 2.            ATTACH STASK1,SAVE=SAVE1,ECB=ECB1
               •
               •
 3.            ATTACH STASK2,SAVE=SAVE2,ECB=ECB2
               •
               •
 4.  STASK1    ENQ       RCBA                    Protect resource TOTAL.
               •
               •
 5.  * Process TOTAL                             Used by STASK1 and STASK2.
               •
               •
 6.            DEQ       RCBA                    Release resource TOTAL.
               •
               •
 7.  STASK2    EQU       *
               •
               •
 8.            B         STASK1                  Process TOTAL
               •
               •
 9. RCBA       RCB                               RCB for resource TOTAL
10. TOTAL      DS                                Shared resource
```

Explanation for Example 2: The resource (TOTAL) in STASK1 is protected by the resource control block named RCBA. The protection remains in effect only if every segment of coding within the partition referring to TOTAL issues the ENQ macro before executing that selection of coding and subsequently dequeues that resource with the DEQ macro. This is effectively accomplished by branching to the same physical set of code.

Note that the coding from statements 4-6 does not necessarily have to be reentrant, but you should ensure that values for constants associated with the subroutine do not have to be retained from one reference to the next, whenever the resource is used. If so, these values should be saved with the appropriate subtask and then later restored.

<u>EXAMPLE 3</u>: In this example, the subtasks again share the same resource, but use different subroutines for processing that resource.

```
 1.  MTASK      START 0
                •
                •
 2.             ATTACH ST1,SAVE=SAVE1,ECB=ECB1
                •
                •
 3.             ATTACH ST2,SAVE=SAVE2,ECB=ECB2
                •
                •
 4.  STASK1     EQU       *
                •
                •
 5.             ENQ       RCBA                   Protect resource RESRCA
                •
                •
 6.  *Update RESRCA                              Process using RESRCA
                •
                •
 7.             DEQ       RCBA                   Release resource RESRCA
                •
                •
 8.  STASK2     EQU       *
                •
                •
 9.             ENQ       RCBA                   Protect resource RESRCA
                •
                •
10.  * Update RESRCA                             Process using RESRCA
                •
                •
11.             DEQ       RCBA                   Release resource RESRCA
                •
                •
12.  RCBA       RCB                              RCB for resource RESRCA
13.  RESRCA     DS or DTF                        Shared resource
```

<u>Explanation for Example 3</u>: RESRCA can be simply an area in main storage or a file defined by a declarative macro. In either case, RESRCA is protected from subtask 2 while subtask 1 is operating on it. Thus, if all tasks enqueue and dequeue each reference to RESRCA, then RESRCA is protected during the time it takes to process instructions from that task's ENQ instruction to its DEQ instruction. This is readily apparent if RESRCA is in main storage. However, if it is a file, the record being operated upon is protected while in main storage, but it is not necessarily protected on the external storage device.

If the file is on DASD, the HOLD function should be utilized, if possible. In any such situation, the priorities of subtasks must be considered for proper operation.

POST Macro Example

The POST macro can be used by one task to inform another task of the completion of some event, or it can release a number of tasks from wait state.

```
 1.  MAINTASK   BALR      2,0
 2.             USING     *,2
                •
                •
 3.             ATTACH ST1,SAVE=AREA1,ECB=ECB1
                •
                •
 4.             ATTACH ST2,SAVE=AREA2,ECB=ECB2
                •
                •
 5.             ATTACH ST3,SAVE=AREA3,ECB=ECB3
                •
                •
 6.             WAIT      ECB1                   Wait for completion of subtask 1
 7.             DETACH SAVE=AREA1                Detach subtask 1
                •
                •
 8.             EOJ
 9.  ST1        ST        1,MTSVAR               Store address of main task save area
                •
                •
10.             WAITM ECB2,ECB3                  Wait for subtask 2 or subtask 3
11.             NI        2(1),X'7F'             Turn off WAIT bit
                •
                •
12.  ST1EOJ     L         0,MTSVAR               Get address of main task save area
13.             POST      ECB1,SAVE=(0)          POST ECB for main task
14.             WAIT      ECB1A                  WAIT to be detached
                •
                •
15.  ST2        EQU       *
                •
                •
16.  ST2A       EQU       *
                •
                •
17.             POST      ECB2                   POST ECB for subtask 1
                •
                •
18.             B         ST2A
                •
                •
19.  ST3        EQU       *
                •
                •
20.  ST3A       EQU       *
                •
                •
21.             POST      ECB3
                •
                •
22.             B         ST3A
                •
                •
23.  MTSVAR     DC        F'0'                   Save area address for main task
24.  ECB1A      DC        F'0'                   Dummy ECB for subtask 1
25.  ECB1       DC        F'0'                   ECBs for subtasks
26.  ECB2       DC        F'0'
27.  ECB3       DC        F'0'
```

Explanation for POST Macro: Subtask 1 (ST1) is dependent on input from subtask 2 (ST2) and subtask 3 (ST3). Therefore, it issues a WAITM on the ECBs for those subtasks. Notice that statement 11 resets the wait bit (set to 0) in the ECB that satisfies the wait condition. This ensures that the wait bit is off before reissuing the WAITM. (Subtask 1 is the highest priority task and, therefore, would gain control before subtask 2 and subtask 3. The result is that the WAITM is always satisfied from the first operation.) Initially, subtask 1 is placed in wait state by the WAITM. Control is then passed to subtask 2 and then to subtask 3. When either of the two subtasks has the necessary data for subtask 1, it posts its ECB that removes subtask 1 from wait state. When subtask 1 finishes its processing, it posts its ECB causing the main task to be taken out of wait state which then detaches subtask 1.

WAITM Macro Example

A task issuing the WAITM macro should ensure that if an event has the possibility of not occurring (perhaps the task posting the event was terminated), the waiting task should allow an eventual outlet. This outlet, as shown in the following example, can also wait on the terminating ECB of the task that was to perform the preferred event.

```
 1.  MAINTASK   BALR      2,0
 2.             USING     *,2
                •
                •
 3.             ATTACH ST1,SAVE=SAVE1,ECB=ECB1
                •
                •
 4.             WAITM ECB2,ECB1                 Wait for preferred or secondary event
 5.             NI        2(1),X'7F'            Turn off WAIT bit
 6.             B         4(1)                  BR to branch in vector table
                •
                •
 7.  PREVENT    EQU       *                     Main task preferred event
                •
                •
 8.  TEVENT     EQU       *                     Main task secondary event
                •
                •
 9.             EOJ                             Main task end of job
                •
                •
10.  ST1        EQU       *
                •
                •
11.             POST      ECB2                  POST completion of preferred event.
12.  ECB1       DC        F'0'                  ECB for secondary event
13.             B         TEVENT                Vector BR to secondary event
14.  ECB2       DC        F'0'                  ECB for preferred event
15.             B         PREVENT               Vector BR to preferred event
```

Explanation for WAITM Macro: In this example, the WAITM macro contains a preferred event as the first operand and a secondary event as the second operand. The preferred event is the posting of ECB2 after subtask 1 completes its calculation. If subtask 1 terminated before its calculation is completed, the supervisor posts the ATTACH macro ECB of subtask 1, ECB1, and the secondary event can satisfy the WAITM macro. In either event, after the WAITM macro is satisfied, the address of the posted ECB is contained in register 1. This address can select a problem program routine.

In this particular case, a branch instruction points to a table containing a list of ECBs with corresponding branch instructions to the routine to be given control when the ECB is posted. This table can easily be expanded to include up to a maximum of 16 ECBs (due to the WAITM format used).

STXIT AB Macro Example

The STXIT AB macro instruction establishes linkage to an abnormal termination routine for either a main task or a subtask.

```
 1.  MAINTASK   BALR     2,0
 2.             USING    *,2
 3.             STXIT    AB,MTABEND,MTABSV      Initialize AB exit
                •
                •
 4.             ATTACH ST1,SAVE=SAVE1,ECB=ECB1,ABSAVE=ST1ABSV
                •
                •
 5.             ATTACH ST2,SAVE=SAVE2,ECB=ECB2
                •
                •
 6.             ATTACH ST3,SAVE=SAVE3,ECB=ECB3
                •
                •
 7.  MTABEND    STC      0,MTABCODE             Save AB code
 8.             C        1,A(ST1ABSV)           Test if ST1ABTERM
 9.             BE       ST1ABEND               BR if YES
                •
                •
10.             CANCEL                          CANCEL for main task cancels all tasks
11.  ST1ABEND   EQU      *
                •
                •
12.             CANCEL ALL                      CANCEL ALL for subtask 1
                •
                •
13.  ST1        EQU      *
                •
                •
14.  ST2        STXIT    AB,ST2ABEND,ST2ABSV    Initialize subtask 2 AB exit
                •
                •
15.  ST2ABEND   STC      0,ST2ABCOD             Save AB code
                •
                •
16.             CANCEL                          CANCEL for subtask 2
                •
                •
                •
17.  ST3        STXIT    AB,ST3ABEND,ST2ABSV    Initialize subtask 3 AB exit
                •
                •
18.  ST3ABEND   STC      0,ST3ABCOD             Save AB code
                •
                •
19.             CANCEL
                •
                •
20.             DC       0D                     Align on doubleword boundary
21.  MTABSV     DS       9D'0'                  Main task AB save area
22.  ST1ABSV    DC       9D'0'                  Subtask 1 AB save area
23.  ST2ABSV    DS       9D'0'                  Subtask 2 AB save area
24.  ST3ABSV    DS       9D'0'                  Subtask 3 AB save area
25.  MTABCODE   DC       X'0'                   Save area for AB codes
26.  ST2ABCOD   DC       X'0'
27.  ST3ABCOD   DC       X'0'
```

Explanation for STXIT Macro: Statement 3 establishes linkage to the abnormal termination routine for the main task. Statement 4 attaches subtask 1 and indicates to the supervisor that it will use the main task's abnormal termination routine by specifying the ABSAVE parameter. Note that the main task's abnormal termination routine tests for a main task or subtask 1 abnormal termination by comparing the address in register 1 to the address of subtask 1's AB save area.

When the main task or subtask 1 cancels (CANCEL ALL), the entire partition is canceled. Subtasks 2 and 3 initialize their own abnormal termination exits because they use their own abnormal termination routines. When either subtask 2 or subtask 3 cancels, only that subtask is terminated; the other tasks within the partition continue processing.

Track Hold and Reentrant Modules Example

Although track hold applies across partitions, this example only shows two subtasks sharing the same DA file and the same DA modules. It is possible that a similar set of routines could be executing in a second partition also sharing the file with the first partition, but that partition would have its own DA module.

```
 1.  MAINTASK   START    0
                •
                •
 2.             ATTACH ST1,SAVE=AREA1,ECB=ECB1
                •
                •
 3.             ATTACH ST2,SAVE=AREA2,ECB=ECB2
                •
                •
 4.  ST1        OPEN     DAFILE1                 OPEN DA master file
                •
                •
 5.             LA       13,DASAVE1              Initialize register 13 with DA save area
 6.             READ     DAFILE1,KEY             Read and hold record
                •
                •
 7.             WAITF    DAFILE1
                •
                •
 8.             WRITE    DAFILE1,KEY             Write updated record
 9.             WAITF    DAFILE1
10.             FREE     DAFILE1                 Release track
                •
                •
11.  DAFILE1    DTFDA HOLD=YES,RDONLY=YES,...
                •
                •
12.  ST2        OPEN     DAFILE2                 OPEN DA master file.
                •
                •
13.             LA       13,DASAVE2              Initialize register 13 with DA save area
14.             READ     DAFILE2,KEY             Read and hold record
15.             WAITF    DAFILE2                 From DA master file
                •
                •
16.             WRITE    DAFILE2,KEY             Write updated record
17.             WAITF    DAFILE2
                •
                •
18.             FREE     DAFILE2                 Release track
                •
                •
19.  DAFILE2    DTFDA HOLD=YES,RDONLY=YES,...
                •
                •
20.             DAMOD HOLD=YES,RDONLY=YES,...
                •
                •
21.  DASAVE1    DS      8D'0'                    Save areas used by DAMOD
22.  DASAVE2    DS      8D'0'                    when shared and reentrant
```

Explanation for TRACK HOLD and REENTRANT Modules: Because both subtasks share the same file, HOLD=YES and RDONLY=YES must be specified in both DTFs and in the DAMOD. In addition, before any I/O operation is issued (READ, WRITE, or WAITF), register 13 must contain the address of a unique save area to store the registers used by the module. Register 13 is not altered between I/O operations executed by a given subtask, and therefore, only needs to be initialized once. If other reentrant access methods were used by the subtask, register 13 would have to be initialized for each LIOCS function.

FOPT Macro

OPERATOR COMMUNICATIONS SUPPORT (OC)

Operator Communications (OC) refers to the processing of an external interrupt by a problem program. In a multitasking environment, only the main task can communicate via the OC linkage. By specifying OC=YES, a table (OC option table) is generated within the supervisor (see Figure 1.13). When the problem program issues the STXIT macro, the address of its external interrupt routine is moved to the OC option table. The user's routine is terminated by issuing the EXIT macro. When OC=YES is specified, support is available to all partitions. Figure 1.20 illustrates a sample program using this support.

The Tape Compare Utility program requires this support. OC=YES is also required if emulator program operator services are to be requested through the INTERRUPT key.

Operational Considerations

To cause an external interrupt for the background partition, the operator presses the INTERRUPT key on the CPU console. To cause an external interrupt for the foreground partitions, the operator presses the REQUEST key on the 1052 console and, in reply to the ATTN routine statement 'READY FOR COMMUNICATIONS', types 'MSG F1' or 'MSG F2'.

INTERVAL TIMER SUPPORT (IT)

This parameter generates programming support for the hardware timer feature. The timer consists of two parts. The first part keeps track of the time-of-day and is used to time-stamp system time. The second part of the timer allows a problem program to set a time interval (via SETIME macro). By using the STXIT, EXIT, and TECB macros, a specific routine within the problem program is entered when this time interval elapses. In a multitasking environment, only the main task can set a Timer Event Control Block (TECB).

The interval timer is in addition to and separate from the time-of-day support generated by the specification of the TIMER= parameter of the CONFG macro. When interval timer support is specified (i.e., IT=BG, F1, or F2), TIMER=YES is assumed for the CONFG macro. Support is only available to one partition at a time as defined at system generation time. The TIMER command can change the assignment from one partition to another after the supervisor has been generated. QTAM requires IT=F1.

PC Option Table and OC Option Table:

	A	B
BG	a	b
F2	a	b
F1	a	b
	word	word

A

No STXIT given: a = 0
STXIT issued: a = address of the user program check (operator communications) routine
STXIT issued when the user routine is already in use: a = complement of user program check (operator communications) routine address

B

No STXIT given: b = 0
STXIT issued: address of the user save area

IT Option Table

C	D
c	d
word	word

C

No TECB or STXIT issued: c = 0
TECB issued: c = address of the timer event control block
STXIT issued: c = address of the user interval timer routine
STXIT issued when user routine is already in use: c = complement of the user interval time routine address

D

No TECB or STXIT issued: d = 0
TECB issued: d = complement of the TECB address
STXIT issued: d = address of the user save area

AB Option Table

E	F
e	f
e	f
e	f
word	word

E

No STXIT given: e = 0

STXIT issued and rtnaddr parameter passed: e=address of entry point of user's abnormal termination routine. If AP (asynchronous processing) is supported, the main task and subtasks may have the same or different AB routines. When a subtask is attached after a STXIT AB macro has been issued by the main task, the subtask will receive the AB routine address specified by the main task only if the ATTACH macro for that subtask has the ABSAVE parameter specified. The subtask can override this by issuing its own STXIT AB macro.

F

No STXIT given or no save area parameter passed: f = 0

STXIT issued and save area parameter passed: f=address of a 72-byte save area used by the supervisor to store the old PSW and general registers 0-15.

Each table address (less 8 bytes) is found in the communications region at the byte locations specified below. The labels shown identify the first byte of the corresponding table.

Table	Bytes in COMREG	Label
PC	100-101 (X'64'-'65')	PCTAB
IT	102-103 (X'66'-'67')	ITTAB
OC	104-105 (X'68'-'69')	OCTAB
AB	12-13 (X'0C'-'0D') of extension	ABTAB

Figure 1.13. Option Tables

PROGRAM CHECK SUPPORT (PC)

Program Check (PC) support generates a PC table within the supervisor (see Figure 1.13). The PC table contains the address of a user program check routine. This address is placed in the table via the STXIT macro issued by the problem program. If the STXIT PC linkage is established and a program check within this program occurs, the supervisor gives control to the user's routine instead of canceling the job being run in this partition. The support is extremely advantageous when using LIOCS (e.g., files can be closed before job termination). If a program check occurs in a routine being executed from the logical transient area (LTA), only the task associated with that routine is abnormally terminated.

In a multitasking environment each subtask and main task may have its own PC routine. A PC routine can be shared by more than one task within a partition. This can be done by issuing a STXIT macro in each task with the same routine address but with separate save areas. To successfully share the same PC routine it must be reentrant (capable of being used concurrently by two or more tasks).

Figure 1.20 shows a sample program using this support. Refer to Supervisor and I/O Macros listed in the Preface for further information on problem program macro formats and their use.

TAPE ERROR RECORDING

The three options and their system generation specifications for tape error recording are:

- Tape Error Block (TEB) by unit, TEB=n.
- Error Statistics by Tape Volume (ESTV), TEBV=(DASD,n) or TEBV=(SYSLOG,n).
- Error Volume Analysis (EVA), EVA=(r,w,n).

Any one or any combination of these options may be selected. If more than one is included, the n must be the same for each. A TEB table is generated for the TEB option. A TEBV table is generated for the ESTV and EVA options. Status information contained in these tables is shown in Figure 1.15 and 1.16. Figure 1.14 shows the I/O interrelationship.

TEB: The system generates the number of tape unit error blocks specified by n. TEB support automatically collects and writes magnetic tape unit status on SYSLOG at the end of every job utilizing these units. The n must be at least equal to the number of tape units and/or tape cartridge readers attached to the system. (TEB is the only one of the three options supporting the tape cartridge reader.) Additional TEBs should be specified for possible future expansion.

ESTV: The system generates a TEBV table with a status block and the number of error blocks specified by the n in TEBV=(DASD,n)|(SYSLOG,n). ESTV transients format and write the error records on DASD or SYSLOG each time a particular volume is ended by CLOSE, EOV, EOJ, or abnormal termination.

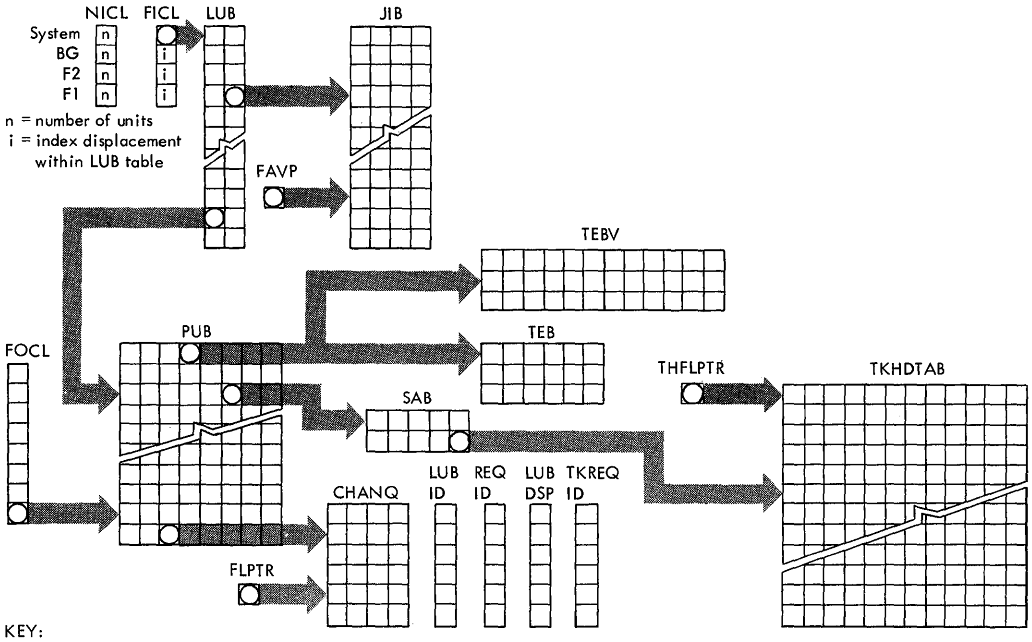
For TEBV=(DASD,n), the ESTV recorder file (ESTVFLE) must be formatted and opened for the collection of statistics using ESTVFMT, the ESTV format program. SYSREC is the system logical unit used to collect ESTV statistics. The information written on ESTVFLE may be retrieved by executing the ESTV dump program, ESTVUT.

For TEBV=(SYSLOG,n), ESTV transients format and write the data on SYSLOG and do not access any intermediate storage device.

EVA: The system generates a TEBV table and prints a message on SYSLOG when the temporary read error threshold (r) or temporary write error threshold (w) has been exceeded on a currently accessed tape volume. The number of SIOs is also included in the message. EVA can be used for both labeled and unlabeled tape volumes. Only the first four bytes of the status block portion of the TEBV table are generated if TEBV is not specified also. The status block is followed by the number of error blocks specified by the n parameter.

System Generation Guidelines

The FOPT macro checks the options in the following order: TEB, TEBV, EVA. An invalid specification for any option (n is outside the range 0-254) results in NO being assumed for that option. If TEB has been validly specified, the TEBV n will take on the value of the TEB n. If either TEB or TEBV has been validly specified, the EVA n will take on the value of the previously set n. If any n does not have the same value as an earlier valid n, an MNOTE is issued.



KEY:

NICL (Number in Class) : The first byte contains the number of system class units. The second, third, and fourth bytes contain the number of programmer class units (BG, F2, F1) (Figure 1.21).

FICL (First in Class) : The first byte points to the first system class unit in the LUB table. (Always the first LUB table entry.) The second byte points to the first programmer class unit in the LUB table BG area. The third points to the first programmer class unit in the LUB table F2 area. The fourth points to the first programmer class unit in the LUB table F1 area (Figure 1.21).

LUB (Logical Unit Block) Table : The first byte points to a PUB table entry (if the logical unit is assigned) or contains X'FF'. The second byte points to a JIB table entry or contains X'FF' (Figure 1.21).

PUB (Physical Unit Block) Table : The first two bytes contain the channel and unit address of the physical device; the third a CHANQ pointer; the fourth a TEB pointer; the fifth device type codes; the sixth a device characteristic code or a SAB pointer; the seventh the channel scheduler flag; and the eighth has the job control flag (Figure 1.30).

FOCL (First on Channel List) : The first byte points to the first PUB (highest priority) on channel zero. The next byte points to the first PUB (highest priority) on channel one, etc. A hexadecimal FF indicates the associated channel is not supported.

TEB (Tape Error Block by Unit) : One TEB is built for each tape unit at supervisor generation time if tape error statistics by unit are required (Figure 1.15).

TEBV (Tape Error Block by Volume) : One TEBV is built for each tape unit at supervisor generation time if tape error statistics by volume or error volume analysis are required (Figure 1.16).

FAVP (First Available Pointer) : A one-byte pointer to the next available JIB entry.

JIB (Job Information Block) : The first two bytes contain extent or LUB information. The third contains ownership and JIB flags. The fourth contains JIB chaining information (Figure 1.22).

CHANQ (Channel Queue) Table : The first byte contains the chain field (a pointer to the next in queue). The last three bytes contain the CCB address (Figure 1.29).

LUBID (LUB Identification) : A one-byte pointer to the LUB making the I/O request.

REQID (Requestor Identification) : A one-byte pointer to the program containing the CCB (Figure 1.29).

LUBDSP (LUB Displacement) : A one-byte value equal to the absolute LUB number (CCB byte 7).

FLPTR (Free List Pointer) : A one-byte pointer to the next free entry in the channel queue (Figure 1.29).

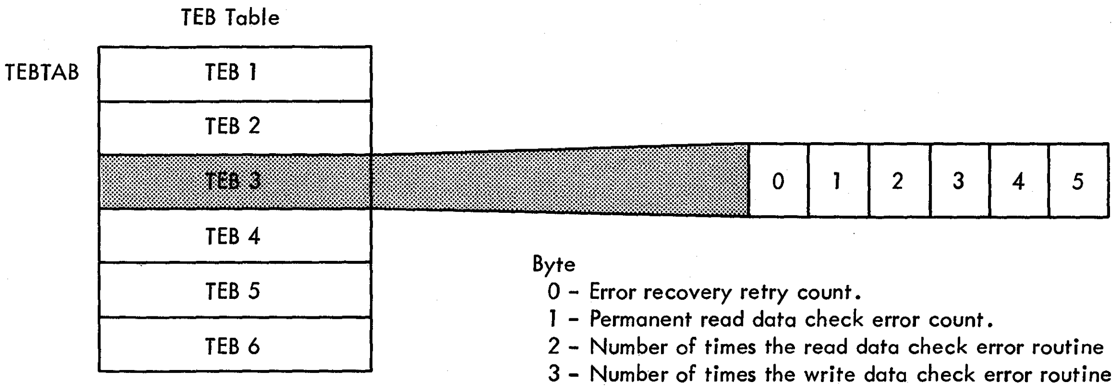
SAB (Seek Address Block) : A four-byte (BCCH) address that is the current disk address of the device plus a fifth byte that contains a Track Hold Table pointer or X'FF'. If the Track Hold function is not supported, the fifth byte contains X'00'.

TKHDTAB (Track Hold Table) : The first byte contains a pointer to the next available entry (or X'FF'); bytes 2-4 have CCB address of the requesting task; bytes 5-10 have disk address (BBCCHH) of track being held; byte 11 has key of owning track; and byte 12 has two uses: bit 0=1 means a task is waiting for the track, and bits 4-7 count the number of holds on the track. Note that the number of holds is one greater than the value of bits 4-7 of the last byte.

THFLPTR (Track Hold Free List Pointer) : A one-byte pointer to the next free entry in the Track Hold Table.

TKREQID (Track Requestor Identification) : A one-byte pointer to the PIB of the task requesting I/O.

Figure 1.14. I/O Table Interrelationship



Byte
0 - Error recovery retry count.
1 - Permanent read data check error count.
2 - Number of times the read data check error routine is entered.
3 - Number of times the write data check error routine is entered.
4 - Write skip (erase gap) count.
5 - Noise record count.

One TEB is generated for each 2400 series or 3420 magnetic tape or 2495 Tape Cartridge Reader unit if the FOPT macro contains the TEB = n parameter. Job control resets each TEB at normal or abnormal End-of-Job. An unused TEB contains HEX'FF0000000000'. A TEB is referenced from byte 3 of a magnetic tape unit PUB.

Bytes 70 and 71 (X'46'-'47') of the communications region contain the address of the TEB table entry. Label TEBTAB identifies the first byte of the table.

Figure 1.15. Tape Error Block

Decimal Displacement	Label	Byte Length	Description
	TEBVTAB		Symbolic address of the TEBV Table
(TEBV Status Block portion of TEBV Table - Note 1)			
0	TEBLEN	1	Length of TEBV Error Block (for each Error Block generated)
1	TSBLEN	1	Length of TEBV Status Block (4, 6, or 22 bytes - see Note 1)
2	EVARTH	1	EVA Read Error Threshold
3	EVAWTH	1	EVA Write Error Threshold
4	TEBSTAT	1	DASD ESTV File Status
5	TEBUDC	1	ESTV Label Update Counter
6	TEBDEV	1	Data Set Device Code
7	UPXTNT	4	Disk Address of Upper Extent of Data Set (cchh)
11	TEBRPT	1	Number of Records per Track
12	NXTESR	5	Disk Address of Next Available Space for Data Record (cchhr)
17	ESTVLABL	5	Pointer to ESTVFLE Label in VTOC (cchhr)
(TEBV Error Block Portion of TEBV Table - Note 2)			
22	TEBV	1	Status Indicator (giving status of posting and writing error conditions)
23		1	Usage Indicator (X'00'=TEBV Error Block in use; X'FF'=Error Block generated but not serving any tape unit)
24		1	Retry Counter
25		1	Permanent Read Errors
26		1	Temporary Read Errors
27		1	Temporary Write Errors
28		1	Erase Gaps
29		1	Noise Blocks
30		1	Permanent Write Errors
31		1	Clearer Actions
32		2	Number of Start I/Os
34		6	Volume Serial Number (Volume ID)
40	(Begin repeating bytes 22-39 for second TEBV Error Block)		

Note 1: The TEBV Table is composed of one Status Block and (n) Error Blocks and is addressed symbolically by TEBVTAB. The table is generated if EVA and/or ESTV are included in the system.

The size of the TEBV Status Block is determined by supervisor options in the FOPT macro at generation time:

- When EVA is chosen without ESTV, the TEBV Status Block is four bytes long (bytes 0-3), followed by TEBV Error Blocks, so that bytes 4-21 are omitted.
- When ESTV output is on SYSLOG, the TEBV Status Block is six bytes long (bytes 0-5), followed by TEBV Error Blocks, so that bytes 6-21 are omitted.
- When ESTV output is on DASD, the TEBV Status Block is 22 bytes long (bytes 0-21), followed by TEBV Error Blocks.

Note 2: The number of TEBV Error Blocks generated corresponds to the (n) parameter in the FOPTmacro for TEB, TEBV, or EVA options. A TEBV Error Block always contains 18 bytes, as shown in bytes 22-39 of this figure. The TEBV Table, therefore, is composed of one TEBV Status Block (with the length dependent upon supervisor generation options as described in Note 1), followed by (n) number of 18-byte TEBV Error Blocks.

Figure 1.16. TEBV Table Showing Status Block and Error Blocks

SEEK SEPARATE SUPPORT (SKSEP)

When DASD devices are command chained, they monopolize the channel until channel end for the device associated with the last CCW in the chain is reached. This means that the channel is unavailable for use by other partitions requiring I/O operation of other DASD devices on the channel.

The seek separation feature was designed to improve the performance of systems running under DOS. This feature enables the supervisor to separate a seek from its associated read or write so that the seek can be separately scheduled. This means that multiple seeks can be issued to devices on a channel, and the reads and writes scheduled as the seeks are completed. As this is a supervisor function, it automatically applies to programs written at any language level and/or operated in any of the three partitions. As the number of devices on a channel increases, the benefits derived from this feature increase. The implementation of this feature is such that when a seek has been issued to a device, the arm cannot again be shifted until the I/O operation that initiated the seek has been completed. In other words, arm stealing has been prevented. After seek is issued, the channel is available for scheduling any other I/O operations. In a multiprogramming environment, this feature is particularly important when the different partitions have a mix of input/output requests for a single channel with multiple direct access devices.

The seek separation capability is provided by unchaining the user's seek and by posting the seek separation bit (X'10' of the 13th byte of the user's CCB) and the "Wait for Device End" bit in the user's CCB. When START I/O is performed on the new unchained seek, channel end is immediately sent back from the control unit, thereby freeing the channel for performing seeks to other units on the channel. When the device end interrupt occurs at the completion of a seek, the seek separation and "Wait for Device End" bits are turned off, the seek is rechained to the remaining CCWs for the I/O request, and the device is not dequeued. A START I/O is now performed on this unit that has its arm already positioned at the correct cylinder.

Specifying this feature generates a Seek Address Block (SAB) within the supervisor (refer to Figure 1.14). Each DASD device has an entry in the SAB containing the current disk address for that device. Each time the user performs a seek, his seek address is compared with the entry for that device in the SAB. If they are equal, no seek separation is performed. If they are not equal, the seek address replaces the current entry in the SAB, and seek separation is performed. Each user's DASD CCW chain must start with a long seek (X'07') in order to use this feature.

System Generation Guidelines

Specifying SKSEP=YES indicates support for all DASD devices specified by the DVCGEN macro at system generation time. N is the number of DASD devices to be supported and cannot be less than the number of DASD devices specified at system generation.

PHYSICAL TRANSIENT OVERLAP SUPPORT (PTO)

Physical Transient Overlap (PTO) support allows the system to:

- Overlap I/O operations in the error recovery routines (ERP) with problem program processing.
- Overlap I/O time required to fetch user programs and system transient routines into main storage with problem program processing.

The PTO option allows other tasks to be selected when an I/O operation is being performed during one of the following:

- Fetching a phase (SVC 1)
- Fetching a logical ($$B) transient (SVC 2)
- Fetching a physical ($$A) transient (SVC 3)
- Loading a phase (SVC 4)
- Loading a phase header (SVC 23)
- Modifying the system communications region via MVCOM macro (SVC 5)
- Fetching of the ATTN routine
- Performing ERP I/O operations

For example, if an error occurs in the background partition while reading a tape record, an error recovery routine is called into the physical transient area to reread the record up to 100 times. Each reread operation requires time to read the record itself plus time to backspace the record before reading it again. Without PTO, the entire 100 rereads by the error recovery routine would not be overlapped with other processing and other tasks would not receive control during this interval. With PTO, I/O time is overlapped and available to the foreground partitions for processing until another error recovery routine or system transient routine is required.

Another advantage of PTO involves fetch I/O time. Fetching requires one or more searches of the library directory on SYSRES, reading of the directory into the physical transient area, and searching for and reading one or more records of the program or transient into the proper area of main storage. This involves several disk I/O operations. PTO allows the fetch I/O time to be overlapped with processing by any partition requiring CPU time (in priority sequence) until another fetch or other use of the physical transient area is required.

CONSOLE BUFFERING SUPPORT (CBF)

Previously, a system component could hold up the system and degrade throughput performance whenever an I/O operation was issued to the console typewriter (1052), followed by a WAIT. The console buffering option (CBF) alleviates this situation by queuing write operations to a 1052, 3210, or 3215 and returning control immediately to the routine that issued the write with the WAIT bit posted. The user's WAIT is satisfied immediately, rather than at actual I/O completion. Each message is assigned a buffer containing the message itself and control blocks for I/O interrupt handling (Figure 1.17). The system writes the message from the buffer on the 1052, 3210, or 3215 as soon as possible while user-processing continues.

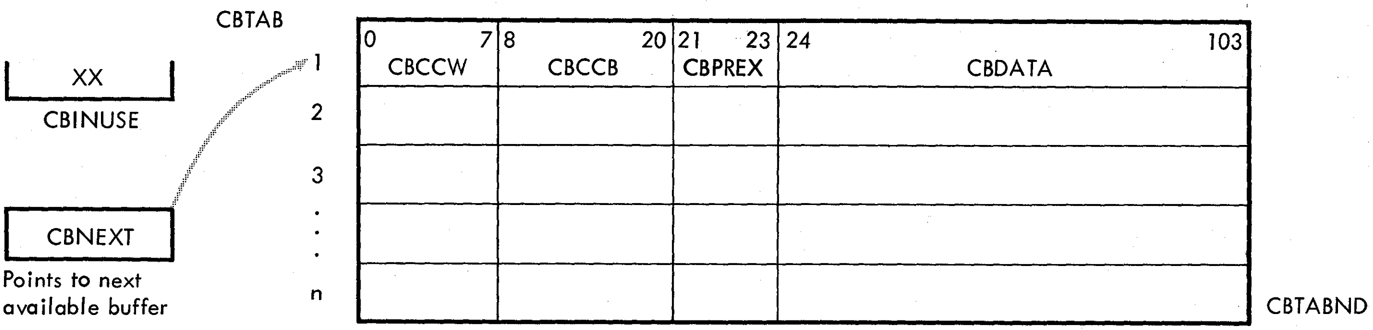
The CBF facility is used only for write operations which require no user error handling. Only those messages that meet the following criteria are buffered:

1. Messages cannot exceed 80 characters.
2. Data chaining and command chaining are not used.
3. The CCB associated with the operations does not request any sense information (CCB+12 set to X'20') and/or does not have either "accept unrecoverable I/O error", "post at device end", or "user error routine" bits on (CCB+2 set to X'15' or any combination of these). In other words, no error checking is required.
4. The CCW must have either an X'01' or X'09' command code (i.e., only a WRITE operation).
5. For user messages, the CCW and the message must reside within the partition issuing the write operation.

If these conditions are not met, the console operation is handled in a normal manner.

You have the option of specifying the number of buffers required for queuing messages, but the total may not exceed 50. If a buffer is not available, your WAIT is not satisfied until a buffer is freed. At least one buffer should be specified for each partition or task issuing messages so that buffers are available and the task can continue processing while the message is being printed. Job control often issues several console messages in succession; therefore several additional buffers should be specified for partitions that use job control frequently.

Another factor to be considered in the selection of the CBF option is extra channel usage. Specification of the CBF option and selection of the CHANQ default (6 channel queue entries), results in the number of buffers specified being added to the CHANQ default. However, when both the CBF and CHANQ options are specified, it is recommended that the number of CHANQ entries desired be increased by the number of buffers specified. Otherwise, the number of entries generated in the channel queue will be less than desired.



The Buffer Table is a 104 multiplied-by-n byte area of main storage, where n is 1-50. Each buffer entry in the table is 104 bytes. CBNEXT is a halfword constant that points to the next available buffer entry. It is initialized with the address of CBTAB and is incremented by 104 every time a buffer is used, so that it points to the next entry. When its value becomes greater than CBTABND, it is reinitialized with the value CBTAB.

CBINUSE is a one-byte counter that contains the number of entries currently in use. It is incremented whenever a buffer is used and is decremented at dequeue time when the buffer becomes free.

Each buffer entry contains the following fields:

Displacement: 0 CCW moved from requestor core. The data address portion of the CCW is modified to point to the data portion of the buffer.

8 CCB moved from requestor core. The CCW address in the CCB is modified to point to the CCW in the buffer.

21 Prefix. SYSLOG ID moved from problem program PIB. The prefix is printed with the data to identify BG, F1, or F2 partitions.

24 Data moved from requestor core.

Figure 1.17. Console Buffering Table and Work Areas

INDEPENDENT DIRECTORY READ-IN AREA (IDRA)

An independent directory read-in area (IDRA) can be included if the option is specified at system generation time. The funcion of this area is to enhance performance by reducing contention for the physical transient area (PTA) on fetches from the core image library. During the time the physical transient area is busy, fetches from other partitions can be processed.

System Generation Considerations

$$\text{IDRA}=\left\{\begin{array}{l}\underline{\text{NO}}\\ \text{YES}\end{array}\right\}$$

The YES option causes generation of the IDRA area in the supervisor. IDRA requires MPS=YES or BJF, and PTO=YES.

If IDRA=NO, the core image library directories are loaded and scanned in the PTA. If IDRA=YES, the core image library directories are loaded and scanned in the IDRA. The IDRA is used for all supervisor calls that require reading of directories.

COMMAND CHAINING SUPPORT (CCHAIN)

Command chaining support (CCHAIN=YES) allows the DOS error routines to retry an I/O operation starting with the last CCW executed rather than at the beginning of the chain. Under normal error recovery procedures, the entire CCW chain is reexecuted. To utilize the CCHAIN support, you must indicate in the CCB macro instruction that you want the error procedures to retry the I/O operation starting with the CCW in error by setting byte 3, bit 7 on (third operand of CCB=X'0001'). In addition, bytes 8-11 of the CCB must be initialized with the address of the first CCW in the chain each time the chain is executed. See Figure 1.18 for an example using CCHAIN support.

```
BEGIN       START  0

            BALR   4,0

            USING  *,4
            •
            •
DO          LA     1,READ                Initializes Bytes 8-11 of CCB
                                         with address of first CCW in
            ST     1,READCCB+8           chain each time chain is executed.

            EXCP   READCCB

            WAIT   READCCB
            •
            •
            BCT    3,DO
            •
            •
READCCB     CCB    SYS004,READ, X'0001'  Indicates Command Chain Retry.

READ        CCW    2,DATA,X'60',100

            CCW    2,DATA+100,X'60',100

            CCW    2,DATA+200,X'20',100
```

Figure 1.18. Example Using CCHAIN Support

CCHAIN=YES must be specified if data or command chaining of IBM 2495 Tape Cartridge Reader input is performed.

TRACK HOLD SUPPORT (TRKHLD)

The track hold feature allows tasks within the same partition or tasks outside of a partition to share DASD data files when using the direct access method or the sequential method of file organization. This track protection facility can be used with or without multitasking to provide protection between partitions. By definition, track protection means a DASD track that is being modified by some task in main storage is prevented from being accessed by another task in that partition or in any other partition.

Any programs using DTFSD/SDMODXX, DTFIS/ISMOD, or DTFDA/DAMOD can use the track protection macros when performing the following functions.

- DTFIS - all functions except LOAD
- DTFSD - updating with work files
- DTFSD - updating without work files
- DTFSD - other functions with work files
- DTFDA - all functions

The actual holding of a track is a combination of supervisor (PIOCS) and data management (LIOCS). Track hold involves the actual request that a track be held and the request to free a track. The hold request is generated if you specify parameters in the DTFs and logic modules for the program(s) involved.

Supervisor Considerations

At system generation time, you must specify the number of unique tracks that can be held at any one time (255 maximum and 10 minimum). A table for the number of entries specified is built into the supervisor for storing the necessary information required for track protection.

The maximum number of holds (without intervening frees) that a task can issue for a given track is 16. If more than 16 holds are attempted, the task is canceled.

If the DASD channel program does not start with a long seek, track hold is not implemented.

If multiple track search operations are issued, only the first specified track is held but it is not necessarily the same track on which the search is satisfied. In this case, the track on which the search is satisfied is not protected.

If a task requests a track already held by another task, the requesting task is placed in wait state (until the track becomes available). In addition, if the task requests a hold on the 256th unique track or exceeds the limit specified at system generation time, the requesting task is placed in wait state until a previously held track is freed. In either of these cases, especially in the first situation which has a greater chance of occurring than the latter, make sure that the following conditions do not occur.

1. If more than one track is being held, it is possible for a task to put the entire system into wait state. This occurs if one task is waiting for a track that is already held by another task. In the first case, if task A holds track 1 on drive 1 and attempts to hold track 2 on drive 2 while task B is holding it, task A is placed in wait state until task B frees that track. Then, if task B attempts to hold track 1 on drive 1, it enters the wait state until task A frees that track. Because task A is already in wait state, neither task ever regains control. If they are the only two tasks running in the system, the entire system is in wait state.
2. A similar situation can occur if two or more tasks attempt to hold more tracks than the maximum number allowed. For example, assume a maximum of 20 unique tracks can be held by the system. Task A issues 15 DASD I/O operations and task B issues 15 DASD I/O operations (altogether requiring track hold for 30 unique tracks). If both task A and task B are alternately issuing DASD I/O operations, then when each task issues its 11th track hold (at this point both task A and task B have held the maximum of 20 unique tracks), it is placed into wait state. Again, neither will regain control, and if they are the only two tasks running in the system, the entire system is in wait state.

In both examples, if the operator is not aware of what has occurred, he may decide to cancel the tasks. If the lockout is between two subtasks within a partition, the partition is canceled. If the lockout is between two partitions, one partition is canceled, its tracks are freed and the second partition is removed from wait state.

When a task issues a free for a particular track, one of two conditions occurs:

1. If a task has more than one hold on the track (16 maximum), a counter associated with that track is decreased by one, the task retains its hold on the track and control is returned to the task issuing the free.
2. If the task has only one hold on the track, the track is freed by removing its entry from the supervisor table. Any tasks waiting for the track or an available entry in the table are removed from wait state. The task with the highest priority gains control of that track or puts a hold on another track.

Track hold only occurs for DOS-supported DASDs. Otherwise, the I/O operation is issued, but no track-hold function is involved. (This is true for any device, including non-DASDs.)

The following conditions cause cancelation of a task:

1. A free was issued to a non-DASD type or a nonsupported DASD,
2. Holds were not issued to that track,
3. A task attempts to free a track held by another task.

In the last case, when the main task detaches a subtask, any track held by that subtask is freed.

LIOCS Considerations

Track hold is only supported for DTFIS, DTFSD and DTFDA files. Support of the track hold feature requires specification of a HOLD parameter in both the DTFIS/DTFDA/DTFSD and the ISMOD/DAMOD/SDMOD macros. In addition, a FREE macro must be issued for all DTFDA or DTFIS files and for DTFSD work files without update.

A maximum of 16 holds can be applied to one track. A free issued for a track with more than one hold decrements the track hold count, but does not release the track. A free issued for a track not held, terminates the task.

The track hold functions can be used in five specific situations:

1. Using DTFSD for updating with work files (via the WRITE macro).
2. Using DTFSD for updating without work files (via the PUT macro).
3. Using DTFSD with work files, when no WRITE is issued for updating.
4. For all DTFDA files.
5. For all DTFIS files except for LOAD.

In the first two situations, the tracks being held are freed automatically by the system. For the last three situations, the task must issue the FREE macro instruction for each hold placed on a track by the READ macro. It should be noted that for DTFDA, a FREE macro need not be given when only WRITE macros are issued. If HOLD=YES and ERREXT=YES, you must issue the ERET macro to return to the ISAM module to free any held tracks.

System Generation Guidelines

TRKHLD=n

The maximum number of tracks that can be held at one time is 255. The default is 10 if n is an invalid parameter (nonnumeric or outside the range 1-255). MPS=YES or BJF must be specified if TRKHLD=n. If a task attempts to hold an additional track after the maximum has been reached, the task goes into the wait state until a previously held track has been freed.

ABNORMAL TERMINATION SUPPORT (AB)

Abnormal termination exits are available for main tasks and/or subtasks, allowing you to gain control before an abnormal condition removes the task from the system. For example, in the abnormal termination routine, you can close your files. This function is provided by the AB operand of the STXIT macro. See Supervisor and I/O Macros listed in the Preface for detailed information on the format and use of the STXIT macro.

STXIT Macro Considerations

If an IT (Interval Timer) condition occurs while executing the OC (Operator Communication) or PC (Program Check) routines, these routines should not use the same save area (see Figure 1.19).

Routine Being Processed	Condition Occurring			
	AB	IT	OC	PC
AB	T	I	I	I
IT	S	I	H	H
OC	S	H	$I_b E_f$	H
PC	S	H	H	T

E_f- Error message issued in foreground program and control returns to interrupted OC routine.

H - Condition honored. When processing of new routine completes, control returns to interrupted routine.

I_b- Condition ignored for all partitions.

I - Interrupt ignored in a background partition.

S - Execution of the routine being processed is suspended and control is transferred to the AB routine.

T - Job abnormally terminated. If AB routine is present and it has not been interrupted by itself, its exit is taken; otherwise, a system abnormal termination occurs.

Figure 1.19. Processing of STXIT Conditions

When a supervisor is not generated to handle the requested facility, the task is abnormally terminated if a STXIT macro is used. This also applies to a program that requests the timer interrupt and has not been allocated the timer.

If an AB or PC condition occurs and linkage has not been established for a main task abnormal termination routine, the partition is abnormally terminated. However, if the AB or PC condition occurs in a subtask without exit linkage established, only the subtask is terminated.

Only one main task at a time can use the STXIT IT macro. The partition of the main task is specified at system generation time but can be changed by the operator with the TIMER command. There are two distinct methods for using the STXIT IT macro. Only one method can be used at a time.

1. The first method allows the main task to set the timer and enter a routine when the time elapses. The SETIME, STXIT, and EXIT macros are used for this. However, only the main task of the partition owning the timer can issue these instructions.

2. The second method allows a given routine to be performed at timed intervals. The time set is a real time interval, and is not stopped or adjusted when the task using the timer does not have control. The SETIME, TECB, and WAIT macros are used. However, if multitasking, only one task within the partition can use the method and the WAIT on the TECB must appear in that task. Consideration should be given to the priority of the task assigned to process interval timer interrupts. Subtasks can use this method.

Figure 1.20 contains a coding example using STXIT support.

```
                                                                                        PAGE    1

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                              FDOS CL3-5 10/30/69

000000                               1 STXEXAM  START 0
000000 0520                          2          BALR  2,0
000002                               3          USING *,2
                                     4          PRINT NOGEN
                                     5          STXIT PC,PRGCK,PCSAVE       ACTIVATE USERS PROG CHECK RTN
                                    11          STXIT OC,COMMUN,OCSAVE      ACTIVATE USERS OPER.COMM. RTN
                                    17 GOAGAIN  SETIME 10,TIMEBLK           SET INTERVAL TIMER TO 10 SECONDS
                                    23 PRINT    EXCP  CCB1                  PRINT TIMER BEGINNING MESSAGE
                                    27          WAIT  CCB1
                                    33          WAIT  TIMEBLK               WAIT FOR TIMER  INTERRUPT
                                    39          EXCP  CCB2                  PRINT TIMEOUT MESSAGE
                                    43          WAIT  CCB2
00005A 9263 2058      0005A         49 CHK      MVI   *,99                  THIS WILL CAUSE A PROG CHECK  THE
                                    50 *                                    NEXT TIME THROUGH THIS RTN
00005E 47F0 2014            00016   51 LOOP     B     GOAGAIN
                                    52 PRGCK    EXCP  CCB3                  PRINT PROGRAM CHECK MESSAGE
                           (See Note below)
                                    56          WAIT  CCB3
                                    62          EXIT  PC                    RETURN FROM PROGRAM CHECK ROUTINE
                                    65 COMMUN   EXCP  CCB4                  PRINT EXTERNAL INTERRUPT MESSAGE
                           (See Note below)
                                    69          WAIT  CCB4
                                    75          STXIT PC                    DEACTIVATE PROGRAM CHECK EXIT
                                    80          STXIT OC                    DEACTIVATE EXTERNAL INTERRUPT EXIT
                                    85          EXIT  OC                    RETURN FROM EXTERNAL INTERRUPT RTN
                                    88 DONE     EOJ
00009C                              91 PCSAVE   DS    CL72
0000E4                              92 OCSAVE   DS    CL72
                                    93 TIMEBLK  TECB
                                    96 CCB1     CCB   SYS000,CCW1
                                   107 CCB2     CCB   SYS000,CCW2
                                   118 CCB3     CCB   SYS000,CCW3
                                   129 CCB4     CCB   SYS000,CCW4
000170 0900019000000064            140 CCW1     CCW   9,MSG1,X'00',100
000178 090001F400000064            141 CCW2     CCW   9,MSG2,X'00',100
000180 0900025800000064            142 CCW3     CCW   9,MSG3,X'00',100
000188 090002BC00000064            143 CCW4     CCW   9,MSG4,X'00',100
                                   144 MSG1     DC    CL100'THE TIME HAS JUST BEEN SET TO A VALUE OF TEN SECONO
000190 E3C8C540E3C9D4C5                               DS,THIS LINE IS PRINTING WHILE THE TIMER IS RUNNING.'
                                   145 MSG2     DC    CL100'THIS LINE HAS PRINTED AS A RESULT OF A TIMER INTERX
0001F4 E3C8C9E240D3C9D5                               RUPT IE.THE TIMER VALUE HAS GONE THROUGH ZERO.'
                                   146 MSG3     DC    CL100'THIS LINE HAS PRINTED DUE TO THE FORCED PROGRAM CHX
000258 E3C8C9E240D3C9D5                               ECK  AT LABEL *CHK*.'
                                   147 MSG4     DC    CL100'AS A RESULT OF THE INTERRUPT KEY THIS PROGRAM HAS X
0002BC C1E240C140D9C5E2                               GONE TO NORMAL END OF JOB.'
000000                             148          END   STXEXAM
000320 0000009C                    149                =A(PCSAVE)
000324 00000062                    150                =A(PRGCK)
000328 000000E4                    151                =A(OCSAVE)
00032C 00000078                    152                =A(COMMUN)
000330 0000000A                    153                =A(10)
000334 00012C00                    154                =F'76800'
000338 0000012C                    155                =A(TIMEBLK)

                                                                                        PAGE    2

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                              FDOS CL3-5 10/30/69

00033C 00000130                    156                =A(CCB1)
000340 00000140                    157                =A(CCB2)
000344 00000150                    158                =A(CCB3)
000348 00000160                    159                =A(CCB4)
```

Note: If LIOCS or other CSECTS that might alter the base register are used, then a base register must be established as the first step in the interrupt routine. For example:

```
BALR   3,0
USING  *,3
```

Figure 1.20. STXIT Sample Program

MULTIPLE WAIT SUPPORT (WAITM)

The multiple wait facility allows a task to be put into the wait state until one of a number of events occur before proceeding. Control passes to a task when one of the ECBs specified in the WAITM macro has been posted. The multiple wait facility can be used without specifying AP=YES. However, AP=YES cannot be used without WAITM=YES being specified.

An Event Control Block (ECB) is a task communications block written by the problem programmer (see Figure 1.11). When a subtask is successfully attached, byte 2 bits 0 and 1 are set to 0. When the subtask terminates, the supervisor posts (sets a bit to 1) byte 2 bit 0 of the ECB after processing of the abnormal termination routine. In addition, byte 2 bit 1 is posted if the subtask abnormally terminates; that is, if task termination is not the result of issuing the CANCEL, DETACH, DUMP, or EOJ macro instructions. See _Supervisor and I/O Macros_ listed in the _Preface_ for more detailed information.

Multiple Wait Considerations

When control passes to the waiting task, its register 1 points to the address of the ECB that had byte 2, bit 0 set.

The user may use CCBs or TECBs in place of ECBs. This is possible because the posting of byte 2, bit 0 occurs upon their event completions. BTAM/QTAM ECBs, QTAM control blocks, and RCBs cannot be waited on because their format would never satisfy a WAITM, i.e., byte 2, bit 0 will never be posted. When waiting on only one ECB, the WAIT macro can be used.

When using WAITM, a provision should be made for ultimate outlet if none of the events being waited on occur. For example, an abnormal ending of all the tasks on which a multiple wait was dependent would never allow the multiple wait to be satisfied.

The user may specify a preferred event. Consider the following example:

```
        WAITM   ECBA, ECBB, ECBC.
```

ECBA is considered the preferred event because it appears first on the list. For the same reason, ECBB is considered the second most preferable event. The user may use this to his advantage in task synchronization. The system checks the ECBs in the order written in the WAITM macro.

DASD FILE PROTECT SUPPORT (DASDFP)

The DASD file protect facility provides read/write file protection for DASDs. All seeks issued by a problem program are monitored by the supervisor to ensure that the seeks do not stray outside of limits that were validated at file open time. (This could occur during the loading of a direct access file.) DASDFP functions on a cylinder basis for the 2311, 2314, and 2319, and on a strip basis on the 2321. DASDFP routines are not executed when the current PSW has a storage protect key of zero. This occurs for supervisor, job control, and transient routines ($$A, $$B, $$R).

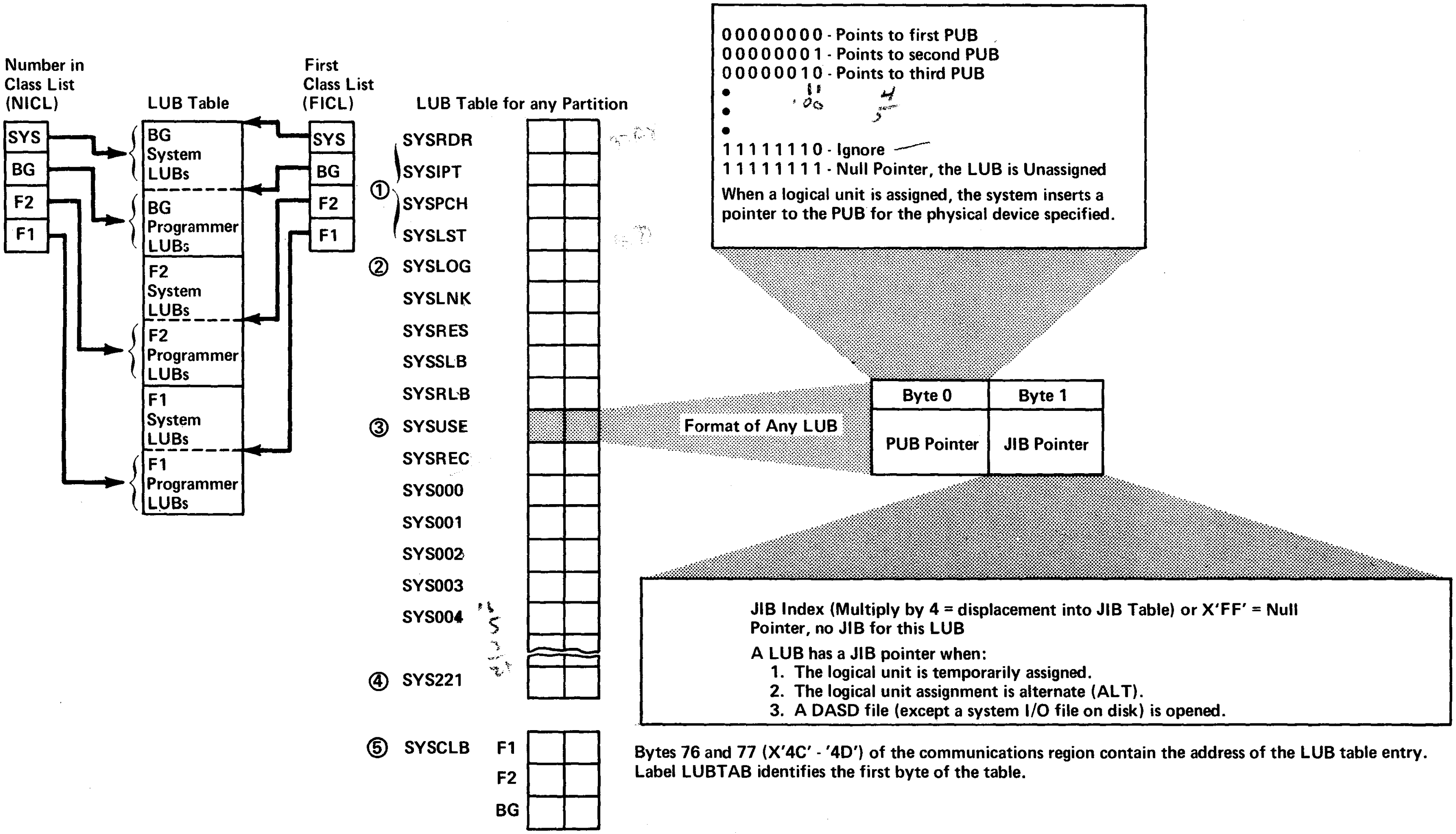
Mode of Operation

Job control or the foreground initiator reads the DLBL/EXTENT (or VOL/DLAB/XTENT) control commands or statements from the input job stream. This information is reformatted and written on the label information cylinder on SYSRES as one or more DASD records. Each of these DASD records is preceded by a key created from the filename (DTF name) entry in the DLBL/VOL command or statement.

When the problem program opens the file (DTF), the OPEN transient routine extracts an 8-byte alphameric constant from the generated DTF table. This constant is called the filename, and is an exact replica of the DTF name.

The OPEN transient routine searches the label information cylinder on SYSRES to find a key equal to this DTF filename constant. When it finds a match, it reads the data portion of this job control DASD record into the label save area in main storage.

The extent information is extracted from this job control DASD record, and is stored in the JIB table. A 2311/2314/2319 extent requires one JIB entry; a 2321 extent requires two JIB entries (see Figures 1.21 and 1.22).



Bytes 76 and 77 (X'4C' - '4D') of the communications region contain the address of the LUB table entry. Label LUBTAB identifies the first byte of the table.

Figure 1.21. Logical Unit Block (LUB) Table

① When in Single Program Initiation mode (Foreground 1 or 2): Must be unit record device and can be referenced by the program.

② When in Single Program Initiation mode (Foreground 1 or 2): Can be referenced by the program.

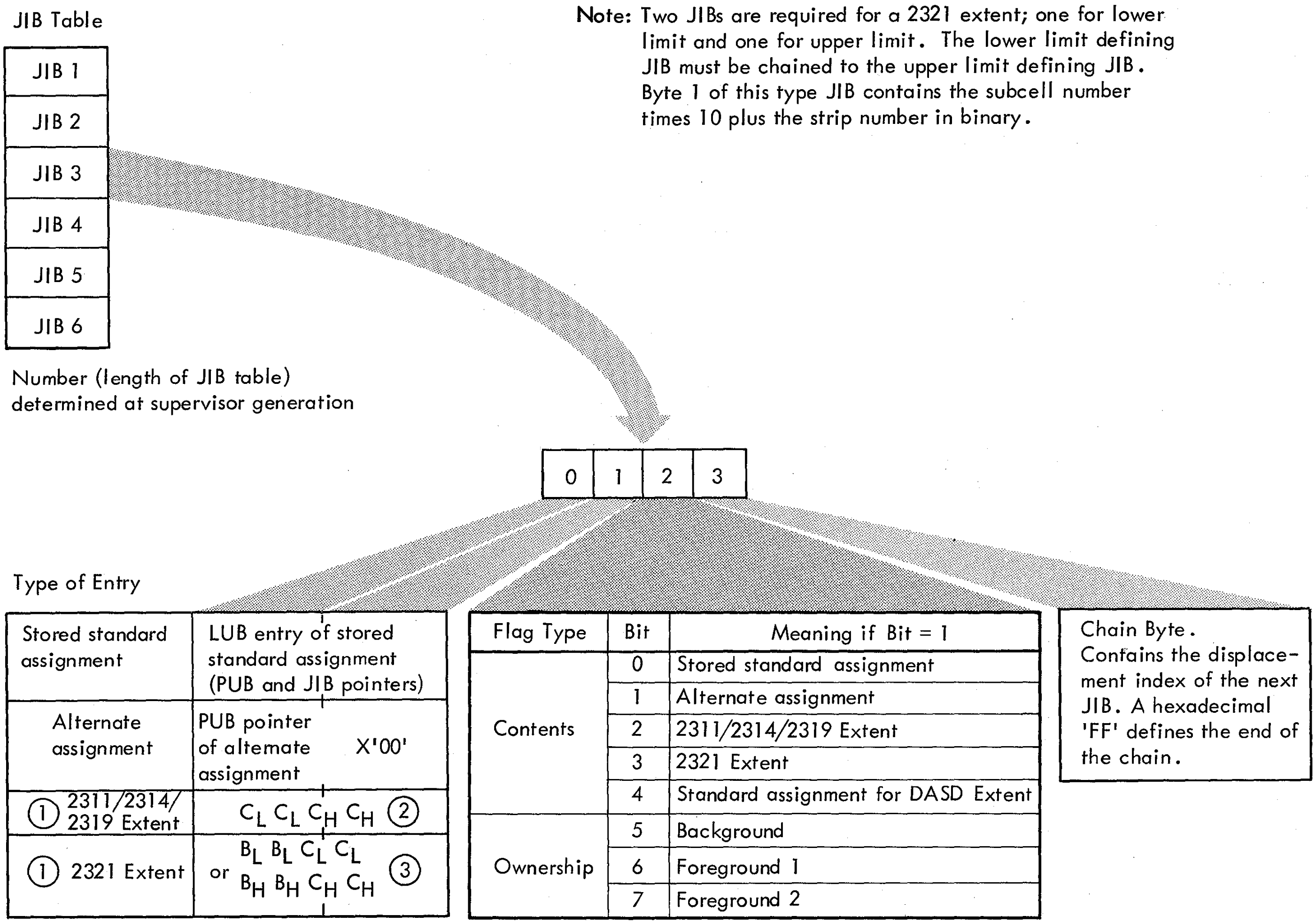
③ SYSUSE may be called SYSCTL in error recovery messages.

④ The maximum number of programmer logical units in the system is 222 if MPS = BJF, or 244 if MPS = YES or NO.

⑤ The SYSCLB (Private Core Image Library) LUB entry functions the same as other LUB entries, but is not part of the LUB Table. To locate the SYSCLB LUB in supervisor, perform the following steps:

1. Divide the PIK by 8.
2. Subtract the result in step 1 from the address of the PIB extension block.
3. If option AP = YES, the result of step 2 is the location of SYSCLB LUB. If option AP = NO, add 16 (for the all-bound PIBX) to the result of step 2.

Note: Two JIBs are required for a 2321 extent; one for lower limit and one for upper limit. The lower limit defining JIB must be chained to the upper limit defining JIB. Byte 1 of this type JIB contains the subcell number times 10 plus the strip number in binary.



Stored standard assignment	LUB entry of stored standard assignment (PUB and JIB pointers)
Alternate assignment	PUB pointer of alternate assignment X'00'
① 2311/2314/2319 Extent	$C_L\ C_L\ C_H\ C_H$ ②
① 2321 Extent	$B_L\ B_L\ C_L\ C_L$ or $B_H\ B_H\ C_H\ C_H$ ③

Flag Type	Bit	Meaning if Bit = 1
Contents	0	Stored standard assignment
	1	Alternate assignment
	2	2311/2314/2319 Extent
	3	2321 Extent
	4	Standard assignment for DASD Extent
Ownership	5	Background
	6	Foreground 1
	7	Foreground 2

Chain Byte. Contains the displacement index of the next JIB. A hexadecimal 'FF' defines the end of the chain.

① Only when file-protect on DASD

② Lower Cylinder
Upper Cylinder

③ Cell or combined sub-cell and strip

Bytes 68-69 (X'44'-'45') of the communications region contain the address of the JIB table entry. Label JIBTAB identifies the first byte of the table.

Figure 1.22. Job Information Block (JIB) Table

An EXTENT/VOL command or statement names the symbolic unit containing the extent. The $$BOFLPT OPEN transient stores each extent into a JIB entry that is related or linked to a specific symbolic unit.

When the problem program requests an I/O operation on DASD, the supervisor extracts the symbolic unit and the seek channel command word (CCW) from the command control block (CCB). The current DASD seek address is compared to the extent limits stored in the JIB table for this specific symbolic unit. If the DASD seek address falls outside the range of extents, the job is canceled.

For a 2311/2314/2319 extent, the upper and lower cylinder numbers comprise the protection boundaries for the named symbolic unit. A 2321 is only protected on strip boundaries.

Note: This is a programmed check to determine if the problem program is trying to access data outside the allowed cylinder or strip limits.

Next, the supervisor builds and executes a small channel program containing three commands: a seek that is identical to the problem program seek, a Set File Mask that prevents any other long seeks (X'07') from being executed, and a TIC that transfers control to the command following the problem program's seek (see Figure 1.23).

Note: This is a hardware check to prevent the problem program's channel from moving the read/write head outside the cylinder or strip limits that were validated by the programmed check.

DASDFP provides file protection on the basis of symbolic unit. It does not provide protection by access method, file, or DTF. However, if each DTF in a problem program is assigned to a different symbolic unit, file protection can prevent one DTF from accessing the data belonging to another DTF.

Also, DASDFP does not prevent file contention between partitions. Thus, more than one partition may access the same file at the same time (and both partitions may attempt to update the same record simultaneously). When using DASDFP, follow these guidelines:

1. For complete protection, files should begin and end on cylinder/strip boundaries.
2. File protection is ensured only if the DASD labels involved are unexpired.
3. In any one program, each DTF should use a different symbolic unit, even if the files reside on the same physical volume.
4. The system residence file must reside on a protected channel. If it does not reside on a protected channel, you will not be able to IPL your system.

System Generation Guidelines

$$\text{DASDFP=}\left\{\begin{array}{l}\underline{\text{NO}}\\ \text{(n,n,}\left\{\begin{array}{l}2311\\2314\\2321\end{array}\right\}\text{)}\end{array}\right\}$$

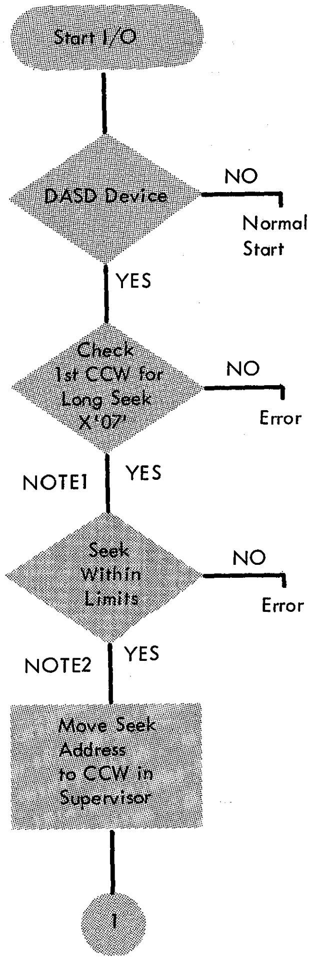
n, n specifies the range of channels on which DASD may be attached. If either 2311 or 2314/2319 is specified, protection for both is provided. If 2321 is specified, 2311, 2314, 2319, and 2321 are supported.

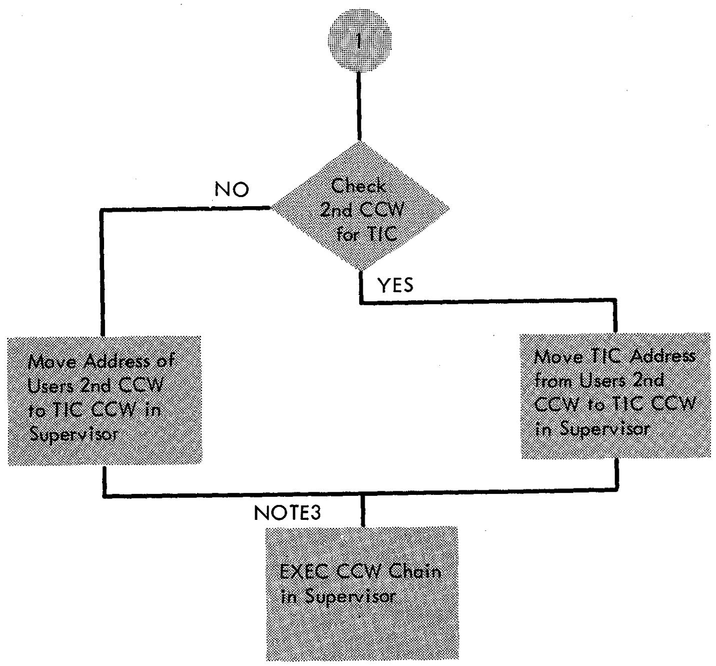
If the parameter for a 2311, 2314 or 2321 is omitted, both 2311 and 2314/2319 protection is assumed.

Example:

DASDFP=(1,3,2321)

Protection is available on channels 1, 2, and 3 for 2321s, 2311s, and 2314s/2319s.





NOTE1. Check JIB for this file.

NOTE2. Supervisor moves user's seek address to a CCW chain in supervisor.

NOTE3. The following CCW's are in the supervisor, one set for each channel that has DASD file protect.
07,User seek address,X'60',6
31,File mask,X'40',1
08,User CCW (2nd or TIC address from 2nd),X'40',0

Figure 1.23. DASD File Protect Logic Flow

SYSTEM FILES ON DISK SUPPORT (SYSFIL)

In systems with at least 24K bytes of main storage, the system logical units SYSRDR, SYSIPT, SYSIN, SYSLST and/or SYSPCH may be assigned to an extent on 2311 or 2314/2319 disk storage.

The system files when used on disk are supported by use of the ASSGN and CLOSE job control statements, and by the supervisor channel scheduler routines.

Job control, via the standard ASSGN statement, opens the file and initializes the disk information block (DIB table) within the supervisor. Figure 1.24 shows the DIB table (also, see Figure 1.21).

Each time a problem program requests I/O on a system logical unit, the supervisor checks the DIB table for a valid seek address. (Note that the job information block is not used for system files on disk). After each successful access to the file, the supervisor updates the current address field in the affected DIB.

	Current Address										End Address						R	U.L.	L.L.		R.C.		Reserved	
SYSLNK	B	B	C	C	H	H	R	P	This area not used for SYSLNK DIB															
SYSIN	B	B	C	C	H	H	R	K	D	D	B	B	C	C	H	H	X	H	H	*	XX	XX		
SYSPCH																				*				
SYSLST																				*				
Number of Bytes	7							3			6						1	1	1	1	2		2	

KEY: Current Address: The next address to be used (for both input and output).
End Address : The last address within the limits of the extent.
R : Maximum number of records per track.
U. L. : Upper head limit
L. L. : Lower head limit
R. C. : Record Count - residual capacity for beginning of operator notification. This is set at system generation time with the SYSFIL parameter, or after IPL with the SET statement (RCLST and/or RCPCH operands). A warning message is issued by job control after end-of-job step when the minimum number of remaining records has been reached or exceeded during the previous job.
P : Starting cylinder of private core image library, if assigned.
KDD : Key and data length for the symbolic device.

KDD for SYSIN = X'000050'
KDD for SYSPCH = X'000051'
KDD for SYSLST = X'000079'

Bytes 96 and 97 (X'60' - '61') of the communications region contain the address of the SYSLNK entry. Label DSKPOS identifies the first byte of the table.

Figure 1.24. Disk Information Block (DIB) Table

When a problem program issues an open to a system file that is currently assigned to disk, the LIOCS open routines transfer the extent information to the DTF table from the DIB instead of from the file label in the volume table of contents. This causes the current address field in the DIB to be used as the beginning extent for the DTF of the file being opened.

When a problem program closes a system file that is assigned to disk, the LIOCS close routine posts the file closed and does not disturb the DIB.

Job control, via the CLOSE statement, closes the system files on disk and deactivates the DIB.

Considerations When Using System Files on Disk

1. The system logical units of SYSIPT, SYSRDR, SYSIN (SYSIN is both SYSIPT and SYSRDR), SYSPCH, and SYSLST can be assigned to disk for any batched job partition.

2. Record lengths for these assignments are:

    - 80 for SYSRDR
    - 80 or 81 for SYSIPT on the IBM 2311 (80 for the IBM 2314)
    - 121 for SYSLST
    - 81 for SYSPCH

3. The creation of files for use as system input, and the printing or punching of system output files is done by user-written programs. (Utility programs are available to simplify this.)

4. Predefined symbolic filenames have been established for all system files that can be assigned to disk. The filenames (used in the DLBL statement and associated with the SYSxxx entry of the accompanying EXTENT statement) are IJSYSIN for SYSRDR, SYSIPT, or SYSIN; IJSYSPH for SYSPCH; IJSYSLS for SYSLST.

5. When SYSRDR and SYSIPT are both assigned to disk, they must reside in the same extent and be referred to as SYSIN. The filename specified in the DLBL/VOL command or statement must be IJSYSIN.

6. Because SYSPCH and SYSLST have different size records, SYSOUT cannot be assigned to disk.

7. Job control issues operator warning messages when the area assigned to disk approaches a predefined residual capacity.

8. The residual capacity for operator warning is established at supervisor generation time with the FOPT macro (SYSFIL parameter). These values can be changed after IPL by the SET job control command (RCLST, RCPCH operands).

System Generation Guidelines

$$\text{SYSFIL=}\left\{\begin{array}{l}\underline{\text{NO}}\\(\left\{\begin{array}{l}2311\\2314\end{array}\right\}[,n_1,n_2])\end{array}\right\}$$

Specification of either 2311 or 2314 gives support for 2311/2314/2319. If MPS=BJF in the SUPVR macro, this parameter supports foreground logical units when running in batched mode. If the emulator program parameter SYSIO=222 or SYSIO=333 is indicated, a value must be specified for SYSFIL.

n_1 = residual capacity for beginning of operator notification when SYSLST is assigned to disk. $100<n_1<65535$. If n_1 is omitted, 1000 is assumed.

n_2 = residual capacity for beginning of operator notification when SYSPCH is assigned to disk. $100<n_2<65535$. If n_2 is omitted, 1000 is assumed.

PRIVATE CORE IMAGE LIBRARIES

The ability to create and maintain private core image libraries provides the single partition (no MPS or BJF) and the multiple partition users with the ability to maintain all the installation programs in core image format. It also augments the multiprogramming facility. A private core image library may reside (starting on any cylinder boundary) on any volume. The organization of the first ten tracks and the private core image directory and library is similar to that portion of the system residence volume from cylinder 0, up to and including the core image library itself (see Figure 1.1).

The areas containing the bootstrap records, volume labels, system directory records 2-4, and the IPL loader program are formatted but unused in a private core image library. The system work area and all directories (transient, open, etc) are formatted, used, and maintained for a private core image library in the same manner as these areas are used and maintained for the system core image library. Multiple private core image libraries may reside on one volume or they may be created on separate volumes.

A private core image library may be assigned to any partition. Output from the linkage editor may be placed in a private core image library. Librarian functions are available to create and maintain this type of library.

If a catalog function is attempted in the foreground partition, the supervisor will be cataloged in the private core image library, and the IPL routine cannot access a private core image library.

System Considerations

In a disk system supporting both batched-job foreground and private core image libraries, several choices are available to the user as to the partition in which his programs are to be link-edited and in which they are to execute. A program may be:

1. Link-edited in the background to execute within the background and placed in the system core image library or a private core image library.

2. Link-edited in the background to execute within a foreground partition and placed in the system core image

library or a private core image library.

3. Link-edited in a foreground partition to execute within the background and placed in a private core image library, which will be assigned to the background at execute time.

4. Link-edited in a foreground partition to execute within that foreground partition and placed in a private core image library assigned to that partition.

User programs written in Assembler should avoid the use of Y-type address constants if they are to properly execute within a foreground partition. Most IBM-supplied language translators can execute in a batched-job foreground partition, providing enough main storage is available in which to execute the translator. For a list of IBM-supplied programs that can execute in the foreground, refer to DOS Concepts and Facilities listed in the Preface.

The selection of both the batched-job foreground and private core image library options provides the most efficient multiprogramming environment. The following is a suggested procedure for creating such an environment. Figure 1.25 shows a private library dedicated to each partition. Figure 1.26 shows the use of private libraries in a single partition environment.

1. For maximum efficiency, the system core image library can be reserved for system program phases.

2. Link-edit the assembler and other language translators in the background and place them in a private core image library. See Note.

3. Repeat this process in each desired foreground partition, cataloging the output from the linkage editor to a private core image library assigned to that foreground partition.

   Some language translators require a partition size of more than 10K to execute.

4. Self-relocating programs may be cataloged in the system core image library or a private core image library.

5. If a non-self-relocating program is to be run only in one partition it should be link-edited and cataloged in a private core image library that will be assigned to that partition. See Note.

6. If a non-self-relocating program is to be run in more than one partition, it should be link-edited and cataloged to the core image library for each partition.

   Note: If it is not desired to dedicate a private core-image library to the background partition, these programs must be placed in the system core image library.

In the environment just created, it is possible to execute a compile, link edit, and go job in any partition. To execute such a job in a foreground partition, a private core image library must be assigned to that partition. When executing the linkage editor in a foreground partition, output must be placed in a private core image library. When executing in the background partition, output from the linkage editor can be permanently placed in either the system core image library or a private core image library, if assigned.

A unique SYSLNK, as well as other work files (for example, SYS001, SYS002), is required for each partition. If the language translators are to execute concurrently, then each work file must be uniquely identified. For example:

```
// DLBL IJSYSLN, 'DOS.BG.SYSLNK'
// EXTENT ,,,,10,200
// DLBL IJSYS01,'DOS.BG.SYS001'
// EXTENT ,,,,410,200
   .
   .
   .
// DLBL IJSYSLN,'DOS.F1.SYSLNK'
// EXTENT ,,,,210,200
   .
   .
   .
```

If the SYSLNK for each partition is assigned to a different drive, no unique identification is necessary. Figure 1.27 illustrates these two cases.

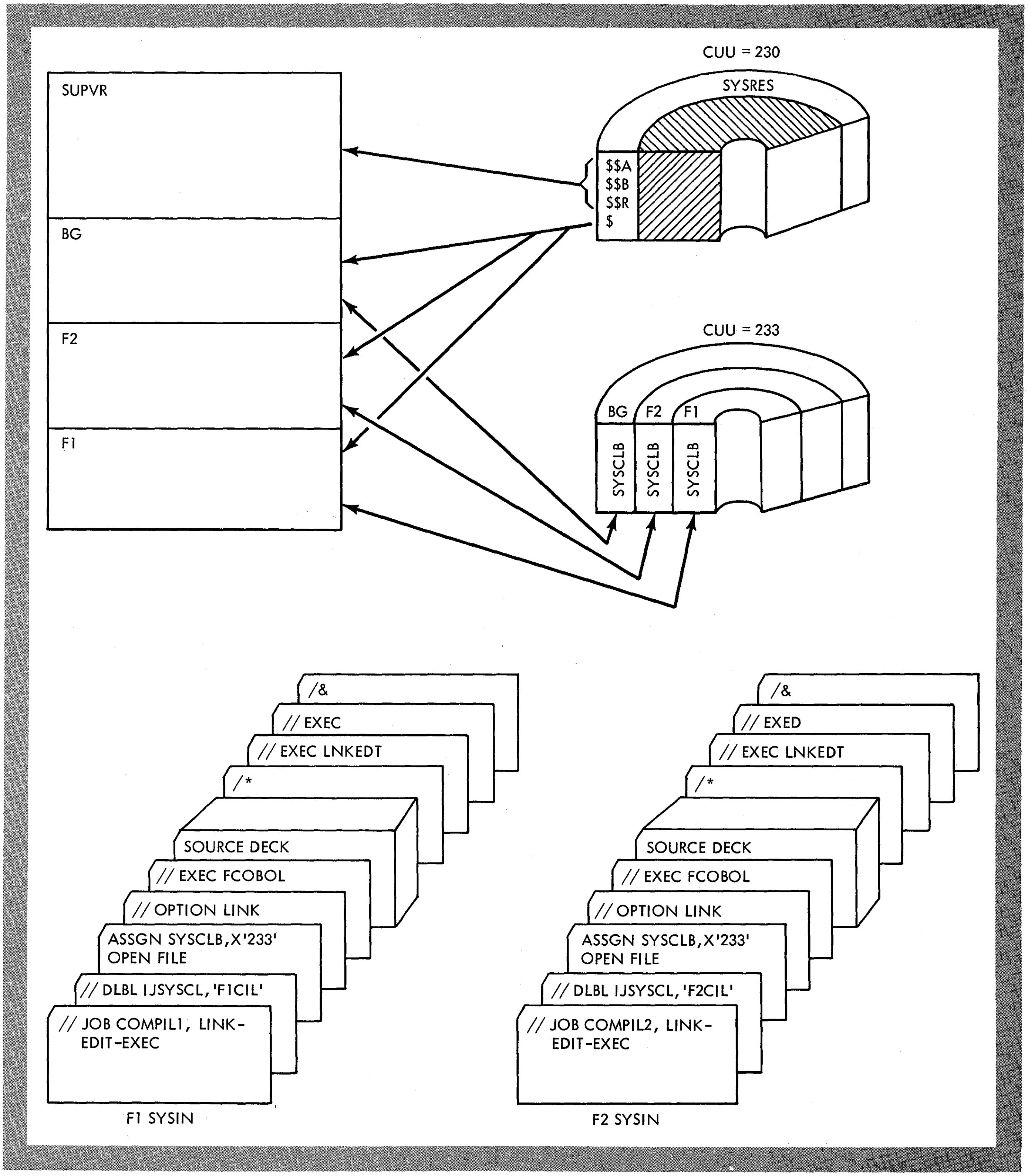


Figure 1.25. Example of Dedicated PCIL in a Multiple Partition Environment

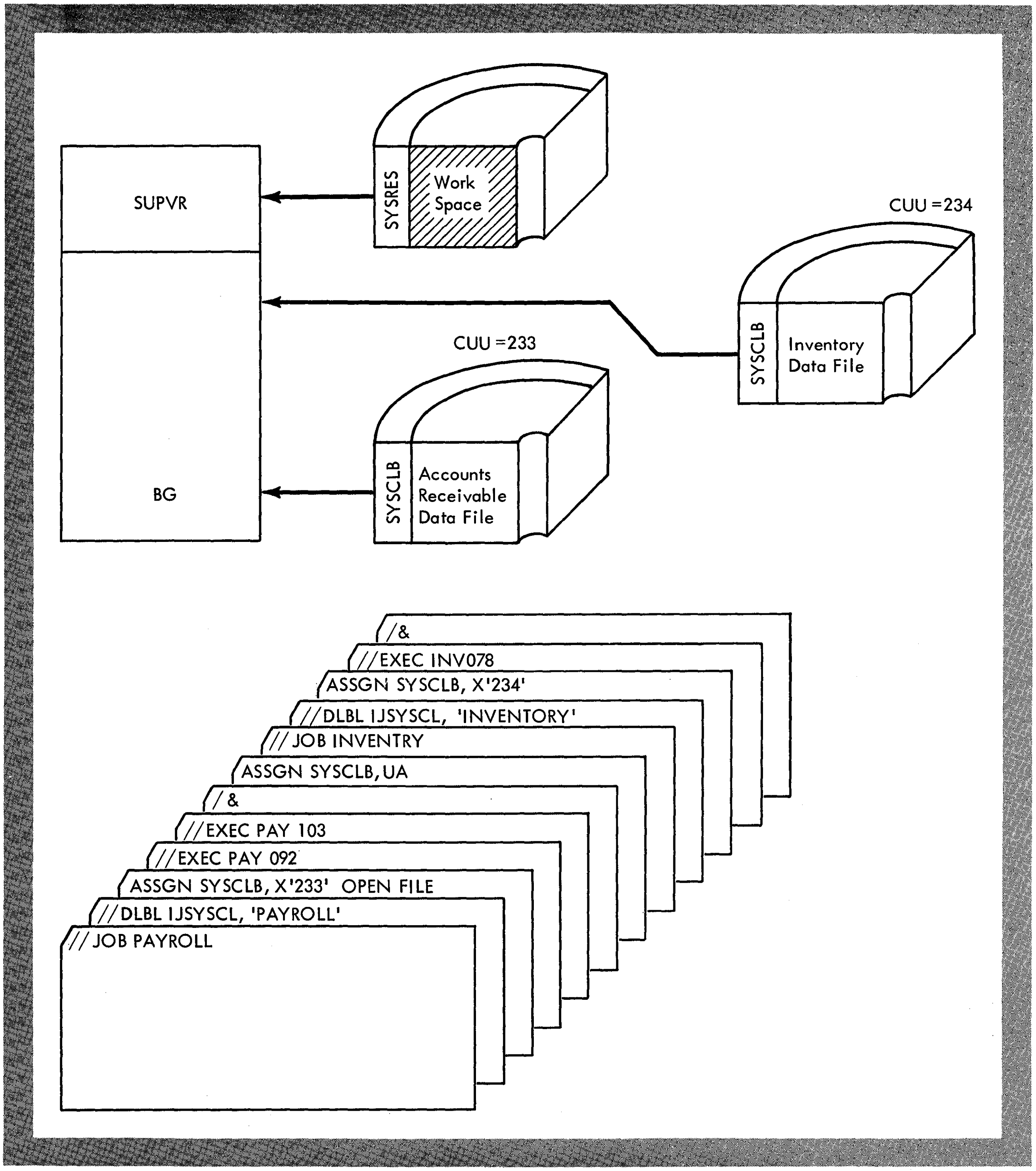


Figure 1.26. Example of PCIL in a Single Partition Environment

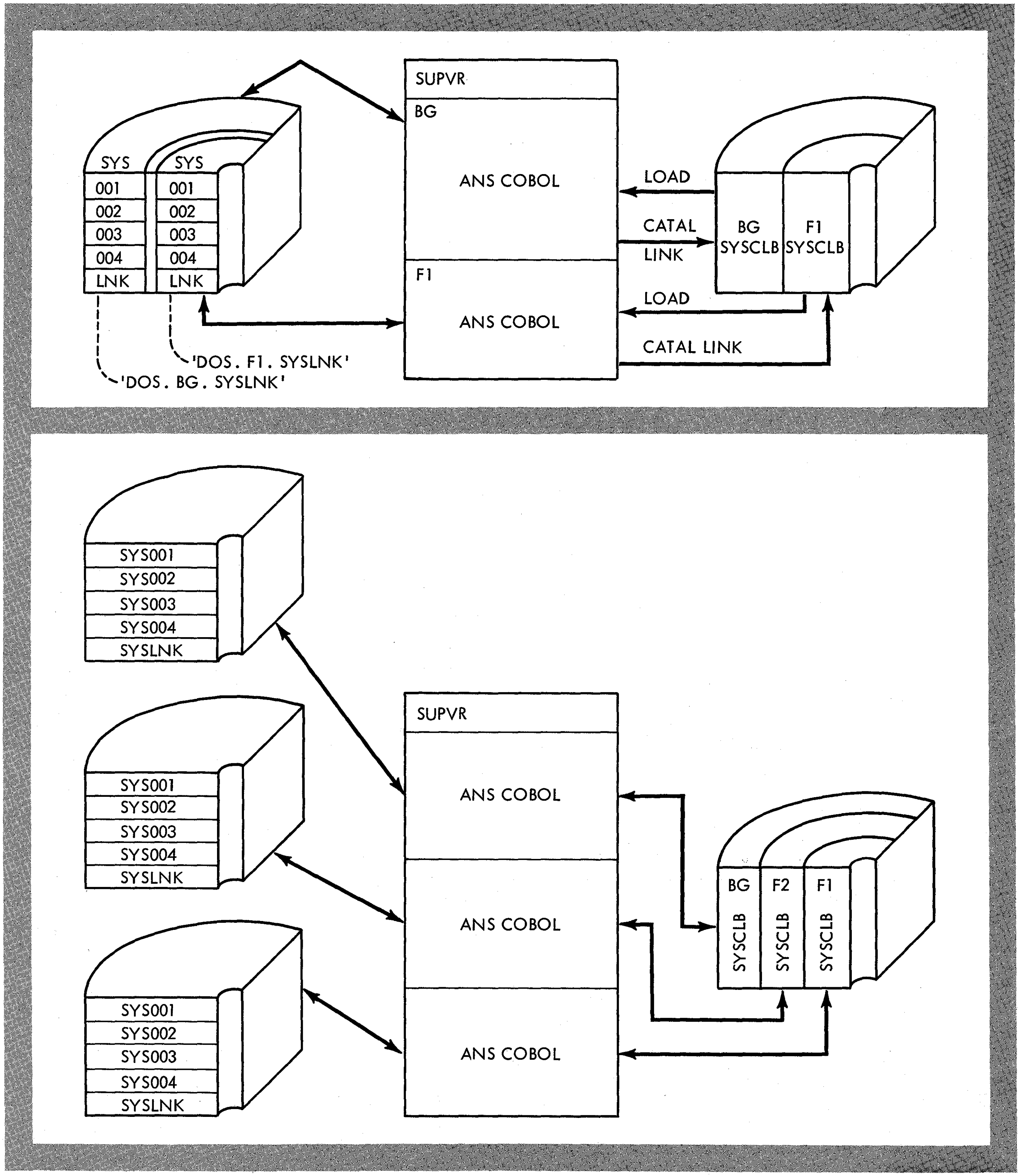


Figure 1.27. Identification of SYSLNK Files

In a batched-job multiprogramming environment the assembler and linkage editor functions can execute simultaneously in different partitions, each with its own assigned private core image library. Non-self-relocating programs are not limited to execute in only one partition because a version may be link edited for each partition and stored in the core image library for that partition.

Note that in fetching or loading a phase into main storage, if a private core image library is assigned, it is searched first. If the phase is not found in the private core image library, the system core image library is searched. The one exception to this sequence is for $ and $$ phases, in which case the system core image library is searched first.

The multiprogramming system just described requires a minimum of 32K of positions of main storage.

System Generation Guidelines

PCIL=$\left\{\begin{array}{l}\underline{\text{NO}}\\ \text{YES}\end{array}\right\}$

To specify supervisor support for private core image libraries, the PCIL=YES keyword parameter must be included in the FOPT macro. If PCIL=YES is specified and MPS=NO or MPS=YES, private core image library partition support is provided for the background only.

JOB ACCOUNTING INTERFACE SUPPORT

The Job Accounting Interface facility provides job step and job information that you can use for charging system use, supervising system operation, planning new applications, etc.

When this option is selected (JA=YES or n_1,n_2,n_3), system functions build accounting tables in the supervisor and accumulate accounting information. (See Figure 1.28 for the job accounting table.) To utilize this information, the user must write a self-relocating routine to store or print the desired portions of the table. This routine must be cataloged in the core image library under the name $JOBACCT. One user job accounting table is built for each partition when MPS=BJF; otherwise, only one table is reserved (for the background). When the system is running in the Single Program Inititation (SPI) mode, tables for the foreground partitions cannot be accessed.

Programming Considerations

If the user I/O routine ($JOBACCT) is written using LIOCS with label processing (for example, standard label tapes, DTFDA, or DTFPH with MOUNTED=ALL), the JALIOCS parameter also must be specified. This parameter is used to reserve a user save area and a label area in the supervisor. The label area replaces the one normally used by LIOCS label processing routines.

$JOBACCT can be as large as 4,096 bytes, but may use overlay loading if more storage is needed. For more efficient loading it should not exceed one core image library block (1728 bytes on an IBM 2311, or 1688 bytes on an IBM 2314/2319). With the one-block length, only one LOAD is required to get the routine into main storage.

Because $JOBACCT is called in at the end of each job step, it should only perform data gathering and recording, but not data reduction and formatting if additional system overhead is to be held to a minimum. Overhead depends largely upon the efficiency of $JOBACCT. The optional SIO accounting (JA=n_1,n_2,n_3) also causes additional overhead.

LIOCS uses registers 13-15. If $JOBACCT needs any of these registers when any LIOCS function has been performed, save and restore the desire registers (register 14 should always be saved when using LIOCS because it is necessary to return to job control via BR 14).

If $JOBACCT uses LIOCS, it should save at least part of the DTF information (status switches, extent information, pointers) in the user save area. If more than one DTF is used, information from each should be saved. The user save area may be used to save any type of information as well as accumulate step to step statistics for end of job accounting. This accumulation reduces the rate of scheduled output records caused by writing a step accounting record for each job step. The user save area is not accessed by system functions.

If an error occurs that causes $JOBACCT to be canceled, $JOBACCT is not called again until the system is re-IPLed. "JOB ACCT" appears in the cancel message, and the problem program name appears in the EOJ message. The STXIT option also may be used to inform the operator that an error occured in $JOBACCT rather than in the problem program. The job in that partition is terminated and normal processing continues with the next job.

The system passes registers 11-15 to the user's I/O routine ($JOBACCT). These registers contain the following information:

- Register 11: length of the job accounting table. Each table may vary in length according to the number of SIO counts specified at system generation time.
- Register 12: base register for $JOBACCT (this eliminates the need for the user to load the base register);
- Register 13: address of the user save area;
- Register 14: link register ($JOBACCT must exit via BR 14 to return to job control);
- Register 15: address of the partition's job accounting table.

Because some of the job step information (see fields 4, 5, and 10-16 in Figure 1.28) is cleared in the step-to-step transition, job control calls $JOBACCT at the end of each step. If $JOBACCT does not save or accumulate this information, it is lost.

System Generation Guidelines

$$\text{JA=}\left\{\begin{array}{l}\underline{\text{NO}}\\ \text{YES}\\ (n_1,n_2,n_3)\end{array}\right\}$$

Specification of either YES or (n_1,n_2,n_3) gives Job Accounting Interface support. If MPS=BJF (in the SUPVR macro) Job Accounting Interface is supported for all partitions. (The foreground job accounting tables cannot be accessed when running in the SPI mode.)

Specification of (n_1,n_2,n_3) gives the additional support of SIO accounting. The value of each n may range from 0-255 and indicates the number of devices available for SIO accounting for each partition (BG, F2, and F1, respectively). These numbers are independent of the system generation option of the number of devices attached to the system. If more I/O devices are accessed than the number specified for that partition, SIO accounting for the additional devices will not be performed.

$$\text{JALIOCS=}\left\{\begin{array}{l}\underline{\text{NO}}\\ (s,\ \ell\ )\end{array}\right\}$$

No indicates that no special LIOCS support is required. Specification of (s, ℓ) indicates that a user save area and a label area are to be reserved.

s is the decimal number of bytes to be reserved for the user save area (located in the supervisor). This save area may be used to save DTF information or for any other purpose desired by the user. The system does not access this area. (The address of the save area is available in register 13 when $JOBACCT is called.) The range of valid s is 0-1024, with a default of 16.

ℓ is the decimal number of bytes needed for a label area. This label area replaces the one normally used by LIOCS label processing. It is required when $JOBACCT uses LIOCS for such things as standard tape labels, DTFDA, and DTFPH with MOUNTED=ALL. The valid range of ℓ is 0-224 with a default of zero. The value that is substituted for ℓ is normally the number of bytes that would be allocated by a given parameter on the LBLTYP statement. See <u>DOS System Control and Service</u> listed in the <u>Preface</u>, to determine the number of bytes allocated for any given LBLTYP statement.

If the JA parameter is specified and JALIOCS is not, Job Accounting Interface is generated but no alternate label area is reserved (16 bytes are reserved for the save area). The routine $JOBACCT must then use a device or method that does not require LIOCS label processing. If the JA parameter is not specified, the JALIOCS parameter is ignored.

When the JA parameter is included, timer support is automatically included. If the CPU is not equipped with a timer, or if the timer is disabled, time fields in the accounting table will not be meaningful.

Field	Displacement	Byte Length	Contents
1	0-7	8	Job Name. 8-byte character string taken from JOB card.
2	8-23	16	User Information. 16 characters of information taken from the JOB card.
3	24-25	2	Partition ID. BG, F2, or F1.
4	26	1	Cancel Code.
5	27	1	Type of Record. S=job step; L=last step of job.
6	28-35	8	Date. mm/dd/yy or dd/mm/yy depending on supervisor option.
7	36-39	4	Start Time. 0hhmmssF, where h=hours, m=minutes, s=seconds, F is a sign (in packed decimal format).
8	40-43	4	Stop Time. zeros except in last record, which has job stop time (in same format as start time).
9	44-47	4	Reserved.
10	48-55	8	Phase Name. 8-byte character string taken from the EXEC card.
11	56-59	4	High Core: Hex address of uppermost byte of any program fetched or loaded (taken from the communications region).
12	60-63	4	CPU Time. 4 binary bytes given in 300ths of a second. Time is calculated from exit of the user-written routine called during job control to next entry of the routine. Time used by the user-written output routine is charged to overhead of the next record.
13	64-67	4	Overhead Time. 4 binary bytes given in 300ths of a second. Includes time taken by functions that cannot be charged readily to one partition (such as attention routine and error recovery). System overhead time is divided by the number of active batch partitions and recorded in each accounting table.
14	68-71	4	All Bound Time. 4 binary bytes in 300ths of a second. This is the time the system is in the wait state divided by the number of partitions running.
15	72-		SIO Tables. Variable number of bytes. Six bytes are reserved for each device specified in the JA parameter. First two bytes are X'0cuu', next four are hex count of SIOs for job step. Unused entries contain X'10' followed by five bytes of zeros. Stacker select commands for MICR devices are not counted. Error recovery SIOs are not charged to the Job Accounting Table. Devices are added to the table as they are used.
16		1	Overflow. Normally X'20'. Set to X'30' if more devices are used than set by the JA parameter at system generation time.

Cancel Code (hex)	Cause
10	Normal EOJ
17	Program Request. Same as 23 but causes dump because subtasks were attached when maintask issued CANCEL macro.
18	Eliminates cancel message when maintask issues DUMP macro with subtasks attached.
19	I/O operator option.
1A	I/O error.
1B	Channel failure.
1C	CANCEL ALL macro issued.
1D	Maintask termination.
1E	Unknown ENQ requestor.
1F	CPU failure.
20	Program check.
21	Illegal SVC.
22	Phase not found.
23	Program request.
24	Operator intervention.
25	Invalid address or insufficient core allocation to partition.
26	SYSxxx not assigned (unassigned LUB code).
27	Undefined logical unit (invalid LUB code in CCB).
28	QTAM cancel in progress.
30	Read past /& on SYSRDR or SYSIPT.
31	I/O error queue overflow (error queue overflow or no CHANQ entry available for ERP).
32	Invalid DASD address (disk).
33	No long seek (disk).
34	I/O error during fetch (irrecoverable I/O error during fetch of non-$ phase).
35	Job control open failure.
40	Load $$BEOJ.
80	Cancel occurred in Logical Transient Area (LTA).
FF	Unrecognized cancel code, or, if the system is placed in the wait state and no further processing is done by the terminator, supervisor catalog failure.

Note: The difference between Start and Stop times will not necessarily equal the sum of CPU, All Bound, and Overhead times. All Bound and Overhead times will vary, depending on the number of active partitions and the type of partition activity. CPU time is accurate for each partition, but it may not be reproducible (recreatible). That is, the same job being executed under different system conditions (varying number of active partitions, logical transient availability, etc) may show differences in CPU time.

Figure 1.28. Job Accounting Table

ALLOC Macro

F1=nK, F2=nK

This macro is optional. It specifies the partition sizes for the foreground areas. Because the foreground partition is storage protected, n must be a multiple of 2. The allocation of the foreground partitions is dependent upon core size and the fact that the background area must be at least 10K. (14K is required if the 14K variants are used in the Assembler or COBOL.) The allocation for the foreground partition may be altered after supervisor generation by the use of the ALLOC control statement.

System Considerations

If the system residence pack is to be used on more than one system, specify the partition sizes for the smallest system in the ALLOC statement. Then, if the system residence pack is used on a system with a larger amount of main storage available, the partition boundaries can be changed by submitting an ALLOC statement after IPL has been completed.

IOTAB Macro

JOB INFORMATION BLOCK (JIB)

The job information block (JIB) table (Figure 1.22) is used by the supervisor to store the permanent LUB assignments for the duration of a temporary assignment (// ASSGN). It is also used to store extent information if the DASDFP feature is implemented, and to handle alternate assignments.

One JIB entry is required for each logical unit temporarily reassigned by a // ASSGN statement. One JIB is required for each alternate logical unit assignment. One JIB is required for each open 2311/2314/2319 extent with the DASD file protect feature (except for system input/output extents). Two JIBs are required for each open 2321 extent with the DASD file protect feature. The minimum value generated is 5 and the maximum value is 255.

CHANNEL QUEUE (CHANQ)

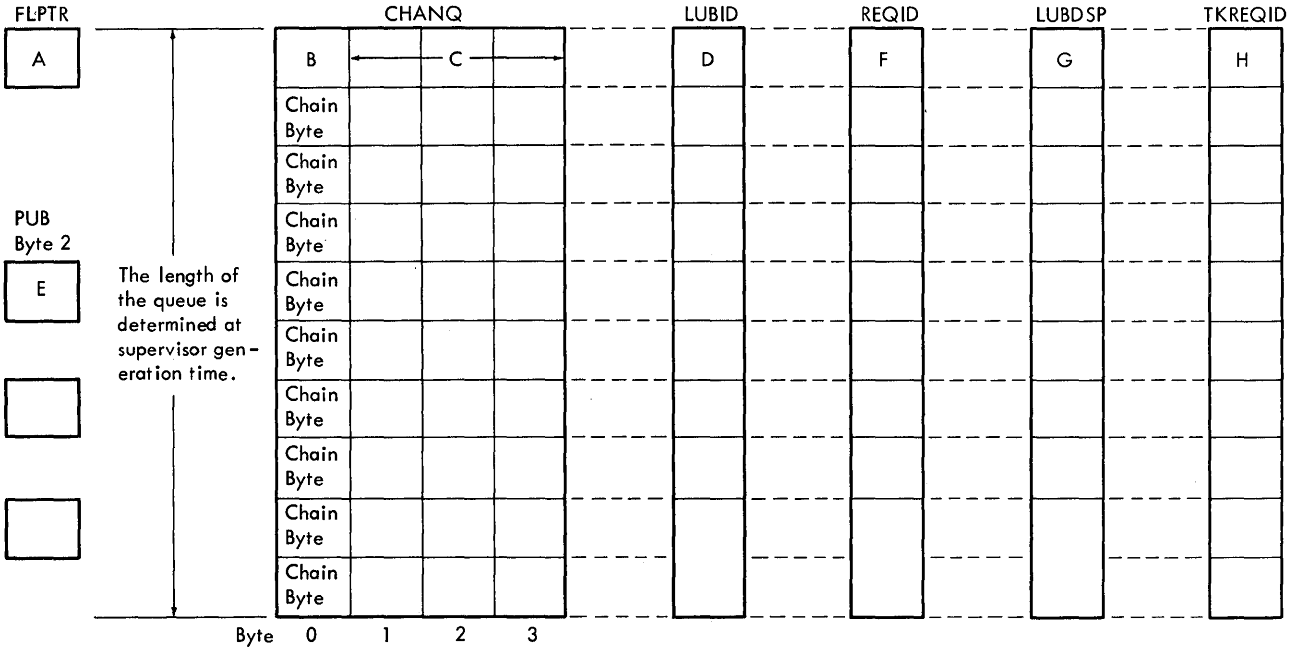
$$\text{CHANQ}=\left\{\begin{array}{c}\underline{6 \text{ or } 6+\text{CBF}}\\ n\end{array}\right\}$$

The channel queue (CHANQ) table is used by the supervisor to schedule I/O operations. The table consists of four byte entries containing a pointer to the next CHANQ entry for that device followed by the CCB address of the specified device. The n parameter specifies the number of entries in the channel queue table. The minimum value generated is 6. If the assumed value is taken and the console buffering (CBF) option is specified, the assumed value is six more than the CBF value. The maximum number of CHANQ entries that can be specified is 255. Figure 1.29 shows the CHANQ table.

Determining Size of the CHANQ

When determining the size of the CHANQ at system generation time consider the following:

1. In a teleprocessing environment with many lines and terminals, the size of the CHANQ could become exceedingly large (see the following discussion and Note 1).
2. If multifile volumes are used on the system, one CHANQ entry is required for each file being accessed at the same time.
3. When the supervisor finishes with an entry in the queue, this entry can be used for some other device. Thus, the first entries in the CHANQ are most often used while the entries near the end of the table are seldom used.
4. If there are not enough entries specified in the CHANQ (the last entry in the table is in use when another I/O request is made), the requesting task or partition must wait in a compute loop until an entry is free to allow scheduling of the I/O operation.



KEY

A The free list pointer contains a displacement index to a free list entry within the channel queue. The free list is a group of entries that function in essentially the same manner as a device queue. When the free list pointer contains a hexadecimal FF, it indicates that no more free list entries are available.

B The first byte of the channel queue entry (chain byte) contains a pointer (displacement index) to the next channel queue entry for that device. A hexadecimal FF indicates the last channel queue entry for that device. New requests on a given device are queued at the end of a given device queue.

C CCB address for the specified device.

D A pointer (displacement index) to the entire LUB table identifying the logical unit making the I/O request. This is doubled to get the actual displacement into the full LUB table.

E Contains a pointer (displacement index) to the first channel queue entry for a specific device (Figure 19).

F Contains a code identifying the program making the I/O request. The one-byte entry is called a RID (Requestor Identification). The RID indicates what program the CCB belongs to. The RID is in the form X'nk'.

n = user-storage protection key (supervisor = 0, BG = 1, F2 = 2, F1 = 3).
k = 0 for all user requests and all supervisor CCBs, where n = 0.
k = 1 for supervisor CCBs to SYSLOG that bypass ID prefix.
k = 2 for a fetch CCB.
nk = FF for any unused channel queue entries.

G Contains X'FF' if the LUB is nonsystem class, or contains the displacement index within the partition LUB if it is a system class LUB.

H Contains X'FF', or the displacement into the PIB table for the PIB of the task requesting I/O.

Bytes 108-109 (X'6C' - '6D') of the communications region contain the address of the LUBID Table. Label LUBIDTAB identifies the first byte of the table. Bytes 98-101 (X'62' - '63') of the communications region contain the address of the Error Recovery Block (ERRBLOC); ERRBLOC + 6 bytes is the address of the channel queue table (CHANQ). The addresses of the other tables are not at fixed locations. They can be found in the program listing cross-reference by using the labels REQIDTAB, LUBDSPTB, and TSKIDTAB. CHANQ can also be found in this way.

Figure 1.29. CHANQ, LUBID, REQID, LUBDSP, and TKREQID Tables

The following two approaches are presented as an aid in determining the number of CHANQ entries to specify.

1. Using the total number of devices on the system as a base, add one to that value for each file (other than the first) on any DASD. For example:

Device	CHANQ Entries	Number of Files After First File	CHANQ Entry Total
2540 Card Reader (X'00C')	1	0	1
2540 Card Punch (X'00D')	1	0	1
1403 Printer (X'00E')	1	0	1
1052 Printer-Keyboard (X'01F')	1	0	1
2311 Disk Drive (X'190')	1	1	2
2311 Disk Drive (X'191')	1	1	2
2311 Disk Drive (X'192')	1	1	2
		TOTAL	10

2. Generate your system using 10 as a value for the CHANQ parameter.

3. Run your worst case or heaviest workload against the system. Then take a storage dump.

4. Determine how many of the channel queue entries were used in the worst case environment.

   As an aid in determining the number of entries used, let us examine the contents of the CHANQ for the same system after IPL and after running our worse case workload. First, assuming the start of the table at X'2D24', the contents of the CHANQ after an IPL would be:

Hex Location	Value
2D24	02002EC0
2D28	FF0043F8
2D2C	03000000
2D30	04000000
2D34	05000000
2D38	06000000
2D3C	07000000
2D40	08000000
2D44	09000000
2D48	FF000000

Note that during the IPL only two of the CHANQ entries were used (i.e., only the first two entries in the table have CCB addresses in bytes 1-3). For the same system after running our worst case workload and taking a storage dump, the contents of the CHANQ appear as follows:

Hex Location	Value
2D24	01004020
2D28	050040B8
2D2C	00002F90
2D30	FF0043F8
2D34	06005898
2D38	040036DD
2D3C	07000000
2D40	08000000
2D44	09000000
2D48	FF000000

Note 1: Only six of the entries were used. There were sufficient entries in the table to handle the I/O requests with a reserve of 4. You would have to decide whether or not you should reduce the amount of entries. In this case, there are only 16 extra bytes involved.

Note 2: In a teleprocessing environment if you had 100 channel queue entries and only used 50 of them, you would have to do some careful evaluation.

Note 3: If the tenth entry in the table had a CCB address, it would indicate the possibility that a task or partition may have been waiting for a place in the table. If this occurs, you should add more entries to the table and repeat the procedure until the last entry contains no CCB address.

If the CHANQ value is too large, storage space is wasted. If too small, the supervisor must wait for an entry in the CHANQ table to clear before enqueuing the next request. If the CHANQ table entries are filled after taking a dump, reassemble your supervisor with a larger CHANQ value. The following approach gives a more efficient way of determining the number of CHANQ entries to specify.

1. Generate your system using the total number of logical units (Line Control Blocks -LCB- or Channel Control Blocks -CCB) as the maximum number of logical I/O requests that can be enqueued. It is unnecessary for the CHANQ parameter value to be larger than the total number of system and programmer units. The value for the CHANQ parameter may be that number or smaller.

2. Run the programs that use the largest number of logical I/O units against the system. If your system supports multiprogramming, have as many partitions running as represent your heaviest workload. Use programs that use logical double buffering or that handle dynamic buffering (teleprocessing). Then use the PDAID DUMPGEN program to print a dump that formats the CHANQ.

3. Determine how many of the channel queue entries were used in this worst case environment.

An as aid in determining the number of entries used, let us examine the contents of the CHANQ. After running our worst case workload (the program that is running with the most double buffered I/O devices), and using DUMPGEN, the contents of the CHANQ might appear as follows:

```
     CHANNEL QUEUE TABLE

DOS  CHAIN   CCB ADDR   CUU
     PTR
 00   02     001CF8
 01   FF     006398     00E
 02   03     000000
 03   04     000000
 04   05     000000
 05   06     000000
 06   07     000000
 07   08     000000
 08   09     000000
 09   0A     000000
 0A   0B     000000
 0B   0C     000000
 0C   0D     000000
 0D   0E     000000
 0E   0F     000000
 0F   10     000000
 10   11     000000
 11   12     000000
 12   13     000000
 13   FF     000000
```

Only two of the entries were used. Therefore, we can reduce the amount of entries. In this example, the minimum CHANQ number, six, can be specified.

DVCGEN Macro

CHANNEL RESCHEDULING CONSIDERATIONS

The type of supervisor generated is an important factor in establishing the order of DVCGEN statements within each channel definition in the PUB table. See Figure 1.30. Figure 1.14 shows the I/O relationship.

When rescheduling a selector channel, the non-MPS supervisor channel scheduler always begins at the first device definition for the channel within the PUB table. Rescheduling a channel is the channel scheduler's way of locating the next device on the channel that has an outstanding I/O request in the CHANQ (Figure 1.29). The channel scheduler examines the third byte in the PUB table entry for a device. If a null pointer (all bits on) is found, the scheduler steps to the next device within that channel in the PUB table. For a non-MPS system, this scan always starts at the first PUB entry for the channel that has just completed an I/O operation. Figure 1.31 lists the device type codes.

When rescheduling a selector channel in an MPS system, the channel scheduler uses a rotating scanning technique to ensure that each device on a channel gets an equal opportunity of being started. The channel scheduler in a MPS system retains the address of the last PUB entry started for each channel. After an I/O interrupt has been serviced, the supervisor determines that the channel is free, and reschedules the channel by going to the last PUB entry started for that channel and stepping to the next entry. A test determines if that device has an outstanding request. If there are no outstanding requests for that device, the scheduler steps to the next PUB entry for that channel. If no requests are found, the scheduler steps to the first PUB entry for the channel and the scan continues until an outstanding request is located. If no outstanding requests are located, the scan stops after the PUB entry for the last device started on that channel is checked.

When rescheduling a multiplexor channel with no burst mode devices running, the channel scheduler starts another I/O operation for this device (if there is another request pending in the CHANQ). If burst mode devices are running (BMPX=YES is specified in the PIOCS macro), a rotating scanning technique as described for a selector channel determines the next I/O request to be initiated.

System Generation Guidelines

The following rules must be used when using DVCGEN macro statements:

1. One DVCGEN macro instruction must be used for each device on system. For a 2314/2319, each individual unit needs a DVCGEN statement.

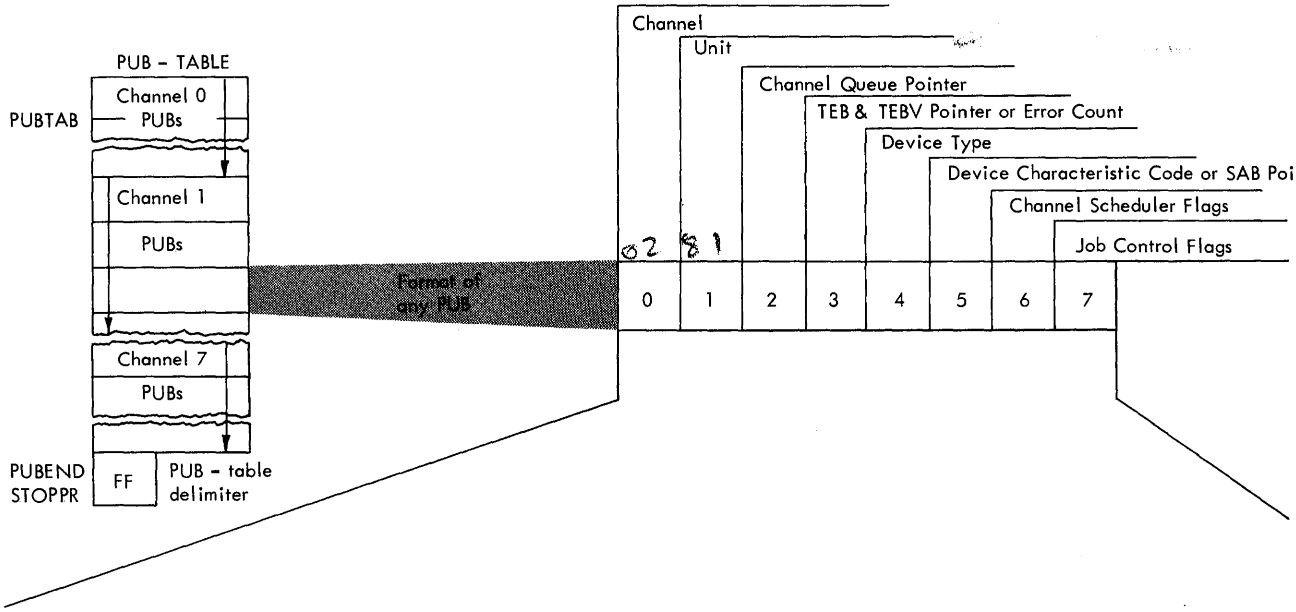
2. The device generated by these macros can be changed with ADD and/or DEL cards at IPL time.

3. In a non-MPS supervisor, DVCGEN macros _must_ be in ascending channel address sequence. SYSRES should be the first DVCGEN statement for its channel if it is to have the highest priority in a non-MPS supervisor. In an MPS supervisor, sequence is not important since each device on a channel gets an equal opportunity at being started.

4. Switchable units (attached to more than one selector channel) are defined once on the lowest channel by which they are addressable. They may not be re-defined as non-switchable units on the higher channels. Any switchable units must be the last devices specified for each channel and must be on consecutive channels.

5. The total number of DVCGEN statements must not exceed the total number of devices specified in the IODEV parameter of the IOTAB macro.

6. IBM 1052 Printer-Keyboards and IBM 3210 or 3215 Console Printer-Keyboards that are not on-line but were defined by DVCGEN statements must be deleted by DEL statements when performing IPL from the card reader.



BYTE 0 - Channel number. (Hex 0-7, FF=NULL)

BYTE 1 - I/O device unit number. (HEX 1F= 1052, HEX 80=magnetic tape unit 0 ...)

BYTE 2 - HEX 0, 1, 2, ... points to the first channel queue entry for this device.

BYTE 3 - If device is a magnetic tape unit* and TEBs and/or TEBVs are specified, this byte is a TEB/TEBV pointer (HEX 1, 2, 3 ...)

If device is a magnetic tape unit* but neither TEBs nor TEBVs are specified, this byte is an error counter.

If device is not a magnetic tape unit*, this byte is an error counter.

BYTE 4 - See Figure 1.31 for device type codes.

BYTE 5 - SS of the MODE=parameter in the DVCGEN macro for tape unit.

For DASD with seek separation, this byte is used as the SAB pointer. With track hold but not seek separation supported, this byte contains a pointer to the Track Hold Table entry or X'FF' (with both SKSEP and TRKHLD specified, the track hold pointer is found in the SAB entry).

For MICR type devices, this byte indicates the external interrupt line is in use.

BYTE 6 -
Bit 0: 1 = Device busy
1: 1 = Switchable device
2: 1 = EOJ for SYSRDR or SYSIPT
3: 1 = I/O error queued for recovery
4: 1 = Operator intervention required
5: 1 = Device end posting required
6: 1 = Burst device on MPX
7: 1 = 7-track tape unit

BYTE 7 -
Bit 0-4: standard MODE assignment for 7-track tape (all ones if not tape, all zeros if device is down).
5: device is assigned to a background job
6: device is assigned to a foreground 1 job
7: device is assigned to a foreground 2 job

NOTE: A null is generated for each device to be supported by the supervisor.
Standard physical unit assignments are made to the PUB table at supervisor generation time. PUBs are ordered by channel and priority within a channel.

Bytes 64 and 65 (X'40' - '41') of the communications region contain the address of the PUB table entry.
Label PUNTAB identifies the first byte of the table.

*2400 series or 3420 Magnetic Tape units or 2495 Tape Cartridge Reader (TEBs); 2400 series or 3420 Magnetic Tape units only (TEBVs).

Figure 1.30. Physical Unit Block (PUB) Table

Card Code	Actual Device	Dev. Type X'nn'	Device Type
2400T9	9-track 2400 Series Magnetic Tape Units	50	Magnetic Tape Units
	9-track 3420 Magnetic Tape Units		
2400T7	7-track 2400 Series Magnetic Tape Units		
	7-track 3420 Magnetic Tape Units		
2495TC	2495 Tape Cartridge Reader	51	Tape Cartridge Reader
1442N1	1442N1 Card Read Punch	30	Card Readers - Punches
2520B1	2520B1 Card Read Punch	31	
2501	2501 Card Reader	10	Card Readers
2540R	2540 Card Reader	11	
2540P	2540 Card Punch	21	Card Punches
2520B2	2520B2 Card Punch	20	
1442N2	1442N2 Card Punch	22	
2520B3	2520B3 Card Punch	20	
1403	1403 Printer	40	Printers
1403U	1403 Printer with UCS Feature	42	
3211	3211 Printer	43	
1404	1404 Printer	40	
1443	1443 Printer	41	
1445	1445 Printer	41	
1050A	1052, 3210, or 3215 Printer-Keyboard	00	
UNSP	Unsupported Device	FF	Unsupported. No burst mode on multiplexor channel
UNSPB	Unsupported Device	FF	Unsupported with burst mode on multiplexor channel
2311	2311 Disk Storage Drive	60	DASD
2314	2314 Direct Access Storage Facility	62	
	2319 Disk Storage Facility		
2321	2321 Data Cell Drive	61	
1412 **	1412 Magnetic Character Reader	75	MICR - Magnetic Ink Character Recognition Devices and Optical Reader/Sorters
1419 **	1419 Magnetic Character Reader	72	
	1255 Magnetic Character Reader		
	1259 Magnetic Character Reader		
1419P **	1419 Dual Address Adapter Primary Control Unit	73	
1419S **	1419 Dual Address Adapter Secondary Control Unit	74	
2701 *	2701 Data Adapter Unit	D0	Teleprocessing lines A = SAD0 command when enabling the line B = SAD1 command when enabling the line C = SAD2 command when enabling the line D = SAD3 command when enabling the line
2702 A B C D		D1	
2703	2703 Transmission Control	D2	
2955	2955 Data Adapter Unit	D7	Data link for RETAIN/370
2671	2671 Paper Tape Reader	70	Paper Tape Reader
1285	1285 Optical Reader	76	Optical Readers
1287	1287 Optical Reader	77	
1288	1288 Optical Page Reader		
1017	1017 Paper Tape Reader with 2826 Control Unit Model 1	78	Paper Tape Reader
1018	1018 Paper Tape Punch with 2826 Control Unit Model 1	79	Paper Tape Punch
2260	2260 or 2265 Display Station	C0	Display Station
7770	7770 Audio Response Unit	D3	Audio Response Units
7772	7772 Audio Response Unit	D4	
1017TP	1017 Paper Tape Reader with 2826 Control Unit Model 2	D5	Paper Tape Reader
1018TP	1018 Paper Tape Punch with 2826 Control Unit Model 2	D6	Paper Tape Punch

Note: The codes used in the DVCGEN macros are the same codes used in IPL statements.

* For other teleprocessing devices, see IBM System/360, DOS BTAM and QTAM PLMs, GY30-5001 and GY30-5002.

** This device type code is also used for the 1270/1275 optical reader/sorters.

Figure 1.31. Device Type Codes

Section 2: Data Management

Section Outline

Section Figures

Concepts of Data Management

This section is written for the reader who seeks a general understanding of the functions and concepts of data structures and storage media. It explains the basic concepts of data organization and defines some of the terms that will be encountered in subsequent sections of this and other IBM publications.

The experienced programmer may find this section of interest, but it is written primarily for the reader whose experience with tape and disk files is limited.

DATA FILES AND RECORDS

Data files stored on such media as paper, cards, tapes, or disk storage devices, are encountered in practically every business activity. These files provide the basis for most manual, mechanical, and electronic data processing. Data files are composed of individual records ranging from a few records up to thousands or millions of records.

A record can be defined as a collection of information comprised of alphameric and/or nonalphameric characters related to a common identifier. The common identifier is known as a record's control field or key. Usually one of the prime information elements (fields) present within a record is used to identify the record. For example, man number could be used as the key or identifier for a payroll record, and policy number could be the key of an insurance policy file.

The size or length of records varies from file to file because the size can range from a single character up to thousands of characters.

A data field is a sequence of one or more characters treated as a processing unit of information. A single record usually includes one or more logical data fields. An individual data field is normally identified by its location within a record.

The logical structure of records and of fields within records has become increasingly important since the advent of computers and high-speed recording media such as magnetic tapes and disks. This logical structure is strongly affected by whether a record is fixed or variable length.

Data records of an ASCII tape file may include a block prefix and/or padding characters. These options are in addition to the fields contained in data records written in EBCDIC mode. When present, the block prefix is the first field of a physical record, and it may be 0-99 bytes long. DOS can use this field to check the length of the physical record only with ASCII variable length records. For fixed length and undefined records, DOS ignores the block prefix on input and does not restore this field on output.

The length of an ASCII physical record includes the block prefix field and the number of padding characters. Padding characters ensure that all blocks conform with the required length. DOS accepts these padding characters (corresponding to EBCDIC X'5F') on ASCII input but does not perform any padding operation on output.

Fixed length, unblocked (Figure 2.1): Each logical record is the same length as the physical record.

For ASCII fixed length, unblocked records, the block prefix and padding characters are optional. Regardless of the presence of these optional fields, ASCII fixed length with only one logical record per physical record is considered to be unblocked. The number of padding characters must always be less than the size of the logical record.

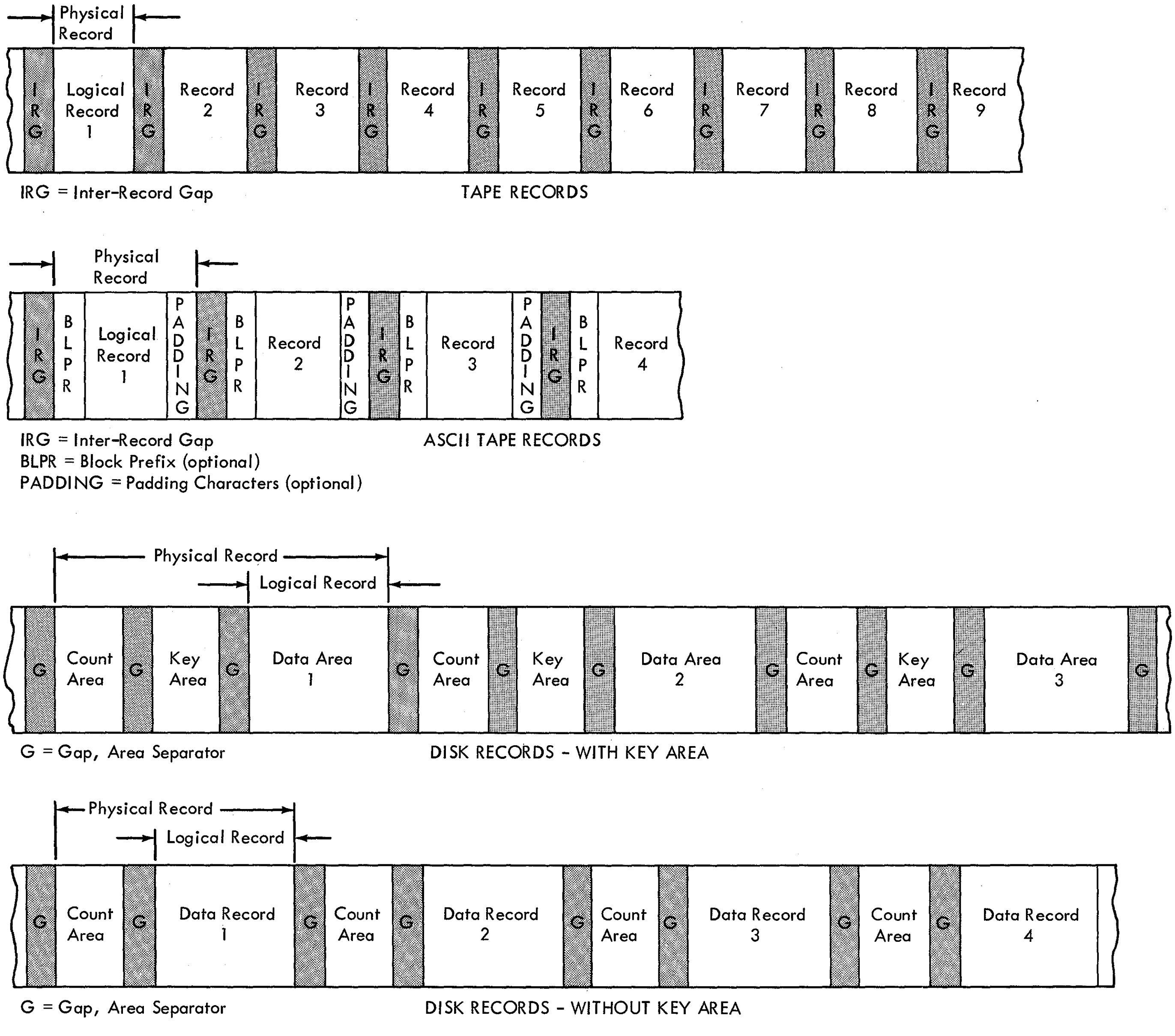
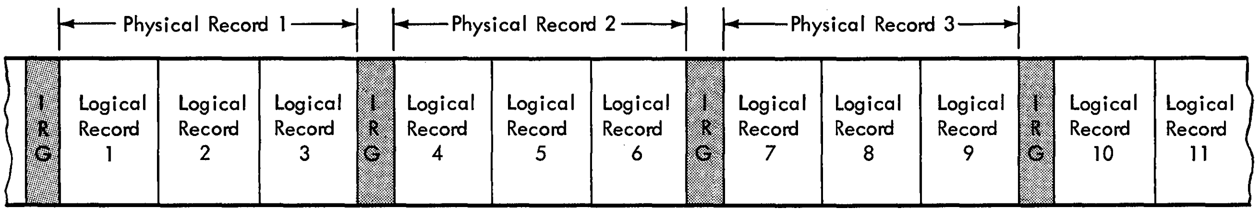


Figure 2.1. Fixed Length Unblocked Record Format

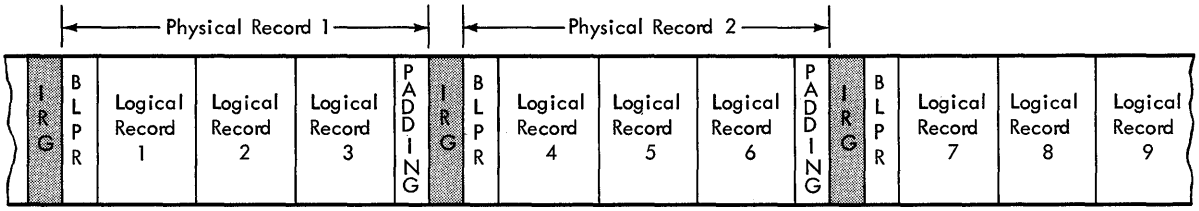
Fixed length, blocked (Figure 2.2): Blocked records are usually considered to be two or more logical records within one physical record. The number of records in each block (blocking factor) is usually kept constant. For example, the illustrations show blocked records with a blocking factor of 3 meaning there are three logical records within each block (physical record).

Each physical record in ASCII fixed length, blocked format may contain a block prefix and/or padding characters. The number of padding characters must be less than the size of the logical record. Physical records are deblocked until the number of bytes remaining in the physical record is less than the specified logical record length.



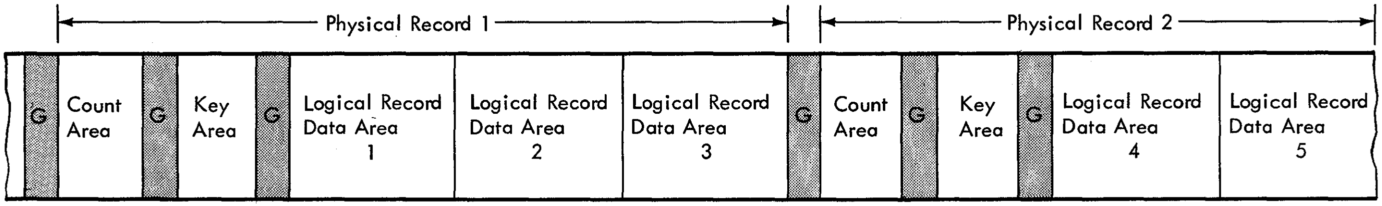
IRG = Inter-Record Gap

TAPE RECORDS



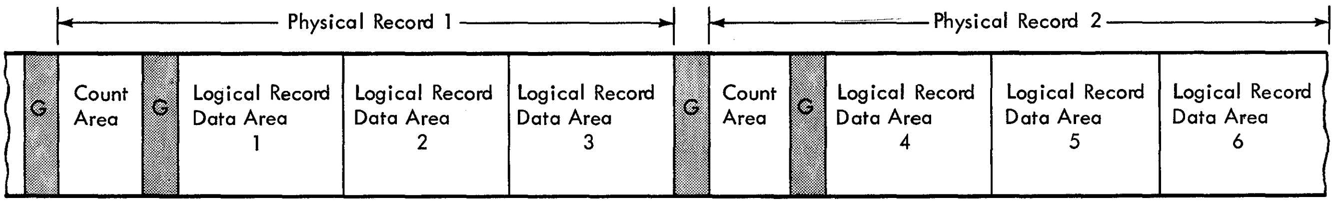
IRG = Inter-Record Gap
BLPR = Block Prefix (optional)
PADDING = Padding Characters (optional)

ASCII TAPE RECORDS



G = Gap, Area Separator

DISK RECORDS-WITH KEY AREAS



G = Gap, Area Separator

DISK RECORDS-WITHOUT KEY AREAS

Figure 2.2. Fixed Length Blocked Record Format

Variable length, unblocked (Figure 2.3): Each physical record contains one logical record that can vary in length. Each record must contain both a block length field (BL) and a record length field (RL) giving the size of the block and the size of the logical record respectively. The first two characters (XX) of the block length field (BL) specify the actual block length in 16-bit binary form. The last two characters (indicated by bb) are binary zero. For variable length unblocked records, BL specifies the logical record length plus 4 bytes (the size of BL).

The first four bytes following the block length field must contain the record length field (RL). The first two bytes (XX) specify the length of the logical record including the bytes used for RL field itself. The remaining two bytes (bb) are binary zero.

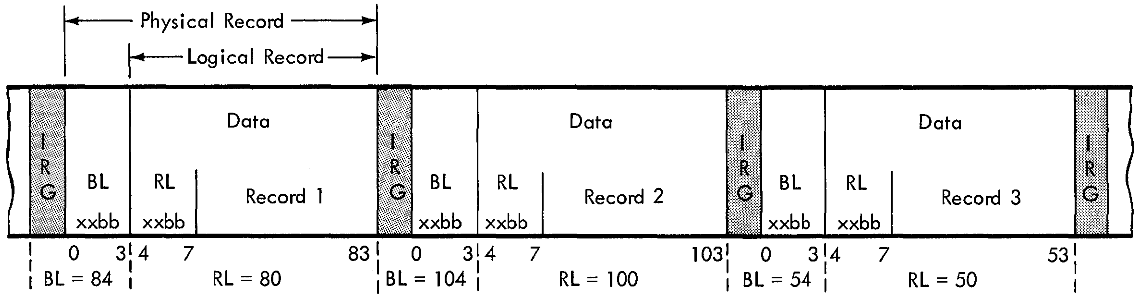
For ASCII variable length, unblocked records, each physical record may contain a block prefix. If the block prefix is four bytes long, it may contain the length of the physical record that DOS can use for checking purposes. Each logical record must contain a record length field (dddd) giving the size of the logical record in unpacked decimal format. If a padding character is found in the first position of a record length field, all remaining bytes in that block are bypassed, and the next logical record is retrieved from the next block.

Variable length, blocked (Figure 2.4): One or more logical records are contained within each physical record. The first four bytes (block length field) of each physical record (block) specifies the total number of bytes in the block. The first two bytes (XX) specify the length of the block (including the four bytes for the block length field itself). The remaining two bytes (bb) are blank. The size of each logical record must be placed in a record length field (RL). The RL must be the first four bytes of the logical record. The first two bytes (XX) of RL specify the length of the logical record including the bytes used for the RL field. The remaining two bytes (bb) are binary zero.

For ASCII variable length, blocked records, each physical record may contain a block prefix. For checking purposes, the block prefix may contain the length of the physical record (in unpacked decimal format) if the block prefix is four bytes long. The length of each logical record, also in unpacked decimal format (dddd), must be placed in the first four bytes of the logical record. If a padding character is found in the first position of a record length field, all remaining bytes in the block are bypassed, and the next logical record is retrieved from the next block.

Undefined: When file records do not conform to any of the four previous formats, they are classified as undefined. For example, any variable length record not conforming to IBM's variable length format is considered undefined.

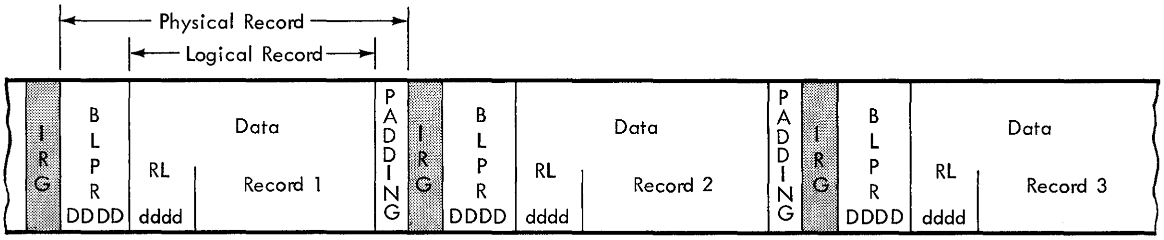
ASCII undefined records may include a block prefix and/or padding characters. DOS will not attempt to distinguish padding characters from the physical record. The entire physical record is passed on to the user.



BL = Block Length } in binary halfword (16 bit) format, plus two blank bytes
RL = Record Length }

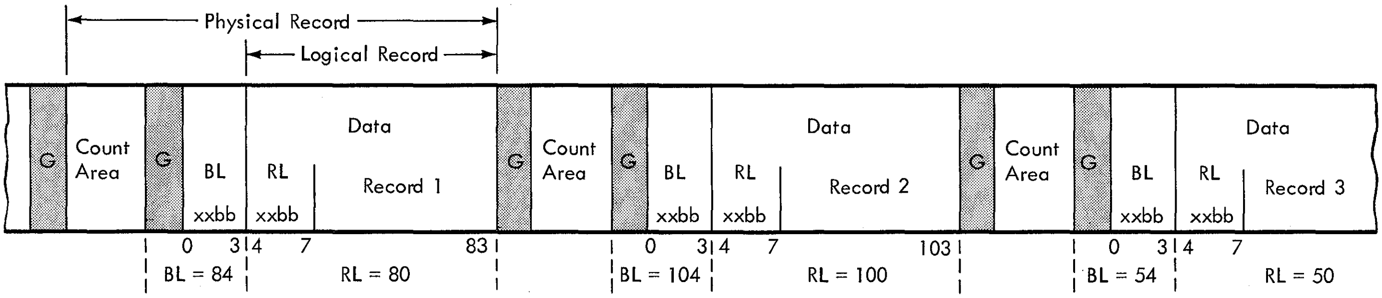
IRG = Inter-Record Gap
G = Gap

TAPE RECORDS



IRG = Inter-Record Gap
BLPR = Block Prefix (optional)
PADDING = Padding Characters (optional)
DDDD = length of the physical record if BLPR is 4 bytes long (unpacked decimal)
dddd = length of logical record in unpacked decimal format
RL = Record Length

ASCII TAPE RECORDS

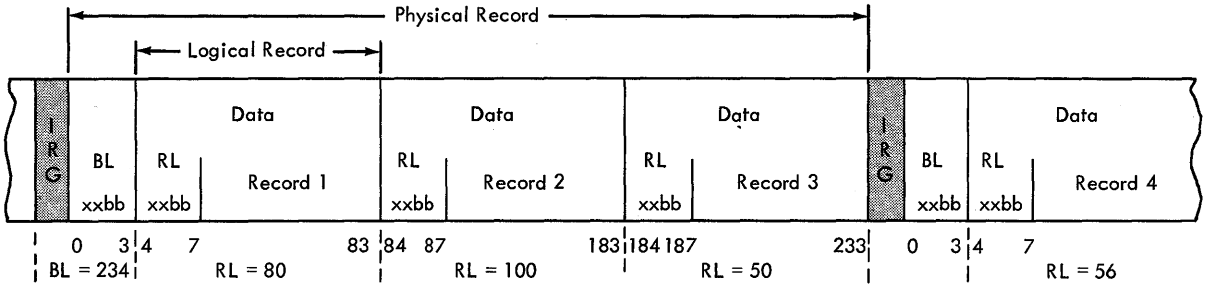


BL = Block Length } in binary halfword (16 bit) format, plus two binary zero bytes
RL = Record Length }

IRG = Inter-Record Gap
G = Gap

DISK RECORDS

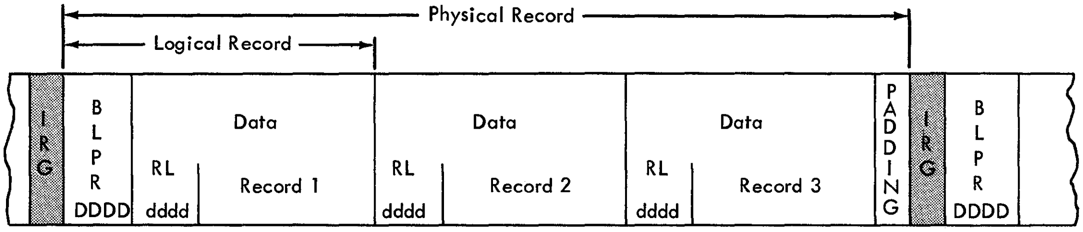
Figure 2.3. Variable Length Unblocked Record Format



BL = Block Length
RL = Record Length } in binary halfword (16 bit) format, plus two blank bytes.

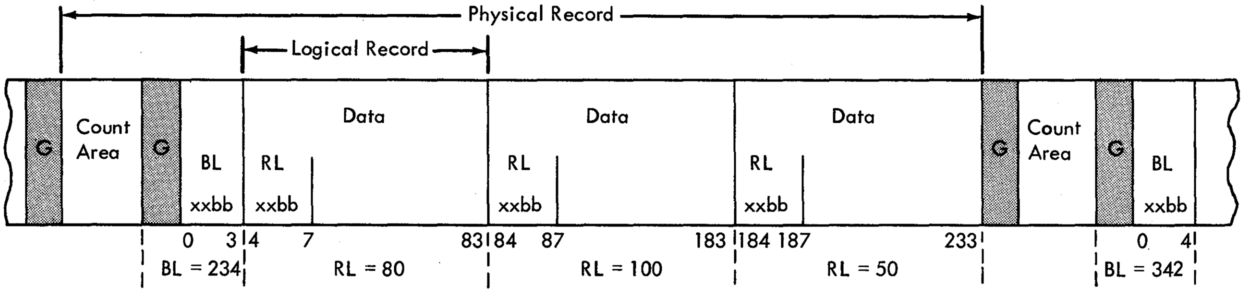
IRG = Inter-record gap
G = Gap

TAPE RECORDS



IRG = Inter-Record Gap
BLPR = Block Prefix (optional)
PADDING = Padding Characters (optional)
DDDD = Length of the physical record if BLPR is 4 bytes long (unpacked decimal)
dddd = Length of logical record in unpacked decimal format
RL = Record Length

ASCII TAPE RECORDS



BL = Block Length
RL = Record Length } in binary halfword (16 bit) format, plus two binary zero bytes.

IRG = Inter-record gap
G = Gap

DISK RECORDS

Figure 2.4. Variable Length Blocked Record Format

BLOCKING RECORDS

The length of individual data records varies with the type of data and the application requiring such data. The format design of a data record is very significant to the efficient use of the various storage media available on the System/360 and System/370. One important element in the design of data records involves what is commonly called blocking and deblocking. Input/output units (storage media) are relatively inefficient when used to store short blocks of information. To increase the efficiency of input/output units, data records are assembled into blocks of records with size convenient and efficient for processing. Each physical record on either tape or disk requires interrecord gaps. These gaps are blank areas that distinguish beginning and ending points of a record. If records are blocked before loading onto a tape or disk, many of these gaps can be eliminated. One of the most important advantages of blocked records is the increased channel/CPU processing time overlap. The average number of reads required to locate a record can usually be reduced by increasing the blocking factor (number of records per block). The greater the blocking factor, the greater the chance that the next record required will be in the same block. This is an important consideration when designing jobs that involve file searching either on tape or disk. It is particularly important when using disk storage techniques that develop overflow records. Overflow records occur when there are more items assigned to a disk track than can be stored on that track.

Blocked records normally require more main storage than unblocked records because main storage has to contain the block of records being read or written on a storage device. Also, more main storage is required to hold blocking and deblocking program instructions. The LIOCS macro instructions are designed to handle the blocking and deblocking of records so that the user need only design the most efficient blocking factor for his particular data file and equipment specifications.

LOGICAL FILE VS PHYSICAL UNIT

A logical file consists of one record for each item of a group (i.e., an inventory file would contain one record for each inventory item). A physical unit(s) is used to store a logical file. For example, the IBM 2400 Magnetic Tape Unit, the IBM 3420 Magnetic Tape Unit, the IBM 2311/2314/2319/2321 Direct Access Storage Devices (DASD) and the IBM 2540 Card Read Punch can be considered physical units when data records punched into cards are being read into the system.

A logical file may be of such a size that it requires several reels of tape or disk packs to contain it. Such a file would be referred to as a multivolume file. (Each reel of tape, disk pack or each cell within the 2321 is considered to be a volume.) If a small file does not require an entire reel of tape or disk pack to contain it, the remaining space on the volume could be utilized by another complete or partial file. This volume would be referred to as a multifile volume.

DATA MANIPULATION

Data manipulation with DOS is implemented at two distinct levels. The first level, physical IOCS requires extensive knowledge of System/360 and/or System/370 input/output devices, as well as a detailed understanding of the basic assembler language. The second level, logical IOCS, uses a series of IBM-supplied macros to construct and process data files. Logical IOCS requires a minimum knowledge of the hardware I/O devices and is easily implemented within the problem program by the coding of macros. This system is also used by most of the DOS high-level languages to control I/O operations.

Physical Input/Output Control System (PIOCS)

Physical IOCS consists of input/output (I/O) routines that handle the actual transfer of data records between external storage devices (cards, tape, disk, etc.) and main storage. Program routines incorporated in the channel scheduler portion of the supervisor handle the following functions of PIOCS:

- Building a schedule of I/O operations for all devices on the system (CHANQ table).
- Starting the actual I/O operations on a device (SIO).
- Scheduling the start of all I/O operations and monitoring all events associated with I/O.

- Performing error recovery procedures (ERP) for all DOS supported devices, when necessary.

A user's problem program normally uses logical IOCS for file processing. Logical IOCS, in turn, uses physical IOCS to perform actual data transfers. There are occasions, however, when a user may need to bypass the logical IOCS routines to perform a particular I/O operation. (Physical IOCS is implemented at the assembler language level.) Three macro instructions are provided to allow the user to communicate directly with physical IOCS.

CCB This macro instruction creates a command control block. This is also considered an event control block (ECB). The CCB contains the user's information about special considerations and/or options that he has chosen for this I/O operation. It is comprised of constant statements (DS) that are used in two-way communications with the supervisor.

EXCP This macro instruction is converted to the proper SVC instruction (SVC 0) to request execution of a channel program. It supplies the location of the corresponding CCB to the supervisor.

The EXCP macro instruction provides more freedom in controlling devices than the logical IOCS macros, yet retains many of the operational advantages of the Disk Operating System. DOS provides scheduling and queuing of I/O requests, efficient use of channels and devices, data protection, interruption procedures, and error recognition and retry. To use physical IOCS, however, the programmer needs detailed knowledge of device control and system functions. He must supply his own channel programs, using the CCW (channel command word) assembler instruction statement (See <u>Channel Command Word</u> section).

WAIT This macro instruction tests CCB byte 2, bit 0 (traffic bit) to determine when an I/O operation has been completed. If the operation is not completed, the supervisor gets control until physical IOCS within the supervisor sets the traffic bit to indicate completion of the operation. The WAIT macro should always be issued for each requested I/O operation.

Whenever physical IOCS macro instructions are used, the programmer must construct the channel command words (CCW) for his input/output operations. He uses the assembler CCW statement to do this. See <u>Channel Command Word</u> for detailed information on the CCW. Figure 2.5 and the following text shows how an I/O operation can be traced through physical IOCS.

A request is made to physical IOCS to start an I/O operation by means of the EXCP macro instruction in the problem program. From information in the CCB, physical IOCS determines the channel for which the request was made and places the request on a queue for that device. If the channel(s) or device is not busy, the I/O is started and control returns to the problem program. If the channel is busy, control returns to the problem program, but the I/O request waits in the channel queue. When the request reaches the top of the channel queue, the I/O is started.

Control returns to the program requesting the I/O unless there was an error condition detected on the START I/O (SIO) instruction. The problem program normally continues processing until it requires that the requested I/O operation be complete (either the information being read into main storage is needed or the output area must be freed on an output operation). At this time, the WAIT macro should be issued. The WAIT macro causes the now waiting task to be removed from task selection until the proper interrupt is processed for this device by the supervisor.

Any problem program that is running will be interrupted when the I/O operation is complete (all data transferred to or from main storage and the external device and no permanent errors have been detected). At this point, the request is removed from the channel queue and normal task selection resumes.

If an error was detected that could not be corrected by the device error routines, the problem program or the computer operator would be notified via a message on SYSLOG. User error routines can be notified via the CCB to handle conditions such as wrong length record.

Physical IOCS always attempts to perform its function so that the time for executing an I/O operation is overlapped with the I/O operations on other channels and also allows the I/O operations to be overlapped with processing.

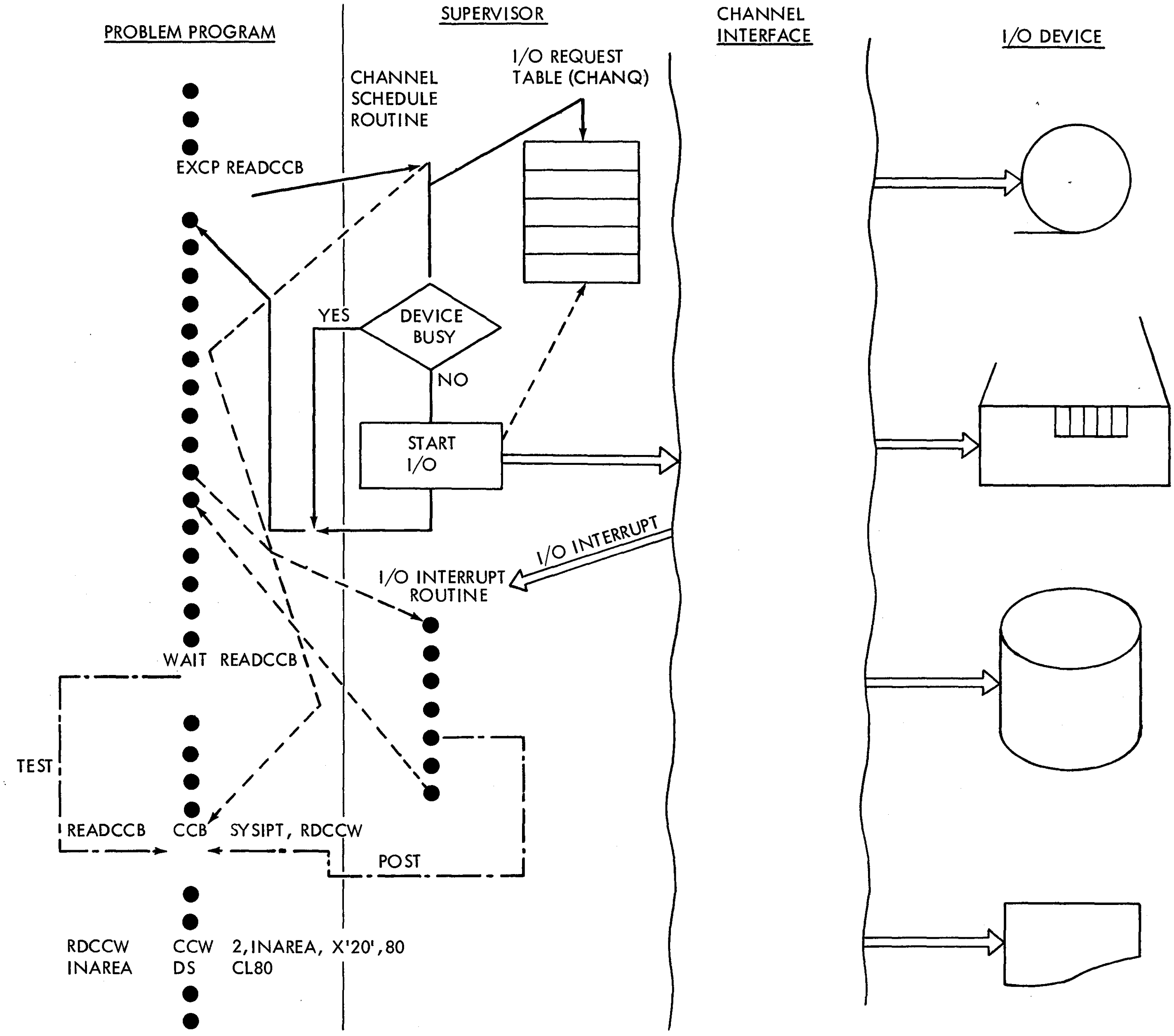


Figure 2.5. I/O Operation Using PIOCS

Command Control Block (CCB)

The CCB establishes communication between the problem program and physical IOCS. The CCB is two doublewords in length with eight major fields, as shown in Figure 2.6. The optional 8 bytes are generated if the user requests that a sense operation be performed on an I/O error. Data transferred from the device to main storage during a sense operation provides information concerning both unusual conditions detected in the last operation and the status of the device. All data in the CCB is in the hexadecimal format. The eight fields of the CCB are listed and described as follows:

1. Count Field (Bytes 0,1): Contains the residual count from the channel status word (CSW), that is stored by PIOCS when the CCB is removed from the queue. The residual count in conjunction with the original count specified in the last CCW used, indicates the number of bytes transferred to or from the area designated by the CCW. When an input operation is terminated, the difference between the original count in the CCW and the CSW is equal to the number of bytes transferred to main storage. For an output operation, the difference is equal to the number of bytes transferred to the I/O device.

2. Transmission Information (Bytes 2 and 3): Used for two-way communication between the supervisor and the problem program. Each bit within these two bytes represents either a condition that was detected by the supervisor and posted to the user, or a user option to be communicated to the supervisor. Refer to Figure 2.6. The designation pr.pr. indicates those bits that the problem programmer may set and are not reset by the supervisor for each I/O request. PIOCS indicates those bits that the supervisor is capable of setting/resetting just before each I/O operation associated with this CCB.

   The user options may be initialized at assembly time by specifying the third operand of the CCB macro. (See Supervisor and I/O Macro listed in the Preface for detailed information for coding the CCB). A third operand with a value X'0100' instructs PIOCS to return to the user after each I/O operation for this CCB whether or not any errors have occurred. It is then up to the user to handle all aspects of error recovery and/or retry, even to the point of determining if an error does exist. The only additional information that the supervisor provides under this circumstance is the CSW status information posted in bytes 4 and 5 of the user CCB. If the user specifies the fourth operand (Sense Address), the sense information is present in the sense area that the user has specified (if an error condition existed at the end of the I/O operation).

   Note: Bytes 2 through 5 are ANDed off (mask setting is X'1F050000') by PIOCS when the CCB is placed in the queue. Communication bits that were set on by the problem program are left on because an AND instruction is used by PIOCS for resetting bytes 2 through 5.

3. CSW Status Bits (Bytes 4,5): Contains the CSW status information that is stored by PIOCS before control returns to the problem program.

   Note: The particular bits that are turned on in bytes 2 through 5 indicate the conditions that were detected by PIOCS and/or the problem program by the specification of the third operand of the CCB macro.

4. Symbolic Unit Number (Bytes 6,7): Contains the 2-byte hexadecimal representation of SYSnnn (symbolic unit). This value represents the location of the logical unit in the LUB table (see Figure 1.19) and is placed in the CCB by the problem programmer's specification of the symbolic unit to be used for this I/O operation. The symbolic unit is converted to a 2-byte hexadecimal representation by the CCB macro. The first byte indicates the class of the symbolic unit (system class=X'00', programmer class=X'01'). The second byte of the representation indicates the number of the unit within one of the previously mentioned classes. An example of such a conversion can be illustrated with the symbolic unit SYS007, which converts to X'0107'.

5. Byte 8: Contains the length of the block prefix. This length is X'00'-X'63' for ASCII input tapes and X'00' or X'04' for ASCII output tapes. For EBCDIC tape files, this byte is always X'00'.

6. CCW Address (Bytes 9-11): Contains the address of the CCW that is associated with this CCB. This address is placed in the CCB by the specification of the second operand of the CCB macro. In the case of chained

CCWs, this operand specifies the address of the first CCW.

7. Byte 12: Contains information used by physical IOCS that must not be modified by the user.

8. CCW Address in CSW (Bytes 13-15): Contains the CCW address from the CSW stored by PIOCS before control returns to the problem program. A CCB that has been queued by PIOCS to service a problem program I/O request cannot be used for a second problem program I/O request until the first request has been completed.

9. Optional Sense CCW (Bytes 16-23): Bytes 16-23 are appended to the CCB by the CCB macro expansion when the user wants sense information on unrecoverable I/O errors returned. If the user specifies a sense address by coding the fourth operand of the CCB, an 8-byte CCW for reading sense information is generated as the last field of the CCB. The name field (sense address) of the area that the user supplies must have an assigned length attribute of at least one byte. Physical IOCS uses this length attribute in the CCW to determine the number of bytes of sense information the user wants at this sense address. For more detailed information concerning this sense information see the Principles of Operation listed in the Preface. For detailed information regarding the actual coding of the physical IOCS macros refer to Supervisor and I/O Macros listed in the Preface.

	Count	Transmission Information	CSW Status Bits	Symbolic Unit Address	Reserved for Logical IOCS	CCW Address	Reserved for Physical IOCS	CCW Address in CSW	Optional Sense CCW
Bytes	0 1	2 3	4 5	6 7	8	9 11	12	13 15	16 23
Used for	Residual Count	Transmition Information Between Physical IOCS and Problem Program	(Note 1)	Hexadecimal Representation of SYSnnn SYSRDR = 0000 SYSIPT = 0001 SYSPCH = 0002 SYSLST = 0003 SYSLOG = 0004 SYSLNK = 0005 SYSRES = 0006 SYSSLB = 0007 SYSRLB = 0008 SYSUSE = 0009 SYSREC = 000A SYSCLB = 000B SYS000 = 0100 SYS001 = 0101 SYS221 = 01DD	Buffer Offset: ASCII Input Tapes X'00'-X'63' ASCII Output Tapes Fixed: X'00' Variable: X'00' or X'04' Undefined: X'00'	Address of CCW Associated with this CCB	X'80'-CCB being used by ERP X'40'-Channel Appendage Routine Present for Teleprocessing Device X'20'-Sense Information Desired X'10'-Message Writer X'08'-EU Tape Error X'02'-Tape ERP Read Opposite Recovery X'01'-Seek Separation or Console Buffering	Address of CCW in the CSW Stored at Channel End, or Address of the Channel Appendage Routine for Teleprocessing Devices	8 Bytes Appended to the CCB when Sense Information is Desired

Byte 4	Byte 5
BIT DESIGNATION	BIT DESIGNATION
32 Attention	40 Program-controlled interruption
33 Status modifier	41 Incorrect length
34 Control unit end	42 Program check
35 Busy	43 Protection check
36 Channel end	44 Channel data check
37 Device end	45 Channel control check
38 Unit check	46 Interface control check
39 Unit exception	47 Chaining check

Byte 2

	Traffic Bit (Wait) (Note 5)	End-of-File (/* or /&) 3211- UCSB Parity Check (Line Complete) (Note 2)	Unrecoverable I/O Error	Accept Unrecoverable I/O	Return DASD Data Checks 2671 errors, or 1017/1018 errors to the user	Post at Device End (Note 5)	Return Tape Read Data Check, 1018 Data Check, 2540 or 2520 Equipment Check, DASD Data Checks on Read or Verify Command, or 3211 Passback Requested (Notes 3 & 6)	User Error Routine
Bits	0	1	2	3	4	5	6	7
Set On By	PIOCS	PIOCS	PIOCS	Pr. Pr.	Pr. Pr.	Pr. Pr.	Pr. Pr.	Pr. Pr.

Byte 3

	DASD - Data Check in Count Area MICR - SCU Not Operational 1285/1287/1288 Data Check 3211- Print Check/ Equipment Check	DASD - Track Overrun MICR - Intervention Required 1285/1287- Keyboard Correction in Journal Tape Mode 1017- Broken Tape 3211- Print Quality/ Equipment Check	DASD - End of Cylinder MICR - (Note 4) 1287/1288- Hopper Empty in Document Mode 3211- Line Position Error (Note 7)	2540, 2520- Equipment Check Tape - Read Data Check DASD - Any Data Check 1285/1287 Equipment Check 1017/1018 Data Check 3211- Print Check/ Data Check	Non-Recovery Questionable Condition: Card - Unusual Command Sequence DASD - No Record Found 1285/1287/1288- Document Jam or Torn Tape 3211- UCSB Parity Check (Command Retry)	No-Record-Found Condition (Retry on 2311, 2314, or 2319)	Carriage Channel 9 Overflow or Verify Error for DASD 1287 Document Mode - Late Stacker Select 1288 - End of Page	Command Chaining Retry from the next CCW to be executed
Bits	0	1	2	3	4	5	6	7
Set On By	PIOCS	PIOCS	PIOCS	PIOCS	PIOCS	Pr. Pr.	PIOCS	Pr. Pr.

PIOCS = Physical IOCS
Pr. Pr. = Problem Program

Note 1. Bytes 4 and 5 contain the status bytes of the Channel Status Word (Bits 32-47). If byte 2, bit 5 is on and device end results as a separate interrupt, device end will be ORed into CCB byte 4.

Note 2. Indicates /* or /& statement encountered on SYSRDR or SYSIPT. Byte 4, bit 7 (unit exception) is also on.

Note 3. DASD data checks on count not returned.

Note 4. For 1255/1259/1270/1275/1412/1419, disengage. For 1275/1419D, I/O Error in external interrupt routine (channel data check or busout check).

Note 5. The traffic bit (Byte 2, bit 0) is normally set on at channel end to signify that the I/O was completed. If byte 2, bit 5 has been set on, the traffic bit and bits 2 and 6 in byte 3 will be set on at device end. Also see Note 1.

Note 6. 1018 ERP does not support the Error Correction Function.

Note 7. This error occurs as an equipment check, data check, or FCB parity check.

Figure 2.6. Command Control Block (CCB) (Part 1 of 3)

Byte	Bit	Condition Indicated		On Values for Third Operand in CCB Macro	Mask for Test Under Mask Instruction
		1 (ON)	0 (OFF)		
2	0 Traffic Bit (WAIT).	I/O Completed. Normally set at Channel End. Set at Device End if bit 5 is ON.	I/O requested and not completed.		X'80'
	1 End of File on System Input. 3211 UCSB Parity Check (line complete)	/* or /& on SYSRDR or SYSIPT. Byte 4, Unit Exception Bit is also ON. Yes	 No		X'40'
	2 Unrecoverable I/O Error	I/O error passed back due to program option or operator option.	No program or operator option error was passed back.		X'20'
	3[1] Accept Unrecoverable I/O Error (Bit 2 is ON)	Return to user after physical IOCS attempts to correct I/O error.[2]	Operator Option: Dependent on the Error	X'1000'	X'10'
	4[1] 2671 data check. 1017/1018 data checks. Return any DASD data checks.	Operator Options: Ignore, Retry, or Cancel. Ignore or Cancel. Return to user.	Operator Option: Retry or Cancel. Cancel.	X'0800'	X'08'
	5[1] Post at Device End.	Device End condition is posted; that is, byte 2, bit 0 and byte 3, bits 2 and 6 set at Device End. Also byte 4, bit 5 is set.	Device End conditions are not posted. Traffic bit is set at Channel End.	X'0400'	X'04'
	6[1] Return: Uncorrectable tape read data check (2400-series, 3420, or 2495); 1018 data check; 2540 or 2520 punch equipment check; DASD read or verify data check; 3211 passback requested. (Data checks on count not returned.)	Return to user after physical IOCS attempts to correct 3211, tape, or DASD error. Return to user when 1018 data check.[4]	Operator Option: Ignore or Cancel for tapes, punches, or paper tape punch (1018). Retry or cancel for DASD.	X'0200'	X'02'
	7[1] User Error Routine	User handles error recovery.[3]	A physical IOCS error routine is used unless the CCB sense address operand is specified. The latter requires user error recovery.	X'0100'	X'01'
3	0 Data check in DASD count Field. Data check - 1285, 1287, or 1288. MICR - SCU not operational. 3211 Print Check (equipment check).	Yes - Byte 3, bit 3 is OFF; Byte 2, bit 2 is ON. Yes Yes Yes	No No No No		X'80'
	1 DASD Track overrun. 1017 broken tape. Keyboard correction 1285 or 1287 in Journal Tape Mode. 3211 print quality error (equipment check). MICR intervention required.	Yes Yes Yes Yes Yes	No No No No No		X'40'
	2 End of DASD Cylinder. Hopper Empty 1287/1288 Document Mode. MICR - 1255/1259/1270/1275/1412/1419, disengage. - 1275/1419D, I/O error in external interrupt routine. 3211 line position error.[5]	Yes Yes Document feeding stopped. Channel data check or Busout check. Yes	No No No No		X'20'

Figure 2.6. Command Control Block (CCB) (Part 2 of 3)

Byte	Bit		Condition Indicated		On Values for Third Operand in CCB Macro	Mask for Test Under Mask Instruction
			1 (ON)	0 (OFF)		
3	3	Tape read data check (2400-series or 2495); 2540 or 2520 punch equipment check; or any DASD data check.	Operation was unsuccessful. Byte 2, bit 2 is also ON. Byte 3, bit 0 is OFF.	No		X'10'
		1017/1018 data check.	Yes	No		
		1285, 1287, or 1288 equipment check.	Yes	No		
		3211 data check (print check).	Yes	No		
	4	Questionable Condition.	Card: Unusual command sequence (2540). DASD: No record found.			X'08'
		Nonrecovery	1285/1287/1288: Document jam or torn tape.			
		UCSB parity check (command retry).	Yes	No		
	5[1]	No record found condition	Retry command if no record found condition occurs (disk).	Set the questionable condition bit ON and return to user.	X'0004'	X'04'
	6	Verify error for DASD or Carriage Channel 9 overflow	Yes. (Set ON when Channel 9 is reached only if Byte 2, bit 5 is ON).	No		X'02'
		1287 document mode - late stacker select.	Yes	No		
		1288 End-of-Page (EOP).	Yes	No		
	7[1]	Command Chain Retry	Retry begins at last CCW executed.	Retry begins at first CCW of channel program.	X'0001'	X'01'

[1] User Option Bits. Set in CCB macro. Physical IOCS sets the other bits OFF at EXCP time and ON when the condition specified occurs.

[2] I/O program check, command reject, or tape equipment check always terminates the program.

[3] For System/360, the user must handle all error or exceptional conditions except Channel Control Check, Interface Control Check, I/O Program Check, and I/O Protection Check. For System/370, the user may handle Channel Control Checks and Interface Control Checks. The occurrence of a channel data check, unit check, or chaining check causes a byte 2, bit X'20' of the CCB to turn on, and completion posting and dequeuing to occur. I/O program and protection checks always cause program termination. Incorrect length and unit exception are treated as normal conditions (posted with completion). Also, the user must request device end posting (CCB byte 2, bit X'04') in order to obtain errors after channel end.

[4] Error correction feature for 1018 is not supported by physical IOCS. When a 1018 data check occurs and CCB byte 2, bit X'02' is on, control returns directly to the user with CCB byte 3, bit X'10' turned on.

[5] A line position error can occur as a result of an equipment check, data check, or FCB parity check.

Figure 2.6. Command Control Block (CCB) (Part 3 of 3)

Channel Command Word (CCW)

To aid the programmer in using physical IOCS, an assembler instruction statement, CCW (channel command word), is provided. This CCW assembler instruction statement is a convenient means to define and generate the 8-byte channel command words needed for the channel program. See Figure 2.7 for the format of the CCW assembler instruction.

Name	Operation	Operand
Any Symbol or Not Used	CCW	Four operands separated by commas specifying the command code, data address, flags, and count.

Figure 2.7. Format of the CCW

Command Code

The CCW command code (1 byte) may be expressed as a decimal number, hexadecimal representation, or as a symbol that has been equated to the proper hexadecimal or decimal value (e.g., 19, X'13', PR, where PR EQU 19 respectively). The assembler moves, or converts and moves, the command code to the first byte of the machine language CCW it is generating. Each I/O device has a specific set of commands to which it will respond; any other commands to that device are rejected.

Data Address

This field must contain a data address unless the CCW command is a control command (for example, skip to channel 1 for a printer). Normally this field is expressed as a symbol for ease of program relocation and reflects the address of the input/output area.

Flags

The third field of the CCW mnemonic is used to communicate special considerations to the channel regarding this CCW. The value of the flag byte may be expressed as a decimal number, hexadecimal character or as a symbol that has been equated to the proper decimal or hexadecimal value. Although it is referred to as the flag byte, only the five high order bits (bits 32-36 of the resultant CCW) represent individual flags. The three low order bits must be zero in their final hexadecimal form. The function of each flag bit is as follows:

Bit 32 (High Order Bit of the Flag Field): This is the chain data (CD) flag. Chaining refers to a series of CCWs in contiguous (consecutive) doubleword storage locations that are linked to each other forming a chain.

Data chaining permits the reading or writing of an I/O record from different areas of main storage.

If the CD flag bit is set to 1 (CD flag on), the data address and count in the next sequential CCW are also used in storing a data record. Both CCWs pertain to the same I/O record.

When data chaining, the command bytes of successive CCWs are ignored (unless it is a transfer in channel (TIC)), but the field must contain a valid command. Only the first CCW's command byte is used. It is important to note that data chaining pertains to only one I/O operation or data record. This could be one punch card or one tape record or one line of print and so forth.

Example: To read columns 1-30 of a punch card into storage beginning at location ONE and columns 31-80 into storage beginning at location TWO, two CCWs could be used. The first CCW would have its CD flag bit set to 1 as shown:

```
CCW1  CCW  2,ONE,X'80',30
      CCW  2,TWO,X'00',50
```

CCW1 causes thirty bytes to be read into storage starting at location ONE. The second CCW causes the next fifty bytes of the card to be read into storage starting at location TWO.

Note: The ability to data chain is dependent upon device and channel speeds.

Bit 33: Bit position 33 of the CCW is the command chaining (CC) flag. This bit when set to one causes the next sequential CCW to initiate another operation on the same I/O device.

For example, it is possible to read two cards into main storage as a result of one EXCP macro instruction. Two CCWs with the CC flag set in the first CCW are required.

```
CCW1  CCW  2,DATA,X'40',80
      CCW  2,DATA+80,X'00',80
```

In this example, CCW1 initiates a read command. The first card is read into storage locations DATA through DATA+79.

Because the CC flag bit in the first CCW is set to one, the second CCW is used to initiate another card read operation. The next card is read into storage locations DATA+80 through DATA+159.

Note that when data chaining, only one I/O operation occurs. The data from the one I/O record is placed in main storage under control of two or more CCWs.

When command chaining, each CCW controls a different I/O operation. The commands that are chained do not need to be the same. For instance, it is possible using one EXCP macro instruction to do a write-backspace-read combination with a magnetic tape unit by command chaining with three CCWs in one EXCP macro instruction.

When command chaining, an I/O interrupt cannot occur at the end of each command, but can occur when the last command is executed.

When command chaining, each command processes one physical record. If the byte count of the record does not agree with the count field of the CCW, an incorrect length indication results.

Bit 34: Bit position 34 of the CCW is the suppress incorrect length indication (SLI) flag. The SLI bit may be turned on to prevent an indication to channel of an unequal compare between the byte count specified in the CCW and the actual bytes read from the record.

Example of Incorrect Length: If you want to read columns 1-50 of a card into main storage starting at location BUFFER and bypass the remaining 30 columns, the following CCW can be used:

```
CCW1  CCW  2,BUFFER,X'20',50
```

An incorrect length indication results because one CCW causes the device to process one complete record and the device

1000 = 80
0100 = 40
0010 = 20

could not be stopped on the 51st byte. When an incorrect length indication occurs, the status bit in the CCB is set for testing by the programmer.

If the SLI flag bit is on, it suppresses the incorrect length indication for each data record. For example, if you want to read columns 1-40 of a card into main storage starting at location BUFFER1 and columns 41-60 into main storage starting at location BUFFER2, the following two CCWs could be used:

```
CCW1  CCW  2,BUFFER1,X'80',40
CCW2  CCW  0,BUFFER2,X'20',20
```

CCW1 causes 40 bytes to be read into main storage starting at location BUFFER1. CCW2 causes the next 20 bytes of the card to be read into storage starting at location BUFFER2. A breakdown of the flag bytes would be:

CCW1 Data Chaining

CCW2 Suppress incorrect length indication

Note that the SLI flag is not required for the first CCW. The incorrect length indication is relevant to an entire data record and not to a given CCW. The SLI flag is present in the second CCW because it is the last CCW for the record and the total value of the two CCW count fields does not equal the actual record length.

Bit 35: Bit position 35 of the CCW is the SKIP flag. The SKIP flag suppresses the transfer of information into main storage. It can be used together with the data chaining feature to read selected portions of an I/O record into main storage. To do this, the CD flag is used along with bit 35 of the CCW. For example, if columns 51-80 of a punch card are to be read into locations DATA through DATA+29, the card reader reads all 80 columns of information and attempts to transfer 80 bytes of data into main storage. The channel needs 2 CCWs.

```
CCW1  CCW  2,DATA,X'90',50
      CCW  2,DATA,X'00',30
```

CCW1 has both the CD and the SKIP flag bits on.

Because the skip flag bit is on, CCW1 is used to suppress the transfer of data into main storage. The count field causes 50 bytes to be bypassed.

Because the CD flag bit is on, the second CCW is fetched after the first 50 bytes have been skipped. The second CCW causes bytes 51 through 80 to be read into storage starting at location DATA.

Bit 36: Bit position 36 of the CCW is the program control interrupt (PCI) flag. CPU normally receives an I/O interrupt at the end of the I/O operation. However, bit position 36 of the CCW can be used to signal an I/O interrupt before the end of the operation.

When a CCW is fetched that has its PCI flag bit on, CPU receives an I/O interrupt as soon as it can accept it. In this way, CPU is notified of the progress of an I/O operation. This notification could be used to initiate the processing of the data records that have been read into main storage by the preceding CCWs in the chain.

I/O interrupts normally occur at the end of the operation. An interrupt caused by PCI would not affect the I/O operation. The interrupt is taken by the CPU and the I/O operation continues in the same manner as if the interrupt had not occurred.

The PCI-caused interrupt occurs as soon as possible after start of execution of the command containing the PCI flag. The occurrence of the interrupt may be delayed, depending on the model of System/360 and/or the particular device that it is operating.

A more detailed explanation of the CCW flags can be found in Principles of Operation listed in the front of this manual.

Count

The count field gives the total number of bytes in the storage area (the physical size of the records being read or written). The count field specifies any number of bytes up to 65,535. Except for a CCW specifying a transfer in channel, the count field cannot contain the value zero.

To illustrate the use of physical IOCS, Figures 2.8 and 2.9 show a sample program with a related flowchart.

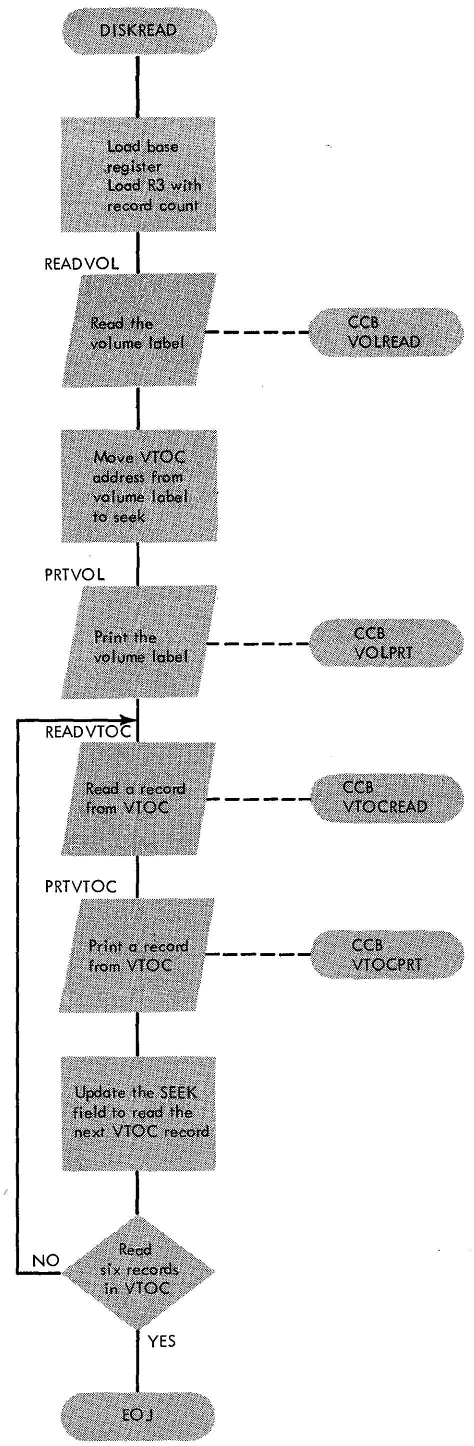


Figure 2.8. Flowchart for EXCP Coding Example

```
                                                                                                              PAGE     1

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                           FDOS CL3-5 10/31/69

000000                                1 DISKREAD START 0                                                            0010
                                      2          PRINT NOGEN
000000                                3          BALR  4,0                                                          0030
000002 0540                           4          USING *,4                                                          0020
000002 4130 0006            00006     5          LA    3,6
                                      6 READVOL  EXCP  VOLREAD     READ THE VOLUME LABEL                           00005
                                     10          WAIT VOLREAD      WAIT FOR THE READ TO COMPLETE
00001A D204 40D2 410B 000D2 0010B    16          MVC SEEK+2(5),VOLUME+11   SET UP SEEK/SEARCH ARGUMENT
                                     17 PRTVOL   EXCP  VOLPRT      PRINT THE VOLUME LABEL                           0070
                                     21          WAIT  VOLPRT      WAIT FOR PRINT TO COMPLETE                       0080
                                     27 READVTOC EXCP  VTOCREAD    READ THE VTOC                                    0090
                                     31          WAIT  VTOCREAD    WAIT FOR THE READ TO COMPLETE                    0100
                                     37 PRTVTOC  EXCP  VTOCPRT     PRINT A VTOC RECORD , KEY AND DATA               0110
                                     41          WAIT  VTOCPRT     WAIT FOR PRINT TO COMPLETE                       0120
00005C 4360 40D6            000D6    47          IC    6,SEEK+6    UPDATE THE RECORD COUNT IN THE
000060 4166 0001            00001    48          LA    6,1(6)      SEEK/SEARCH ARGUMENT                             0140
000064 4260 40D6            000D6    49          STC   6,SEEK+6    STORE UPDATED ID INTO SEEK/SEARCH ARG.
000068 4630 4034            00034    50          BCT   3,READVTOC
                                     51 EOJ      EOJ                                                                0180
00006F 0700                          54          CNOP  0,4
                                     55 VOLREAD  CCB   SYS004,VOLCCW                                                0190
                                     66 VTOCREAD CCB   SYS004,VTOCCCW
                                     77 VOLPRT   CCB   SYS006,VOLPTCCW                                              0210
                                     88 VTOCPRT  CCB   SYS006,VTOCPCCW                                              0220
000080 070000D040000006              99 VOLCCW   CCW   LONGSEEK,SEEK,CC,6    SEEK TO CYLINDER 0, HEAD 0
0000B8 290000D760000004             100 SEARCHKY CCW   SRCHKEYE,KEYSCH,CC+SLI,4   SEARCH KEY EQUAL (VOL1)
0000C0 080000B800000000             101          CCW   TIC,SEARCHKY,0,0           TIC BACK TO SEARCH
0000C8 0600010020000050             102          CCW   READDATA,VOLUME,SLI,L'VOLUME READ VOL LABEL DATA FIELD
0000D0 00000000000000               103 SEEK     DC    X'00000000000000'                                            0270
0000D7 E5D6D3F1                     104 KEYSCH   DC    CL4'VOL1'
0000DB 0000000000
0000E0 070000D040000006             105 VTOCCCW  CCW   LONGSEEK,SEEK,CC,6        SEEK TO THE VTOC CYLINDER/HEAD
0000E8 310000D260000005             106 SEARCHID CCW   SRCHIDE,SEEK+2,CC+SLI,5    SEARCH ID EQUAL FOR VTOC RCD
0000F0 080000E800000000             107          CCW   TIC,SEARCHID,0,0           TIC BACK TO SEARCH
0000F8 0E0001002000008C             108          CCW   READKDTA,VTOCRCD,SLI,L'VTOCRCD READ KEY/DATA FROM VTOC
000100                              109 VTOCRCD  DS    0CL140                                                       0330
000100                              110 VOLUME   DS    0CL80                                                        0340
000100                              111 KEY      DS    CL44                                                         0350
00012C                              112 DATA     DS    CL96                                                         0360
00018C E5D6D3E4D4C540D3             113 VOLHEAD  DC    C'VOLUME LABEL'                                              0370
000198 E5E3D6C340D2C5E8             114 KEYHEAD  DC    C'VTOC KEY'                                                  0380
0001A0 E5E3D6C340C4C1E3             115 DATAHEAD DC     C'VTOC DATA'                                                0390
                                    116 *  ********************************************************************
                                    117 *  PRINT VOLUME HEADER AND SPACE ONE
                                    118 *  PRINT VOLUME LABEL AND SPACE THREE
                                    119 *  ********************************************************************
0001A9 00000000000000
0001B0 0900018C6000000C             120 VOLPTCCW CCW   PRINTSP1,VOLHEAD,CC+SLI,L'VOLHEAD
0001B8 1900010020000050             121          CCW   PRINTSP3,VOLUME,SLI,L'VOLUME
                                    122 *  ********************************************************************
                                    123 *  PRINT VTOC KEY HEADING
```

```
                                                                                                              PAGE     2

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                           FDOS CL3-5 10/31/69

                                    124 *  PRINT KEY OF VTOC RECORD
                                    125 *  PRINT DATA HEADING
                                    126 *  PRINT DATA PORTION OF VTOC RECORD
                                    127 *  ********************************************************************
0001C0 0900019860000008             128 VTOCPCCW CCW   PRINTSP1,KEYHEAD,CC+SLI,L'KEYHEAD
0001C8 190001006000002C             129          CCW   PRINTSP3,KEY,CC+SLI,L'KEY
0001D0 090001A060000009             130          CCW   PRINTSP1,DATAHEAD,CC+SLI,L'DATAHEAD
0001D8 1900012C20000060             131          CCW   PRINTSP3,DATA,SLI,L'DATA
                                    132 * ********************************************************************
                                    133 *  SAMPLE DASD CHANNEL COMMAND EQUATES
                                    134 * ********************************************************************
000007                              135 LONGSEEK EQU   07
000029                              136 SRCHKEYE EQU   41
000008                              137 TIC      EQU   08
000006                              138 READDATA EQU   06
000031                              139 SRCHIDE  EQU   49
00000E                              140 READKDTA EQU   14
                                    141 * ********************************************************************
                                    142 *  SAMPLE 1403 OR 1443 PRINTER COMMAND EQUATES
                                    143 * ********************************************************************
000009                              144 PRINTSP1 EQU   09
000019                              145 PRINTSP3 EQU   25
                                    146 * ********************************************************************
                                    147 *    FLAG OPERANDS
                                    148 * ** ADD SYMBOLS TOGETHER FOR COMBINATIONS SUCH AS CD+SLI*************
                                    149 * ********************************************************************
000080                              150 CD       EQU   128  CHAIN DATA
000040                              151 CC       EQU   64   COMMAND CHAIN
000020                              152 SLI      EQU   32   SUPPRESS LENGTH INDICATION
000010                              153 SKIP     EQU   16   SKIP PORTIONS OF THE RECORD
000008                              154 PCI      EQU   8    PROGRAM CONTROLLED INTERRUPT
000000                              155 ENDCCWS  EQU   0    END OF CCW LIST
                                    156          END                                                                0460
0001E0 00000070                     157                =A(VOLREAD)
0001E4 00000090                     158                =A(VOLPRT)
0001E8 00000080                     159                =A(VTOCREAD)
0001EC 000000AC                     160                =A(VTOCPRT)
```

Figure 2.9. Physical IOCS Sample Program

Logical Input/Output Control System (LIOCS)

Logical IOCS (LIOCS) provides data management functions required to locate and access logical records in a file (just as the problem programmer would). LIOCS uses physical IOCS to accomplish actual data transfer and device control. The data management functions include:

- Organization (blocking and deblocking) of logical records.
- Control (switching) of I/O areas when more than one area is used.
- Handling of the open and close requirements, and end-of-file and end-of-volume conditions.
- Resolving symbolic references to physical I/O devices.
- Translating data in the I/O areas from ASCII to EBCDIC (on input) and from EBCDIC to ASCII (on output).

With IBM-supplied macro instructions, logical IOCS eliminates the repetitive detail coding required for standardized input and output routines. These logical IOCS macros, referred to as imperative macros, supply the facilities for reading, writing, blocking and deblocking records, file labeling, and error checking. To make use of these facilities, logical IOCS imposes certain requirements that must be handled by the problem programmer. These include descriptive entries in other IBM-supplied macros, called declarative macros, to specify the characteristics of a file that is to be processed by logical IOCS. The IBM assembler program uses the descriptive information when processing the macro statements to tailor the logical IOCS routines for the specific application.

Logical IOCS handles data transferred to or from I/O devices as logical files of data. When logical IOCS determines that an I/O area contains no logical record (or records, in the case of blocked records) needing processing, it issues a physical IOCS macro to execute the actual data transfer. Figure 2.10 shows the relationship between logical and physical IOCS for a logical IOCS imperative GET macro issued to an input file when one I/O area is used.

Logical IOCS is a generative system that uses the capabilities of a macro language. It is this generative nature that provides the problem programmer with the ability to include only those routines needed for a specific job or job step.

LOGICAL IOCS PROCESSING METHODS

The IBM Disk Operating System provides logical IOCS routines to process records in sequential order, in random order by the Direct Access Method (DAM), or randomly and sequentially by the Indexed Sequential File Management System (ISFMS). Sequential processing applies to all files on serial I/O devices (such as card reader, tape, printer, etc), and to records on IBM 2311, 2314, and 2319 disk or 2321 data cell devices when they are processed serially. The types of processing performed by DAM and ISFMS apply only to files of Direct Access Storage Device (DASD) records.

Logical disk files can be data set secured. A data-secured file cannot be accessed accidentally by problem programs. The data set security function provides a record on the system log each time a data-secured file is opened. It is the user's responsibility to insure data security.

In addition, logical disk files can have the track hold option. When track hold is specified in the DTF, a track that is being modified by a task in one partition cannot be concurrently accessed by a task or subtask in the same or another partition provided that all programs competing for a track specify the track hold option. Any program that does not use the track hold option will not be denied access to a track, and can modify a track that is being held by another program. Therefore, all programs accessing the same file should have the track hold option.

Sequential Processing

Sequential processing reads, writes and processes successive records in a logical file. For example, card records are processed in the order the cards are fed. Tape records are processed starting with the first record after a header label and continuing through the records to the trailer label. DASD records are processed starting with a beginning DASD address and continuing in order through the records on successive tracks and cylinders to the ending address.

A sequential file on DASD is contained within one or more sets of limits called extents that are specified by the user in job control EXTENT/XTENT statements or commands. If the logical file consists of more than one set of limits, logical IOCS automatically processes each set as required by the user. The records within

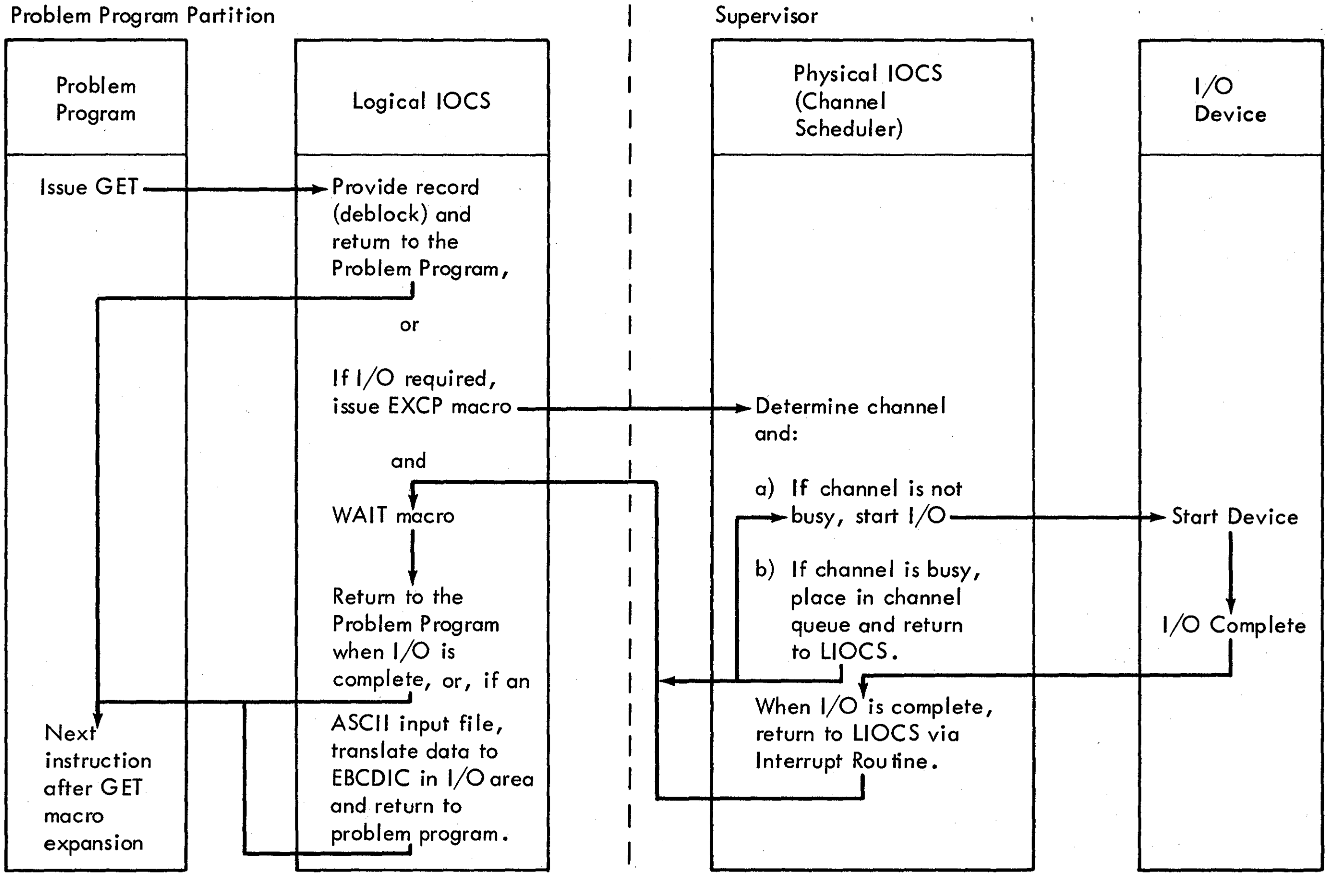


Figure 2.10. Retrieving a Record Using Logical IOCS

each set must be adjacent and contained within one volume (disk pack or data cell). The sets are not required to be adjacent or on the same volume. Sequential processing of a DASD file written by the Direct Access Method can be performed if the physical structure of a sequential file is present.

Direct Access Method (DAM)

The Direct Access Method (DAM) processes records contained on IBM 2311, 2314, 2319, or 2321 DASD devices that are usually organized in a random manner.

Logical IOCS locates a DASD record for processing by referring to a record location reference supplied by the problem program. The location reference consists of two parts: a track reference and a record reference. The track reference specifies the track (or the first of multiple tracks) to be searched for the record. The record reference may be the record key, if records contain key areas, or the record identifier (ID) that is in the count area of each DASD record. Logical IOCS seeks the specified track and searches for the record on that track, or on the succeeding tracks in the cylinder.

Indexed Sequential File Management System (ISFMS)

DASD records contained within an indexed sequential file can be processed in a random order or in sequential order by control information. Both orders use the control information of the records (such as employee number, part number, etc), that is available in the key area of each DASD record. Any record stored at any location in the logical file can be processed by the random method. The user supplies ISFMS with the key (control information) of the desired record. ISFMS searches for the record and makes it available for processing.

In sequential processing, ISFMS makes a series of records available. The records are available one after the other in order by the control information (key) in the records. The user specifies the first record to be processed. ISFMS retrieves the succeeding records (on demand) from the logical file, in key order, until the problem program terminates the operation.

ISFMS creates an organized file and then adds to, reads from, and updates records in that file. The file is organized from records that are presorted by control information. As the DASD records are loaded, ISFMS constructs indexes for the logical files. The indexes permit individual records to be found in subsequent processing operations. The indexes are created in such a way that records can be retrieved randomly or sequentially. If records are added to the file at a later date, ISFMS updates the indexes to reflect the new records.

MAIN STORAGE REQUIREMENTS

Logical IOCS routines are generated as part of the problem program. Imperative macro expansions, that serve as linkages to the logical or physical IOCS routines, are generated in-line at the point the macro is used in the problem program. The open, close, EOF/EOV, and other special purpose routines are called into the B-transient (logical transient) area as required. The physical IOCS routines used by logical IOCS are generated as part of the supervisor program.

MODULAR/TABULAR SYSTEM

The terms modular and tabular mean that the system uses tables in conjunction with data handling modules to implement its functions.

The modular/tabular system has the following advantages:

- Saves assembly time by allowing the data handling modules to be generated separately and stored in the relocatable library for subsequent use.
- Uses one module for many files if the device types are the same and the files are similar.

The major advantage is the ability to use one logical IOCS module with many different files. It not only saves a large portion of main storage but also makes the system more versatile.

The modular/tabular combination for a specific file is generated by two macros: the file definition macro in the form DTFxx, and the module generation macro in the form xxMOD.

The file definition macros describe the logical file, indicate the type of processing to be used for the file, and specify main storage areas (work area, I/O area, etc.) for the file. A number of file definition macros define the files processed by logical IOCS, and one defines files processed by physical IOCS (DTFPH). The file to be processed determines the type of file definition macro to use.

The module generation macros generate the data handling logic modules. These modules contain generalized routines needed to perform the functions of the logical IOCS imperative macros. The generalized routines in the logic modules are altered and made more specific through various parameters (specified by the problem programmer) included in the xxMOD macro statements. It is possible, therefore, to generate many variations of a particular type of logic module, each specifically suited to the need of the problem programmer.

DTF (Define the File) Macros

Whenever logical IOCS imperative macro instructions are used in a problem program to control the transfer of records in a file, that file must be defined by a declarative DTF macro instruction. The DTF macro instruction describes (through various parameters specified by the problem programmer) the characteristics of the logical file, indicates the type of processing for the file, and specifies the main storage areas and routines. Detailed descriptions of the logical IOCS file definition (DTF) macros and their parameters appear in Supervisor and I/O Macros listed in the Preface.

In general, the IBM-supplied file definition (DTF) macros are device oriented. In addition, three macros, DTFSR, DTFBG and DTFEN are supported by the Disk Operating System to provide upward compatibility from the IBM System/360 Basic Operating System (8K system). A brief description follows for each of the DTF macros available to users of the IBM Disk Operating System.

DTFCD: Define The File for a Card Device. To define a file associated with the records on a card unit.

DTFCN: Define The File for a CoNsole. To define a file associated with the console (1052, 3210, 3215) printer.

DTFDA: Define The File for Direct Access method. To define a file when DASD (Direct Access Storage Device) records are to be processed by the Direct Access Method.

DTFDI: Define The File for Device Independent system files. To define files assigned to the device independent system logical units SYSRDR, SYSIPT, SYSPCH, and SYSLST.

DTFIS: Define The File for Indexed Sequential file management system. To define a file organized and processed by the indexed sequential file management system.

DTFMR: Define The File for Magnetic Recognition. To define a file associated with a magnetic ink character recognition (MICR) device.

DTFMT: Define The File for Magnetic Tape. To define a file associated with a magnetic tape device.

DTFOR: Define The File for an Optical Reader. To define a file associated with an optical character reader device.

DTFPH: Define The File for processing by PHysical IOCS. To define a magnetic tape or DASD file with standard labels that is to be processed by physical IOCS when the OPEN and CLOSE macros are used for label processing. DTFPH parameters define the magnetic tape and DASD files. No other files processed by physical IOCS require definition.

Only the following logical IOCS functions can be performed for files defined by a DTFPH macro:

- Check the header labels on input files, and close these files when requested.
- For ASCII tape files only: Translate labels to EBCDIC for input files and check them. Create labels and translate them to ASCII for output files.
- Create header labels on output files, and create trailer labels when the file is closed.
- Force end-of-volume on an output file when requested.

When a DTFPH macro instruction is encountered at assembly time, the assembler builds a DTF table that includes only the parameters needed for the OPEN, CLOSE(R), FEOV, and FEOVD routines. The OPEN, CLOSE(R), FEOV, and FEOVD macro expansions call the open and close routines into the supervisor B-transient area at object time.

DTFPR: Define The File for a PRinter. To define a file associated with a printer device.

DTFPT: Define The File for Paper Tape. To define a file associated with a paper tape device.

DTFSD: Define The File for Sequential DASD. To define sequential files on a direct access storage device (DASD).

DTFSR: Define The File in a SeRial type file device. To define a file for sequential processing of records on any IOCS supported I/O device.

The DOS DTFSR macro definition accepts either the BOS or BPS DTFSR macro as valid input. After determining the device type required, the DOS DTFSR macro calls the appropriate DOS DTF macro from the source statement library. The DTF macro called by the DOS DTFSR then sets up a DTF table in the usual manner.

The DOS DTFSR macro definition only allows upward compatibility and should not be used as a statement in the user's DOS source deck.

DTFBG: Define The File for BeGin-definition. Must be punched with DTFBG in the operation field and DISK in the operand field. The name field is left blank. DTFBG is included in DOS to provide compatibility with the BOS DTFSR macro instruction.

DTFEN: Define The Field ENd. To show there are no more DTF source statements to process. The DOS DTFEN macro definition allows upward compatibility for BOS and BPS users.

MOD (Module Generation) Macros

Each DTF (except DTFCN, DTFPH and DTFSR) is linked to a logical IOCS module generated by an xxMOD macro instruction. These modules provide the necessary instructions to perform the input/output functions required by the problem program. For example, the module can read or write data, test for unusual input/output conditions, block or deblock records, or place logical records in a work area.

Some of the module functions are provided on a selective basis, according to the parameters specified in the xxMOD macro instruction. The problem programmer has the option of selecting (or omitting) some of these functions according to the requirements of his program. Omitting some of these functions results in smaller main storage requirements for a particular module.

There are two options for MOD macros. The user can:

1. Insert the MOD macro instruction with its file parameters in the problem program source deck. In this case, the logic module is assembled in-line with the problem program.
2. Choose at system generation time to generate the logic modules needed for his file formats and system configuration. To do this, source decks using macro parameters to describe the file attributes are punched for each MOD macro statement. The logic module macro definition generates its own unique name, or the user can name the module in the name field of the MOD macro statement. The user name overrides the name the macro definition normally generates.

For each type of xxMOD macro, the problem programmer can generate many logic modules by issuing the macro with varying parameters for each required module. The logic modules can be cataloged in the relocatable library. The CATALR control cards are automatically generated when the module is assembled.

At assembly time, the assembler produces an EXTRN (External Symbol) card for every V-type constant (or EXTRN statement), in the user program. The assembler expansion of the DTF statement produces an EXTRN card with the name of the logic module needed to support the parameters that were specified in the DTF macro. The IBM-generated module names indicate the type of file and the support that each is capable of supplying for the DTF. Refer to Figure 2.11 for a breakdown of these names. Because of the descriptive nature of the IBM standard names, the programmer should be careful when specifying his own names for the logic modules or overriding the IBM standard names. At the time this program is link-edited, the linkage editor resolves these EXTRN symbols (AUTOLINK). If the program is not to be executed immediately, the linkage editor catalogs the program into the core image library.

Figure 2.12 gives the module name prefixes used in the IBM-supplied programs. Figure 2.13 shows the relationship of the DTFxx and the xxMOD macros. The DTFCN macro is unique in that it generates its own logic module and combines it with the DTF table. The logic module for DTFCN is always punched in the object deck, along with the DTF table (A of Figure 2.13).

Logic Module	Prefix	4th Character	5th Character	6th Character	7th Character	8th Character	Subsetting/ Supersetting + Permitted * Not Permitted
CDMOD	IJC	F RECFORM=FIXUNB V RECFORM=VARUNB U RECFORM=UNDEF	A CTLCHR=ASA Y CTLCHR=YES C CONTROL=YES Z Neither CTLCHR nor CONTROL specified	B RDONLY=YES and TYPEFLE=CMBND C TYPEFLE=CMBND H RDONLY=YES and TYPEFLE=INPUT I TYPEFLE=INPUT N RDONLY=YES and TYPEFLE=OUTPUT O TYPEFLE=OUTPUT	Z Neither WORKA nor IOAREA2 specified W WORKA=YES I IOAREA2=YES B Both WORKA=YES and IOAREA2=YES Z WORKA=YES not specified (CMBND file only)	0 DEVICE=2540 1 DEVICE=1442 2 DEVICE=2520 3 DEVICE=2501 4 DEVICE=2540 and CRDERR=RETRY 5 DEVICE=2520 and CRDERR=RETRY	* * * * * IJC F A B B 0 V Y C I 1 U + H W 2 C I Z 3 Z N 4 O 5
PRMOD	IJD	F RECFORM=FIXUNB V RECFORM=VARUNB U RECFORM=UNDEF	A CTLCHR=ASA Y CTLCHR=YES C CONTROL=YES S STLIST=YES Z Neither CTLCHR nor CONTROL nor STLIST specified	C = B if ERROPT=YES (ERROPT=name in DTFPR) and PRINTOV=YES = P if PRINTOV=YES and ERROPT is not specified (ERROPT=RETRY or is omitted in DTFPR) = E if ERROPT=YES (ERROPT=name in DTFPR) = Z if neither ERROPT (ERROPT=RETRY or is omitted in DTFPR) nor PRINTOV is specified	I IOAREA2=YES Z IOAREA2 not specified	V RDONLY=YES and WORKA=YES W WORKA=YES Y RDONLY=YES Z Neither RDONLY nor WORKA specified	* * + * * IJD F A B I V V Y E Z W U S + Y + P Z C Z Z
PTMOD	IJE	S SCAN=YES Z SCAN not specified	T TRANS=YES and SCAN not specified Z TRANS not specified	F RECFORM=FIXUNB and SCAN=YES U RECFORM=UNDEF and SCAN=YES Z SCAN not specified and/or DEVICE=1018	1 DEVICE=1017 2 DEVICE=1018 Z DEVICE=2671 or not specified	Z	* * * * IJE Z Z Z Z Z Z T Z Z S Z F Z S Z U Z Z Z Z 1 Z T Z 1 S Z F 1 S Z U 1 S Z Z 2 Z T Z 2
MTMOD (GET/ PUT)	IJF	F RECFORM=FIXUNB or FIXBLK N RECFORM=UNDEF (ASCII files) V RECFORM=VARUNB or VARBLK (EBCDIC mode) U RECFORM=UNDEF (EBCDIC mode) X RECFORM=FIXUNB or FIXBLK (ASCII files) R RECFORM=VARUNB or VARBLK (ASCII mode) S RECFORM=SPNUNB or SPNBLK (spanned records)	B READ=BACK Z READ=FORWARD or not specified	C CKPTREC=YES Z CKPTREC not specified	W WORKA=YES Z WORKA not specified	M ERREXT=YES and RDONLY=YES N ERREXT=YES Y RDONLY=YES Z Neither ERREXT nor RDONLY specified	* + + + + IJF F B C W M N Z Z Z Y R + U N X Z + S V
MTMOD (Work Files)	IJE	W TYPEFLE=WORK	E ERROPT=YES Z ERROPT not specified	N NOTEPNT=YES S NOTEPNT=POINTS Z NOTEPNT not specified	Z	M ERREXT=YES and RDONLY=YES N ERREXT=YES Y RDONLY=YES Z Neither ERREXT nor RDONLY specified	+ + + IJF W E N Z M Z S Y Z + N Z

Figure 2.11. Generated Name Structure for Logic Modules (Part 1 of 2)

Logic Module	Prefix	4th Character	5th Character	6th Character	7th Character	8th Character	Subsetting/ Supersetting + Permitted * Not Permitted
SDMOD (GET/ PUT)	IJG	C SDMODFx specifies HOLD=YES F SDMODFx does not specify HOLD=YES R SDMODUx specifies HOLD=YES U SDMODUx does not specify HOLD=YES P SDMODVx specifies HOLD=YES (spanned records) Q SDMODVx does not specify HOLD=YES (spanned records) S SDMODVx specifies HOLD=YES V SDMODVx does not specify HOLD=YES	U SDMODxU I SDMODxI O SDMODxO	C ERROPT=YES and ERREXT=YES E ERROPT=YES Z Neither ERROPT nor ERREXT specified	M TRUNCS=YES and FEOVD=YES T TRUNCS=YES W FEOVD=YES Z Neither TRUNCS nor FEOVD specified	B CONTROL=YES and RDONLY=YES C CONTROL=YES Y RDONLY=YES Z Neither RDONLY nor CONTROL specified	+ * + * + IJG C U C T B F I E Z Y + O Z + R C U Z + P Q V + P S V
SDMOD (Work Files)	IJG	T HOLD=YES W HOLD=YES not specified	C ERROPT=YES and ERREXT=YES E ERROPT=YES Z Neither ERROPT nor ERREXT specified	N NOTEPNT=YES R NOTEPNT=POINTRW Z NOTEPNT not specified	C CONTROL=YES Z CONTROL not specified	T RDONLY=YES and UPDATE=YES U UPDATE=YES Y RDONLY=YES Z Neither RDONLY nor UPDATE specified	+ + + + + IJG T C N C T W E R Z Y Z Z + U Z
ISMOD	IJH	A RECFORM=BOTH and IOROUT=ADD or ADDRTR B RECFORM=FIXBLK and IOROUT=ADD or ADDRTR U RECFORM=FIXUNB and IOROUT=ADD or ADDRTR Z RECFORM not specified (IOROUT specifies LOAD or RETRVE)	A IOROUT=ADDRTR I IOROUT=ADD L IOROUT=LOAD R IOROUT=RETRVE	B TYPEFLE=RANSEQ G IOAREA2=YES and TYPEFLE=SEQNTL or IOROUT=LOAD R TYPEFLE=RANDOM S TYPEFLE=SEQNTL Z Neither TYPEFLE nor IOAREA2 specified (IOROUT=ADD or LOAD)	B CORINDX=YES and HOLD=YES C DORINDX=YES O HOLD=YES Z Neither CORINDX nor HOLD is specified	F CORDATA=YES, ERREXT=YES and RDONLY=YES G CORDATA=YES and ERREXT=YES O CORDATA=YES and RDONLY=YES P CORDATA=YES S ERREXT=YES and RDONLY=YES T ERREXT=YES Y RDONLY=YES Z Neither CORDATA nor RDONLY nor ERREXT specified	+ + + + + IJH A A B B F B I R O O Z + + + + + A B C S A R S Z Y U * + + Z L G G S P + + G T Z Z
DAMOD	IJI	F RECFORM=FIXUNB B RECFORM=UNDEF (handles both UNDEF and FIXUNB) S RECFORM=SPNUNB V RECFORM=VARUNB	A AFTER=YES Z AFTER not specified	I IDLOC=YES E IDLOC=YES and FEOVD=YES R FEOVD=YES Z Neither FEOVD nor IDLOC specified	H ERREXT=YES and RELTRK=YES P ERREXT=YES R RELTRK=YES Z Neither ERREXT nor RELTRK specified	W HOLD=YES and RDONLY=YES X HOLD=YES Y RDONLY=YES Z Neither HOLD nor RDONLY specified	+ + + + + IJI B A E H W F Z I P X + Z Z Z S + + + V E H W R R Y Z Z Z
DIMOD	IJJ	F Fixed unblocked record format	C ASA and System/360 control character support for printers and punches	B TYPEFLE=OUTPUT I TYPEFLE=INPUT	I IOAREA2=YES Z IOAREA2 not specified	C RDONLY=YES D RDONLY not specified	+ + * IJJ F C B I C I Z D
ORMOD	IJM	F RECFORM=FIXUNB X RECFORM=FIXBLK U RECFORM=UNDEF D RECFORM=UNDEF and BLKFAC=YES	C CONTROL=YES Z CONTROL not specified	I IOAREA2=YES W WORKA=YES B IOAREA2=YES and WORKA=YES Z Neither IOAREA2 nor WORKA specified	T Device is in tape mode D Device is in document mode	Z	* + * * IJM D C B D Z F Z I T U W X Z
MRMOD	IJU	S Single address adapter D Dual address adapter	Z	Z	Z	Z	* IJU D Z Z Z Z S

Figure 2.11. Generated Name Structure for Logic Modules (Part 2 of 2)

$I\left\{\begin{matrix}H\\J\\K\\L\end{matrix}\right\}xxxxxx$

	IHD	S/360 COBOL processing subroutines
	IJB	System Service and System Control
DATA MANAGEMENT	IJC	Card logic
	IJD	Printer logic
	IJE	Paper tape logic
	IJF	Magnetic tape logic
	IJG	Sequential DASD logic
	IJH	Indexed Sequential DASD logic
	IJI	Direct Access DASD logic
	IJJ	Device independent logic
	IJK	PL/I processing subroutines
	IJL	Teleprocessing routines
DATA MANAGEMENT	IJM	Optical Reader logic
	IJN	Audio Response Vocabulary File Utility
	IJO	Disk Sort/Merge
	IJP	Tape Sort/Merge
	IJQ	D-level Assembler
	IJR	RPG compiler
	IJS	S/360 COBOL compiler
	IJT	Basic FORTRAN, and FORTRAN subroutines
DATA MANAGEMENT	IJU	Magnetic Readers logic
	IJV	Autotest
	IJW	Utilities
	IJX	PL/I compiler
	IJY	F-level Assembler
	IJZ	OLTEP
	IKL	COBOL LCP
	ILA	ANS COBOL compiler
	ILB	ANS COBOL processing subroutines
	ILC	CE Serviceability Programs
	ILF	F-level FORTRAN IV and FORTRAN IV subroutines
	ILH	Tape and Disk Sort/Merge

Figure 2.12. DOS Relocatable Library Module Name Prefixes

Reentrant Modules

A reentrant module is a logic module that can be used asynchronously, or shared by more than one file. The RDONLY=YES parameter in the module generation macro generates a reentrant logic module. The RDONLY (read only) parameter implies that the generated logic module is never modified in any way regardless of the processing requirements of any file(s) using the module. To provide this feature, unique save areas external to the logic module are established, one for each task using the module. Each save area must be 72 bytes and doubleword aligned. Before a logic module is entered or an imperative macro is issued to the file, the task must provide the address of its unique save area in register 13.

Reentrant modules include: CDMOD, DAMOD, DIMOD, ISMOD, MTMOD, and SDMOD.

Interrelationships of the DTF and Module Macro Instructions

The DTFCD, DTFDA, DTFDI, DTFIS, DTFMR, DTFMT, DTFOR, DTFPR, DTFPT, and DTFSD declarative macros are similar in that they each generate a DTF table that references an IOCS logic module (refer to B of Figure 2.13). The first 20 bytes of each table have the same format, that is, a command control block (CCB) and a logic module address. The remainder of each table is tailored to the particular device and file type.

The description that follows is general and includes all of the DTF types included in B of Figure 2.13.

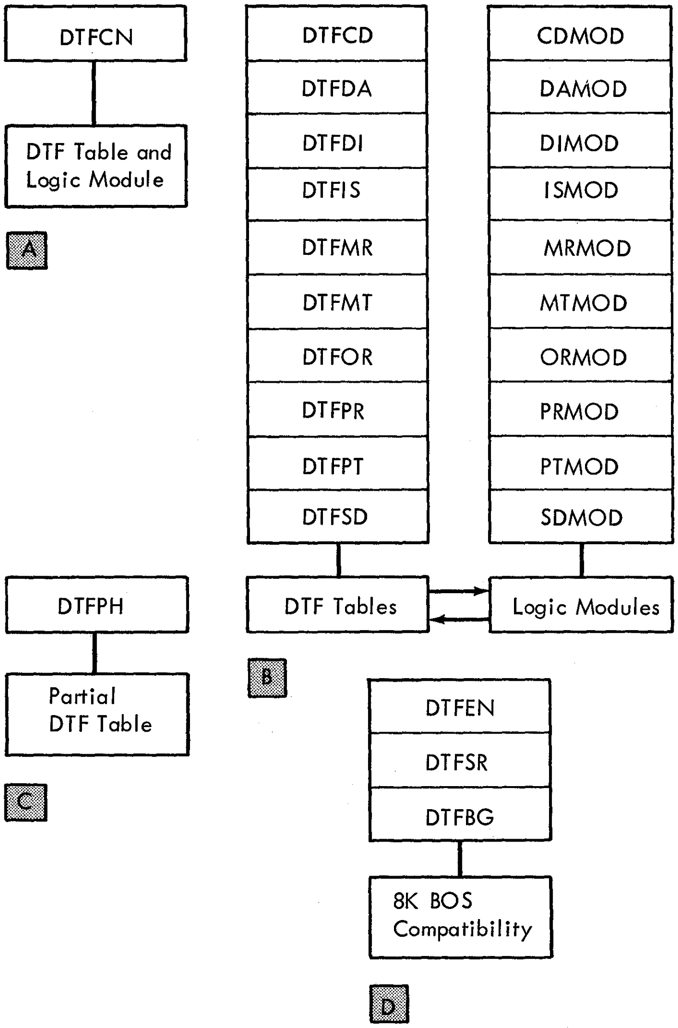


Figure 2.13. DTF and Module Macro Relationships

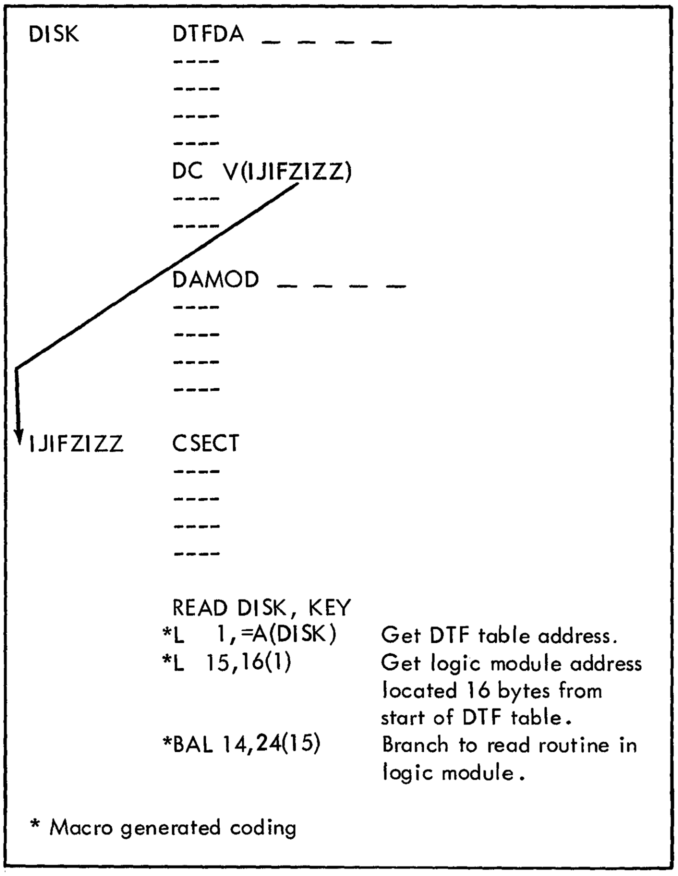
When one of these DTF macro instructions is encountered at assembly time, the assembler builds a DTF table tailored to the DTF parameters. The table contains:

- Device CCB (Figure 2.6).
- A V-type statement used by the Linkage Editor to resolve the linkage to the logic module associated with this DTF.
- Logic indicators, that is, one I/O area, two I/O areas, device type, etc.
- Addresses of all of the areas and controls used by this device (except work files).

Regardless of the method of assembling logic modules and DTF tables (with the main program or separately), a symbolic linkage results between the DTF table and the logic module. The Linkage Editor resolves these linkages at edit time.

To accomplish the linkage between the DTF table and the logic module, the assembler generates a V-type address constant in the DTF of a named CSECT in the logic module. To resolve this linkage, the linkage symbols (module names) must be identical.

Figure 2.14 shows the relationship of the program, the DTF and the logic module. It also shows a DTFDA macro with a filename of DISK. The assumed parameters have generated a request for a DAMOD named IJIFZIZZ. Based on this name, the linkage editor was able to locate the module. The read statement generated coding to load the address of the DTF table into register 1. This gives the program access to the DAMOD address, and the program branches to the required routine within the module.



```
DISK      DTFDA _ _ _ _
          ----
          ----
          ----
          ----
          DC  V(IJIFZIZZ)
          ----
          ----

          DAMOD _ _ _ _
          ----
          ----
          ----
          ----

IJIFZIZZ  CSECT
          ----
          ----
          ----
          ----

          READ DISK, KEY
         *L   1,=A(DISK)     Get DTF table address.
         *L   15,16(1)       Get logic module address
                             located 16 bytes from
                             start of DTF table.
         *BAL 14,24(15)      Branch to read routine in
                             logic module.

* Macro generated coding
```

Figure 2.14. Coding Example of DTF and Module Relationship

IMPERATIVE MACROS

The problem programmer issues imperative logical IOCS macro instructions to initiate such functions as opening a file, making records available for processing, writing records that have been processed, controlling physical device operations, etc.

For each imperative macro issued by the problem programmer, the assembler program generates an in-line expansion that links the instruction to the DTF table (thus the logic module) for the specified file. For an operand, the imperative macro instruction must always contain the filename in the DTFxx macro describing the file.

Figure 2.15 summarizes the macro instructions provided by IBM for logical IOCS. Figure 2.16 further defines the general function of each of the macro instructions and indicates the devices with which they are used. The Supervisor and I/O Macros listed in the Preface gives a detailed explanation of each of the imperative macros.

MACROS	DTFCD	DTFCN	DTFDA	DTFDI	DTFIS	DTFMR	DTFMT	DTFOR	DTFPH	DTFPR	DTFPT	DTFSD	DTFSR
CHECK						X	X					X	
CLOSE(R)	X		X	X	X	X	X	X	X	X	X	X	X
CNTRL	X		X				X	X		X		X	X
DISEN						X							
DSPLY								X					
ENDFL					X								
ERET			X				X					X	
EXETL					X								
FEOV							X					X	X
FREE			X									X^1	
GET	X	X		X	X	X	X	X			X	X	X
LBRET			X				X					X	X
LITE						X							
NOTE							X					X	
OPEN(R)	X		X	X	X	X	X	X	X	X	X	X	X
POINTR							X^1					X^1	
POINTS							X^1					X^1	
POINTW							X^1					X^1	
PRTOV										X			X
PUT	X	X		X	X		X			X	X	X	X
RDLNE								X					
READ			X		X	X	X					X	
RELSE							X					X	X
RESCN								X					
SEOV							X						
SETFL					X								
SETL					X								
TRUNC							X					X	X
WAITF			X		X	X		X					
WRITE			X		X		X					X	

1. Work files only.

Figure 2.15. Logical IOCS Imperative Macros and DTFs

Macro Instruction		TYPE OF PROCESSING WITH LOGICAL IOCS																	
		Sequential													Direct Access Method	Indexed Sequential File Management System			
		1052 Printer - Keyboard	1285 Optical Reader	1287 Optical Reader	1403/1404/1443/ 1445/3211 Printer	1412/1419/1255/ 1259 Magnetic Ink Character Reader	1270/1275 Optical Reader/Sorter	1442/2520/2540 Punch	1442/2501/2520/ 2540 Reader	2311/2314/2319 Disk Drive	2321 Data Cell	2400 and 3420 Magnetic Tape Units	2671/1017 Paper Tape Reader	1018 Paper Tape Punch		Load File	Add Records	Random Retrieve	Sequential Retrieve
Initialize	OPEN(R)		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	LBRET [1]									X	X	X			X				
Process	GET	X	X	X		X	X		X[2]	X	X	X	X						X
	PUT	X			X			X	X[4]	X[3]	X	X		X					X
	READ			X		X	X			X		X			X			X	
	WRITE									X		X			X	X	X	X	
	CHECK					X	X			X		X							
	RELSE [5]									X		X							
	TRUNC [6]									X		X							
	WAITF			X		X	X								X	X	X	X	
	RDLNE		X	X															
	RESCN			X															
	DSPLY			X															
Set Mode	SETFL															X			
	ENDFL															X			
	SETL																		X
	ESETL																	X	
Non Data Operations	CNTRL		X	X	X			X	X	X	X	X			X				
	CHNG [7]											X							
	PRTOV				X														
	DISEN					X	X												
	LITE					X[9]	X[9]												
	ERET									X	X	X				X	X	X	X
Work Files for DASD and Magnetic Tape	READ									X		X							
	WRITE									X		X							
	CHECK									X		X							
	NOTE									X		X							
	POINTR									X		X							
	POINTW									X		X							
	POINTS									X		X							
Complete	CLOSE(R)		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	FEOV											X							
	FEOVD									X	X								
	FREE									X[8]	X[8]			X					
	LBRET [1]									X	X	X							
	SEOV											X							

Notes:

1. Applies only if DTFSR, DTFMT, DTFDA, or DTFPH LABADDR or XTNTXIT is specified.
2. In the 2520 or 2540, GET normally reads cards in the read feed. If TYPEFLE =CMBND is specified, GET reads cards at the punch - feed - read station.
3. PUT rewrites on input DASD record if UPDATE is specified.
4. In the 1442, 2520, or 2540, PUT punches an input card with additional information if TYPEFLE = CMBND is specified.
5. Applies only to blocked input records.
6. Applies only to blocked output records.
7. Provided only for upward compatibility for BPS and BOS.
8. Workfiles only.
9. Applicable to 1419 and 1275 with the Pocket Light Feature.

Figure 2.16. Logical IOCS Imperative Macros and Devices

SEQUENTIAL FILE ORGANIZATION

Card Files

The DTFCD/CDMOD macros provide the user with the ability to read (GET) a record from a card reader or punch (PUT) a record on a card punch (up to 80 characters in both cases). The user has the option of specifying one or two I/O areas (IOAREA1 and IOAREA2), also called buffer areas, to hold the data transferred to main storage from the device, or vice versa. The second buffer area allows the user to overlap processing while the following record is read or the previous record is punched. The user can process the data read (or data to be punched) in the buffer area if only one area is specified. If two buffer areas are specified, a register (IOREG) can be specified to point to the start of the data field (leftmost position) in the current buffer area, or the data can be placed into a work area (WORKA) for processing (or punching). In the latter case, DTFCD/CDMOD transfers the record from the buffer area to the work area or vice versa. The capability for reading a card record and punching information into the same card (CMBND) is also provided if the user has an IBM 1442, and IBM 2520, or an IBM 2540 with the punch-feed-read (PFR) feature. Many data processing installations use the stacker selection capabilities made available by the following three methods:

1. The user may issue a CNTRL macro instruction after a GET or before a PUT to select the desired stacker,
2. First character control (CTLCHR) may be used, where the first character of a record may be an ASA (American Standard Association) or System/360 control character for stacker selection, and
3. The selection for a given file may be specified (SSELECT) when the DTF table is generated.

Console Typewriter

The DTFCN macro allows reading and writing of a record from or to an IBM 1052 Printer-Keyboard or an IBM 3210 or 3215 Console Printer-Keyboard by issuing a GET or PUT macro instruction. The record may be processed in the buffer area or in a work area (WORKA). Only one buffer area can be used with this access method. This file does not require the use of OPEN and CLOSE routines.

Line Printers

The DTFPR/PRMOD access method provides the ability to print a record by issuing a PUT macro instruction. The record to be printed can be presented to the access method via a work area (WORKA) or can be placed in the buffer area. Two buffer areas (IOAREA1 and IOAREA2) can be specified to allow overlap processing. In this case, a work area or a general register must be used to indicate the proper buffer area. Three types of printer-form control are provided by the access method:

- CNTRL macro instruction for line spacing or page skipping.
- PRTOV (printer overflow) macro instruction for page skipping or exiting to a user-supplied routine (indicated in the macro instruction) that can perform certain end-of-page and/or start-of-page functions.
- First character control (CTLCHR) that can be used where the first character of a record may be an ASA or System/360 control character for line spacing or page skipping.

Paper Tape Files

The DTFPT/PTMOD access method provides the ability to retrieve a data record from a supported Paper Tape device. Two buffer areas (IOAREA1 and IOAREA2) can be used for overlap processing. In this case, a general register must be specified to point to the record in the buffer area currently being used. This access method also handles shifted code for figure shift (FTRANS and SCAN) and/or letter shift (LTRANS and SCAN), or nonshifted code to be translated into System/360 and System/370 code (TRANS). The user must supply the various translation tables needed.

Magnetic Tape Files

The DTFMT/MTMOD access method provides the ability to create or retrieve magnetic tape records in sequential order. The file is created by indicating via the DTFMT parameters that it is an output file (TYPEFLE) and then issuing PUT macro instructions. If the records are to be retrieved, an input file (TYPEFLE) is indicated and a GET macro instruction is issued. In either case, the records may be processed in a work area (WORKA) or in the buffer area by using a general register (IOREG). When an input file is to be processed, the access method can also support the read-backward feature for magnetic tape units.

When an ASCII file is to be processed, each record read into the I/O area is translated from ASCII to EBCDIC. When an ASCII file is to be created, the record in the output area is translated from EBCDIC to ASCII just before it is written on the tape.

Other optional features that greatly enhance the DTFMT/MTMOD access method are:

- Specifying two buffer areas for overlap processing capabilities (IOAREA1 and IOAREA2).
- Alternate tape switching between two tape units.
- Bypassing of checkpoint records on input files (CKPTREC). An ASCII file must not contain checkpoint records.

Other macro instructions allow the user to rewind, rewind and unload, execute various other magnetic tape device functions (CNTRL), or release (SKIP) the remaining records in an input block (RELSE). This can be useful if records are grouped by specific categories. An inverse function of RELSE allows the programmer to truncate or write short blocks of records (TRUNC) for an output file.

In addition to GET/PUT functions, DTFMT/MTMOD provides the highly useful feature of issuing READ/WRITE macro instructions to create a file or retrieve records from a work file (WORK). Overlap processing can take place while the I/O operation is being performed. The user awaits the READ/WRITE operation by issuing a CHECK macro instruction that ensures completion of the operation. The particular features that enhance this facility are the NOTE and POINT macro instructions. By issuing a NOTE macro instruction, the location of the data block in the file can then be obtained. POINT macro instructions provide the ability to reposition to a given block in the file. POINTR can position the tape to the block indicated, POINTW can position the tape after the block indicated, and POINTS can position the tape to the beginning of the file.

Sequential Disk

Records can be created or retrieved and updated from a direct access device by using the DTFSD/SDMOD access method. The file can be created by specifying an output file (TYPEFLE) and issuing PUT macro instructions. If the records are to be retrieved, an input file is indicated (TYPEFLE) and GET macro instructions are issued. It is also possible to update those records in the same location on DASD that were retrieved by a GET macro instruction. In this case, a PUT macro instruction must be issued for the file after the GET for the record to be updated and preceding the GET for the next record. The access method also provides double buffering (IOAREA1 and IOAREA2) ability for overlap processing. The user can either process the record in a work area (WORKA) or use a general register to point to the record in the current buffer area. Another macro instruction allows the user to skip the remaining records in an input block (RELSE). This can be useful if records are grouped by specific categories. An inverse function of RELSE, allows the programmer to truncate or write short blocks of records (TRUNC) for an output file. The CNTRL macro instruction may also be used to seek the track address of the next record to be processed. In the case of a data cell, CNTRL can restore a strip if the user knows that processing on it has been completed.

DTFSD/SDMOD also provides the READ/WRITE, CHECK, NOTE, and POINT macro instructions described earlier under magnetic tape files, although an IBM 2311/2314/2319/2321 DASD is used as an I/O device in this case. NOTE/POINT uses cylinder, track, and record identification for noting and locating blocks in the file. Also, if a NOTE follows a WRITE, the unused space on a track can be returned when the file is being created or when records are to be written in the count, key, and data format of DASD. If a POINTR or POINTW is issued before a READ or WRITE (UPDATE), the READ or WRITE macro instruction processes the block indicated. If a formatting WRITE (write count, key, and data) is issued, the WRITE macro instruction writes a new record after the block indicated.

Another feature available with this access method that can be used in either of the two mentioned processing modes, is the split cylinder mode. This mode allows two or more files to share the same cylinder. Each file occupies the same track positions through the range of assigned cylinders. The technique has the advantage of minimizing access-arm movement when cross referencing among two or more files that perform similar functions. The use of this facility is indicated in one of the job control statements (XTENT/EXTENT) needed at execution time of the user's program. See System Control and System Service listed in the Preface for detailed information on the job control statements.

DIRECT ACCESS METHOD (DAM) FILE ORGANIZATION

Great flexibility in reading or writing a record from or to a direct access device is achieved by the DTFDA/DAMOD access method. With the WRITE (AFTER) macro instruction, the user may create a file in any manner desired. For example, a part number or control field can be converted to a DASD address using a randomizing algorithm, and the record can be written at that disk address. Also, the user can create a sequential file with keys (control fields) to be processed later in a skip-sequential fashion.

In skip-sequential processing, a file that has been sorted on a key control information field is created in a sequential manner. Records are retrieved by scanning or searching the file using the key of each record desired. In this way, only these records are retrieved and the job throughput is improved when a large percentage of the file is processed.

Record retrieval is accomplished by issuing a READ macro instruction. Two types of READ macro instructions may be issued for record retrieval. The user may read the record by simply supplying the track and record location (ID), or by supplying the track location along with the record key (KEY) to be used for record retrieval. Record updating is performed by two corresponding WRITE macro instructions (ID and KEY). They work in the same way as the READ macro instructions.

In addition to the READ/WRITE macro instructions, there is a WAITF macro instruction. This macro instruction is issued after each READ/WRITE. The WAITF macro instruction serves a two-fold purpose:

1. It allows overlap processing or issuing of another I/O device macro instruction (within the limitations of the system configuration) while the present I/O operation is taking place, and
2. It communicates termination of the READ/WRITE operation with which it was associated, and returns any error/status information indicating whether the I/O operation was successful to the problem program in the field specified by DTFDA ERRBYTE.

Three other options, enhancing the capabilities of the access method, are:

1. The ability to utilize the multiple track search facility of a DASD (SRCHM).
2. The ability to return the record location (the first five bytes of the record identification, namely CCHHR) when retrieving a record by its key (IDLOC). These features facilitate skip-sequential processing. For example, by issuing a READ (KEY) macro instruction and starting at the beginning of the file, the direct access device searches multiple tracks until the record is found. The record location is returned and is used as a reference point in starting the search for the next record to be retrieved. In this manner of processing, only the required records are read, whereas in the normal sequential mode of processing, all records are read even if they are not all processed.
3. The ability to seek ahead by issuing a CNTRL macro instruction (or restore a strip to an IBM 2321 Data Cell) and then continue normal processing or issue other I/O device macro instructions.

When creating a file using the WRITE (AFTER) macro instruction, the access method automatically maintains the amount of space available on each track on which the records are stored along with the address of the last record on each track (provided the DASD has been properly initialized). This access method does not provide blocking and deblocking facilities. They must be performed by the user.

Random Addressing Techniques

In addition to the specification of the DTFDA and DAMOD macros, the problem programmer must supply the DASD address of the record to be read or written before issuing each READ/WRITE imperative macro. The following discussion presents various methods for determining a DASD address from a record control field.

File addressing involves the file of records that must be stored and retrieved in a data processing system, and the direct access storage device itself. The data records that must be stored in a direct access storage device are usually identified by a control field, such as part number, and employee number. Normally, the numbers or characters in the control field are unevenly distributed. For example, a seven-positioned control field may be used to identify 25,000 items in a parts master file. However, with a seven-position number, it is possible to identify ten million items. In this example, only 0.25% of the available numbers are used.

The direct access storage devices, on the other hand, are usually composed of physical locations that are identified by an evenly distributed set of numbers. The addressing problem converts an unevenly distributed set of numbers to an evenly distributed sequential set of numbers within the address limits of the direct access device. Many addressing techniques have been developed to accomplish this task. In choosing a technique for address conversion, it is important to remember that an ideal distribution of control fields is a completely uniform one. Uniform distribution means that the difference between any pair of successive control fields taken in ascending order is constant.

The worst distribution of control fields is a random one. There is no way to transfer from random keys to addresses with better than random distribution. In practice, purely random control field sets and completely uniform ones are rare. A data file is likely to have control fields that distribute in groups or clusters of irregular length and separation. This kind of grouping of numbers introduces a degree of uniformity. The irregular length and separation of the number groups implies a degree of randomness. A well chosen conversion technique produces an address set that reflects both elements and has a distribution intermediate between random and uniform. To be ideal for use in direct access storage devices, the conversion technique should produce a unique storage address for every record in a file. This is seldom possible. Most control-field conversion routines result in assigning some address to more than one record. These duplicate addresses are sometimes referred to as synonyms. The selected conversion routine should convert the control fields (keys) of the records in a data file to a series of addresses with a minimum number of synonyms and within the desired storage address range. The following sections discuss briefly the most successful conversion routine, followed by a discussion of synonym handling.

Random Addressing Formula

The simplest method of file organization is that in which a unique DASD address is obtained from the control data of each record. This is referred to as the random addressing method. If the control numbers of a set of data records in a file are consecutive numbers without gaps, they may be converted to DASD addresses by simple arithmetic. For example, if the account numbers for a customer file run from 10000 to 17563 (7564 account numbers), and ten account records can be stored on each disk track, 757 tracks are needed. By subtracting 10,000 from an account number and then dividing by 10, a numeric address in the range 000 to 756 is obtained. To place this file on a 2311/2314/2319 disk drive, starting at track address 1200 (cylinder 120 head 0), a constant 1200 is added to the quotient and a constant (1) is added to the remainder. This constant (1) is required because record zero (R0) of each track is reserved to facilitate the handling of defective recording areas that may occur during the life of the disk pack. Using this approach, a record containing the data for account number 16349 would be stored at track reference 1834 in record-reference ten, calculated as follows:

16349 - 10000 = 6349

6349 ÷ 10 = 634 with remainder 9

634 + 1200 = 1834 = track reference

9 (remainder + 1) = 10 = record reference

When processing this file randomly, any record can be found with a single seek. When it is possible to process sequentially, only one seek is needed per cylinder. Record retrieval time is thus at a minimum. This is an optimum situation, and it rarely occurs in actual practice.

Normally, the control data of a file of records can seldom be used directly as DASD addresses. If a file does not have control fields that can be used directly as DASD addresses, it is sometimes possible to preassign addresses. For example, the item number 513XP could become 513XP-13472, that could then be converted to a track and record reference.

Prime Number Division

If the control fields of a file of records are not consecutive or contain numerous unused numbers, as is usually the case, the random addressing technique under the topic Random Addressing Formula makes inefficient use of storage locations. All possible numbers are assigned locations, and those numbers not used leave empty record areas in the storage unit. Files established with control numbers composed of coded information usually have a much higher potential range of items than is required for storage. To handle this situation, initial conversion is made on the control numbers to reduce the range to a practical size. This conversion is often referred to as randomizing.

Randomizing generally refers to the techniques developed to convert a set of control numbers with numerous unused numbers to a tightly packed set, to result in very few unused storage areas. There are many techniques used for this conversion of numbers, a few being: folding, extracting, squaring, and radix transformation. One method, sometimes called prime number division* or divide remainder, is adaptable and usually satisfactory for converting a file of numbers.

To illustrate the prime number division technique, suppose the customer file in the example under the topic Random Addressing Formula used a coded control number of ten digits. The first three could be a geographical code (branch office number), the next two could describe the nature of the business, the next one could be a size-of-customer code, and the final four could be sequentially assigned within class.

Thus, account number 139 457 0307 would be the 307th account assigned branch office 139. It would belong to a customer-of-size code 7 in industry class 45. Because this ten-digit number cannot be used efficiently to describe 7564 accounts, it is converted by dividing by the closest prime number to the number of storage locations available. Assume 10,000 locations available, then divide by 9973. The remainder serves as the control number and a technique similar to the example under the topic Random Addressing Formula calculates a track and record reference.

1394570307 ÷ 9973 = 139834 with a remainder of 5825.

This remainder is operated on as in the example, under the topic Random Addressing Formula, assuming 10 records per track.

5825 ÷ 10 = 582 with a remainder of 5.

To load the file starting at track address 1100, add 1100 to 582 for a sum of 1682. The track reference for this record is 1682, and the record is the sixth record on the track (remainder of 5 + 1 = 6).

To summarize prime number division:

1. Select a divisor equal to or greater than the number of records to be stored (10 to 20% greater is recommended). The best divisors are primes. Even numbers or multiples of five should never be used - divisors must end in 1, 3, 7, or 9. The divisor is called the range.

2. Divide the control number by the range and use the remainder to generate the track address.

Prime number division always works; that is, it always converts control numbers into the desired range because in division, the remainder is always less than the divisor and the highest valued remainder is the divisor-1. Dividing any number, no matter what size, by a desired range, always produces remainders in the desired range. Using a prime number as the divisor usually results in relatively few duplicate remainders, and therefore relatively few address synonyms.

A prime number is not always the best choice of divisor for a given set of keys. Also, it is not necessarily true that all primes produce equally good results. However, primes avoid serious maldistribution and may be safely used with little analysis of the control field set of the data files.

---

* A prime number is a number divisible only by itself, or one.

Synonym Records

The transformation of record control fields to direct access storage device addresses usually produces some synonym records.

The file organization used with a data file employing indirect addressing (addresses converted by a random addressing formula) must be able to accommodate the synonyms or duplicate addresses.

The first consideration in organizing the file is sometimes called the packing factor. The number of synonyms produced by a random addressing conversion routine can be reduced by assigning more DASD storage space than is actually required by the file. The percentage of the file area actually used for record is called the packing factor. The packing factor for an efficiently organized file can vary from 65% to 95%. A packing factor of 80% usually proves to be a good starting point. After all efforts have been made to design a file conversion technique with few synonyms, an approach to handling the remaining synonyms must be chosen.

One such technique is often referred to as the chaining method. As each record is read into the computer for loading into a direct storage access device, its control field or key is converted to a physical address. These converted addresses are called home addresses.

Note: The home addresses discussed here are not directly related to the track home address used to control the physical operation of the DASD. These home addresses are related to the record identifier (ID) associated with DAM.

The first record converted to a particular address is stored in the home address location. The additional records converted to this address are stored in synonym locations. The address of the first synonym location is stored in the home address location. The address of the second synonym location is stored in the first synonym location, etc. Chaining requires that in the home address and all synonym locations, space be reserved for the address of the next location or link in the chain (Figure 2.17).

Retrieval of records is accomplished by converting the control information (record key) to the home address. The record in the home address location is read into main storage, and its control information is compared to that of the record being sought. If the control fields are not equal, the address of the first overflow record is extracted from the home record, and another read command is issued using this address. The process is repeated until the desired record is found.

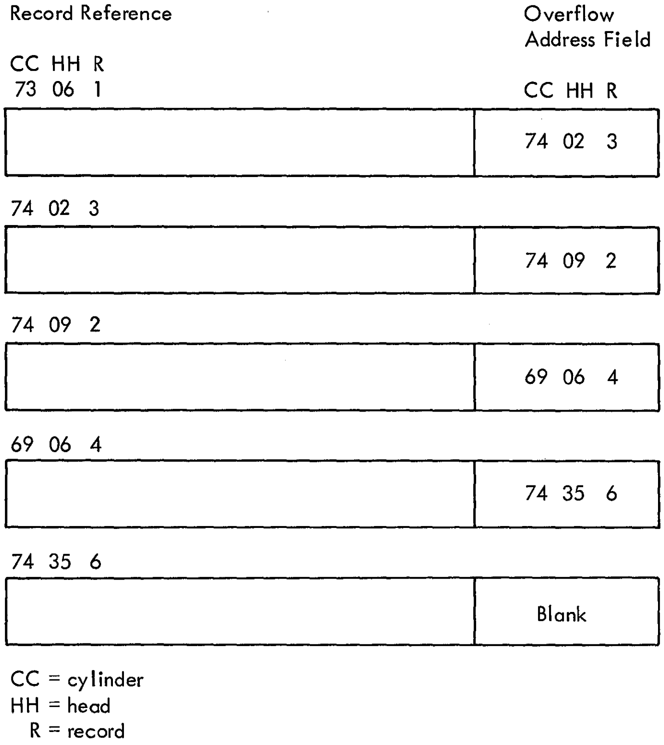


Figure 2.17. Direct Access Address Chaining

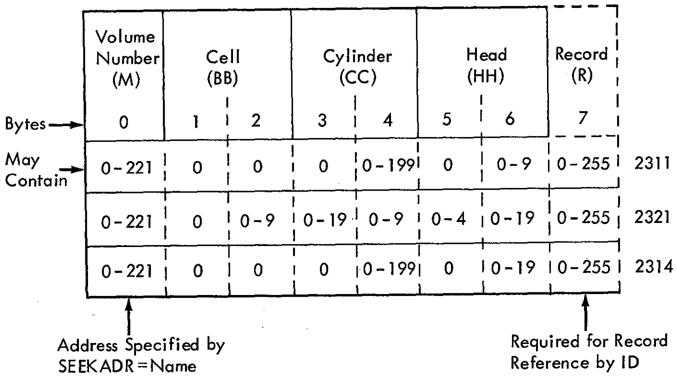
Two other techniques that solve the synonym problem are similar in concept to the chaining method but do not require a chaining field to be present in each record.

1. Preassigned tracks synonym technique. When there is not room to store a record in its home location, a specific preassigned synonym track is used. The synonym track(s) should be defined to be in the same cylinder containing the home location to reduce the number of seeks required to locate a record.
2. Consecutive spill synonym technique. When a synonym occurs, a sequential search is made starting at the next record within the cylinder until an empty record storage location is found. If the last track in a given cylinder overflows, a return is made to the first track. This technique does not require the use of a chaining field, nor is a seek required to locate the synonym record (must be on same cylinder).

DASD Address Specification For Read/Write Operations

The direct access method requires DASD addresses for all read/write operations. These addresses may be supplied in one of two ways: as an actual physical address (MBBCCHHR) or as a relative track address (see Relative Track Addressing).

The actual physical address can be specified as an 8-byte binary address (see Figure 2.18) in the form MBBCCHHR. These 8-byte addresses are used either as the starting point for a search on record key or as the actual address for a READ or WRITE ID. When searching for a record key, the programmer may specify that the search be only within the specified track, or from track to track, starting at the address given and continuing either until the record is found or until the end of the cylinder is reached.



Bytes	Volume Number (M)	Cell (BB)		Cylinder (CC)		Head (HH)		Record (R)	
	0	1	2	3	4	5	6	7	
May Contain	0-221	0	0	0	0-199	0	0-9	0-255	2311
	0-221	0	0-9	0-19	0-9	0-4	0-19	0-255	2321
	0-221	0	0	0	0-199	0	0-19	0-255	2314

Figure 2.18. DASD Address Formats

For certain types of operations, the system can be requested to return the ID (CCHHR) of the record read or written or of the next record following the one read or written. The programmer can place these in the 8-byte address field to either READ or WRITE a new record or to update the one read. For example, to delete a record from a random file with keys, the programmer can randomize the record key to a starting location, search on key to read the record, and then use the ID returned to write a blank or zeroed record (key and data) back into the same location. The descriptions of the READ and WRITE macro instructions explain when the ID can be returned and whether the ID returned is that of the same or of the next record. See Data Management Concepts listed in the Preface for a description of these macros.

When the ID returned is that of the next record, the system obtains the ID by chaining to a read-count command. This command skips to the next track if the record read or written was the last currently on the track. The system does not read the next ID if the end of a cylinder is reached. In this case, it adds one to the CC portion of the previous ID, forces the HH portion to 0, and forces R to 1 for a 2311, 2314, or 2319 file. For a 2321 file, it adds one to the high order H, forces the low order H to zero, and forces R to one. An overflow from the high order H increases the low order C by one, forces both Hs to zero, and forces R to one. Subsequent overflows of address locations cause increases in the next higher positions of the addresses. (It is the user's responsibility to check the validity of the address returned in IDLOC.)

Relative Track Addressing

The required DASD addresses may also be given as a relative address. This address is then converted by IOCS to the actual physical address (see Figure 2.18). Relative track addressing is more convenient to use than the actual physical address for the following reasons:

1. The data in the file appears to be one logically-continuous area, although it may be physically noncontiguous.
2. The user needs to know only the relative position of the data within the file; its actual physical address is not required.

The relative address may be specified by the user in either of two formats: hexadecimal (in the form TTTR), or zoned decimal (in the form TTTTTTTTRR).

In both the hexadecimal and zoned decimal format, the Ts represent the track number relative to the start of the data file, and the Rs represent the record number on that track. The hexadecimal format requires 4 bytes, while the zoned decimal format requires 10 bytes. Relative track addressing is implemented through the DTFDA macro. Parameters in this macro specify the number of extents in the file, the form of relative addressing used (hexadecimal or zoned decimal), and other required information. For specific information on the implementation of relative track addressing, see the Supervisor and I/O macros listed in the Preface.

INDEXED SEQUENTIAL FILE MANAGEMENT SYSTEM (ISFMS)

The DTFIS/ISMOD facility is both an access method and a DASD file organization technique. Facilities are provided for you to create a file, add new records in any order to a previously created file, and retrieve all records in the file either randomly or sequentially.

The track hold facility is also provided for protecting DASD tracks that are currently being processed. Track hold prevents two or more programs from updating the same record at the same time provided all the programs use the facility.

The file is created in ascending sequential order from the input that has been previously sorted on the record keys.

As the file is created, an index hierarchy is developed. The lowest level is called the track index and occupies the first track (in the case of an IBM 2321, one or possibly more tracks) of each cylinder that is contained in the file area called the prime data area. This index includes a pair of index entries for each track of the cylinder containing the user's data records. The first entry of each pair indexes the highest record key on the appropriate track being referenced. The second entry locates overflow records that can occur from that track when new records (additions) are added to the file.

The second level of index, called a cylinder index, is generated on a DASD area separate from the prime data area. An entry is made in the cylinder index for the highest record key of each cylinder in the prime data area, and each entry points to the track index on the appropriate cylinder.

A third, optional level of index, the master index, is generated in the same area as the cylinder index and precedes it. The master index has an entry for the highest key on each track within the cylinder index area. For a small file, this level of index is generally not needed because searching a cylinder index of two or three tracks is as fast as searching the master index and then the cylinder index.

An index entry is composed of a key and an address. The track index is composed of two types of entries, a normal entry and overflow entry for each prime data track within that cylinder. In the example illustrated in Figure 2.19, the normal entry indicates that the highest key on track 1 is 8, and the address is at the beginning of track 1. The overflow entry indicates the same key after loading as the normal entry. The address of hexadecimal Fs indicates no entries for this track in the overflow area.

Note: If the track index does not occupy all of track 0, track 0 also contains data records.

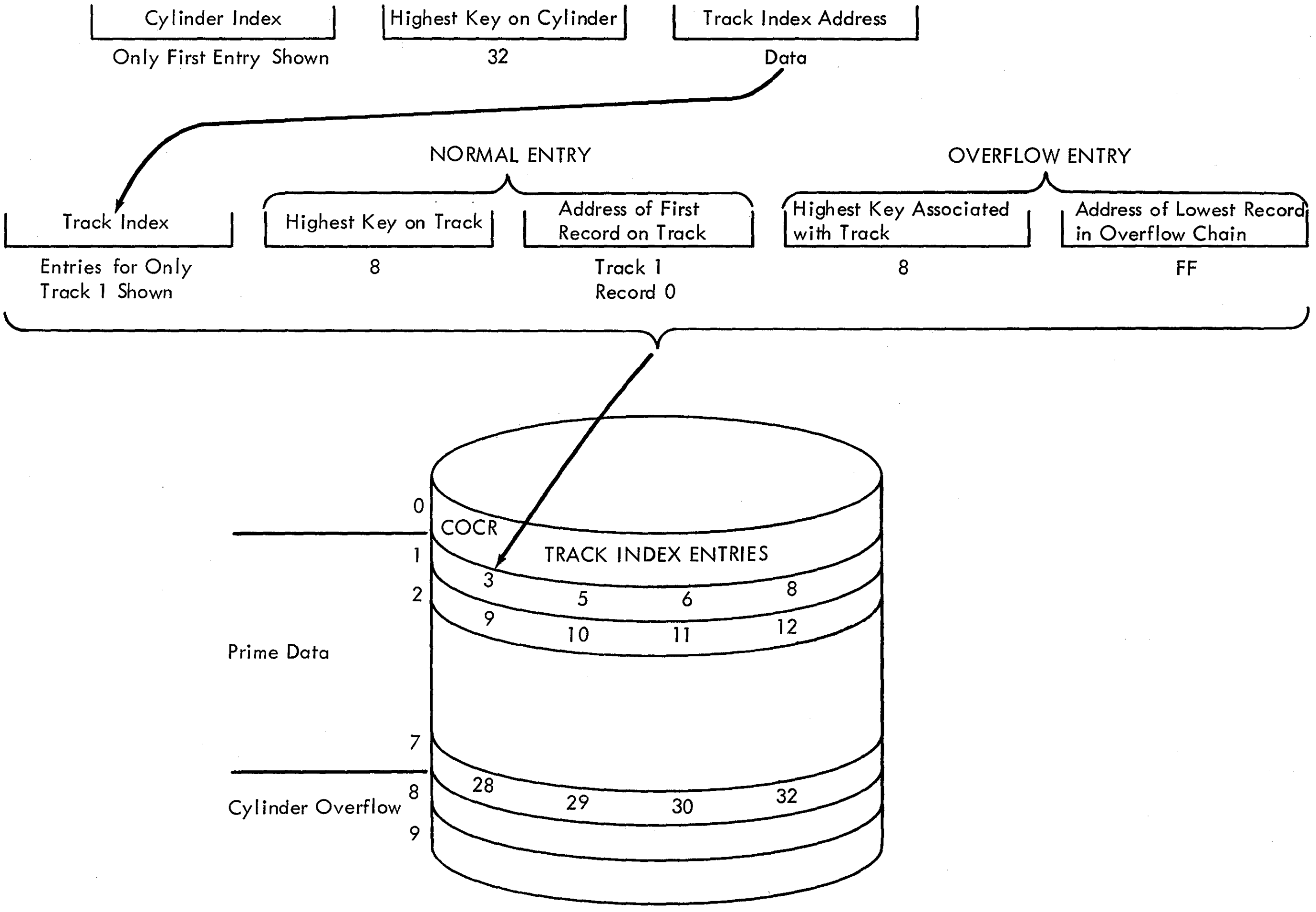


Figure 2.19. Example of Track Index

If record 7 is added to the file as illustrated in Figure 2.20, record 8 would be forced into the overflow area, and record 7 would be the highest record on track 1. In this case, the key of normal entry is changed to 7, and the address of the overflow entry indicates the location of record 8, which is on track 8 record 1. No other changes to the indexes are required.

The cylinder overflow control record (COCR), that is maintained in the data portion of record zero, would indicate that no records were in the cylinder overflow area after loading. After the addition of record 7, it would be updated to indicate that the last record in the cylinder overflow area was on track 8, record 1.

The cylinder index indicates that record 32 is the highest key on the cylinder, and its address points to the track index. The cylinder index has its own extents that must be defined at job control time. These extents must be outside the limits of any data extents. (Note that at least two sets of extent information must be defined at job control time. These would define a data extent and a cylinder index extent.) If the file exceeds one disk pack, additional data extents are provided, and at the user's option, an independent overflow area extent may be defined. The cylinder index extent must be on-line when the file is processed. It may be on the same pack as the data file, or it may be on a separate pack.

Because the file is sequential with a hierarchy of indexes, it is called an indexed sequential file. The indexes provide direct reference to records, allowing their random retrieval with a minimum of search time. The sequential order of the data records, coupled with the ability to reference overflow records in sequence via the track index, provides sequential retrieval capability. The indexed sequential method consists of four

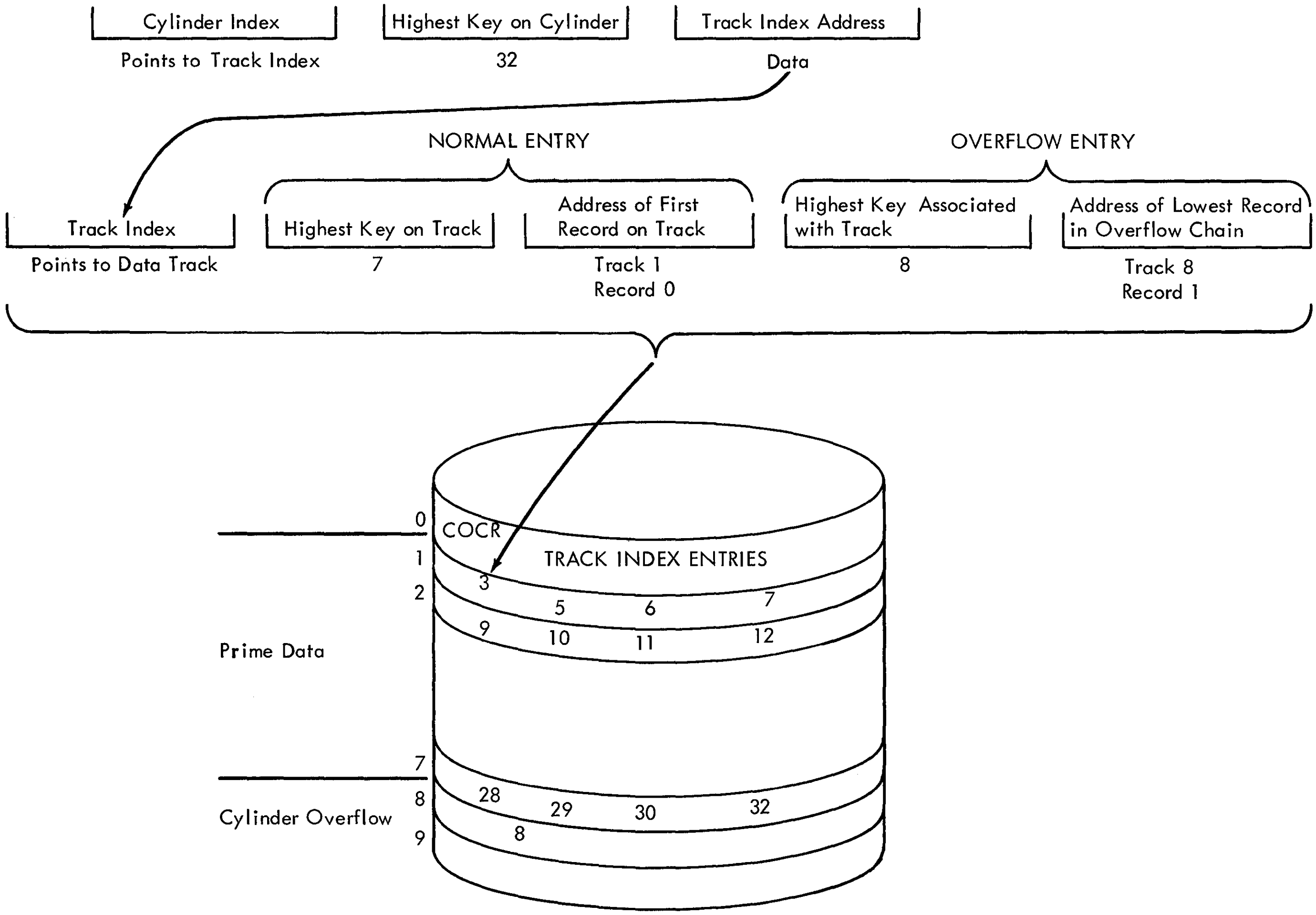


Figure 2.20. Example of Track Index after Addition to File

basic functions that provide capabilities for creating a file, adding new records to it, and retrieving the records. A description of each function follows.

Loading an Indexed Sequential File

A file and its associated indexes are created by the load function. In addition to the DTFIS macro instruction specifying IOROUT=LOAD, there are three other macro instructions used to create the file. The first one to be issued is a SETFL macro instruction (set file load mode) that does the initializing needed for file creation. When the WRITE (NEWKEY) macro instruction is issued, the key and data record placed by the user in a work area (WORKL) are moved to an output block in the buffer area (IOAREAL) and then written in the prime data area. The appropriate index entries are also made. After the user has presented all the data records needed to create the file, an ENDFL macro instruction (end file load mode) is issued to terminate the load.

Facility is provided for protecting DASD tracks that are currently being accessed. For ISAM, the hold applies to both data records and index records. Because track hold cannot be performed on a LOAD file, HOLD=YES cannot be specified when IOROUT=LOAD.

Facility is also provided to extend the file by adding new records higher in collating sequence than the current last (high) record in the file. Using the same user program, this can be done by specifying ISE in the DLBL/DLAB job control statement (required at execution time of the user's program) to indicate a file extension is to take place.

The same macro instructions that load a file originally can be used to extend the file. If it is necessary to increase the size of the file to contain records with keys higher than the last key on the original file, the records can be loaded at the end of the file. A job control XTENT/EXTENT statement extends the upper limit of the prime data area of the file so that the new records can be loaded into the file. (Overflow area is not required because the file is merely extended further on the DASD.)

Adding Records to the File

The add function provides the ability to insert new records in the file. In addition to the DTFIS macro instruction specifying IOROUT=ADD, there are two other macros used for adding records to a file. The first is a WRITE (NEWKEY), that initiates the addition process (searching the indexes, etc.) and returns control to the user to allow overlap processing. To complete the addition operation, a WAITF macro instruction must be issued. This not only ensures that all necessary I/O operations have been completed but also returns status conditions indicating any abnormal operation. When additions are made to the file, the user presents the key and data record to be added in a work area (WORKL). A search is made through the index structure to determine where the record is to be inserted in the file. The record is either inserted in key sequence in the prime data area, or placed in the overflow area by use of a chaining technique that maintains the proper sequence of the file.

Two overflow area options that may be used in any combination are provided. One option allows the user to specify that one or more tracks be reserved at the end of each cylinder to store overflow records (CYLOFL). The second option allows an independent overflow area separate from the prime data area to be reserved for storing overflow records. The first option has the advantage of reducing access time for the retrieval of overflow records associated with a given cylinder. The second option has the advantage of utilizing DASD space more efficiently.

Random Retrieval

The random retrieval function is used for random retrieval and updating of records. In addition to the DTFIS macro specifying either IOROUT=RETRVE or IOROUT=ADDRTR and TYPEFLE=RANDOM, there are three other macros for randomly retrieving and updating records. The first of three macro instructions to be issued for this purpose is a READ (KEY). The READ macro instruction performs a search of the indexes, using the key of the requested record (KEYARG) provided by the user as the search argument. While the I/O operations that perform this function are taking place, control returns to the user to allow overlap processing. To complete the READ function and receive the record, the user must issue a WAITF macro instruction. If the operation is completed successfully, this macro instruction either places the record into a work area (WORKR) or points to the starting location (leftmost position) of the record within the buffer area by using a general register (IOREG). If the operation was not successful, indications of the resulting abnormal conditions are given in the filenameC status byte in the DTF.

If the user wants to update and return the record to the file, he must issue a WRITE (KEY) macro instruction. The WRITE follows the READ of the record to be updated and precedes the READ for the next record. Again, processing can overlap execution of the WRITE instruction. To complete the operation, the user must issue a WAITF macro instruction.

Sequential Retrieval

The sequential retrieval function makes it possible to sequentially retrieve and update records. In addition to the DTFIS macro instruction specifying either IOROUT=RETRVE or IOROUT=ADDRTR and TYPEFLE=SEQNTL, there are four other macros for sequentially retrieving and updating records. The first of four macro instructions to be issued for this purpose is the SETL (set lower limit), that locates the starting point where retrieval begins. This macro instruction provides four methods of starting retrieval:

- From the beginning of the file (BOF)
- At any record location in the prime data area (ID)
- With any record in the file by supplying the key of the desired starting record (KEY)

- With the first record of a group of records in the same class by supplying the generic key for that class; a class being any group of records that contains identical control information in the first few high-order bytes of the record keys (GKEY, a key equal to or lower than the first record of the desired group)

After the SETL has been successfully executed, the user can issue GET macro instructions to retrieve each record in the file in key sequence. The record can be placed in a work area (WORKS), or a general register can be used to point to the starting location of the record in the buffer area. If the user wants to update the record and return it to the file, he must issue a PUT macro instruction. This PUT must follow the GET of the record to be updated and precede the GET of the next record. After all of the desired records have been processed, the sequential retrieval function is terminated by issuing an ESETL macro instruction. This process of issuing the SETL, GET, PUT, and ESETL macro instructions can be repeated as many times as desired. By combining the macro instructions of the sequential retrieval and load functions, the file can be reorganized. The user can retrieve the current file in its proper sequence (both prime data area records and the associated overflow records) and recreate the file in a new prime data area.

CHOOSING THE RIGHT FILE ORGANIZATION AND RETRIEVAL METHOD

The flexibility of a disk system lends itself to several different file organization and processing methods. It is important, therefore, to analyze each file and the program(s) that process it to ensure that the chosen method constitutes the optimum solution with respect to the data processing requirements of the installation.

In many cases, the type of organization and processing best suited to a file is immediately evident. However, some applications may require additional study, because of their complexity, their unusual processing requirements, or because of the wide range of processing programs that use a file. This is an important aspect of planning for a data processing system. Decisions in this area may affect system configuration requirements and should be made before programming begins. The general level of efficiency of the data processing installation may be affected.

There are no absolute rules for the resolution of an uncertain situation regarding the organization and retrieval alternatives. However, there are several criteria that may provide an indication of the optimum solution.

Criteria

The following items form a basis for a decision concerning the organization of a file.

File Activity: Activity refers to the number of records in a file for which there are transactions. This is usually expressed as a percentage.

For example, 10% activity in an inventory file means that, during some specific period, there are transactions to be posted to 10% of the records contained in the inventory file. As the activity increases, sequential processing becomes more efficient. Sequential processing implies either sequential or indexed sequential organization. Activity implies batch processing. This means that transactions do not need to be posted the moment they occur. In fact, the time that may lag between the occurrence and the post may vary from a few hours to weeks or even months, depending on the application.

Although the activity of a file is measured over time, there are applications where transactions cannot be batched. An example would be an on-line inventory file where the transactions would have to be handled as they occur.

Low activity may justify random instead of sequential retrieval.

Another important consideration involves the level of activity when sequential processing becomes more efficient than random processing. In order to make this evaluation, you must know:

- the record length and the blocking factor
- the average number of additions to the file
- whether the input is sorted.

If the Indexed Sequential File Management System is being evaluated for random or sequential processing, you must also know:

- the average time required for access to the cylinder index.

- which overflow options were chosen.

- if the resident cylinder index facility has been selected, and if so, what percentage of the cylinder index can be main storage resident.

During random processing with ISFMS it is advantageous to presort the transactions if the resident cylinder index facility has been implemented.

File Volatility: Volatility refers to the number of additions to and deletions from a file. First, consider the effect of making additions to or deletions from a sequentially organized file. Two files must be defined, and the operation must be handled as it would be with tape, reading from the input file and writing to the output file. With high volatility (that is, many additions and deletions), indexed sequential organization provides a practical solution. One of the advantages of an indexed sequential file is that additions and deletions can be handled without copying the file. However, as the number of additions increases, the efficiency of processing an indexed sequential file decreases. Additions cause records to be placed in overflow areas. Retrieval of these records in the collating sequence of the file requires more time than simply retrieving the records from contiguous tracks of the original file in their physical sequence. This is due to the additional access-arm movement required to read from the separate overflow area(s) and additional reads even when cylinder overflow is used.

With relatively few additions, the decrease in efficiency is minimal. However, there is a point at which it becomes advisable to reorganize the file. Reorganization means building a new indexed sequential file from the old one and, in the process, physically excluding all records that are tagged for deletion. In the same operation, all records in the overflow area are merged into the main file. At this point, the cycle begins again. Processing with the reorganized file is highly efficient. As additions and deletions occur, this high level of efficiency gradually diminishes, until we reach a point where reorganization again becomes advisable.

Additions to a direct access file do not necessitate the creation of a new file as they do for sequential organization. However, as the DA file extents fill up, the randomizing algorithm and its corresponding synonym processor is more heavily taxed.

Therefore, the following considerations should be made:

- At what point is it most advantageous to reorganize an indexed sequential file?

- What level of volatility excludes indexed sequential organization as a practical method of file organization?

- What are the operational considerations for each method of organization?

Many variables must be considered to answer these questions. It is impossible to provide direct answers except in terms of a specific file and a well defined application.

File Size: The user must consider the fact that his on-line capacity is limited. Three important file organization considerations are affected by the size of a file:

1. An indexed sequential file must be entirely on-line for any type of processing.

2. A sequential file on disk may be written on any number of packs, that are then mounted and processed consecutively. Each disk pack may be mounted as needed but the manual intervention that is required is time consuming.

3. A direct access file must be entirely on-line for any type of processing.

The fact that an indexed sequential or direct access file must be entirely on-line whenever it is to be processed imposes obvious physical restrictions on maximum file size. We have talked about the necessity of periodically reorganizing an indexed-sequential file. The user must also consider how this requirement affects file size. For the purpose of file reorganization, two files must be defined: the file to be reorganized and the newly created file. If the user's installation does not include tape units, and if the user does not want to punch his entire file into cards, the on-line disk capacity must be sufficient to accommodate both these files. If the installation includes tape units, the reorganization can be accomplished in two steps. The first program sequentially retrieves records from the indexed sequential file that is to be reorganized and creates an output file of these records on tape. The second program uses this tape file as input and writes the reorganized (output) file on disk. Thus, the maximum size of an indexed sequential file can be doubled by using magnetic tape as an intermediate storage medium. (An alternative would be a series of disk volumes forming a sequential disk file and sequentially sharing the same drive.) Figure 2.21 illustrates organization on a disk-tape system.

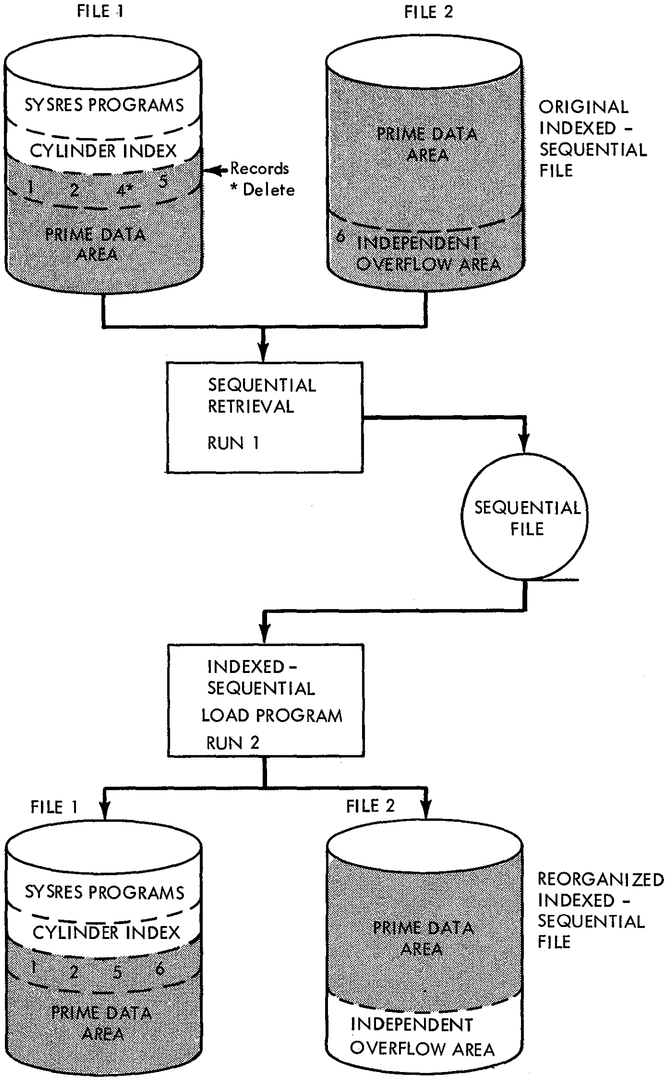


Figure 2.21. File Organization on a Disk/Tape System

It is possible and frequently desirable to divide a large indexed sequential file into several smaller files. Although this approach reduces the flexibility of processing the records, it provides many significant advantages for applications with a low activity rate.

In many cases, if a disk file exceeds the size limitations imposed by indexed sequential organization it can be organized sequentially.

First consider a DOS system with two disk drives. In this case, a sequential file may be contained on any number of packs. When a processing program reaches the record with the highest control information of the disk packs on-line, it stops. The operator can then mount the next pack on the same drive and the operation can be resumed. When a multivolume file is assigned to a disk drive, all subsequent volumes (disk packs) of the file must be mounted on the same drive.

Most sequential files are altered periodically by additions and deletions. In this case, two drives should be used with multivolume files; one for the input file volumes and one for the output file volumes. Each time the end of the available capacity of a pack on either drive is reached, the system issues a message on the console to alert the operator that it is time to change disk packs.

Consider the one-drive system. In this environment, multivolume sequential files, that are retrieved only, or retrieved and updated, can be processed on the single drive. If records are to be added and deleted, two files are involved: one input and one output file. Both files must be completely contained in one volume and the area available to each of the two files must be large enough to accommodate any net increase in file size brought about by subsequent additions and deletions. The single-drive system does not readily accommodate multivolume sequential files to which additions and deletions must be made.

This limitation can be circumvented by dividing a large file into smaller segments, each of which can be defined as a complete file and accommodated within the available portion of a single pack. This area is less than half a pack. The user must treat these segments consistently as separate files, going through the normal cycle of job-to-job transition and program retrieval as each volume is mounted. This approach, while solving one problem, may create others. If totals or statistics must be computed when processing the entire file, a means must be devised for carrying results forward from one volume (file segment) and one program to the next. This can be accomplished but it introduces an additional programming complexity with which the user of the two-drive system need not be concerned.

Response Time: One of the important advantages of a computer system with a direct access storage device is the ability to answer inquiries. Not all applications require the use of an inquiry capability. In some data processing installations, there are no inquiry applications at all. Where it is required, response time to an inquiry is a critical consideration. The less critical the response time, the greater the choice of organization and retrieval options.

The user should consider the following:

- Can the answer to an inquiry wait until the next batched, sequential updating of the relevant file? If it can, then inquiries become an additional transaction type and are processed sequentially with all other transactions against the file. File organization, in this case, could be either sequential or indexed sequential. If the response provided by this method is not fast enough, random access is required.
- Can the answer wait until the end of the present computer run? If so, the relevant file is mounted at the completion of the current job; the inquiry program is loaded; and the file is processed to produce the required answers. Obviously, the time delay involved here varies considerably depending on the job that is in progress when the inquiry arrives.

Random Retrieval Consideration: Many files that could be organized sequentially are organized as indexed sequential files to facilitate system design. It is often possible to reduce the number of peripheral operations by using random retrieval from an indexed sequential file. This is true, for example, of files that have fields which are used in several jobs.

As an example, assume that invoice summary cards are to be listed in the sequence of invoices. Further assume that the cards do not contain customer names, but that these names are required in the listing. Customer names may be obtained from the customer master file by random retrieval (if that file is organized as an indexed sequential file) or by sequential retrieval. Note that, in the latter case, the invoice summary cards must first be sorted into customer number sequence. Figure 2.22 illustrates the two solutions and shows the additional steps required if the customer master file is organized sequentially.

The example for the sequential file is a typical procedure for sequential processing equipment. If this job were run frequently, the system design considerations would probably preclude the use of a sequential file organization.

The considerations previously proposed establish criteria for choosing the organization and retrieval method for a file. In the subsequent paragraphs, these criteria are applied to a number of sample files.

Sample Files

This part illustrates the choosing of file organization and retrieval methods for some typical sample files.

The characteristics of the sample file are chosen arbitrarily. Different characteristics could be attributed to files of similar functions. The examples are furnished to demonstrate the application of the criteria we have just discussed to certain specific file characteristics; and to show, under these circumstances, the optimum organization and retrieval methods.

Example 1

File: table file

Characteristics: the file is stable and requires few changes and infrequent additions and deletions. When alterations are required, the source card file is altered and the file is reloaded onto disk. Normal processing involves retrieval only.

Organization: sequential

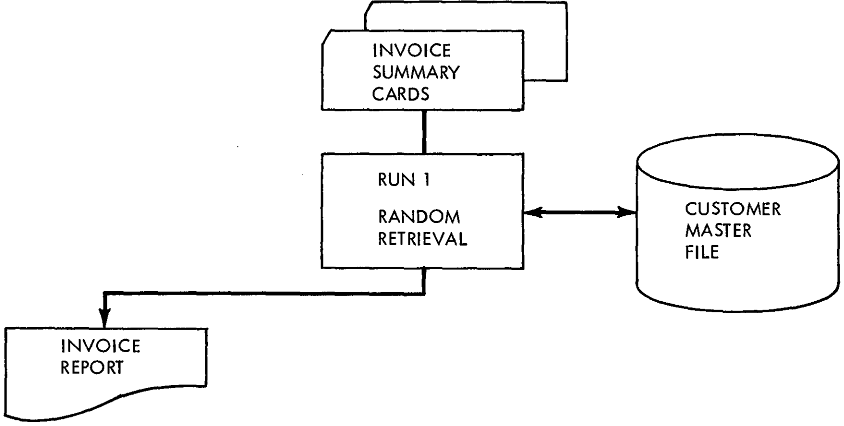
Example 2

File: payroll file

Characteristics: the file has generally low volatility; and a relatively low level of additions and deletions. However, there is a high activity rate. Processing for each pay period involves updating of a high percentage of the year-to-date master payroll information. Batching of transactions, (time cards, changes, etc.) is normal. Fast response to inquiries is not required.

Organization: sequential

EXAMPLE A : INDEXED - SEQUENTIAL ORGANIZATION



EXAMPLE B : SEQUENTIAL ORGANIZATION

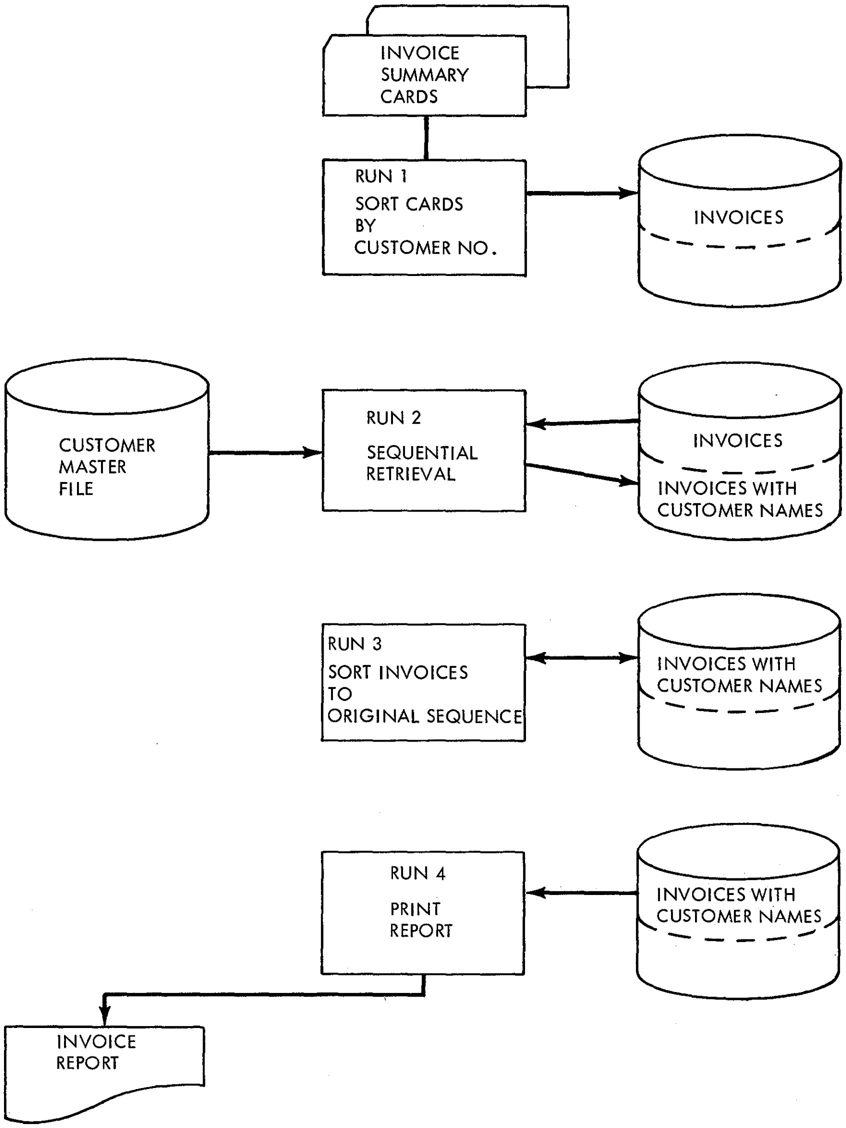


Figure 2.22. Indexed Sequential Versus Sequential File Organization

Example 3

File: wholesale inventory file

Characteristics: the file has moderate volatility and moderate activity. Normal transactions may be batched for posting once or twice a day. Recurring stock status, activity, and reorder reports are sequential. Response to inquiries concerning availability and stock level is required within one hour.

Organization: indexed sequential

Example 4

File: on-line inventory, parts

Characteristics: the file has a low volatility but a high activity. Transactions are processed as they are received. Responses to inquiries concerning availability and stock level is required within 2 minutes. Recurring stock status, activity, and reorder reports are sequential but are only produced bimonthly.

Organization: random

Example 5

File: accounts receivable file

Characteristics: the file has low volatility and low activity. Transactions are combined in batches for daily posting. Billing is cyclic. Statements are written throughout the month by sequentially retrieving records from the file between specific limits. Inquiries are processed twice daily.

Organization: indexed sequential

Summary

The method of organization best suited to a particular file of disk records depends upon many factors. These factors must be analyzed for each file in any one particular application. Often, more than one organization scheme can be considered for the same file. In one application, records could be processed purely at random; in another, the same records could be processed in sequence by various control fields. For example, records within a file might be processed at random during an updating run and sequentially within certain groups such as branch office or due date when producing reports or billing. A file such as this would be analyzed to determine whether it should be organized:

1. Randomly, thus keeping process time at a minimum during one run but destroying the advantage of the sequential nature of the other.
2. Sequentially, thus minimizing the time required to produce reports but increasing updating time.
3. Randomly for updating, and then sorted into sequence for reports.

The decision would depend on the nature of the file. Other considerations might be:

1. Can transactions be batched and sorted before processing, or must they be processed as they occur?
2. Is the activity distributed throughout the file in such a manner as to warrant passing the entire file when updating?
3. Would the processing time saved by sorting warrant the time and effort required?

Questions of this kind apply to each file in an installation. In choosing organization methods, the over-all processing objectives of the system must be kept in mind at all times.

Section 3: Program Design

Section Outline

Section Figures

The following are included in this section which presents techniques for the effective use of some DOS facilities:

- Link Editing. Includes the linkage editor control statements and overlay structures, design and communication. In addition, examples illustrating module dependency and overlay tree structure are included.
- Self-Relocating Program. Explains how to write a self-relocating program. It presents the rules, advantages, disadvantages, and techniques for writing self-relocating code.
- Checkpoint/Restart. Includes the programmer's responsibilities when a checkpoint is taken during problem program execution. This also explains the CHKPT macro for checkpointing a program and describes checkpoint considerations for tape and disk, as well as for repositioning of files and restarting checkpointed programs.
- IBM 3211 Printer Support. Includes the error recovery techniques.
- Macro Writing. Includes the macro language and its effective use. Also included are examples illustrating how to write and use macros.

Link Editing

Link editing provides the user with the capability of combining separately assembled or compiled program sections or subprograms. To make this possible, the output of the language translators must be processed. At first, the separate program sections are in relocatable form; that is, the address constants are identified for later modification to absolute execution time values. The linkage editor (LNKEDT) links and relocates separate program sections into a single phase that can be loaded by the control program and then executed.

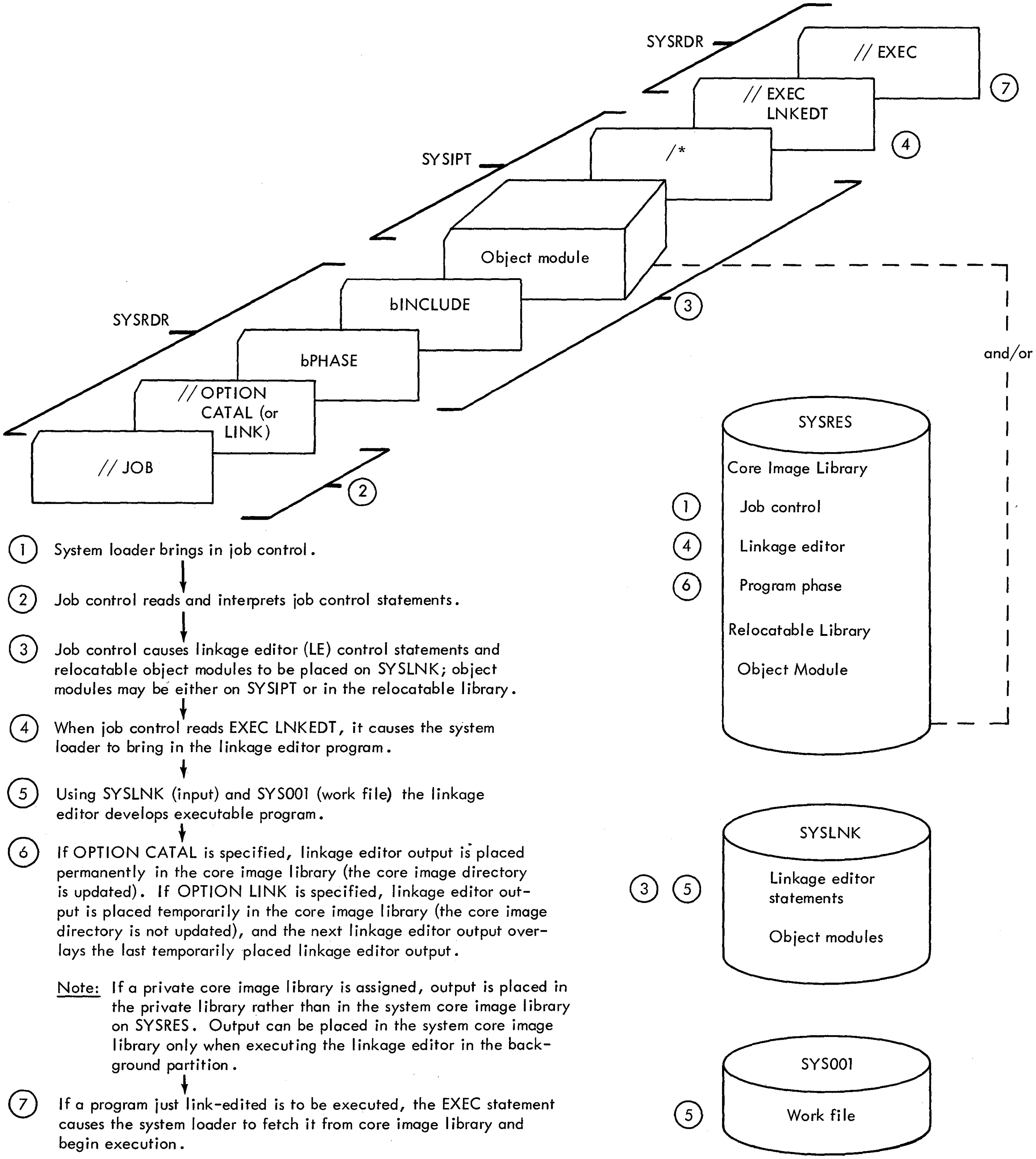
Every relocatable program must be processed by the linkage editor before it can be executed. Once a program is edited, it can be executed immediately, cataloged as a permanent entry in the core image library, or both cataloged and executed immediately. When a program is cataloged in the core image library, the linkage editor is no longer required for that program, because it can be loaded directly from the resident pack by the system loader of the control program. On the other hand, if a program is edited and executed immediately without cataloging, the linkage editor is required again the next time the program is to be executed. Cataloging is a system design decision based on such factors as frequency of use and space available in the core image library.

In a system having a minimum of 32K positions of main storage, batched-job foreground multiprogramming, and private core image library support, the linkage editor can execute in any partition. A private core image library must be assigned when executing the linkage editor in a foreground partition. When executing the linkage editor in the background partition, if a private core image library is not assigned, the default is to the system core image library. Without the two options specified the linkage editor can execute in the background partition only.

SYSTEM FLOW

Figure 3.1 shows the system flow for the linkage editor program. Before the linkage editor program is executed, job control must perform these functions:

- Process the OPTION statement. OPTION LINK or CATAL turns on the control program switches which cause job control to open the SYSLNK file. Unless these switches are on, the linkage editor control statements are invalid.
- Copy the linkage editor control statements onto SYSLNK. The linkage editor control statements are ACTION, PHASE, INCLUDE, and ENTRY. The ACTION, PHASE, and ENTRY statements are copied directly on SYSLNK. There are two forms of the INCLUDE statement. INCLUDE statements with no operand are not copied but cause the data (object module) on SYSIPT to be written until the end-of-data (/*) occurs. If the object module to be linkage edited is cataloged in the relocatable library, the INCLUDE statement must have the name of the module as an operand. The format of the INCLUDE statement is copied, but the module is not.
- Write an ENTRY statement with a blank operand if the job stream does not already contain one. When the EXEC LNKEDT statement is encountered, an ENTRY statement is created to ensure termination of the link edit input.



① System loader brings in job control.

② Job control reads and interprets job control statements.

③ Job control causes linkage editor (LE) control statements and relocatable object modules to be placed on SYSLNK; object modules may be either on SYSIPT or in the relocatable library.

④ When job control reads EXEC LNKEDT, it causes the system loader to bring in the linkage editor program.

⑤ Using SYSLNK (input) and SYS001 (work file) the linkage editor develops executable program.

⑥ If OPTION CATAL is specified, linkage editor output is placed permanently in the core image library (the core image directory is updated). If OPTION LINK is specified, linkage editor output is placed temporarily in the core image library (the core image directory is not updated), and the next linkage editor output overlays the last temporarily placed linkage editor output.

Note: If a private core image library is assigned, output is placed in the private library rather than in the system core image library on SYSRES. Output can be placed in the system core image library only when executing the linkage editor in the background partition.

⑦ If a program just link-edited is to be executed, the EXEC statement causes the system loader to fetch it from core image library and begin execution.

Figure 3.1. Linkage Editor System Flow

- Inform the system loader to load the linkage editor program. The linkage editor program uses the data on SYSLNK as input. It handles the relocatable modules as directed by the PHASE and INCLUDE statements to develop executable program phases. Some of the linkage editor program functions are:
  1. Extracting the modules named in INCLUDE statements from the relocatable library. If, in extracting a module, another INCLUDE statement occurs, this module is also retrieved. The nesting of modules is possible up to a depth of five (a level of six).
  2. Constructing composite dictionaries for ESD and RLD data, to resolve all linkages between different control sections.
  3. Relocating each control section as necessary within a phase.
  4. Assigning the entire phase to a contiguous area in main storage.
  5. Modifying all relocatable address constants to contain the relocated value of their symbols.
  6. Searching the relocatable library for a cataloged object module with the same name as each unresolved external reference. The automatic library lookup feature (AUTOLINK) is particularly useful for retrieving IOCS modules. It may be suppressed.
  7. Building the core image directory phase headers and cataloging to the core image library, if CATAL is specified.

If a phase by the same name was cataloged previously, the old phase is deleted and the new one is cataloged. Deletion removes the item from the directory, but it does not release the space in the library until a condense function occurs.

If a private core image library is assigned, the linkage editor output is placed in the private rather than the system core image library, permanently if OPTION CATAL is specified or temporarily if OPTION LINK is specified (see Figure 3.1, point 6). In this case, the library need not be on SYSRES as shown in Figure 3.1. Output may be placed in the system core image library (either permanently or temporarily) only when executing the linkage editor in the background partition.

SYMBOLIC UNITS REQUIRED

The symbolic units required by the linkage editor are basically a subset of those needed by the language translators:

SYSIPT	Module input
SYSLST	Programmer messages and listings
SYSLOG	Operator messages
SYSRDR	Control statement input (via job control)
SYSLNK	Input to the linkage editor
SYS001	Work file

Note that SYSRDR and SYSIPT may contain input for the linkage editor. This input is written on SYSLNK by job control.

If output from the linkage editor is to be placed in a private core image library, the following symbolic unit is required:

SYSCLB the private core image library may be assigned anywhere in the job stream but must be before the // EXEC LNKEDT statement.

LINKAGE EDITOR CONTROL STATEMENTS

Position 1 must be blank on linkage editor control statements. Otherwise, they follow the same format as job control commands. Refer to <u>DOS System Control and System Service</u> listed in the <u>Preface</u> for a detailed explanation of the control statements.

<u>ACTION Statement</u>

$$\text{ACTION}\left\{\begin{array}{l}\text{CLEAR}\\\text{MAP}\\\text{NOMAP}\\\text{NOAUTO}\\\text{CANCEL}\\\text{BG}\\\text{F1}\\\text{F2}\end{array}\right\}$$

This statement specifies linkage editor options. It is not required, but if used, it must appear as the first linkage editor statement in the input stream. If multiple operands are required, they may be placed on separate ACTION statements or on one

ACTION statement separated by commas. The ACTION statement is effective only for the next linkage editor execution. The parameters have the following meanings:

CLEAR	Set the unused area of the core image library to binary zeros. The linkage editor clears from the next available entry (taken from the core image directory) to the end of the core image library. Because this is time-consuming, use it only if the areas defined by DS statements must be filled with zeros.
MAP	Write main storage map and error diagnostics on SYSLST. Whenever SYSLST is assigned, MAP is automatic unless NOMAP is specified.
NOMAP	Nullify MAP action.
NOAUTO	Suppress AUTOLINK function for the entire program, not just one phase.
CANCEL	Cancel the job if the content of the linkage editor input is in error. See messages 2100I to 2170I in the <u>DOS Messages</u> listed in the <u>Preface</u>.
BG F1 F2	Causes the end of supervisor address used in linkage editor calculations to be set to the beginning of the partition specified, plus the length of the label area and of the save area. The end of supervisor address in the communications region is not changed.

The BG, F1 and F2 operands provide the capability of link editing a program to execute in a partition other than that in which the link edit function is taking place. Programs that have a phase origin of S (or * for the first phase of a program) can be origined to the specified partition by use of the operands.

Use of the ACTION BG statement is possible only in a system supporting the batched-job foreground and private core image library options when the linkage editor is executing in a foreground partition.

Use of the ACTION F1 (or F2) statement in a multiprogramming environment requires that the partition be allocated. If these operands are used in a non-MPS environment, they are ignored.

If none of these operands is present, the program will be link-edited to execute in the partition in which the link edit function is taking place, unless otherwise specified on the PHASE statement.

An example of the use of the ACTION F1 statement follows. Assume a 64K machine with:

8K supervisor

24K background area

16K foreground 2 area

16K foreground 1 area.

When executing the linkage editor in the background the statement PHASE PHASE1,S causes PHASE1 to be origined at 8K (the end of the supervisor area). The sequence

```
ACTION F1

PHASE PHASE1,S
```

causes PHASE1 to be origined at 48K (the beginning of the foreground 1 area) plus the length of the foreground save area.

When executing the linkage editor in foreground 2, the statement PHASE PHASE1,S causes PHASE1 to be origined at 32K (the beginning of the foreground 2 area) plus the length of the foreground save area. The sequence

```
ACTION BG
PHASE PHASE1,S
```

causes PHASE1 to be origined at the end of the supervisor area.

PHASE Statement

PHASE name,origin[,NOAUTO]

A program phase is the section of a program that is loaded by the system loader as a single overlay with a single FETCH or LOAD. The input for building a single phase consists of the text from one or more complete control sections. Therefore, programs may consist of many phases, or a phase may consist of many subprograms or control sections.

The PHASE statement provides the linkage editor with the phase name and an origin point for the phase. The phase name catalogs the phase in the core image library and retrieves it for execution. Job control uses the phase name to construct a single track phase directory before each job step is executed. This phase directory is built for all background program executions. If executing in the foreground, this phase directory is built under link, edit, and go conditions. The entries to this directory are taken from the core image directory for any phase where the first four characters of the name are identical to those in the name specified in the EXEC statement. The directory entry contains such information as loading address, entry point, and starting disk address in the core image library. The separate phase directory permits faster retrieval of the phases.

The entries in the operand field represent the following:

name — Symbolic name of the phase, consisting of 1 to 8 alphameric characters. The first 4 characters of a multiphase program should be the same to achieve maximum retrieval efficiency.

origin — Specification of the load address of the phase. The load address can be one of six forms:

1. S [+relocation]. If link editing in the background, the origin point is at the end of the supervisor, the save areas, and the area assigned to the COMMON pool, if any. If link editing in the foreground, the beginning address of the partition is substituted for the end of supervisor address.
2. ROOT. This phase is designated as the root phase, which remains in main storage throughout execution. Its location is the same as with format S.
3. +displacement. The origin point is set at a specified location; +0 must be used for any self-relocating program.
4. F+address. This format is used to begin the program at the start of a foreground partition when link editing in the background and the foreground partition is not allocated. If the foreground partition is allocated, ACTION F1 or F2 has the same effect as F+address.
5. *[±relocation]. This is the most frequently used format and specifies an origin point for a phase at the next available core location.
6. symbol [(phase)] [±relocation]. This format specifies an origin point for a phase at the same point as previously defined symbol (for overlays).

A detailed explanation of the origin parameter is given in the <u>DOS System Control and System Service</u> listed in the <u>Preface</u>. Also refer to <u>Link Editing Examples</u>.

NOAUTO — Suppress the AUTOLINK function for this phase only.

INCLUDE Statement

INCLUDE [modulename] [,(namelist)]

This statement specifies that an object module is to be included for editing by the linkage editor. The system assumes the location of the module as follows:

1. Both operands missing. The object module is on SYSIPT; it is copied onto SYSLNK.
2. Modulename given. The object module is cataloged in the relocatable library under the same name.
3. Second operand only given. The object module is in the input stream on SYSLNK. The parameters represent the following:

modulename — Use this parameter only if the module is cataloged in the relocatable library. It consists of 1 to 8 alphameric characters and must be the same as the name used when the module was cataloged.

(namelist) — This parameter provides the ability to select particular control sections from a given module. It is expressed as (csname1, csname2, ...).

ENTRY Statement

ENTRY [entrypoint]

The ENTRY statement signals the end of program input to the linkage editor. The entrypoint operand indicates the transfer address for the first phase as follows:

1. If omitted, the first significant address in an END record encountered during the generation of the first phase is used; or, if no such operand is found, the load address of the first phase is used.

2. If given, it must be the name of a CSECT or a label definition defined in the first phase.

The ENTRY statement can be completely omitted because job control automatically writes an ENTRY statement with a blank operand when it encounters the EXEC LNKEDT statement.

LBLTYP Job Control Statement

$$\text{// LBLTYP} \begin{Bmatrix} \text{TAPE (nn)} \\ \text{NSD (nn)} \end{Bmatrix}$$

The label storage records for standard labeled tape files and nonsequential DASD files (direct access, indexed sequential, or DTFPH with all packs mounted) are brought into the label save area of the partition containing the processing program. Therefore, main storage must be reserved by the user whenever such files are processed. Because this area is used during OPEN for one file at a time, the total area needed is that required by the largest file.

Main storage reservation is accomplished by the LBLTYP statement. The amount of main storage reserved is governed by the operand TAPE or NSD as follows:

- TAPE reserves 80 bytes of main storage. This format is used when standard labeled tape files and <u>no</u> nonsequential DASD files are processed. nn is ignored by job control. This same 80-byte area is used by all labeled tape files.

- NSD reserves 84 bytes plus 20 bytes per extent. The number of extents is specified in the nn parameter for the nonsequential file that has the largest number of extents. This format is used when nonsequential DASD files are processed, regardless of whether labeled tapes are processed. This same area is used by nonsequential DASD files with fewer extents, and by labeled tape files.

The LBLTYP statement is not required if only unlabeled tape files and/or sequential DASD files are being processed. Only one LBLTYP statement is submitted. The placement of the statement in the job stream varies as shown:

- Non-self-relocating (background/foreground). Immediately preceding the EXEC LNKEDT statement at linkage editor time.

- Self-relocating (background/foreground). Ahead of the EXEC for the program.

Examples of various linkage editor functions follow.

SUMMARY OF CONSIDERATIONS FOR LINK AND CATAL OPTIONS

1. SYSLNK must be assigned, or LINK and CATAL options are ignored (switches are not set). If executing the linkage editor in a foreground partition, a private core image library (SYSCLB) must be assigned. This is possible only in a system supporting the batched-job foreground and private core image library options.

2. Unless the switches are set by the LINK or CATAL option, the linkage editor control statements are ignored.

3. The CATAL option sets the LINK and CATAL switches.

4. When the LINK switches are set, the output of the language translators is placed on SYSLNK.

5. LINK and CATAL switches are turned off by:

   a. /& or JOB statement

   b. An error during compilation.

6. Completion of cataloging (update of transient, library routine, and foreground program directories, and system status report) occurs when the /& statement is read by job control.

7. If a successful link edit has not occurred, cataloging does not take place.

8. If multiple linkage editor job steps are set up as one job, keep these points in mind:

   a. It is not possible to CATAL into the core image library with // OPTION CATAL and then have another linkage editor job step with // OPTION LINK in the same job. Operator message 1S1nD (STATEMENT OUT OF SEQUENCE) results.

   b. If a compilation is being performed, the link switches may be turned off by an error. When cataloging to the core image library, therefore, it is advisable to handle multiple job steps as separate jobs (each with /&) to be sure that the cataloging operation is finished on the /&.

   c. SYSLNK extents are reset each time before Job Control writes a new series of link edit control statements onto SYSLNK. Core image library directory and subdirectory are updated at /& if the CATAL option is included.

LINKAGE EDITOR PROGRAM CONSIDERATIONS

The linkage editor program consists of eight phases. The efficiency of link editing programs into the core image library depends upon the amount of main storage allocated to the partition where the linkage editor is executing. If a minimum of 14K of main storage is available several phases of the linkage editor program are contained within main storage during linkage editor processing. If less than 14K is available, only one linkage editor processing phase may be resident in the partition at one time. Some time is lost fetching the additional phases as they are needed.

PROGRAM OVERLAY STRUCTURES

Overlay is a programming technique that minimizes the main storage requirements of a program. To use overlay, the programmer should be familiar with two related techniques:

1. Organizing the program as an overlay structure.

2. Communicating with the control program during execution through FETCH and LOAD macros.

Overlay Tree Structure

To place a program in an overlay structure, the programmer should be familiar with the following terms:

Phase: A phase is the smallest functional unit (one or more control sections) that can be loaded as one logical entity during the execution of the program. A phase can contain up to 524,288 bytes of text. The root phase (first phase) remains in main storage throughout execution.

Tree: A tree is the graphic representation that shows how phases can use main storage at different times. It does not imply the order of execution, although the root phase is the first to receive control.

The design of an overlay program requires the organization of the control sections of the program in an overlay tree structure. The tree structure is developed considering:

1. The amount of available main storage.

2. The frequency of use of each control section.

3. The dependencies between control sections.

4. The manner in which control should pass within a path, from one path to another, and from one region to another.

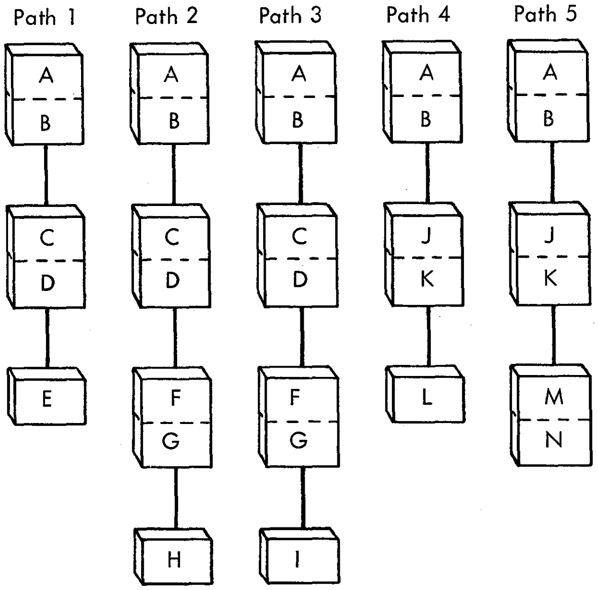
When the overlay tree structure for a program is determined, PHASE statements are prepared which segment the program in that manner.

Overlay Tree Design

To begin constructing an overlay tree, the programmer should select those modules that receive control at the beginning of execution plus those that should always remain in main storage; these form the root phase. The rest of the tree can be developed by determining the dependency of the remaining phases and how they can use the same main storage locations at different times during execution.

Module dependency is determined by the requirements of a control section or module for a given routine in another control section. A module depends upon a control section to which it branches or whose data it must process. The required control section must be in main storage before execution can continue beyond a given point in the program. Figure 3.2 illustrates how modules depend on each other, and the paths that result from these dependencies.

The module containing control sections A and B can be used to form the root phase. The module containing control sections C and D can use the same main storage as the module containing control sections J and K. Phases that use the same main storage area can overlay each other during execution. The module containing control section E can use the same main storage as the module containing control sections F and G. The module containing control section H can use the same main storage as the module containing control section I. The module containing control section L can use the same main storage as the module containing control sections M and N.



Note: A through N are Control Sections.

Figure 3.2. Module Dependency

Figure 3.3 shows the resulting overlay tree structure. The longest path in this structure is formed by phases 1, 2, 4, and 5, because the program requires 21,000 bytes when they are in main storage. Thus, the minimum main storage requirement for the program is 21,000 bytes. The program would require 46,000 bytes if not put in an overlay structure. The linkage editor assigns the relocatable origin of the root phase (the origin of the program) at 0. The relative origin of each phase is determined by 0, plus the length of all phases in the path. For example, the origin of phases 3 and 4 is equal to 0 + 5,000 (the length of phase 2) + 6,000 (the length of the root phase).

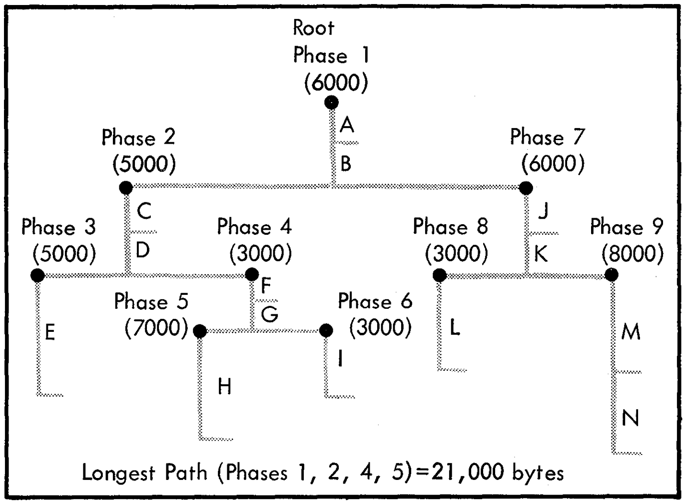


Figure 3.3. Overlay Tree Structure

When a phase is in main storage, all phases in its path are in main storage. (Each time a phase is loaded, all phases in its path are also loaded if they are not already in main storage.) In Figure 3.3, when phase 4 is in main storage, phases 1 and 2 are also in main storage. This does not imply that phase 5 or 6 is in main storage because neither phase is in the path of segment 4.

The position of the phases in an overlay tree structure does not necessarily imply the order in which they are executed. A phase can be loaded and overlaid as many times as required by the logic of the program. If a phase is modified during execution, that modification remains only until the phase is overlaid. However, a phase cannot be overlaid by itself.

Phases that can be in main storage simultaneously are considered to be inclusive. Phases not in the same path are considered to be exclusive; they cannot be in main storage simultaneously.

Phases on which two or more exclusive phases are dependent are called common phases. A phase common to two other phases is part of each. In Figure 3.3, phase 7 is common to phases 8 and 9, but not to phase 2.

Overlay Communication

The programmer must know how his program can communicate with the control program during execution. Two ways in which he can overlay phases in his program are:

1. By a LOAD macro instruction, that loads the named phase into main storage and returns control to the calling phase.
2. By a FETCH macro instruction, that loads the specified phase into main storage and passes control to the entry address of the fetched phase.

Some of the advantages are:

1. The LOAD macro allows the programmer to load his phases in violation of the tree structure that was defined at linkage editor time.
2. The LOAD macro allows the programmer to load phases from other programs.

However, these responsibilities are associated with the use of these macros:

1. The programmer must keep track of which phases are in main storage.
2. The programmer must be aware of overlay structure and he must know the phase names at compilation or assembly time.

A phase that is loaded into main storage by the FETCH macro is always relocated unless the self-relocating option was specified at linkage editor time. A phase that is loaded into main storage by the LOAD macro is relocated relative to the same structure as when it was linkage edited; i.e., LOADs can be made outside the defined tree, but it is not possible for the system to ensure that references outside the tree will be valid.

Self-Relocating Programs

A system supporting multiprogramming has the capability of executing self-relocating programs. A self-relocating program is one that can be executed at any location in main storage. Writing a self-relocating program is an efficient coding technique because self-relocating programs are link edited only once for execution in any partition. When linkage editing, use OPTION CATAL and a PHASE card such as:

```
PHASE   Phasename,+0
```

This causes the linkage editor to assume that the program is loaded at core location zero, and to compute all absolute addresses from the beginning of the phase. The job control EXEC function recognizes a zero phase address and adjusts the origin address to compensate for the current partition boundary save area and label area (if any). It then gives control to the updated entry address of the phase.

RULES FOR WRITING SELF-RELOCATING PROGRAMS

In general, if a problem program is written to be self-relocating, the following rules must be adhered to:

1. The supervisor must support multiprogramming (that is, MPS=YES or BJF must be specified as a parameter in the SUPVR macro at system generation time).
2. The PHASE card must specify an origin of +0.
3. The program must relocate all address constants used in the program. Whenever possible, use the LA instruction to load an address in a register instead of using an A-type address constant. For example,

   Instead of using:

```
      USING  *,12
      BALR   12,0
      LA     12,0(12)
      BCTR   12,0
      BCTR   12,0
      LA     1,EOF
      ST     1,AEOF
      •
      •
      L      10,AEOF
      •
      •
EOF   EOJ
      •
      •
AEOF  DC     A(EOF)
```

   Use:

```
      USING  *,12
      BALR   12,0
      LA     12,0(12)
      BCTR   12,0
      BCTR   12,0
      •
      •
      LA     10,EOF
      •
      •
EOF   EOJ
```

4. If logical IOCS is used, the program must use the OPENR and CLOSER macros to open and close files.

5. If physical IOCS is used, the program must relocate all CCW address fields.

6. Register notation must be used when issuing an imperative macro (I/O, I/O control, and supervisor communication). Register notation utilizes less main storage and permits faster execution.

The following rules apply to multimodule programs.

7. The relocation factor should be calculated and stored in a register for future use. For register economy, the base register can hold the relocation factor.

   For example:

   ```
   USING  *,12
   BALR   12,0
   LA     12,0(12)
   BCTR   12,0
   BCTR   12,0
   ```

   Register 12 now contains the relocation factor and the program base.

8. When branching to an external address, use one of the following techniques:

   a.
   ```
   L      15,=V(EXTERNAL)
   BAL    14,0(12,15)
   ```

   b.
   ```
   L      15,=V(EXTERNAL)
   AR     15,12
   BALR   14,15
   ```

   where register 12 is the base register containing the relocation factor.

9. The calling program is responsible for relocating all address constants in the calling list(s). See Figure 3.4 for an example of calling program relocating the address constants in a calling list.

ADVANTAGES OF SELF-RELOCATING PROGRAMS

Self-relocating programs have the ability to run in any one of the three problem program partitions without needing linkage editing again. The program can also be loaded anywhere within a partition. The restriction of specific partition allocations need not be adhered to with a self-relocating program because it relocates itself.

DISADVANTAGES OF SELF-RELOCATING PROGRAMS

Self-relocating programs are slightly more time consuming to write and they usually require slightly more main storage.

PROGRAMMING TECHNIQUES

A self-relocating program is capable of proper execution, regardless of where it is loaded. DTFDI should be used to resolve device differences between partitions. A self-relocating program must also adjust all of its own absolute addresses to point to the proper address. This must be done after the program is loaded, and before the absolute addresses are used.

```
// JOB A
// OPTION LINK
// EXEC ASSEMBLY
CSECT1   START 0
         USING *,12      Use load point value as the base to
         BALR  12,0      find the load point value.
         LA    12,0(12)
         BCTR  12,0
         BCTR  12,0
         •
         •
         LA    1,A
         LA    2,B
         LA    3,C       Modify the CALL address constant list.
         LA    4,D
         STM   1,4,LIST
         LA    13,SAVEAREA
         L     15,=V(EXTERNAL)
         AR    15,12     Adjust CALL address by relocation
                         factor.
         CALL  (15),(A,B,C,D)
LIST     EQU   *-16      For address constants (4 bytes each).
         EOJ
SAVEAREA DC    9D'0'
         END
/* *
// EXEC ASSEMBLY
CSECT2   START 0
         ENTRY EXTERNAL
EXTERNAL SAVE  (14,12)
         USING *,12
         BALR  12,0      Establish new base
         •
         •
         RETURN(14,12)
         END
/* *
// EXEC LNKEDT
```

Figure 3.4. Relocating Address Constants in a Calling List

Within these self-relocating programs, some macros generate self-relocating code. For example, the MPS utility macros are self-relocating (that is, they modify all of their own address constants to their proper values before using them). OPENR and CLOSER macros are designed to be used in self-relocating programs. OPENR and CLOSER can be used in place of OPEN and CLOSE, and adjust all of the address constants in the DTFs opened and closed. OPENR and CLOSER can be used in any program because the OPENR macro computes the amount of relocation. If relocation is 0, the standard open is executed. In addition, all of the module generation (xxMOD) macros are self-relocating.

The addresses of all address constants containing relocatable values are listed in the relocation dictionary in the assembly listing. This dictionary includes both those address constants that are modified by self-relocating macros, and those that are not. The address constants not modified by self-relocating macros must be modified by some other technique. After the program has been linkage edited with a phase origin of +0, the contents of each address constant is the displacement from the beginning of the phase to the address pointed to by that address constant.

The following techniques place relocated absolute addresses in address constants. These techniques are required only when the LA instruction cannot be used.

Technique 1

Named A-type address constants:

```
         •
         LA    4,ADCONAME
         ST    4,ADCON
         •
         •
ADCON    DC    A(ADCONAME)
```

Technique 2

A-type address constants in the literal pool:

```
         •
         LA    3,=A(ADCONAME)
         LA    4,ADCONAME
         ST    4,0(3)
         •
         •
         LTORG
               =A(ADCONAME)
```

Technique 3

A-type address constants with a specified length of three bytes, and a nonzero value in the adjacent left byte (as in CCWs):

A. If CCW list dynamically changes during program execution:

1.

```
         •
         IC    3,TAPECCW
         LA    4,IOAREA
         ST    4,TAPECCW
         STC   3,TAPECCW
         •
TAPECCW  CCW   1,IOAREA,X'20',100
         •
         •
IOAREA   DS    CL100
```

2.

```
         •
         USING *,12
         BALR  12,0
         LA    12,0(12)
         BCTR  12,0
         BCTR  12,0 Reg. 12 contains re-
                       location factor
         •
         L     11,TAPECCW
         ALR   11,12
         ST    11,TAPECCW
         •
TAPECCW  CCW    1,IOAREA,X'20',100
         •
IOAREA   DS    CL100
```

B. If CCW list is static during program execution:

```
         •
         LA    4,IOAREA
         ST    4,TAPECCW
         MVI   TAPECCW,1
         •
         •
TAPECCW  CCW   1,IOAREA,X'20',100
         •
         •
         DS    CL100
```

Technique 4

Named V-type or A-type address constants:

```
         •
         LA    3,ADCONAST  Determine
         S     3,ADCONAST  Relocation
                           factor
         •
         •
         L     4,ADCON
         AR    4,3  Add Relocation factor
         ST    4,ADCON
         •
         •
ADCONAST DC    A(*)
ADCON    DC    V(NAME)
```

Note that the load point of the phase is not synonymous with the relocation factor as developed in register 3 (technique 4). If the load point of the phase is taken from register 0 (or calculated by a BALR and subtracting 2) immediately after the phase is loaded, it may be added to address constants with varying results. If the phase was linkage edited with an origin of +0, the correct results are obtained. If the phase was linkage edited with an origin of * or S, incorrect results are obtained because, both the linkage editor and the program itself have added the load point to all address constants. See Figure 3.5 for an example of a self-relocating program.

```
    SOURCE STATEMENTS
             REPRO
             PHASE EXAMPLE,+0                +0 ORIGIN IMPLIES SELF-RELOCATION
             PRINT NOGEN
PROGRAM      START 0
             BALR  12,0
             USING *,12
*       ROUTINE TO RELOCATE ADDRESS CONSTANTS
             LA   1,PRINTCCW                 RELOCATE CCW ADDRESS
             ST   1,PRINTCCB+8                 IN CCB FOR PRINTER
             LA   1,TAPECCW                  RELOCATE CCW ADDRESS
             ST   1,TAPECCB+8                  IN CCB FOR INPUT TAPE
             IC   2,PRINTCCW                 SAVE PRINT CCW OP CODE
             LA   1,OUTAREA                  RELOCATE OUTPUT AREA ADDRESS
             ST   1,PRINTCCW                   IN PRINTER CCW
             STC  2,PRINTCCW                 RESTORE PRINT CCW OP CODE
             LA   1,INAREA                   RELOCATE INPUT AREA ADDRESS
             ST   1,TAPECCW                    IN TAPE CCW
             MVI  TAPECCW,READ               SET TAPE CCW OP CODE TO READ
*       MAIN ROUTINE...READ TAPE AND PRINT RECORDS
READTAPE     LA   1,TAPECCB                  GET CCB ADDRESS
             EXCP (1)                        READ ONE RECORD FROM TAPE
             WAIT (1)                        WAIT FOR I/O COMPLETION
             LA   10,EOFTAPE                 GET ADDRESS OF TAPE EOF ROUTINE
             BAL  14,CHECK                   GO TO UNIT EXCEPTION SUBROUTINE
             MVC  OUTAREA(10),INAREA         EDIT RECORD
             MVC  OUTAREA+15(70),INAREA+10   IN
             MVC  OUTAREA+90(20),INAREA+80   OUTPUT AREA
             LA   1,PRINTCCB                 GET CCB ADDRESS
             EXCP (1)                        PRINT EDITED RECORD
             WAIT (1)                        WAIT FOR I/O COMPLETION
             LA   10,CHA12                   GET ADDRESS OF CHAN 12 ROUTINE
             BAL  14,CHECK                   GO TO UNIT EXCEPTION SUBROUTINE
             B    READTAPE
CHECK        TM   4(1),1                     CHECK FOR UNIT EXEC. IN CCB
             BCR  1,10                        YES-GO TO PROPER ROUTINE
             BR   14                          NO-RETURN TO MAINLINE
CHA12        MVI  PRINTCCW,SKIPTO1           SET SEEK TO CHAN 1 OP CODE
             EXCP (1)                        SEEK TO CHAN 1 IMMEDIATELY
             WAIT (1)                        WAIT FOR I/O COMPLETION
             MVI  PRINTCCW,PRINT             SET PRINTER OP CODE TO WRITE
             BR   14                         RETURN TO MAINLINE
EOFTAPE      EOJ                             END OF JOB
             CNOP 0,4                        ALIGN CCB'S TO FULL WORD
PRINTCCB     CCB  SYS004,PRINTCCW,X'0400'
TAPECCB      CCB  SYS001,TAPECCW

PRINTCCW     CCW  PRINT,OUTAREA,SLI,L'OUTAREA
TAPECCW      CCW  READ,INAREA,SLI,L'INAREA
OUTAREA      DC   CL110' '
INAREA       DC   CL100' '
SLI          EQU  X'20'
READ         EQU  2
PRINT        EQU  9
SKIPTO1      EQU  X'8B'
             END  PROGRAM
```

Figure 3.5. Self-Relocating Sample Program

Linkage Editor Examples

LINK EDIT-AND-EXECUTE-EXAMPLE

```
    // JOB LINKEXEC
    *  LINK EDIT AND EXECUTE IN BACKGROUND, SINGLE PHASE, SINGLE OBJECT MODULE
    *  RELOCATABLE MODULE NOT CATALOGED, BACKGROUND PROGRAM
    *  NONSEQUENTIAL DASD & LABELED TAPE FILES TO BE PROCESSED
1.  // ASSGN SYSLNK,X'190'
2.  // OPTION LINK
3.     PHASE PROGA,*
       INCLUDE

4.     Relocatable object deck

    /*
5.  // LBLTYP NSD(2)
6.  // EXEC LNKEDT

7.     Any job statements required for execution such as ASSGN or label statements.

8.  // EXEC
       Data input as required.

    /*
    /&
    *  1.  TO CATALOG AND EXECUTE, CHANGE STATEMENT 2 TO // OPTION CATAL.
    *  2.  TO CATALOG ONLY, CHANGE STATEMENT 2 TO // OPTION CATAL AND
    *      REMOVE ALL STATEMENTS FOLLOWING LNKEDT EXCEPT /&
    *  3.  TO USE MODULE FROM RELOCATABLE LIBRARY, CHANGE STATEMENT 3
    *      TO INCLUDE MODULES AND REMOVE ALL STATEMENTS UP TO // LBLTYP.
```

Explanation for Link Edit and Execute

This example illustrates the basic concept of linkage editing and executing by using a single phase that is constructed from a single relocatable object deck contained in punched cards. The program is executed in the background partition. Labeled tape and nonsequential DASD files are to be processed when the phase is executed. No more than two extents are used by any DASD file.

Statement 1: No assignments are necessary, because the system units required for linkage editing are in the assumed configuration. However, an ASSGN for SYSLNK is included to illustrate its position relative to the OPTION statement in case assignment is required.

Statement 2: The OPTION LINK statement sets switches to indicate that a linkage editor operation is to be performed. If SYSLNK has not been assigned, the statement is ignored. Linkage editor control statements are not accepted unless the OPTION statement is processed. Because option is LINK, not CATAL, only link editing is performed; cataloging to the core image library does not occur.

Statement 3: The PHASE statement is copied on SYSLNK, because position 1 is blank and the LINK switch is on. The operands are not examined until SYSLNK becomes input to the linkage editor program.

When the PHASE statement is processed by the linkage editor, only one phase is constructed, because only one PHASE statement is submitted for the entire LNKEDT. The name of this phase is PROGA, as specified in the first operand. The second operand indicates the origin point for the phase. Because an * has been used, the phase begins in the next main storage location available, with forced doubleword alignment. Because this is the first and only phase, it is located at the end of the supervisor plus length of the label save area (reserved by LBLTYP) plus length of any area assigned to the COMMON pool (as designated by a CM entry in the relocatable module).

A relocation factor, either plus or minus, is used with the *, such as *+1024. This causes the origin point of the phase to be set relative to the * by the amount of the relocation term. This term can be expressed as:

```
X'hhhhhh' -- 1 to 6 hexadecimal digits
dddddddd  -- 1 to 8 decimal digits
nK        -- where K = 1024
```

*+1024 uses the second format and adds 1024 bytes to the origin location. +1K or +X'400' gives the same result as +1024.

Statement 4: The INCLUDE statement has no operands, so the system reads the records from SYSIPT and writes them on SYSLNK until SYSIPT has an end-of-data (/*) record. The data on SYSIPT is expected to be the object module in card image format that is used in this linkage editor operation. If the output of the language translator (SYSPCH) is placed on 2311/2314/2319 instead of cards, it cannot be used directly as SYSIPT in a linkage editor operation because the records contain a stacker select code in position 1. SYSPCH must be converted to an 80-position card image record.

Statement 5: The LBLTYP statement causes a computation of the number of bytes that are required for label storage data in the program to be linkage edited. In this example, 124 bytes are reserved (84 + [2x20]). The calculation is saved by job control and passed on first to the linkage editor and later to LIOCS.

Statement 6: The EXEC LNKEDT writes an ENTRY statement with no operand on SYSLNK and causes the system loader to bring in the linkage editor program.

Using the data just placed on SYSLNK as input, the linkage editor develops executable code. The output is placed in the next available space of the core image library (immediately after the last cataloged phase). This is true regardless of whether the program is cataloged or not. Cataloging causes the updating of the directory to reflect a new ending point for the library. If cataloging does not occur, the next program that is linkage edited overlays it. For this reason, a linkage edited program that is not cataloged is said to be placed in the temporary area of the core image library. Also, a program that is linkage edited without cataloging must be linkage edited whenever it is used. No ACTION options are specified. Therefore, in resolving the external references, the system makes use of the AUTOLINK feature. Error diagnostics and a main storage map are written on SYSLST, because SYSLST is assigned.

Statement 7: Because the program is not cataloged, it must be executed immediately. Any pertinent job control statements are entered at this point.

Statement 8: An EXEC statement with no operand indicates that the phase to be executed was just linkage edited. Therefore, no search of the core image directory is required, and the system loader brings the program into main storage from the temporary area and transfers control to its entry point. In this example, the entry point is either the address specified in the END record, or the phase load address if the END address is omitted, because the automatic ENTRY statement is in effect.

This example can be modified to illustrate the following:

1. Catalog and execute. To cause this phase to be cataloged rather than merely linkage edited, change the OPTION (statement 2) from LINK to CATAL. The core image library directory still refers to the old version of the program. It is not updated until /& has been read.

2. Catalog only. To catalog only, change the OPTION (statement 2) from LINK to CATAL and remove all data following the EXEC LNKEDT (statement 6) up to the /& statement.

3. Catalog object module in relocatable library. The name to catalog the object module into the relocatable library must be added to the INCLUDE statement. If the name is RELOCA, the statement becomes INCLUDE RELOCA. The relocatable object deck and /* statement are removed. This form of the INCLUDE statement is written on SYSLNK when it is read by job control. The linkage editor retrieves the object module when it encounters the INCLUDE statement because it uses SYSLNK for input.

CATALOG TO CORE IMAGE LIBRARY EXAMPLE

```
    // JOB CATALCIL
    *  LINK EDIT AND CATALOG TO CORE IMAGE LIBRARY
    *  SINGLE PHASE, MULTIPLE OBJECT MODULES, FOREGROUND PROGRAM
    *  MIXTURE OF CATALOGED AND UNCATALOGED RELOCATABLE MODULES
    *  LABELED TAPE FILES AND SEQUENTIAL DASD FILES TO BE PROCESSED
1.  // ASSGN SYSLNK,X'190'
2.  // OPTION CATAL
3.     PHASE PROGB,F+32768
4.     INCLUDE


       Relocatable object deck

    /*

       INCLUDE SUBRX
       INCLUDE SUBRY
       INCLUDE

       Relocatable object deck

    /*

5.  // LBLTYP TAPE
6.  // EXEC LNKEDT
7.  /&
```

Explanation for Catalog to Core Image Library

This example illustrates the cataloging of a single phase composed of multiple relocatable object modules. These modules are located in the input stream and in the relocatable library. Labeled tape files and sequential DASD files are processed when the phase is executed. The program is executed in a foreground partition. Assume that the foreground partition begins at location 32768.

Statement 1: The SYSLNK assignment indicates the relationship to the OPTION statement, although it is not required because of the assumed configuration.

Statement 2: The OPTION CATAL statement sets the LINK switches, as well as a CATAL switch. If SYSLNK is not assigned, the statement is ignored. The linkage editor control statements are not accepted unless the OPTION statement is processed. Linkage editing and cataloging to the core image library will occur.

Statement 3: Only one PHASE is constructed. It is cataloged to the core image library and retrieved by the name PROGB. Because this is to be a foreground phase, F plus the location address in the foreground partition must be specified.

A program may be linkage edited to any address that falls within a foreground partition. The load address does not have to coincide with the partition address. However, the program must be of such a size that it can reside in the available core defined from the load address to the end of the partition.

The address may be expressed in one of three forms:

X'hhhhhh' -- A hexadecimal number of 4 to 6 digits
dddddddd -- A decimal number of 5 to 8 digits
nnnnK -- Where K = 1024 and n is 2 to 4 digits

+X'8000', +32768, and +32K are equivalent. The actual origin point of the phase is adjusted upward from the address specification to allow for the partition save area, and the label information (LBLTYP statement reservation).

Statement 4: Four modules make up this phase. The first and last are not cataloged in the relocatable library; therefore the object decks must be on

SYSIPT, and each must be followed by the end-of-data record (/*). SUBRX and SUBRY are cataloged previously to the relocatable library by those names. Job control puts the uncataloged modules on SYSLNK in place of their INCLUDE statements. Job control copies the INCLUDE statements for the cataloged modules.

Statement 5: The LBLTYP statement has the operand TAPE, rather than NSD because labeled tapes and sequential DASD files are processed when the phase is executed. Eighty bytes are reserved ahead of the actual phase for label information. LBLTYP NSD is also satisfactory because it generates a minimum of 104 bytes and tapes require only 80.

Statement 6: The EXEC LNKEDT statement causes the system loader to bring in the linkage editor program. SYSLNK now becomes input to the linkage editor. It contains the following:

```
PHASE PROGB,F+32768
First uncataloged relocatable deck
INCLUDE SUBRX
INCLUDE SUBRY
Second uncataloged relocatable deck
ENTRY
```

The modules are linkage edited so that they occupy contiguous areas in main storage in the sequence in which they appear in the input stream. When the linkage editing is completed, cataloging to the core image library occurs because of the CATAL option. The core image directory is checked to make sure the new phase entries fit. If not, the job is canceled. The directory is scanned for any match to a phase being cataloged. A match is deleted from the directory. The system directory is updated to reflect the changes. Job control is brought into main storage.

Statement 7: Because CATAL was specified, a special routine is executed when the /& control statement is read by job control. This routine updates the transient, library-routine, and foreground-program directories. A system status report is printed to reflect the usage and available space in each of the libraries and directories. These operations do not occur in a LINK situation. The /& resets the CATAL option, that is, it turns off the LINK and CATAL switches.

The example can be modified to illustrate a catalog-and-execute operation by inserting the following data between the EXEC LNKEDT and /& statements:

1. Any job control statements required for execution or PROGB
2. A // EXEC statement
3. Any card reader input for PROGB

Note that the actual update of the directories and the system status report are delayed until completion of the execution of PROGB, when /& is read. From a system design standpoint this is not desirable because of possible operational problems. Making the execution of PROGB a separate job avoids any difficulties. All core image library directories are updated at /&. This is time consuming and should not be done for each module cataloged.

EXECUTE LINKAGE EDITOR IN FOREGROUND AND CATALOG TO PRIVATE CORE IMAGE LIBRARY EXAMPLE

```
    // JOB CATLCIL
    *  LINK EDIT AND CATALOG TO PRIVATE CORE IMAGE LIBRARY
    *  LINKAGE EDITOR EXECUTING IN FOREGROUND
    *  SINGLE PHASE, MULTIPLE OBJECT MODULES, FOREGROUND PROGRAM
    *  MIXTURE OF CATALOGED AND UNCATALOGED RELOCATABLE MODULES
    *  LABELED TAPE FILES AND SEQUENTIAL DASD FILES TO BE PROCESSED
1.  ASSGN SYSCLB,X'191'
2.  // ASSGN SYSLNK,X'190'
3.  // OPTION CATAL
4.     PHASE PROGB,S
5.     INCLUDE


       Relocatable object deck

    /*

       INCLUDE SUBRX
       INCLUDE SUBRY
       INCLUDE

       Relocatable object deck

6.  // LBLTYP TAPE
7.  // EXEC LNKEDT
8.  /&
```

Explanation for Catalog to Private Core Image Library

This example illustrates the execution of the linkage editor in a foreground partition and the cataloging of a phase to a private core image library. This function is possible only in a system supporting the batched-job foreground and private core image library options. The phase being cataloged is the same as that in the previous example where the link edit was executed in the background.

Statement 1: The assignment of a private library is accomplished by the ASSGN SYSCLB statement. The label for SYSCLB must be stored on PARSTD or STDLABEL cylinder, or, if the DLBL statement is included in the job stream, it must follow the ASSGN SYSCLB statement.

Statement 2: The SYSLNK assignment indicates the relationship to the OPTION statement, although it is not required because of the assumed configuration.

Statement 3: The OPTION CATAL statement sets the LINK switches, as well as a CATAL switch. If SYSLNK is not assigned, the statement is ignored. The linkage editor control statements are not accepted unless the OPTION statement is processed. Linkage editing and cataloging to the core image library will occur.

Statement 4: Only one PHASE is constructed. It is cataloged to the private core image library and retrieved by the name PROGB. An origin point of S origins PROGB at the starting address of the foreground partition, plus the length of the save areas and the area assigned to the COMMON pool, if any.

Statement 5: Four modules make up this phase. The first and last are not cataloged in the relocatable library; therefore, the object decks must be on SYSIPT, and each must be followed by the end-of-data record (/*). SUBRX and SUBRY are cataloged previously to the relocatable library by those names. Job control puts the uncataloged modules on SYSLNK in place of their INCLUDE statements. Job control copies the INCLUDE statements for the cataloged modules.

Statement 6: The LBLTYP statement has the operand TAPE, rather than NSD because labeled tapes and sequential DASD files are processed when the phase is executed. Eighty bytes are reserved ahead of actual phase for label information. LBLTYP NSD is also satisfactory because it generates a minimum of 104 bytes and tapes require only 80.

Statement 7: The EXEC LNKEDT statement causes the system loader to bring in the linkage editor program. SYSLNK now becomes input to the linkage editor. It contains the following:

PHASE PROGB,S
First uncataloged relocatable deck
INCLUDE SUBRX
Second uncataloged relocatable deck

The modules are link-edited so that they occupy contiguous areas in main storage in the sequence in which they appear in the input stream. When the linkage editing is completed, cataloging to the private core image library occurs because of the CATAL option. The private core image directory is checked to make sure the new phase entries fit. If not, the job is canceled. The directory is scanned for any match to a phase being cataloged. A match is deleted from the directory. The system directory is updated to reflect the changes. Job control is brought into main storage.

Statement 8: Because CATAL was specified, a special routine is executed when the /& control statement is read by job control. This routine updates the transient, library-routine, and foreground-program directories for the private core image library. A system status report is printed to reflect the usage and available space in each of the libraries and directories. These operations do not occur in a LINK situation. The /& resets the CATAL option, that is, it turns off the LINK and CATAL switches.

The example can be modified to illustrate a catalog-and-execute operation by inserting the following data between the EXEC LNKEDT and /& statements:

1. Any job control statements required for execution or PROGB
2. A // EXEC statement
3. Any card reader input for PROGB

Note that the actual update of the directories and the system status report are delayed until completion of the execution of PROGB, when /& is read. From a system design standpoint this is not desirable because of possible operational problems. Making the execution of PROGB a separate job avoids any difficulties. All core image library directories are updated at /&. This is time consuming and should not be done for each module cataloged.

COMPILE AND EXECUTE EXAMPLE

```
    // JOB COMPEXEC
    *  COMPILE OR ASSEMBLE, LINK EDIT AND EXECUTE
    *  SINGLE PHASE, MULTIPLE OBJECT MODULES, BACKGROUND PROGRAM
    *  SEQUENTIAL DASD FILES TO BE PROCESSED
    *  INPUT TO LINKAGE EDITOR FROM LANGUAGE TRANSLATOR, RELOCATABLE LIBRARY AND SYSIPT.
1.  // ASSGN SYSLNK,X'190'
2.  // OPTION LINK
3.     PHASE PROGA,S
4.  // EXEC COBOL


       COBOL source statements

    /*
       INCLUDE SUBRX
5.     INCLUDE

       Relocatable object module

    /*
6.     ENTRY BEGIN1
    // EXEC LNKEDT

7.     Any job control statements required for PROGA execution.

    // EXEC

       Any input data required for PROGA execution.

    /*
    /&
```

Explanation for Compile and Execute

The language translators provide the option of placing their output on SYSLNK rather than SYSPCH. Because the linkage editor uses SYSLNK for input, a program can be assembled or compiled, then linkage edited and executed. This operation, known as assemble/compile and execute, is illustrated by this example.

All three sources of object module input to the linkage editor are used: SYSIPT, the relocatable library, and the output from a language translator. It is assumed that the phase is executed in the background partition, and that only sequential DASD files or unlabeled tape files are processed.

Statement 1: The SYSLNK assignment is given to illustrate the relationship to the OPTION statement, although it is not required because of the assumed configuration.

Statement 2: Because SYSLNK is assigned, the OPTION LINK statement sets the link indicator switches.

Statement 3: The PHASE statement must always precede the relocatable modules to which it applies; therefore, it is written on SYSLNK first for later use by the linkage editor. S is the origin point, that is, the phase originates with the first doubleword at the end of the supervisor plus length of the label save area (as defined by LBLTYP) plus length of the area assigned to the COMMON pool (if any). This gives the same effect as * gives for a single phase or the first phase. As with the *, the S may be used with a relocation factor, for example, S+1024. The factor must always be positive, because a negative factor could cause the origin point to overlay the supervisor.

Statement 4: The appropriate language translator is called (in this case, COBOL). The normal rules for compiling are followed; the source deck must be on the unit assigned to SYSIPT and the /* defines the end of the source data. Because the LINK switches are set, the output of the language translator is written on SYSLNK. Except for PL/I, FORTRAN (F) and the ASSEMBLER (F) and 14K variant, the DECK option is ignored when SYSLNK is used.

Statement 5: The INCLUDE SUBRX statement is written on SYSLNK. The linkage editor retrieves the named module from the relocatable library. Because the operand is blank, the next INCLUDE statement signifies that the relocatable module is on SYSIPT. The data on SYSIPT is copied on SYSLNK until the /* statement.

Statement 6: The ENTRY statement is written on SYSLNK as the last linkage control statement. The symbol BEGIN1 must be the name of a CSECT or a label definition defined in the first phase. The address of BEGIN1 becomes the transfer address for the first phase of the program. The ENTRY statement is used because the user wishes to provide a specific entry point rather than to use the point specified in the END record or the load address of the phase. The ENTRY statement affects the first, or only, phase.

Statement 7: No LBLTYP statement is required, because only sequential DASD files are to be processed. The rest of the statements follow the same pattern as discussed in the Linkage Editing and Execute example. The input from SYSLNK to the linkage editor is:

```
PHASE PROGA,S
Relocatable module produced by COBOL
  compilation
INCLUDE SUBRX
Relocatable module from SYSIPT
ENTRY BEGIN1
```

If an error is detected during compilation of a source program, the LINK option is suppressed. Under these circumstances the EXEC LNKEDT and EXEC statements are ignored in this example. This LINK option suppression should be kept in mind if a series of programs is to be compiled and cataloged as a single job. Failure of one job step would cause failure of all succeeding steps. Remember that an OPTION LINK cannot be given if OPTION CATAL is in effect, because message 1S1nD (STATEMENT OUT OF SEQUENCE) results. This is an error in instruction to the system because CATAL has functions that must be performed when the next /& statement is read. Therefore, the CATAL switch must remain on, and linkage editing only cannot be performed.

CATALOG FOR PHASE OVERLAY EXAMPLE

```
    // JOB MULTPHAS
    *  LINK EDIT AND CATALOG TO CORE IMAGE LIBRARY
    *  MULTIPLE PHASES, MULTIPLE OBJECT MODULES, BACKGROUND PROGRAM
    *  NO LABELED TAPE OR NONSEQUENTIAL DASD FILES TO BE PROCESSED
    // ASSGN SYSLNK,X'190'
1.  // OPTION CATAL
2.     PHASE PHASEA,ROOT
       INCLUDE MOD1
3.     PHASE PHASEB,*
       INCLUDE MOD2
4.     PHASE PHASEC,PHASEB
       INCLUDE MOD3
5.  // EXEC LNKEDT
    /&
```

Explanation for Catalog for Phase Overlay

Sometimes it is not possible to bring an entire program into main storage because it requires more bytes of main storage than are available. To solve this problem the program can be broken into separate phases that can be brought into main storage as required, overlaying all or part of another phase if desired. The linkage editing of three cataloged relocatable modules into phases for overlay is illustrated by this example.

Statement 1: The OPTION CATAL sets the switches so that the phases can be linkage edited and cataloged into the core image library.

Statement 2: PHASEA is considered the ROOT phase, that is, it is always resident in main storage during program execution. The origin point is the first doubleword address after the supervisor plus length of the label save area (if any) plus length of the area assigned to the COMMON pool (if any). Only the first phase statement is permitted to specify ROOT.

Statement 3: The * in the PHASE card for PHASEB causes MOD2 to be linkage edited at the end of PHASEA.

Statement 4: Because PHASEB is specified as the load address of PHASEC, it is linkage edited into the same address as PHASEB. The symbol that designates the origin point may be a previously defined phase name as in this example, a previously defined control section, or a previously defined external label. A plus or minus relocation factor may be used (for example, PHASE2+100).

Statement 5: The EXEC LNKEDT causes all three phases to be linkage edited and cataloged. When the phases are executed, the ROOT phase normally is loaded by the system loader. The other phases would probably be brought into main storage by means of the FETCH or LOAD macro issued by the calling phase.

SUBMODULAR STRUCTURE EXAMPLE

```
    // JOB SUBMOD
    *  LINK EDIT AND CATALOG TO CORE IMAGE LIBRARY
    *  MULTIPLE PHASES, ONE OBJECT MODULE (SUBMODULAR STRUCTURE)
    *  BACKGROUND PROGRAM, NO LABEL STORAGE RESERVATION
    // ASSGN SYSLNK,X'190'
    // OPTION CATAL
1.     PHASE PHASE1,ROOT
       INCLUDE ,(CSECT1,CSECT3)
2.     PHASE PHASE2,*
       INCLUDE ,(CSECT2,CSECT5)
3.     PHASE PHASE3,PHASE2
       INCLUDE ,(CSECT4,CSECT6)
4.     INCLUDE


       Relocatable object deck
    /*
    // EXEC LNKEDT
    /&
```

Explanation for Submodular Structure

Several relocatable modules are structured into several phases. In this example, a single object module is broken into several phases. The object module is composed of CSECT1-CSECT6. It is structured into three phases with overlay. The module is not cataloged in the relocatable library. Only the PHASE AND INCLUDE statements are discussed.

Statement 1: The INCLUDE statement tells the linkage editor to place CSECT1 and CSECT3 into PHASE1. The sequence in which the CSECTs are linkage edited is determined by the sequence in the input module rather than the sequence in the INCLUDE statement. (CSECT3,CSECT1) would give the same result as (CSECT1,CSECT3). The sequence can be controlled by issuing separate INCLUDE statements. For example, INCLUDE ,(CSECT3) followed by INCLUDE ,(CSECT1) causes CSECT3 to be linkage edited before CSECT1, regardless of the sequence in the object module.

Note that the first operand is missing in the INCLUDE statement, as indicated by the leading comma. This format of the INCLUDE statement searches the next succeeding object module on SYSLNK to locate the named CSECTs. See Statement 4.

PHASE1 is located at the end of the supervisor plus length of the label save area, and the COMMON area (if any).

Statement 2: The INCLUDE statement causes CSECT2 and CSECT5 to be used for PHASE2. This phase is located at the end of PHASE1.

Statement 3: PHASE3 is made up of CSECT4 and CSECT6 and overlays PHASE2 because its origin point is at the same address as PHASE2.

Statement 4: This INCLUDE statement with a blank operand is required to write the object module that follows in the card reader onto SYSLNK, to satisfy the INCLUDE statements with a blank first operand.

With the sequence of statements shown in the example, the PHASE and INCLUDE statements are read from SYSRDR. However, it is permissible to read PHASE and INCLUDE statements from SYSIPT. To do this, Statement 4 (INCLUDE blank) is placed ahead of Statement 1. The INCLUDE with the blank operand directs job control to read the following data (which includes the PHASE, INCLUDE, and then the object module) on SYSLNK from SYSIPT to the /* statement. If SYSRDR and SYSIPT are separate devices, take care to place the PHASE and INCLUDE statements on the correct device.

PHASE and INCLUDE statements can also be in the relocatable library. If the object module is in the relocatable library under the name MOD1, the following changes are made:

1. Remove Statements 1 through 3, and add module name to Statement 4.

```
// JOB SUBMOD
// OPTION CATAL
   INCLUDE MOD1
// EXEC LNKEDT
```

2. When the relocatable module is cataloged to the library, precede it with the following statements:

```
PHASE PHASE1,*
INCLUDE MOD1,(CSECT1,CSECT3)
PHASE PHASE2,*
INCLUDE MOD1,(CSECT2,CSECT5)
PHASE PHASE3,PHASE2
INCLUDE MOD1,(CSECT4,CSECT6)
Relocatable object deck
```

This form of the INCLUDE statement causes the linkage editor to search the module that follows the last INCLUDE statement in the library for the required control sections.

SELF-RELOCATING AND MULTIPLE LINK EDITS EXAMPLE

```
    // JOB MULTCATL
    *  SEVERAL LINK EDITS AS A SINGLE JOB
    // OPTION CATAL
1.     PHASE PROG1,+0
       INCLUDE PROG1
    // EXEC LNKEDT
2.     PHASE PROG2,*
       INCLUDE

       Relocatable object module
    /*
    // EXEC LNKEDT
       PHASE PROG3,*
       INCLUDE PROG3
    // EXEC LNKEDT
    /&
    // JOB LINKGO
3.  // OPTION LINK
       PHASE PROG4,*
       INCLUDE

       Relocatable object module

    /*
    // EXEC LNKEDT

       Any job control statements required for PROG4 execution.
    // EXEC

       Any input data required for PROG4.
    /*
       PHASE PROG5,*
       INCLUDE PROG5
    // EXEC LNKEDT

       Any job control statements required for PROG5 execution.
    // EXEC

       Any input data required for PROG5.
    /*
    /&
```

Explanation for Self-Relocating and Multiple Link Edits

The linkage editing requirements for a self-relocating program and the combining of several cataloging or link-and-execute job steps into a single job are illustrated in this example. Use discretion in deciding how many steps should be combined before a /&, because a failure in one step can cause successive steps to be bypassed unnecessarily.

Statement 1: The +0 displacement as the origin part for PROG1 designates this program as self-relocating. Once this program is cataloged to the core image library, it may be executed in any partition. It can be initiated into the background partition by job control, if the supervisor was generated with multiprogramming capabilities. A non-MPS supervisor will give message 0P77I (CANCELED DUE TO INVALID ADDRESS). However, the program can be loaded by using the LOAD macro in a calling phase.

Statement 2: PROG2 and PROG3 are linkage edited and cataloged as job steps within the job MULTCATL. Note that OPTION CATAL holds for these steps.

Statement 3: A new job is initiated because the succeeding job steps are linked and executed without cataloging. (OPTION LINK cannot be issued when OPTION CATAL is in effect.) Note that OPTION LINK need not be reissued before the next job step.

Checkpoint/Restart

When a background program or a batched-job foreground program is expected to run for an extended period of time, provision may be made for taking checkpoint records periodically during the run. The records contain the status of the job and system at the time the records were written. Thus, they provide a means of restarting at some midway point rather than at the beginning of the entire job, if processing must be terminated for any reason before the normal end of job.

For example, some malfunction, such as a power failure, may occur and cause such an interruption. If checkpoint records are written periodically, operation can be restarted with a set of checkpoint records written before the interruption. These records contain everything needed to reinitialize the system when processing is restarted.

Any programmer logical unit (SYS000-SYSmax) assigned to tape, or the 2311, 2314, or 2319 can be used for recording checkpoints if the proper file definitions are made and the correct label statements are submitted. Checkpoints must not be taken on ASCII tape files.

The Disk Operating System includes routines to take checkpoint records and to restart a job at a given checkpoint. The checkpoint and restart routines are included in the core image library when the system is generated. The checkpoint routine is executed in the logical transient area and is called in response to a CHKPT macro instruction in the problem program. The restart routine is called by job control when it reads a RSTRT control statement. When a program is restarted, the user must reissue any STXIT macro instructions that are desired because the STXIT linkages established before the checkpoint was taken are destroyed. Checkpoint/restart does not save or restore floating point registers. (If needed, these registers should be stored in the problem program area before issuing CHKPT macro, and restored in a user restart routine.)

Only background programs or batched-job foreground programs may be checkpointed. Checkpoint records are written on a 2311, 2314, or 2319 DASD or on magnetic tape. Each checkpoint is uniquely identified. When it is restarted, the RSTRT control statement specifies which checkpoint is to be loaded. If multireel files are being used, the operator must be aware of which reels were being processed when the checkpoint was taken.

Multitasking users should only issue the CHKPT macro in the main task with no subtasks attached. In addition, no tracks on any DASD should be in the hold state. A multitasking abnormal termination routine should not contain a CHKPT macro. Checkpoints should be taken while a program is running successfully, not while it is canceling. Checkpointed programs must be restarted in the same partition in which they were checkpointed. Multiple jobs maybe checkpointed on the same tape.

Checkpoint records written by previous versions of the system are not acceptable to the current version of the system. However, if they are embedded on magnetic tape, they are bypassed by the current version.

It is possible to increase partition allocation between the time the checkpoint is taken and the time the program is restarted, if the starting address of the partition remains unchanged.

PROBLEM PROGRAM RESPONSIBILITIES

Use of CHKPT Macro

Any partition, except a foreground partition in a single program mode, can issue the CHKPT macro successfully. If multitasking, only the main task can successfully checkpoint. CHKPT is ignored when issued by a subtask, a foreground partition in single program mode, or in any of the following additional conditions:

1. The device on which the checkpoint records are to be written is not a magnetic tape or a disk pack. (The device must be a 2311, 2314, or 2319 disk if the filename operand is present.)
2. End of reel is detected while writing the checkpoint on tape.
3. The area on disk is not large enough for a single checkpoint.
4. The macro is issued by a teleprocessing program that has any I/O operation(s) pending on a teleprocessing device.
5. The user-specified end address is greater than the end of the problem program area.
6. The CHKPT macro is issued before the disk checkpoint file is opened.

7. Any of the required DTFPH parameters for the disk checkpoint file contain errors.

8. If a subtask is attached in the partition being checkpointed.

9. If any DASD track for the partition being checkpointed is in the HOLD state.

Note: Checkpoint records are not permitted on ASCII tape files.

If a checkpoint is ignored, control returns to the user with binary zeros in register 0. Otherwise, register 0 contains the appropriate checkpoint number (in unpacked decimal).

Checkpoints are usually taken after a specified period of time has elapsed, or after a certain volume of input is processed. When multitasking, use the following as a guide for selecting a method:

1. The multitasking operation requires checkpoints to be taken on a time interval basis. Therefore, at main task execution time, a STXIT macro establishes linkage for an interval timer interrupt. In the main task interval timer routine, the problem program issues WAIT macros to wait for the detachment of each subtask in the partition, and then takes the checkpoint. If the main task must take an immediate checkpoint, the interval timer routine in the main task must first detach all subtasks, disregarding current processing, before it can successfully issue the CHKPT macro.

2. The multitasking operation requires checkpoints to be taken on a volume basis. Therefore, the main task attaches the subtasks necessary to perform the job, and then issues WAIT macros to wait for each subtask in the partition to detach. Each subtask keeps a count on the unit of work to be performed and detaches when it is finished. When all subtasks are detached, the main task can take the checkpoint.

After the checkpoint is taken, the main task can then either attach more, or the same, subtasks to continue processing.

CHKPT Macro

Name	Operation	Operand
name	CHKPT	SYSnnn,{restart address (r1)} [,{end address (r2)}] [,{tpointer (r3)}] [,{dpointer (r4)}] [,{Filename (r5)}]

SYSnnn specifies the logical unit on which the checkpoint information is stored. It must be a magnetic tape or a disk pack. (See Checkpoint File.)

Restart address (or r1) specifies a symbolic name of the problem program statement (or register containing the address) at which execution is to restart if processing must be continued later.

End address (or r2) is a symbolic name (or register containing the address) of the uppermost byte of the problem program area required for restart. This address must follow the logic modules being included from the relocatable library.

If this operand is omitted, all of main storage allocated to the partition are checkpointed.

This operand provides two advantages:

1. Less time and space is required for recording the checkpoint record set.

2. If a program using 24K of storage is being run in a larger system and only 24K is checkpointed, that program can be restarted, either on a 24K system or as a 24K partition in a multiprogramming system.

Tpointer (or r3) is the symbolic name of an eight-byte field contained in the problem program area. (See Repositioning Magnetic Tape.)

Dpointer (or r4) is the symbolic name of a DASD operator verification table that the user can set up in his own area of main storage. (See DASD Operator Verification Table.)

Filename (or r5) is used only for checkpoint records on disk. It is the name of the associated DTFPH macro. (See Checkpoint on Disk.) Special register notation cannot be used with any of these operands.

Information That Is and Is Not Saved: When the CHKPT macro is issued, the following information is saved:

- Information for the restart and other supervisor or job control routines.
- The general registers.
- Bytes 8-10 and 12-45 of the communication region.
- The problem program area (see End Address Operand).
- All DASD file protection extents attached to logical units belonging to the checkpointed program.

The following information is not saved:

- The floating point registers. (If needed, these registers should be stored in the problem program area before issuing CHKPT, and restored in a user restart routine.)
- Any linkage to user routines set by the STXIT macro. (If needed, STXIT should be used in user's restart routine.)
- Any timer values set by the SETIME macro. (If needed, SETIME should be used in a user's restart routine.)
- The program mask in problem program PSW. (If other than all zeros is desired, the mask should be reset in user's restart routine.)

NOTES FOR DASD AND MICR FILES

DASD system input or output files (SYSIPT, SYSLST, etc.) must be reopened at restart time. In the user's restart routine, the programmer must be able to identify the last record processed before checkpoint.

Magnetic Ink Character Reader (MICR) files require the DTFMR supervisor linkages to be initiated at restart time. This can be accomplished by reopening the MICR file in the user's restart routine. Because the OPEN macro clears the document buffer, the problem program must disengage the device and process all follow up documents in the document buffer before taking each checkpoint.

CHECKPOINT FILE

The checkpoint information must be written on a disk pack or a magnetic tape (either 7- or 9-track, EBCDIC only). The 7-track tape can be in either data conversion or translation mode; however, the magnetic tape unit must have the data conversion feature. On 7-track tapes, the 20-byte checkpoint header and trailer labels are written in the mode of the tape (Figure 3.9). The data records are written in data convert mode, odd parity.

Checkpoints On Tape

The programmer can either establish a separate file for checkpoints or embed the checkpoint records in an output data file (EBCDIC only). When the data file is read at a later time using logical IOCS, the checkpoint records are automatically bypassed. If physical IOCS is used, the user must program to bypass the checkpoint record sets.

If a separate magnetic tape checkpoint file with standard labels is maintained, the labels should be either checked by an OPEN routine or bypassed by a MTC command before the first checkpoint is taken.

Checkpoints On Disk

If checkpoints are written on disk, the following must be observed:

1. Define area of disk to be used by writing a DTFPH macro and using a DLBL, EXTENT label set.
2. The number of tracks required is computed as follows:

$$n\left[1 + \frac{\frac{x}{30} + \frac{y}{20}}{18} + \frac{c}{z}\right]$$

where:

n = the number of sets of checkpoint records to be retained. (When the defined extent is full, the first set of checkpoint records is overlaid.)

c = The number of bytes to be checkpointed in the user's problem program up to the end address specified in the CHKPT macro operand.

x = The number of disk extents including nonoverlapping split-cylinder extents. If split-cylinder extents overlap on the same cylinder, the number of extents counted is the one used by the program. (This number is zero if DASD file-protect is not used.)

y = same as preceding for 2321.

z = 3625, if checkpoint records are written on a 2311. 7249, if checkpoint records are written on a 2314/2319.

For each division, the remainder is rounded to the next highest whole number before multiplying by n.

3. Open the area on disk by issuing an OPEN to the DTFPH.
4. Issue a CHKPT macro that points to the DTFPH to be used.
5. When restarting checkpointed jobs, the DTFPH filename is specified in the RSTRT job control card.
6. Each program can use a common checkpoint file or define a separate one. If a common file is used, only the last program using the file can be restarted.

See Figure 3.6 for an example using the checkpointed facility on disk.

```
PAYROLL      START
               •
               •
CHKPDSK      DTFPH      DEVICE=2311,MOUNTED=SINGLE,TYPEFLE=OUTPUT
               •
               •
             OPEN       CHKPDSK
               •
               •
             CHKPT      SYS004,RSTRT,END,,DVER,CHKPDSK
               •
               •
             END
```

```
// JOB CHKPT

 ASSGN SYS004,X'190'

 ASSGN SYS000,X'180'

 ASSGN SYS001,X'181'

 ASSGN SYS002,X'182'

// DLBL CHKPDSK,'CHECKPOINT FILE',,,

// EXTENT SYS004,DOS-II,,,1900,89

// EXEC PAYROLL
```

Figure 3.6. Using Checkpoint Facility on Disk

REPOSITIONING I/O FILES

The I/O files used by the checkpointed program must be repositioned on restart to the next record that the user wants to read or write. The checkpoint facility does not provide aids for repositioning unit record files. The programmer must establish his own repositioning aids and communicate these to the operator, when necessary. Some suggested ways are:

1. Take checkpoints at a logical breaking point in the data, such as paper tape end-of-reel.
2. Switch card stackers after each checkpoint.
3. Print information at the time of checkpoint to identify the record in process.
4. Issue checkpoints on operator demand.

User sequential DASD input, output, or work files require no repositioning.

When updating DASD records in an existing file, the programmer must be able to identify the last record updated at the time of the checkpoint in case he needs to restart. This can be done in various ways:

1. Create a history file to record all updates by dumping an image of the direct access record on tape as soon as it has been read. When a restart is initiated, these records can be used to rewrite the file and establish the status that existed when the corresponding checkpoint was taken. When this is completed, normal restart procedures are accomplished and reprocessing begins.
2. Create a field in updated records to identify the last transaction record that updated it. This field can be compared with each transaction at restart time.

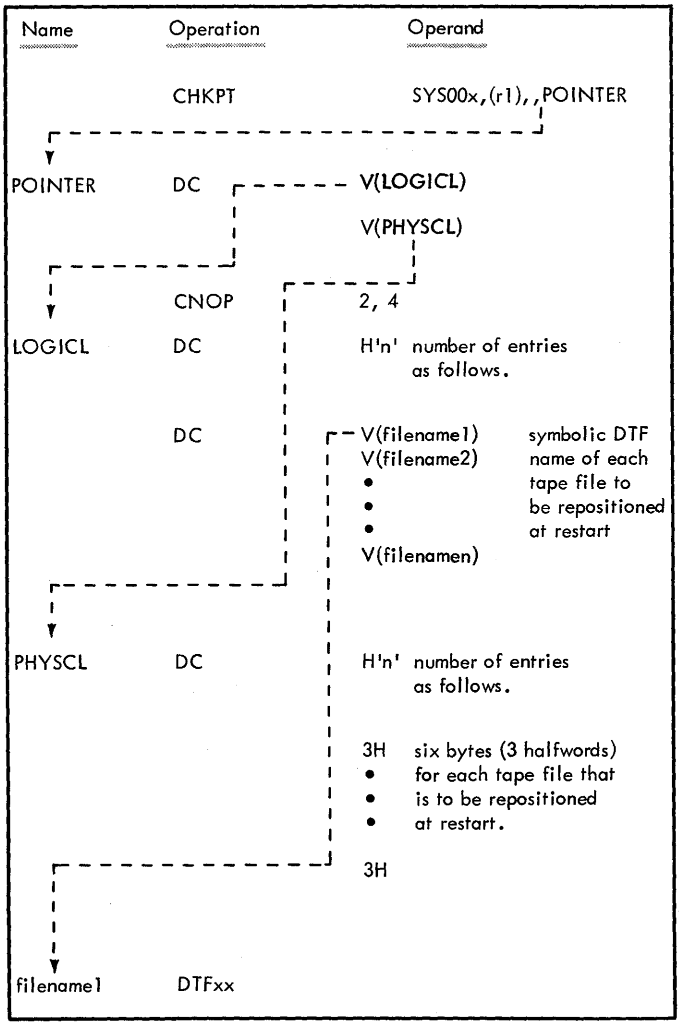
Repositioning Magnetic Tape

Checkpoint provides some aid in repositioning magnetic tape files at restart. Files can be repositioned to the record following the last record processed at checkpoint.

The following discussion presents the procedure in correlation with Figure 3.7.

The fourth operand of the CHKPT macro points to two V-type address constants that the user specifies in his coding. The order of these constants is important.

1. The first constant points to a table containing the filenames of all the logical IOCS magnetic tape files that are to be repositioned.
2. The second constant points to a table containing repositioning information for physical IOCS magnetic tape files that are to be repositioned.
3. If the first, second, or both constants are zero, no tapes processed by logical, physical, or both types of IOCS, respectively, are repositioned.



```
Name        Operation   Operand

            CHKPT       SYS00x,(r1),,POINTER

POINTER     DC          V(LOGICL)
                        V(PHYSCL)

            CNOP        2, 4
LOGICL      DC          H'n'   number of entries
                               as follows.

            DC          V(filename1)    symbolic DTF
                        V(filename2)    name of each
                        •               tape file to
                        •               be repositioned
                        •               at restart
                        V(filenamen)

PHYSCL      DC          H'n'   number of entries
                               as follows.

                        3H     six bytes (3 halfwords)
                        •      for each tape file that
                        •      is to be repositioned
                        •      at restart.
                        3H

filename1   DTFxx
```

Figure 3.7. Procedure for Building Tape Repositioning Tables

If the tables are contained in the same CSECT as the CHKPT macro, the constants may be defined as A-type constants. The user must build the tables discussed. Each filename in the logical IOCS table points to the corresponding DTF table where IOCS maintains repositioning information. The user should note the following:

1. Magnetic tapes with nonstandard labels should be repositioned past the labels at restart time (presumably the labels are followed by a tapemark so that foreward space file may be used).

2. If a tape that is to be repositioned is processed with nonstandard labels and is read backwards, the user must keep a physical IOCS repositioning table, because the physical record count kept by IOCS will be incorrect. The physical record count must be the number of forward reads necessary for restart to position the tape.

3. Restart does not rewind magnetic tapes when repositioning them.

4. A multifile reel should be prepositioned to the beginning of the desired file.

5. The correct volume of a multivolume file must be mounted for restart.

6. For tapes with a standard VOL label, restart writes the file serial number and volume sequence number on SYSLOG, and gives the operator the opportunity to verify that the correct reel is mounted.

7. IOCS can completely reposition files on system logical units (SYSIPT, SYSLST, etc), if the tape is not shared with any other program and if the user keeps a physical IOCS repositioning table. However, if a system logical unit file is shared with other programs, a problem exists. Output produced after the checkpoint is duplicated at restart. Input records must be reconstructed from the checkpoint, or the user restart routine must find the last record processed before checkpoint.

The entries in the physical IOCS table are as follows:

First halfword: hexadecimal representation of the symbolic unit number of the tape (copy from CCB bytes 6 and 7).

Second halfword: number of files within the tape in binary notation. That is, the number of tapemarks between the beginning of tape and the position at checkpoint.

Third halfword: number, in binary notation, of physical records between the preceding tapemark and the position at checkpoint.

DASD Operator Verification Table

If the Dpointer operand in the CHKPT macro is used, the user can build a table in his own area of main storage to provide the symbolic unit number and the bin (cell) number of each DASD file used by his program. At restart, the volume sequence number of these files is printed on SYSLOG, and the operator can verify them.

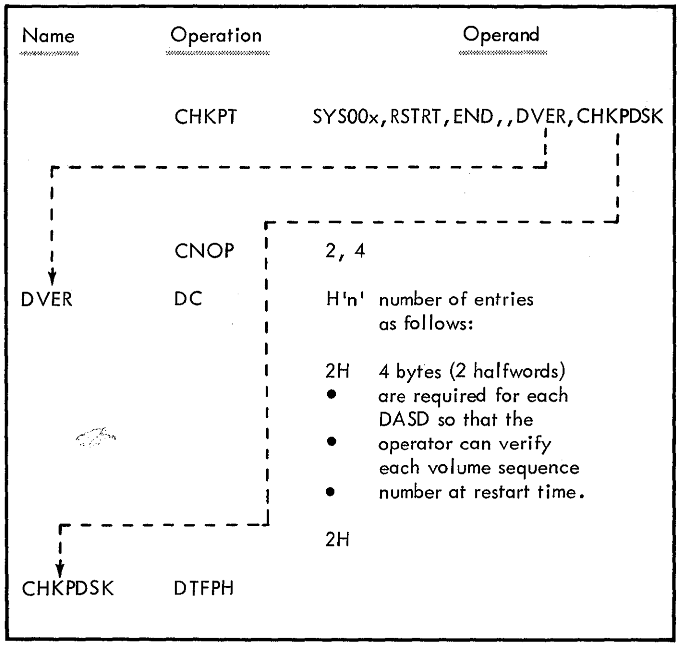
The entries in the DASD operator verification table must consist of the following two halfwords, in the order stated:

1. The symbolic unit in hexadecimal notation copied from CCB bytes 6 and 7.

2. The bin (cell) number in hexadecimal notation is always zero, except for a 2321, in which case the bin number varies with the cell (0-9) being verified.

There must be one table entry for each DASD unit to be verified by the operator.

See Figure 3.8 for the procedure for building a DASD operator verification table.



```
Name      Operation   Operand

          CHKPT       SYS00x,RSTRT,END,,DVER,CHKPDSK

          CNOP        2, 4

DVER      DC          H'n'  number of entries
                            as follows:

                      2H    4 bytes (2 halfwords)
                      •     are required for each
                            DASD so that the
                      •     operator can verify
                            each volume sequence
                      •     number at restart time.

                      2H

CHKPDSK   DTFPH
```

Figure 3.8. Procedure for Building DASD Operator Verification Table

BYPASSING EMBEDDED CHECKPOINT RECORDS ON TAPE WITH PHYSICAL IOCS

The checkpoint information saved is written as a set of magnetic tape records consisting of a 20-byte header record, as many core-image records as required to save the necessary parts of main storage, and a 20-byte trailer record identical to the header. See Figure 3.9 for the format of header and trailer record.

If checkpoint sets are embedded in a file being read with physical IOCS, they must be recognized and bypassed. On any mode input tape, checkpoint sets may be identified by the first 12 bytes of the header or trailer records. Note that when reading backwards, the checkpoint header occupies the 20 low-order bytes of the input area.

Bytes	Contents
0-11	/// CHKPT //
12-13	The number, in binary, of core image records following the header.
14-15	The total number, in unpacked hexadecimal, of records following the header.
16-19	The serial number of the check point.

Figure 3.9. Format of the Checkpoint Header/Trailer Records

When bypassing checkpoint sets, three methods are possible:

1. Go into a read loop (forward or backward) until the checkpoint trailer (header if backward) is encountered.
2. Extract the count from bytes 12-13 of the header (or trailer if backwards), add 2 to this, and forward-space (or backspace) that number of records. Read commands could also be used.
3. Extract bytes 14-15 of the header (or trailer if backwards), pack and convert the field to binary, and forward-space (or backspace) that number of records. Read commands could also be used.

When bypassing checkpoint sets on 7-track tapes in translate mode, only method 3 can be used and only forward-space (or backspace) record commands (not reads) can be used. Reads would create data checks.

BYPASSING CHECKPOINT RECORDS ON TAPE WITH LOGICAL IOCS

When a tape input file contains checkpoint records interspersed among the data records, the DTFMT macro parameter CKPTREC=YES is required. When this parameter is specified, logical IOCS bypasses the embedded checkpoint records.

RESTARTING CHECKPOINTED PROGRAMS

Job control prepares the system for restarting from a checkpoint by loading the restart program that repositions tape units, reinitializes the communication region, and stores the information from the RSTRT statement. The restart program handles the actual restarting of the problem program.

RSTRT Statement

The restart facility allows the programmer to continue execution of an interrupted job at a point other than the beginning. The procedure is to submit a group of job control statements including a restart (RSTRT) statement. The control statements necessary to restart a job from a checkpoint are:

1. JOB statement specifying the same job name used when the checkpoint was taken.
2. ASSGN statements for assigning I/O devices to the symbolic unit names.
3. RSTRT statement specifying the unit that contains the checkpoints and the checkpoint ID number taken from the message printed when the checkpoint was taken. The format of the RSTRT statement is:

```
// RSTRT SYSxxx,nnnn,filename
```

SYSxxx	Symbolic unit name of the device on which the checkpoint records are stored. This unit must have been previously assigned.
nnnn	Identification of the checkpoint record to be used for restarting. This serial number is four characters and corresponds to the checkpoint identification used when the checkpoint was taken. The serial number is supplied by the checkpoint routine.
filename	Symbolic name of the 2311 or 2314 disk checkpoint file to be used for restarting. It must be identical to the filename of the DTFPH used to describe the disk checkpoint file and the fifth parameter of the CHKPT macro instruction. This operand applies only when specifying a 2311 or 2314 disk as the checkpoint file.

When a checkpoint is taken, the completed checkpoint is noted on SYSLOG. Restarting can be done from any checkpoint record, not just the last. The job name specified in the JOB statement must be identical to the job name used when the checkpoint was taken. The proper I/O device assignments must precede the RSTRT control statement.

Assignment of input/output devices to symbolic unit names may vary from the initial assignment. Assignments are made for restarting jobs in the same manner as assignments are made for normal jobs.

IBM 3211 Printer Support

The addition of tapeless forms control and improvements in the use of the loadable print character buffer required special programming support for the IBM 3211 Printer. SYSBUFLD is the service program that loads the Universal Character Set Buffer (UCSB) and the Forms Control Buffer (FCB) with buffer load programs for the 3211 printer. SYSBUFLD is self-relocating, requires 2K of main storage, and is executed as a job step under BJF or SPI. It is initiated by the command:

```
// EXEC SYSBUFLD
```

$$BUFLDR is another 3211 program, called by IPL to load the UCSB and FCB buffer loads from the core image library.

System Considerations

When a 3211 PUB is encountered, the IPL program calls the buffer load transient, $$BUFLDR. $$BUFLDR in turn calls $$BUCB to load the UCSB, and $$BFCB to load the FCB. As supplied by IBM, these two phases contain:

$$BFCB	the configuration for a 66-line page (at 6 lines per inch), with 56 lines available for printing, a channel 1 for line 1, and a channel 12 for line 56.
$$BUCB	the character configuration for the All train.

Also, $$BUFLDR sets the UCSB for folding and suppressing data checks.

SYSBUFLD can be used to change the UCSB or FCB configurations, or to reload these buffers if a hardware failure occurs.

Only one FCB load is supplied in the DOS system. Additional FCB loads can be created using the procedure in the DOS System Control and Service listed in the Preface. FCB loads can be either cataloged in the core image library or can be card images. Any of your FCB loads can be cataloged in the core image library as $$BFCB to enable the IPL program to load it. If a phase name is not specified during an FCB load, SYSBUFLD loads the buffer from SYSIPT.

All UCSB programs must be cataloged in the core image library. They are loaded by specifying them in the SYSBUFLD control card.

The four remaining standard train configurations (G11, H11, P11, or T11) are in the relocatable library under the name IJBTRx11, where x is G, H, P, or T. These can be cataloged in the CIL to be loaded according to the train configuration.

Non-standard UCSB loads can be created following the procedure in the DOS System Control and Service listed in the Preface. Any of these standard or nonstandard loads can be cataloged as $$BUCB to enable the IPL program to load that particular train configuration.

Error Recovery Techniques

The simplest error recovery technique for the 3211 printer is specifying ERROPT=RETRY in the DTFPR, which sets CCB byte 2, bit 5 (PRINTOV=YES also sets this bit). This causes one automatic retry of the equipment-check/command-retry error.

A more comprehensive technique is specifying ERROPT=YES in the PRMOD and ERROPT=name in the DTFPR, which sets CCB byte 2, bits 5 and 6. These bits indicate linkage to your error recovery routine named in the DTFPR, and provide automatic retry of the equipment-check/command-retry error.

Return from your error recovery routine is by register 14. Both registers 14 and 15 must be saved if LIOCS is used during error recovery.

PIOCS users can provide linkage to error recovery by testing the applicable bits in the CCB. See Figure 3.10 for the error indicators.

If PIOCS or LIOCS is used, the sense information is not available to the user. The sense command, issued by the DOS error recovery routines, clears the 3211 sense information.

CCB Byte	Bit	Error
2	1	UCSB Parity Check - Line Complete: There has been a parity error in at least one position of the UCSB. All characters in the line have been printed and line spacing has taken place. The position in error has been cleared, so errors will not be encountered during subsequent scans of the UCSB. Printer speed is degraded as a result of this error. Full printing speed cannot be regained until the UCSB is reloaded. You need a routine to reload the UCSB during the job step, or use SYSBUFLD to reload the UCSB before the next job step.
3	0	Equipment Check/Print Check: This is a hardware error that has resulted in an incomplete line, but line spacing has taken place. Check for possible line position or print quality errors. The line in error must be either accepted or the page reprinted (a user-written routine is needed).
3	1	Equipment Check/Print Quality: This indicates a hardware error has occurred that caused light or blurred printing. Line spacing has taken place. Check for possible print check or line position errors. The line in error must be accepted or the page reprinted (a user-written routine is needed).
3	2	Line Position Error: This can be a hardware error, a parity error in the FCB, or the result of a skip to a channel code not in the FCB. Check for possible print check or print quality errors. There is no way to tell where the carriage is positioned relative to the FBC; physical repositioning, as well as reloading of the FCB, may be necessary.
3	3	Data Check/Print Check: An unprintable character has been sent to the printer. The applicable position(s) in the print line are blank and the paper has been spaced. The line in error must be either accepted or the page reprinted (a user-written routine is needed). Note: Check or reload the UCSB to ensure that the correct load has been used.
3	4	USCB Parity Check/Command Retry: There has been a parity error in at least one position of the UCSB. The applicable position(s) in the print line is blank, but the paper has not been spaced. Subsequent attempts to print the applicable character will result in a data check/print check. The UCSB can be reloaded (a user-written routine is needed), and the applicable command(s) reissued.

Figure 3.10. 3211 Error Status Indicator Bits in the CCB

Macro Writing

The macro-definition language discussed here provides a systematic means by which the DOS/360 assembler language programmer can develop macro instructions, thereby expanding the set of machine-oriented instructions that serve as the basis of the assembler language. This enables the programmer to reduce programming effort and shorten the assembler language source programs. With the aid of the macro language, any sequence of statements can be summarized into a single macro definition. Once written, this definition can be stored and referred to at any time, thus supplying the programmer with precoded routines. The programmer only writes a single statement, a macro instruction, to access the macro definition and retain access to all machine facilities.

Systematic use of macro instructions simplifies the coding of programs, reduces the frequency of programming errors, and encourages the use of carefully standardized sequences of assembler language statements for routine functions.

There are two classes of macros in the Disk Operating System: system macros, which are IBM-written macros supplied with the system, and user macros, which are defined by the user. The user macros may be included in the source program and/or may be entered into the source statement library.

The source statement library contains both user and system macro definitions. This library, which can be a part of System Residence (SYSRES) or a private library, eliminates the need for including definitions in the source module.

MACRO INSTRUCTION

The macro instruction statement is in assembler statement format. Symbols are used as a shorthand method of representing rules, definitions, etc. These macro instructions result in a one-for-one assembler statement.

The name field of the macro instruction may contain a symbol that is not defined unless a symbolic parameter appears in the name field of the prototype and the same parameter appears in the name field of the generated model statement (see _The Macro Definition_).

The operation field contains the mnemonic operation code of the macro instruction and has to be the same as the mnemonic operation code in the source program or in the source statement library.

The placement and order of the operands in the macro instruction statement is determined by the placement and order of the symbolic parameters in the operand field of the prototype statement. The operand field contains from 0-200 entries, separated by commas (entries are commonly referred to as parameters). Any combination of up to 255 characters may be used as a macro instruction operand if the rules concerning apostrophes, parentheses, equal signs, ampersands, commas and blanks are observed. These are described in the _Tape Operating Systems Assembler Language_ publication listed in the front of this manual.

The operand may be written in a format different than that used for assembler language statements. The alternate format described here allows the programmer to write an operand on each line and allows the interspersing of operands and comments in the statement. Figure 3.11 illustrates the operand formats.

Name	Operation	Operand Comments	Col. 72
NAME1	OP1	OPERAND1,OPERAND2,	x
		OPERAND3 THIS IS THE NORMAL FORMAT	
NAME2	OP2	OPERAND1, THIS IS THE	x
		OPERAND2,OPERAND3 ALTERNATE FORMAT	
NAME3	OP3	OPERAND1, THIS IS A COMBINATION	x
		OPERAND2,OPERAND3,OPERAND4,	x
		OPERAND5 OF BOTH FORMATS	

Figure 3.11. Operand Field Formats

When a program is written in the Disk Operating System macro language (an extension of DOS assembler language), one of three macro instruction formats can be used: keyword, positional, or mixed.

Figure 3.12 shows the typical form of a keyword macro instruction to be used with a keyword macro definition.

Name	Operation	Operand
A symbol, sequence symbol, or not used	mnemonic operation code	Zero to 100 or 200 operands, separated by commas.

Figure 3.12. Keyword Macro Instruction

Each operand consists of a keyword immediately followed by an optional value. Nested keywords are not permitted. A keyword consists of one to seven letters and digits, the first of which must be a letter. The operands of a keyword macro instruction may be written in any order. If an operand is omitted, the comma that would have separated it from the next operand need not be written.

The following are valid keyword macro instruction operands:

A4=F'6041'
DUPE4=MEMBER
SO=

The following are invalid keyword macro instruction operands:

&X4.P3=0(1,4)	Keyword does not begin with a letter.
CARDAREA=B+1	Keyword is more than seven characters.
=(TO(8),(AFTER))	No keyword.

The typical macro instruction is positional unless otherwise indicated. The positional macro instruction operands are written in a fixed order. Figure 3.13 illustrates the positional macro instruction.

Name	Operation	Operand
A symbol, sequence symbol, or not used	mnemonic operation code	Zero to 100 or 200 operands, separated by commas.

Figure 3.13. Positional Macro Instruction

The positional operands, if omitted from the macro instruction but appear in the prototype, are replaced by the comma that would have separated them from the next operand. If the last operand is omitted from a macro instruction, then the comma(s) separating the last operand from the previous operand may be omitted.

Figure 3.14 shows a macro instruction preceded by its corresponding prototype statement. The third and sixth operands of the macro instruction corresponding to the third and sixth operands of the prototype statement are omitted in this example.

Name	Operation	Operand
	EXAMPLE	&A,&B,&C,&D,&E,&F
	EXAMPLE	17,*+4,,AREA,FIELD(6)

Figure 3.14. Macro Instruction with Prototype

Mixed-mode macro instruction operands are a combination of both positional and keyword operands. Certain operand entries (positional) must be written in a fixed order; other operand entries (keyword) can be specified in any order. Figure 3.15 illustrates the mixed-mode macro instruction.

Name	Operation	Operand
A symbol, sequence symbol or not used	mnemonic operation code	Zero to 100 or 200 operands, separated by commas.

Figure 3.15. Mixed Macro Instruction

The operand consists of two parts. The first part corresponds to the positional operands and is written in the same way that the operand entry of a positional macro instruction is written. The second part of the operand corresponds to the keyword operands. This part is written in the same way that the operand entry of a keyword macro instruction is written. Figure 3.16 illustrates these facilities.

Name	Operation	Operand
	MACRO	
&N	MOVE	&TY,&P,&R,&TO=,&F=
&N	ST&TY	&R,SAVE
	L&TY	&R,&P,&F
	ST&TY	&R,&P,&TO
	L&TY	&R,SAVE

Figure 3.16. Mixed-Mode Definition

MACRO DEFINITION

A macro instruction cannot be assembled unless a macro definition is made available to the assembler. A macro definition is a set of statements that provide the assembler with:

1. The name entry, the mnemonic operation code, and the form of the macro instruction operand, and

2. The sequence of statements that the assembler uses when the macro instruction appears in the source program.

Elements of the Macro Definition

Every macro definition is made up of four elements: header statement, prototype statement, model statement, and a macro-definition trailer statement.

The macro instruction header and trailer statements denote the beginning and end of a macro definition respectively. The header statement's name field contains blanks, the operation field contains the word MACRO, and the operand field contains blanks. The header statement is the first statement of the macro definition. The trailer statement follows the same general outline as the header statement; only the operation field, that contains the word MEND, is different. The trailer statement must be present to denote the end of the macro definition.

A prototype is defined as an original model on which something is patterned. The prototype statement of a macro definition specifies in the operation column, the name of the macro and the format for the operand of all macro instructions that make use of this definition. This prototype statement must be the second statement in the macro definition.

Like the operand of the macro instruction, the prototype statement may be written in a form different from that used for machine or assembler instructions. The alternate form is described under The Macro Language.

ATTRIBUTES

The assembler assigns attributes to macro instruction operands and to symbols in the program. These attributes may be referred to in conditional assembly instructions under Conditional Assembly Statements.

There are six kinds of attributes:

1. Type (T'): The type attribute of a macro instruction is a letter. The type attribute may be referred to in the operand of a SETC instruction, or in character relations in the operands of SETB or AIF instruction. The letters used with the type attribute and their meanings can be found in the Assembler Language publication listed in the front of this manual.

2. Length (L'): The length attribute of a symbol (or of a macro instruction operand that is a symbol) is the length of the specified operand. Reference to the length attribute of a variable symbol is illegal except for symbolic parameters in SETA, SETB, and AIF statements. Reference must not be made to the length attributes of symbols whose type attributes are the letters M, N, O, T, or U.

3 and 4. Scaling (S') and Integer (I'): Scaling and integer attributes are provided for symbols that name fixed point, floating point, and decimal DC or DS statement. The programmer may refer to the length, scaling, and integer attributes in the operand field of a SETA instruction, or in arithmetic relations in the operand fields of SETB or AIF instructions.

5. Count Attribute (K'): The programmer may refer to the count attribute of macro instruction operands only. The count attribute is a value equal to the number of characters in the macro instruction operand after substituting for variable symbols excluding commas. If the operand is a sublist, the count attribute includes the beginning and ending parentheses and the commas within the sublist. The count attribute of an omitted operand is zero. The count attribute is one of the following:

   a. the operand field of a SETA instruction, or

   b. in arithmetic relations, the operand field of SETB or AIF instructions which are part of a macro definition can be referenced by the user.

6. Number Attribute (N'): The number attribute of macro instruction operands, only, is referenced. The number attribute is a value equal to the number of operands in an operand sublist. The number of operands in an operand sublist is equal to one, plus the number of commas that indicate the end of an operand sublist. If the macro instruction operand is not a sublist, the number attribute is one. If the macro instruction operand is omitted, the number attribute is zero. Reference may be made to the number attribute in the operand field of a SETA instruction, or in arithmetic relations in the operand fields of

SETB and AIF instructions that are part of a macro definition.

SUBLIST NOTATION

A sublist is one or more operands, separated by commas and enclosed in paired parentheses. An operand of a macro instruction may be a sublist. The entire sublist, including all operands, commas and parentheses is considered one macro instruction operand. Each operand entry within the parentheses is called a sublist member.

Sublists provide the user with a convenient way to refer to a collection of macro instruction operands as a single operand, or a single operand in a collection of operands. For the accessing of individual members, the left parenthesis of the sublist notation must immediately follow the last character of the symbolic parameter. A period should not be placed between the left parenthesis and the last character of the symbolic parameter (Figure 3.17).

	NAME	OPERATION	OPERAND
HEADER		MACRO	
Prototype		ADDNUM	&NUM,&REG,&AREA
Model		L	&REG,&NUM(1)
Model		A	&REG,&NUM(2)
Model		A	&REG,&NUM(3)
Model		ST	&REG,&AREA
Trailer		MEND	
MACRO		ADDNUM	(A,B,C),6,SUM
Generated		L	6,A
Generated		A	6,B
Generated		A	6,C
Generated		ST	6,SUM

Figure 3.17. Sublist Illustration

The operand of the macro instruction that corresponds to symbolic parameter &NUM is a sublist. One of the operands in the sublist is referred to in the operand entry of three of the model statements.

Model statements are the macro definition statements from which the desired sequences of machine instructions and certain assembler instructions are generated. Zero or more model statements may follow the prototype statement. A model statement consists of one to four entries:

1. Name field entry which can contain blanks, a symbol, a symbolic parameter or a sequence symbol.

2. The operation entry may contain any machine instruction, conditional instruction, assembler instruction, or symbolic parameter except COPY, END, ICTL, ISEQ and PRINT; or it may contain a variable symbol, depending on the statement.

3. The operand entry may contain ordinary symbols or variable symbols. After substitution, the operand must not be greater than 127 characters. Model statements must follow the rules for paired apostrophes, ampersands, and blanks, as macro instruction operands.

4. The comments field entry which can contain descriptive items of information about the program is inserted after the operand. All 256 valid characters, including blanks, may be used in writing a comment. The entry cannot extend beyond the end column (normally column 71) and a blank must separate it from the operand.

VARIABLE SYMBOLS

The three types of variable symbols are symbolic parameters, SET symbols and system variables.

Symbolic Parameter

The symbolic parameter consists of an ampersand (first character) followed by one to seven letters and/or numbers, the first of which must be a letter. Symbolic parameters appear in prototype and model statements. They are assigned values by the programmer when he writes a macro instruction. The programmer should not use &SYS as the first four characters of a symbolic parameter.

Example:

Valid	Invalid
&LOOP2	&2BAC (first character after & not letter)
&READER	&AREA2456 (too long)

SET Symbols

SET symbols follow the same rules for structure as symbolic parameters. SET symbols differ from symbolic parameters in three ways:

1. their position in an assembler language source program,
2. how they are assigned values, and
3. how their assigned values can be changed.

Defining SET Symbols

A SET symbol must be defined by the programmer before it can be used. It is defined by appearing as an operand of a global or local instruction (GBLA, GBLB, GBLC, LCLA, LCLB, LCLC). Figure 3.18 shows the typical format of global and local instructions.

Name	Operation	Operand
Not used; must not be present	GBLA, GBLB, GBLC, or LCLA, LCLB, LCLC	One or more variable symbols used as SET symbols and separated by commas. (If more than one macro defines one global SET symbol and these macros are assembled together, the global SET symbol value is set by the first macro, but is not altered by subsequent macros.)

Figure 3.18. Format of Globals and Locals

A global instruction (GBLA, GBLB, GBLC) defines one or more operands as names associated with arithmetic, binary, or character data. These operands are called global SETA, SETB, or SETC symbols. If a global instruction is part of a macro definition, it must immediately follow the prototype statement or another GBLA, GBLB, or GBLC. The global definition of these operands indicates that each SET symbol is defined both inside and outside the defining macro and enables communication between discrete macros.

A local instruction (LCLA, LCLB, LCLC) defines one or more operands as names associated with arithmetic, binary, or character data. These operands are called local SETA, SETB, or SETC symbols. If a local instruction is part of a macro definition, it must immediately follow the prototype statement and any global instructions or another LCLA, LCLB, or LCLC. The local definition of these operands indicates that each SET symbol is defined only within the defining macro.

The GBLA or LCLA, GBLB or LCLB, GBLC or LCLC operands are assigned initial values of 0 (X'F0'), 0 (X'00'), and null character (no hexadecimal number), respectively.

The SETA, SETB, and SETC symbols are assigned the initial values of 0, 0, and null character value, respectively.

The SETA instruction in the operand entry is evaluated as a signed 32-bit arithmetic value that is assigned to the SETA symbol in the name entry. Figure 3.19 shows the format of the SETA instruction.

Name	Operation	Operand
A SETA symbol	SETA	One term, or an arithmetic expression, not less than -2^{31} nor greater than $+2^{31}-1$.

Figure 3.19. Format of SETA Instruction

The expression may consist of one term or an arithmetic combination of terms, the minimum and maximum values of which are -2^{31} and $+2^{31}-1$, respectively. The arithmetic value assigned to a SETA symbol is substituted for the SETA symbol when it is used in an arithmetic expression.

The SETB instruction may assign the binary value 0 or 1 to a SETB symbol. Figure 3.20 illustrates the format of this instruction.

Name	Operation	Operand
A SETB symbol	SETB	A 0 or a 1, (0) or (1), or a logical expression within parentheses

Figure 3.20. Format of SETB Instruction

The operand may contain a 0 or a 1 or a logical expression enclosed in parentheses. No explicit binary zeros or ones are allowed in parentheses other than in the form (0) or (1). A logical expression is evaluated to determine if it is true or false. The SETB symbol in the name entry is then assigned the binary value 1 or 0 corresponding to true or false, respectively.

The following are valid operand fields of SETB instructions:

(&AREA+2 GT 29)

(T'&T02 EQ 'C')

The following are invalid:

&B (not enclosed in parentheses)

(T'&P12 EQ 'F' &B) two terms in succession

The SETC instruction assigns a character value to a SETC symbol. Figure 3.21 shows the format of a SETC instruction.

Name	Operation	Operand
A SETC symbol	SETC	One operand of the type attribute, character expression or a substring notation. A SETA symbol may appear here.

Figure 3.21. Format of SETC Instruction

The character value assigned to a SETC symbol may be a type attribute. If the type attribute is used, it must appear alone in the operand field.

A character expression usually appears in the operand field. A character expression consists of any combination of characters enclosed in apostrophes. The maximum length of a character expression is 127 characters. The character value enclosed in apostrophes in the operand field is assigned to the SETC symbol in the name entry. The maximum length character value that can be assigned to a SETC symbol is eight characters. If a value greater than 8 is specified, the leftmost 8 characters are used.

System Variables

System variable symbols are assigned values automatically by the assembler. There are four system variable symbols: &SYSNDX, &SYSPARM, &SYSECT, and &SYSLIST. System variable symbols may be used in the name, operation, and operand entries of statements in macro definitions, but not in statements outside of macro definitions with the exception of &SYSPARM. They may not be defined as symbolic parameters or SET symbols, nor may they be assigned values by SETA, SETB, and SETC instructions.

The &SYSNDX symbol is assigned the four-digit number 0001 for the first macro instruction processed by the assembler, and it is incremented by one for each subsequent inner and outer macro instruction processed. &SYSNDX may be combined with other characters to create unique names for statements generated from the same model statement. The &SYSECT symbol carries a character value that is the name of the last START, CSECT, or DSECT statement encountered before the expansion of the USING macro.

The &SYSPARM is specified in the STDJC macro at system generation time. &SYSPARM allows the user to control conditional assembly flow and source code generated through the use of the parameter specified in the job control OPTION statement. &SYSPARM acts as a global SETC, except its value is set by the job control OPTION statement.

If no named CSECT, DSECT, or START statements occur before a macro instruction, &SYSECT is assigned a null character value for that macro instruction. The &SYSLIST symbol (not available in keyword macro definitions) is the symbol reference for the entire macro instruction operand field. This symbol refers to the nth macro instruction operand. If the nth operand is a sublist, then &SYSLIST(n,m) may refer to the mth operand in the sublist, where n and m may be any arithmetic expressions allowed in the operand field of a SETA statement.

CONCATENATION

Concatenation is defined as a linking together in a series or chain; a process of linking or joining together in a sequence, with a specified order.

If a symbolic parameter in a model statement is immediately preceded or followed by other characters or another symbolic parameter, the characters that correspond to the symbolic parameter are combined, in the order given in the generated statement, with the other characters or the characters that correspond to the other symbolic parameter. This process is called concatenation. When a symbolic parameter is concatenated with any following character value, the extent of the symbol must be defined (delimited). If the first character of the following character value is not a recognized delimiter, a special delimiter character (a period '.'), must be used when the first character is a letter, digit, left parenthesis or a period. A period is optional when the first character is an ampersand (&). See Figure 3.22.

	Name	Operation	Operand
Header		MACRO	
Prototype	&NAME	MOVE	&P,&S,&R1,&R2
Model	&NAME	ST	&R1,&S.(&R2)
Model		L	&R1,&P.B
Model		ST	&R1,&P.A
Trailer		L	&R1,&S.(&R2)
		MEND	
Macro	HERE	MOVE	FIELD,SAVE,2,4
Generated	HERE	ST	2,SAVE(4)
Generated		L	2,FIELDB
Generated		ST	2,FIELDA
Generated		L	2,SAVE(4)

Figure 3.22. Concatenation and Generated Coding

SEQUENCE SYMBOLS

The name entry of a statement may contain a sequence symbol that provides the programmer with the ability to vary the sequence in which the assembler processes statements. These symbols are never variables. They name a branch point in the definition and consist of a period followed by a letter and seven letters and/or digits.

A sequence symbol in the operand entry of an AIF or AGO statement (see Conditional Assembly Statements) references the statement named by the sequence symbol. It can be used in the name entry of any statement that does not contain a symbol or SET symbol, except a prototype statement, or a MACRO, LCLA, LCLB, LCLC, GBLA, GBLB, GBLC, ACTR, ICTL, ISEQ, or COPY instruction. For example, .READER, .LOOP2, .A23456 and .X4F2 are valid sequence symbols.

CONDITIONAL ASSEMBLY INSTRUCTIONS

The conditional assembly instructions allow the programmer to:

1. Define and assign values to SET symbols that can be used to vary parts of generated statements and
2. Vary the number and sequence of generated statements.

These conditional assembly instructions give true flexibility to the macro definition language.

There are 13 conditional assembly instructions: LCLA, LCLB, LCLC, GBLA, GBLB, GBLC, SETA, SETB, and SETC, that are discussed under SET Symbols, and AIF, AGO, ACTR, and ANOP that are discussed in this section.

AIF--Conditional Branch

The AIF instruction alters conditionally the sequence in which source program statements are processed by the assembler. The conditional branch is located within or outside of the macro definition. If the logical expression in the operand field is true, the macro generator branches to the sequence symbol following the logical expression. Figure 3.23 illustrates the typical form of this instruction.

Name	Operation	Operand
A sequence symbol or not used	AIF	A logical expression enclosed in parentheses, immediately followed by a sequence symbol.

Figure 3.23. Conditional Branch Instruction

The following are valid operands of AIF instructions:

(&AREA+X'2D' GT 29).READER

(&NAME+FIVE GT 2).POSSIBLE

The following are invalid operands of AIF instructions:

(T'&ABC NE T'&XYZ)	No sequence symbol
(T'&ABC NE T'&XYZ).X4F2	Blank between logical expression and sequence symbol
.X4F2	No logical expression

AGO--Unconditional Branch

The AGO instruction causes an unconditional branch to the sequence symbol in the operand. Figure 3.24 illustrates the typical form of this instruction.

Name	Operation	Operand
A sequence symbol or not used	AGO	A sequence symbol

Figure 3.24. Unconditional Branch Instruction

The statement named by the sequence symbol in the operand is the next statement processed by the assembler. The statement named by the sequence symbol may precede or follow the AGO instruction.

ACTR--Conditional Assembly Loop Counter

The ACTR limits the number of AGO and AIF branches executed within a macro definition. When used, the ACTR must appear after the globals and locals symbol definition statements and before any other type of model statement. The ACTR instruction assigns a maximum count to the number of AGO and AIF branches executed within the macro definition. When the count reaches zero, an END card is generated. If the count is zero before decrementing, the assembler takes one of two actions:

1. If a macro definition is being processed, the processing of it and any nested macros above it is terminated, and the next statement in the main portion of the program is processed.

2. If the main portion of the program is being processed, conditional assembly is terminated, and the portion of the program generated so far is assembled. If an ACTR statement is not given, the assumed value of the counter is 150.

ANOP--Assembly No Operation

The ANOP instruction facilitates branching to a statement that has a symbol or variable symbol in the name field. The ANOP instruction causes no operation and is inserted immediately before the statement to be branched to. Figure 3.25 illustrates the typical form of this instruction.

Name	Operation	Operand
A sequence symbol	ANOP	Not used, must not be present.

Figure 3.25. Assembly No Operation Instruction

If the programmer wants to use an AIF or AGO instruction and has already entered a symbol or variable symbol in the name entry of the statement to which he wishes to branch, he cannot place a sequence symbol in the name entry. An ANOP instruction can be placed before that instruction, and then branched to. This has the same effect as branching to the statement immediately after the ANOP statement.

EXTENDED CAPABILITIES

The macro language provides additional features that allows the system to:

1. Terminate processing of macro definition

2. Generate error messages

3. Define global SET symbols (discussed in the section SET Symbols)

4. Use system variable symbols (discussed under System Variable Symbols)

5. Prepare keyword and mixed-mode macro definitions and write keyword and mixed-mode macro instructions (discussed under Macro Instruction Formats)

MEXIT -- Macro Definition Exit

The MEXIT instruction allows exit from the macro definition at various points in the definition. However, when this instruction terminates the macro definition, it does not signify the physical end of the definition. Figure 3.26 illustrates the typical form of the instruction.

Name	Operation	Operand
A sequence symbol or not used	MEXIT	Not used, must not be present.

Figure 3.26. Macro Definition Exit Instruction

MEXIT should not be confused with MEND. MEND indicates the physical end of the macro definition. MEND must be the last statement of every macro definition, including those that contain one or more MEXIT instructions.

MNOTE Statement

The MNOTE instruction may generate a message and indicate the level of severity of the error. The severity code is for the programmer's information only and is not used by the DOS assembler or control program. This instruction requests a message to be printed on the output listing. Figure 3.27 illustrates the typical form of this instruction.

Name	Operation	Operand
A sequence symbol or not used	MNOTE	See examples in text

Figure 3.27. MNOTE Instruction

The operand entry of the MNOTE instruction may be written in one of the following forms:

1. severity code,'message'
2. ,'message'
3. 'message'

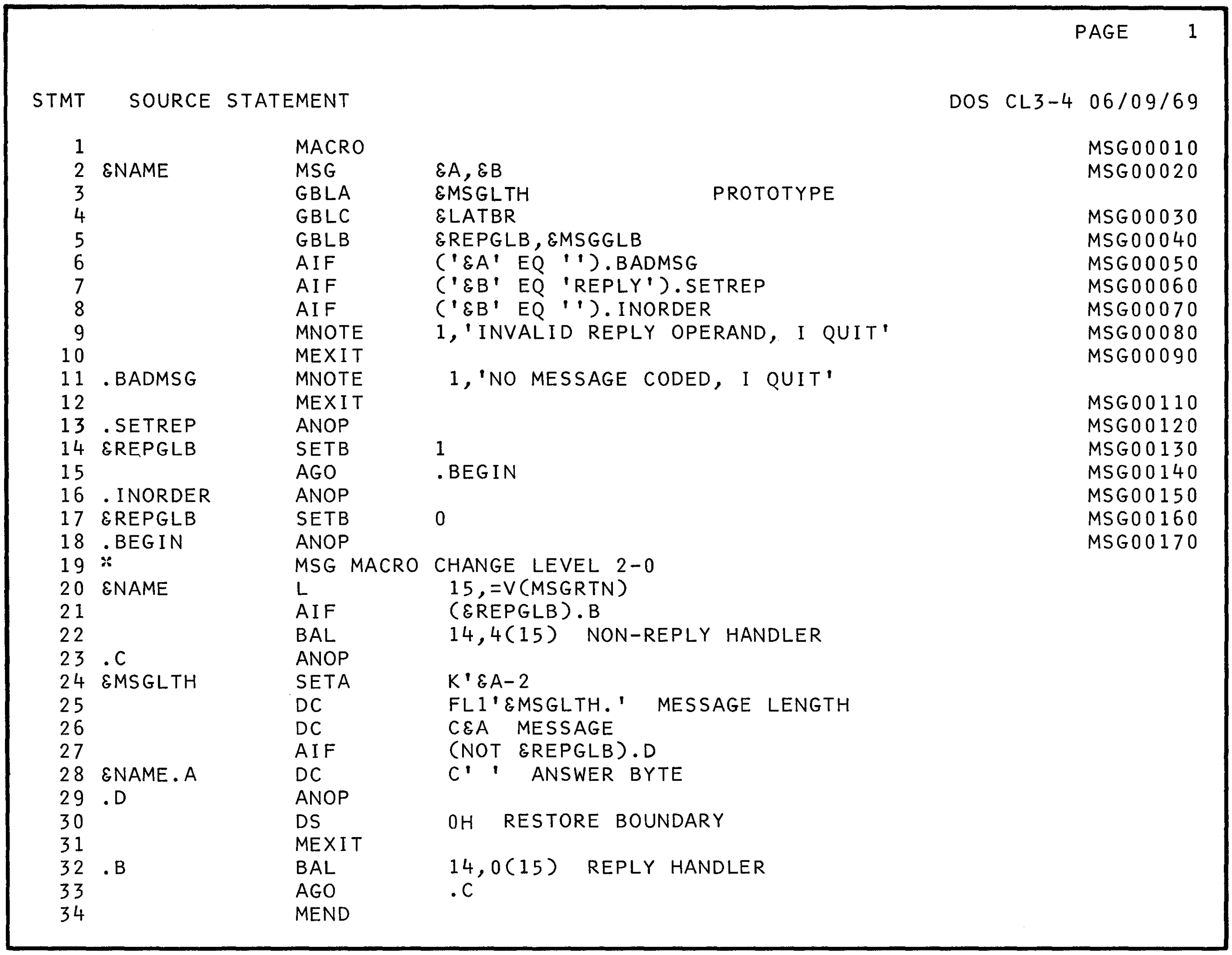
For two and three, the severity code is assumed as one.

The MNOTE statement appears in the listing with a statement number at the point where it was generated. Because the message portion of the MNOTE operand is enclosed in apostrophes, two apostrophes must be used to represent a single apostrophe. Two ampersands must be used to represent a single ampersand that is not part of a variable symbol (see Figure 3.28).

Name	Operation	Operand
	MNOTE	1,'ERROR**--NOT RECOVERABLE'

Figure 3.28. Sample MNOTE

Figures 3.29 and 3.30 are examples of macro writing and usage. Figure 3.29 defines the MSG macro. There are no locally defined symbols. All are globals. Figure 3.30 is an example using the MSG macro to write an appropriate message pertaining to a particular step of the routine. The generated coding of the MSG macro is indicated by a '+' after the statement number.



```
                                                                          PAGE    1

STMT   SOURCE STATEMENT                                          DOS CL3-4 06/09/69

   1               MACRO                                                   MSG00010
   2 &NAME         MSG       &A,&B                                         MSG00020
   3               GBLA      &MSGLTH             PROTOTYPE
   4               GBLC      &LATBR                                        MSG00030
   5               GBLB      &REPGLB,&MSGGLB                               MSG00040
   6               AIF       ('&A' EQ '').BADMSG                           MSG00050
   7               AIF       ('&B' EQ 'REPLY').SETREP                      MSG00060
   8               AIF       ('&B' EQ '').INORDER                          MSG00070
   9               MNOTE     1,'INVALID REPLY OPERAND, I QUIT'             MSG00080
  10               MEXIT                                                   MSG00090
  11 .BADMSG       MNOTE      1,'NO MESSAGE CODED, I QUIT'
  12               MEXIT                                                   MSG00110
  13 .SETREP       ANOP                                                    MSG00120
  14 &REPGLB       SETB      1                                             MSG00130
  15               AGO       .BEGIN                                        MSG00140
  16 .INORDER      ANOP                                                    MSG00150
  17 &REPGLB       SETB      0                                             MSG00160
  18 .BEGIN        ANOP                                                    MSG00170
  19 *             MSG MACRO CHANGE LEVEL 2-0
  20 &NAME         L          15,=V(MSGRTN)
  21               AIF        (&REPGLB).B
  22               BAL        14,4(15)  NON-REPLY HANDLER
  23 .C            ANOP
  24 &MSGLTH       SETA       K'&A-2
  25               DC         FL1'&MSGLTH.'  MESSAGE LENGTH
  26               DC         C&A  MESSAGE
  27               AIF        (NOT &REPGLB).D
  28 &NAME.A       DC         C' '  ANSWER BYTE
  29 .D            ANOP
  30               DS         0H  RESTORE BOUNDARY
  31               MEXIT
  32 .B            BAL        14,0(15)  REPLY HANDLER
  33               AGO        .C
  34               MEND
```

Figure 3.29. Sample MSG Macro

```
STMT   SOURCE STATEMENT                                               DOS CL3-4

1176 *                                                                         *
1177 ***********************************************************************
1178 *                                                                         *
1179 *         THE FOLLOWING ROUTINE CHECKS THE RETURN CODE AFTER A            *
1180 *         READ,WRITE,CONTROL COMMAND IS GIVEN                             *
1181 *         REGISTER EIGHT CONTAINS THE ADDRESS OF ROUTINE WHICH
1182 *         JUST GAVE THE READ OR WRITE COMMAND. IF THE COMPLETION
1183 *         IS BAD REG 8 IS USED TO TURN THIS DEVICE OFF (NO MORE
1184 *         OPERATIONS WILL OCCUR ON THIS DEVICE),THE OPERATOR MUST
1185 *         RELOAD THE PROGRAM INORDER TO RESTART THIS DEVICE
1186 *
1187 ***********************************************************************
1188 *



1190 CKCONDC  LTR   RF,RF                   CK CONDITION CODE
1191          BNZ   CONTCK
1192          BR    R2                      EXIT BACK CK GOOD
1193 CONTCK   MSG   'ERROR OCCURRED ON CKING RETURN CODE'
1194+*           MSG MACRO CHANGE LEVEL 2-0
1195+CONTCK   L     15,=V(MSGRTN)
1196+         BAL   14,4(15) NON-REPLY HANDLER
1197+         DC    FL1'35' MESSAGE LENGTH
1198+         DC    C'ERROR OCCURRED ON CKING RETURN CODE' MESSAGE
1199+         DC    0H RESTORE BOUNDARY
1200          LR    R3,RF                   SWITCH REGS
1201          BAL   R2,FORMAT               GO FORMAT CONDITION CODE
1202          STC   R3,ERRMSG+28            INSERT CHAR
1203          LR    R3,RF
1204          BAL   R2,FORMAT1              GO FORMAT SECOND CHAR
1205          STC   R3,ERRMSG+27            INSERT SECOND CHAR
1206 ERRMSG   MSG   'CONDITION CODE =      '
1207+*           MSG MACRO CHANGE LEVEL 2-0
1208+ERRMSG   L     15,=V(MSGRTN)
1209+         BAL   14,4(15) NON-REPLY HANDLER
1210+         DC    FL1'21' MESSAGE LENGTH
1211+         DC    C'CONDITION CODE =      ' MESSAGE
1212+         DS    0H RESTORE BOUNDARY
1213 X        MVC   MSG2+27(7),OPERATIN
1214 MSG2     MSG   'LAST OPERATION -
1215+*           MSG MACRO CHANGE LEVEL 2-0
1216+MSG2     L     15,=V(MSGRTN)
1217+         BAL   14,4(15) NON-REPLY HANDLER
1218+         DC    FL1'36' MESSAGE LENGTH
1219+         DC    C'LAST OPERATION -                     ' MESSAGE
1220+         DS    0H RESTORE BOUNDARY
1221 PDUMP    PDUMP START,END
1222+* 360N-CL-453 PDUMP    CHANGE LEVEL 3-0
```

Figure 3.30. Sample MSG Coding

Section 4: Debugging Aids

Section Outline

Section Figures

This section presents debugging aids considered helpful to both the application and system programmer. Information contained in this section includes:

- System action on all cancel conditions
- Register conventions for following program flow
- When a storage print is useful
- The types of documentation used in locating program problems
- The action taken when a hard wait or unending loop is encountered
- The importance of low-core messages and system error messages as an aid in determining a starting point for approaching a programming problem.

Sample programs in COBOL, PL/I, FORTRAN, and RPG, together with their respective linkage editor maps to show how to locate programs and partition save areas in main storage are included.

Gathering Documentation

This is an explanation of the types of documentation useful in debugging problem programs. All SYSLOG and SYSLST error messages, program listings, the supervisor listing, linkage editor map, and a core dump should be gathered.

A system dump of main storage should be available, but if the system is in a hard wait or an unending loop, a stand-alone (self-loading) dump will have to be taken.

The DOS stand-alone dump generator, DUMPGEN, produces a stand-alone dump program tailored to system requirements. The dump can either be a conventional dump program or a formatting dump program. A DSERV of the core image library directory may be helpful when program checks occur in the logical transient area. The directory can be displayed by an alphamerically sorted listing of the directory entries, or a listing of the entries in the order they appear in the directory.

The label cylinder display program (LSERV) can also be used for error analysis. LSERV displays the TLBL and the DLBL and EXTENT information contained on the SYSRES label cylinder. Information about secured data files is not displayed.

DOS I/O error logging, MCRR (Machine Check Recording and Recovery), RMS (Recovery Management Support), and the DUMP option of job control are additional facilities for error analysis. The RMS consists of two functions: MCAR (Machine Check Analysis and Recording), and CCH (Channel Check Handler).

Other facilities for error analysis are EREP (Environmental Recording, Editing, and Printing Program), ESTVUT and ESTVFMT (Error Statistics by Tape Volume Utility programs). EREP edits and prints data that has been stored in the recorder file (SYSREC) by the I/O error logging and/or MCRR and/or MCAR/CCH functions. For the IBM System/370, EREP creates and maintains a history tape, and, if specified at system generation time, RDE (Reliability Data Extractor) of OBR/MCAR/CCH and, if specified by the ROD command, IPL/EOD (End of Day) data. Figures 4.1, 4.2, and 4.3 show the SDR communications region, the MCRR linkage table, and the RMS linkage area.

0		SDR Flags (SDR-TABLE)
1		Partition ID
2	3	Number of SDR Records
4	10	First SDR ID BBCCHHR
11	17	First OBR ID BBCCHHR
18	24	Current OBR ID BBCCHHR
25	31	Last OBR ID BBCCHHR
32	35	Address of SDR Accumulator
36	39	Address of SDR Unit Switches
40	43	Reserved
44	71	List Save Area
72	75	Mask Bytes
76	95	SDR1 Work Area
96	103	Test Under Mask Table
104	107	Temporary Work Area
108	111	Test Under Mask Instruction
112	115	F'65536'
116	117	SDR Queue Save Area
118	135	SDR2 Work Area
136	155	Area Modified by A-Transients
156	159	SDR Error Message Save Area
160	163	Branch Instruction
164	167	OBR/SDR Flag Byte Address
168	250	Data Area for OBR/SDR Records

Key to SDR Communications Region Displacements:

0 SDR Flags:

Bit 0: Key of OBR
1: RDE option
2: Initial IPL time
3: RF option = NO, recording is suppressed
Bit 4: RF option = CREATE
5: RF option = YES
6: Error while recording
7: Recorder file ready

Set and tested by Job Control.

1 Set by EREP transient $$BSDRUP to identify the partition making the call for EREP recording.

Settings: X'10' if EREP is running in BG.
X'20' if EREP is running in F2.
X'30' if EREP is running in F1.
X'01' with one of the above if recorder file is ready.
X'00' with one of the above if recorder file is not ready.

2 Initial number of SDR records specified. If SDR record count is not specified, the file is formatted for OBR records only ($JOBCTLM, see IPL and Job Control PLM, GY24-5086).

4 Disk address of first SDR record.

11 Disk address of first OBR record.

18 Disk address of current OBR record.

25 Disk address of last OBR record.

Figure 4.1. SDR Communications Region (Part 1 of 2)

Key to SDR Communications Region Displacements:

32 Address of SDR accumulator area which contains half-byte counters and accumulated error conditions.

36 Address of SDR unit switches.

SDR switch byte (1 for each PUB):
X'80' - Update operations complete
X'40' - Counters on external file overflowed
X'20' - I/O error during write
X'08' - SDR update half-byte counters routine required
X'04' - Update SDR record routine required
Other - Reserved

When entry contains X'01000000', indicates MCRR, no SDR supported.

40 Reserved.

44 SDR1 register save area.

72 Mask formats for interpretive error accumulator, SDR1:

X'FF' - End of update
X'FE' - Bypass counter
X'FD' - Set up 'OR' condition to previous counter
X'FC' - Ignore list item
Other - Test bit in error queue

76 Used by the interpretive error accumulator routine to process list passed by OBR/SDR A-transient.

96 Used by the interpretive error accumulator routine.

104 Used by the interpretive error accumulator routine for address alignment.

108 Executed by the interpretive error accumulator routine.

112 Loop counter for the SDR counter update.

116 Save area for pointers to entries in the SDR error queue.

118 Work area where half byte error counters are unpacked and updated.

136 List of devices passed to the SDR processor from $$ANERAD.

156 Used by SDR/OBR recorder phases to pass error message displacements and disk error addresses in event of an error.

160 Entry point from OBR/SDR A-transients. Branches to label SDRMM.

164 Pointer into the OBR/SDR unit switches. Status posted by recorder phases. (See byte 36).

168 OBR and SDR records formatted by the recorder phases.

Figure 4.1. SDR Communications Region (Part 2 of 2)

MCRRPSW1 (See Note)

0 (Hexadecimal Displacement)	8	10	14
0 (Decimal Displacement)	8	16	20
MCRR PSW Reentrant Address of MCRR Routine	MCRR PSW Address of MCRR Routine	Address of Channel Failure Routine	Address of Machine Check Routine
XXXXXXXX	XXXXXXXX	XXXX	XXXX

Key to displacement:

0	Machine Check Recording and Recovery PSW. Loaded to enable machine check interrupts. Second word (displacement 4-7) contains reentrant address (MCRETURN) to MCRR routine.
8	Machine Check Recording and Recovery PSW. Loaded to enable machine check interrupts. Second word (displacement 12-15) contains initial address (MCRRRTN) of the MCRR routine.
16	Address of channel failure routine (MACHEK1).
20	Address of machine check routine (MACHEK).

Note: MCRRPSW1 is the label of the first byte of the MCRR Linkage Table.

Figure 4.2. Machine Check Recording and Recovery (MCRR) Linkage Table

RASLINK

0 (Decimal Displacement)	8	9	10	11	12	16
CPUID	RASDMC	RASFLAGS	MCFLAGS	RASMODEL	RASTABA	RASBASE
CPU ID field	Damaged Channel byte	RAS flag byte	Machine Check flags	CPU Model	RAS Table (RASTAB) address	Base address for RAS Monitor
XXXXXXXX	X	X	X	X	XXXX	XXXX

Key to RAS Linkage Area displacements:

0 CPU ID field.

8 Address of damaged channel, or X'FF' if no channel damaged.

9 RAS Flag byte:

bit	flag	description
0	X'80'	RAS active
1	X'40'	RAS SIO flag
2	X'20'	RTA in control
3	X'10'	RAS I/O delayed
4	X'08'	Channel check on error SIO
5	X'04'	Reserved
6	X'02'	Channel check on SIO
7	X'01'	I/O active for SIO

10 Machine Check Flags:

bit	flag	description
0-4	——	Reserved
5	X'04'	Hard machine check
6	X'02'	All machine records built
7	X'01'	All channel check records built

11 Largest CPU Model.

12 Address of RAS Table (RASTAB).

16 Address used for base register in RAS Monitor Program.

Figure 4.3. RMS Linkage Area (RASLINK)

Further documentation can be obtained by executing the PDAID program, which records (traces) certain events and either writes them on the I/O device specified or maintains them in the CE area (or alternate address area). The program can trace:

1. fetching or loading of programs or phases (Fetch/Load Trace).

2. input/output activity (I/O Trace).

3. supervisor calls (GSVC Trace).

4. QTAM input/output activity (QTAM Trace).

The On-Line Test Executive Program (OLTEP), together with the On-Line Tests (OLTS), make up the On-Line Test System, which tests I/O devices with minimum interference to other programs running on the system. RETAIN/370 is an OLTEP function that allows the OLTEP programs to be executed on the System/370 from a remote location. RETAIN/370 is yet another problem determination tool. See the DOS OLTEP listed in the Preface for a detailed description.

The facilities mentioned form the DOS problem determination aids. Problem determination is a process or a procedure for determining the cause of an error. The DOS Messages listed in the Preface recommends a specific procedure to follow when an error condition occurs; the DOS System Control and Service gives a detailed explanation of problem determination aids.

Error messages are very important and can supply useful information in determining where to start looking for the trouble (i.e., in what partition the failure took place). A message number or code is supplied to give further information about the error. The program check message gives the location of the failing operation code and the condition code from the program status word (PSW). This gives you a starting point for reconstructing the cause of the error.

The program listing is an extremely useful tool in determining if the error condition was caused by a logic error or a particular condition that you had not considered when the program was written.

The supervisor (SUPVR) listing allows you to check facts at the time of failure and to determine if the error indications were valid. The listing and the main storage dump allow you to locate the Program Interrupt Key (PIK) to determine the task in control of the system at the time of failure.

By locating the Program Information Block (PIB) table, you can easily locate programs in main storage. See Figures 4.4, 4.5, and 4.6 for a description of the PIB table. Using the information in the PIB table, you can check the cancel code and find the partition save area address. The partition save area supplies you with such useful information as the PSW and register values that you can use to locate the last instruction executed.

The system communications region (Figure 1.7) within the supervisor contains the address of the PIB table, and other useful information for determining the nature of the error.

PIB TABLE

Byte Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	= 16 Byte Length
All Bound PIB	Flag Byte See A *	Reserved	SP Prefix		Branch Instruction to the All Bound Routine				Reserved								
Problem Program PIB (Note 1)	Flag Byte See B *	Cancel Code	SYSLOG ID (BG, F2, or F1)		NOP Instruction (CR)	Address of the Partition Save Area			Number of Core Blocks (Note 2)	Address of the Origin of the Partition			PIB Assign Flag See D *	User LUB Index	Number of Program LUBs	Flag Byte See C *	
Attention PIB	Flag Byte See E *	Cancel Code	SYSLOG ID (AR)		Branch Code (BC)	Active=Address of Save Area Inactive = Remainder of BC Instruction			Switch Byte See F *	Logical Transient Bucket (contains save area address)			X'07' See D *	Reserved	Address of the Logical Transient		
Quiesce PIB	Flag Byte See A *	Cancel Code	C'/&'		Branch Instruction to Quiesce I/O Routine				Scratch Byte X'00'	X'00'	Channel PUB Table Index Values X'04'	X'08'	X'OC'	X'10'	X'14'	X'18'	
Supervisor PIB	Flag Byte See A *	Cancel Code	SP Prefix		Branch Instruction to General Exit Routine				Address of SYSRES PUB		Length of Error Queue Entry		Constants to Clear Bytes 2-5 of CCB X'1F'	X'05'	X'00'	X'00'	
Subtask PIB for AP (Note 3)	Flag Byte See B *	Cancel Code	SYSLOG ID (BG, F2, or F1)		NOP Instruction	Address of the Save Area			Number of Core Blocks (Note 2)	Address of the Origin of the Main Task			PIB Assign Flag See D *	User LUB Index	Number of LUBs	Flag Byte See C *	

Note 1: Three problem program PIBs are built in this sequence when the MPS or BJF feature is selected as a generation option:
- Background PIB
- Foreground 2 PIB
- Foreground 1 PIB

When a batch-only environment is established at generation time, the All Bound and Foreground PIBs are excluded from the table, and only one (BG) problem program PIB is built. However, the X'20' bytes that F2 and F1 PIBs normally occupy (between PIBBG and PIBAR) are filled with 32 bytes of DIBs data.

Note 2: Number is in multiples of 2K for F2 and F1. BG is always 10K (X' 0A').

Note 3: Total of nine subtask PIBs are generated, and only when AP is specified at generation time.

* See Figure 4.5 for flag byte expansions A, B, C, D, E and F.

Bytes 90 and 91 (X'5A' - '5B') of the communications region contain the address of the first part of the PIB Table. Label PIBTAB identifies the first byte of the table.

Figure 4.4. First Part of Program Information Block (PIB) Table

A Supervisor, Quiesce, and ALL Bound PIB Flags:

Bit 0: 1 = Always one
1-4 : 0 = Always zero
5 : 1 = Always one
6 : 1 = Active
0 = Inactive
7 : 1 = Active
0 = Inactive

Note: If PTO=YES is specified, Bit 6 is a one in the Quiesce I/O PIB when attached by the supervisor. Otherwise it is always zero.

B Problem Program PIB Flag (First Byte in PIB):

Bit 0: 1 = Registers stored
0 = Registers not stored
1-3 : 0 = Always zero
4 : 1 = QTAM Wait active
0 = QTAM Wait inactive
5 : 0 = Normal execution
1 = Program has seized the system
6 : 1 = Unbound
0 = SVC 2-bound (B-transient in progress)
7 : 1 = Unbound
0 = SVC 7-bound (waiting for an I/O interrupt)
X'80' indicates the program is not present in the system
X'87' indicates the program is PTO bound
X'89' indicates the program is IDRA bound

C Problem Program PIB Flag (Last Byte in PIB):

Bit 0: 1 = Batched Job in Foreground
0 = No BJF
1: Cancel in LTA and Device not Assigned
2: 1 = /& on SYSIN if DASD
0 = No /& on SYSIN
3-4: Reserved
5: 1 = Task is cancelled
0 = Task not cancelling
6: 1 = Subtask (s) attached
0 = No subtasks attached
7: 1 = In AB Routine
0 = Not in AB Routine

D PIB Assign Flag

X'80' = SYSRES DASD file protect inhibited (allow write operation on SYSRES)
X'40' = Channel appendage exit allowed (BTAM)
X'20' = Cancel in progress (used in terminator function)
X'10' = Cancel control (set on a foreground cancel)
X'08' = Hold-Release flag for foreground assignments
X'07' = Supervisor or Attention routine PIB assign flag setting
X'04' = Background program PIB assign flag setting
X'02' = Foreground 1 program PIB assign flag setting
X'01' = Foreground 2 program PIB assign flag setting

E Attention PIB Flag

Bit 0: 1 = Registers stored
0 = Registers not stored
1-5 : 0 = Always zero
6 : 1 = Attention routine active
0 = Attention routine SVC 2-bound
7 : 1 = Active
0 = SVC 7-bound

X'80' indicates the attention routine is not present in the system.
X'89' indicates the program is IDRA bound

F Attention PIB Switch Byte

Bit 0-2: Reserved
3: 1 = PTAFTCH (Fetch $$ANERRY, Z, or 0) Switch ON
0 = PTAFTCH (Fetch $$ANERRY, Z, or 0) Switch OFF
4: 1 = Detach Logical Attention Routine ($$BATTNA) Switch ON
0 = Detach Logical Attention Routine ($$BATTNA) Switch OFF
5: 1 = Physical Attention Recall Switch ON
0 = Physical Attention Recall Switch OFF
6: 1 = Attention Request Switch ON
0 = Attention Request Switch OFF
7: 1 = External Interrupt Request Switch ON
0 = External Interrupt Request Switch OFF

Figure 4.5. PIB Flag Expansions

Byte Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
All Bound PIB	Reserved						H'16' Priority of All Bound PIB (Lowest)		Reserved				H'0' All Bound PIB Displacement		Reserved	
Background PIB	Address of BG Comm. Region		System LUB Index		Reserved		Priority of BG PIB (Note 4)		Address of Termination ECB, if any, or F'0'				X'0010' BG PIB Displacement		Reserved	
FG2 PIB (Note 1)	Address of Area Comm. Region (Note 2)		System LUB Index		Reserved		Priority of F2 PIB (Note 4)		Address of Termination ECB, if any, or F'0'				X'0020' F2 PIB Displacement		Reserved	
FG1 PIB (Note 1)	Address of Area Comm. Region (Note 2)		System LUB Index		Reserved		Priority of F1 PIB (Note 4)		Address of Termination ECB, if any, or F'0'				X'0030' F1 PIB Displacement		Reserved	
Attention PIB	Address of BG Comm. Region		0	0	Reserved		H'3' Priority of Attention PIB		F'0'				X'0040' Attention PIB Displacement		Reserved	
Quiesce I/O PIB	Reserved						H'2' Priority of Quiesce I/O PIB		F'0'				X'0050' Quiesce PIB Displacement		Reserved	
Supervisor PIB	Reserved						H'1' Priority of Supervisor PIB (Highest)		F'0'				X'0060' Supervisor PIB Displacement		Reserved	
Subtask PIB (Note 3)	Address of Area Comm. Region		System LUB Index		Reserved		Priority of Subtask (Note 4)		ECB Address for Subtask, or F'0'				PIB Displacement of Maintask		Reserved	

= 16 Byte Length

Note 1. Generated only if MPS is specified.

Note 2. Always background communications region except when MPS = BJF.

Note 3. Total of nine subtasks generated, and only when AP is specified.

Note 4. Will be filled in with halfword indicating the relative priority of task in the system (range H'4' to H'15', the lower the number the higher the priority).

Bytes 124 and 125 (X'7C'-'7D') of the communications region contain the address of the second part of the PIB table. Label PIB2AD identifies the first byte of the table. The second part of PIB table comes before the first part in storage allocation.

Figure 4.6. Second Part of Program Information Block (PIB) Table

SYSTEM ACTION UNDER CANCEL

The following lists all cancel codes and their message prefixes. Some do not appear in a foreground PIB, such as the X'FF' code (supervisor catalog failure). This type of function can be performed only in the background partition. The linkage editor and system maintenance functions must also be performed in the background area.

Byte one of the PIB table contains a cancel code stored by the system any time a cancel condition is encountered. The PIB table can be located by displaying on the console the communication region address (located at X'16'-X'17') plus the displacement of a X'5A' and X'5B'. This is the address of the first part of the PIB table. Remember each entry is 16 decimal (X'10') bytes in length. Each byte of the PIB is numbered starting with 0 and continuing through 15. The layout of the PIB table and the communications region can be found in Figures 4.4-4.6 and Figure 1.7.

Cancel Code (Hexadecimal): 10

Message Code: None

Description, Action or Condition: This is normal end of job (EOJ). Cancel code X'10' is posted in byte 1 of the PIB for the program issuing the SVC 14. The next time the canceled program is selected on general exit, an SVC 2 is taken to call in a B-transient program, which, in turn, calls job control to perform the end-of-job step.

Cancel Code (Hexadecimal): 17

Message Code: 0S02I

Description, Action or Condition: This is caused by the main task in a partition issuing the CANCEL macro without detaching all subtasks running under its control.

Cancel Code (Hexadecimal): 18

Message Code: None

Description, Action or Condition: This is caused by the main task issuing the DUMP macro with subtasks attached. It allows the dump to take place without the error cancel message being printed. All subtasks are detached and EOJ is taken after the dump is complete.

Cancel Code (Hexadecimal): 19

Message Code: 0P74I

Description, Action or Condition: This is caused by the operator responding to an I/O error message with the cancel option on the 1052.

Cancel Code (Hexadecimal): 1A

Message Code: 0P73I

Description, Action or Condition: This is caused by an I/O error that cannot be handled by the program (task), thus causing the program to be canceled. If the DUMP option is specified at system generation time, a dump of the supervisor and the partition in which the program was running will be taken.

Cancel Code (Hexadecimal): 1B

Message Code: 0P82I

Description, Action or Condition: This is caused by a channel failure.

Cancel Code (Hexadecimal): 1C

Message Code: 0S14I

Description, Action or Condition: This is caused by a subtask issuing the CANCEL ALL macro. This causes all other subtasks to be detached and canceled. The main task is canceled, and a dump of the supervisor and partition involved results.

Cancel Code (Hexadecimal): 1D

Message Code: 0S12I

Description, Action or Condition: This is caused when the main task terminates before all subtasks have been detached. This indicates the subtasks were canceled before they came to a normal EOJ. The subtasks are detached, and the complete partition is canceled.

Cancel Code (Hexadecimal): 1E

Message Code: 0S13I

Description, Action or Condition: This is caused by the combination of one task issuing an enqueue for a resource, and another task issuing a dequeue for that same resource. As a result, the previous owner cannot be identified because register 0 in the save area has been modified.

Cancel Code (Hesadecimal): 1F

Message Code: 0P81I

Description, Action or Condition: This is caused by a CPU failure.

Cancel Code (Hexadecimal): 20

Message Code: 0S03I or 0S11I

Description, Action or Condition: This is caused by a program check interrupt. The program is canceled by the system. The user may supply a PC or AB routine to handle this condition via the STXIT macro. This code is also used when a routine in the transient area is canceled due to a program check in the task or subtask using it.

Cancel Code (Hexadecimal): 21

Message Code: 0S04I or 0S09I

Description, Action or Condition: This can be caused by many user errors. See Figure 4.7 for a list of the causes.

Cancel Code (Hexadecimal): 22

Message Code: 0S05I or 0S06I

Description, Action or Condition: This is caused by the issuing of a FETCH (SVC 1) or a LOAD (SVC 4) macro whose phase name cannot be found. This cancel code is also used when a logical transient is canceled.

Cancel Code (Hexadecimal): 23

Message Code: 0S02I

Description, Action or Condition: This is caused by a program, task or subtask issuing a CANCEL macro. If issued by a program or task, the program or partition is canceled. If issued by a subtask, the subtask only is canceled.

Cancel Code (Hexadecimal): 24

Message Code: 0S01I

Description, Action or Condition: This is a result of an operator entering CANCEL from the 1052.

Cancel Code (Hexadecimal): 25

Message Code: 0P77I

Description, Action or Condition: This is a result of attempting to load a problem program phase at an address outside main storage or outside the requester's area (background or foreground). This condition also occurs:

1. if the program requires more main storage than is allocated to the partition where the program is to run or
2. if an improper address is detected on an SVC interrupt (i.e., CCW address in CCB is invalid).

Cancel Code (Hexadecimal): 26

Message Code: 0P71I

Description, Action or Condition: This is a result of a program issuing an I/O request for a logical unit that is not assigned to a device. If a dump is taken, general register 1 contains the address of the CCB. If the CCB is unavailable, the logical unit message contains SYSxxx.

Cancel Code (Hexadecimal): 27

Message Code: 0P70I

Description, Action or Condition: This is a result of a program issuing an I/O request for a logical unit for which there is no logical unit block (LUB) entry (invalid LUB code in CCB). If a dump is taken, general register 1 contains the address of the CCB.

The complete text for message 0S041 is:

ILLEGAL SVC - HEX LOCATION nnnnnn - SVC CODE nn

where nn is in hexadecimal notation.

This message results from the following causes:

1. When nn is 02: The phase name given does not start with $$B, or

   For LIOCS, macros called in invalid sequence. As a result, an SVC 8 is issued after an SVC 2 before an SVC 9 has been issued to free the transient area, or

   For other conditions, the user specified a temporary exit (SVC 8) for a logical transient. In the temporary exit routine, another routine is called (by an SVC 2) before an SVC 9 is issued to free the transient area.

2. When nn is 05: The 'to' range specified in the MVCOM macro is invalid, or

   MVCOM macro was issued by a foreground program, operating under single program initiation.

3. When nn is 0A, 12, 13, or 18: The interval timer was not allocated to this partition, or

   The supervisor was generated without the timer option.

4. When nn is 0B: The call was not given by a logical transient routine.

5. When nn is 16, 17, or 1A: The caller did not have a PSW key of zero. This is applicable only in a multiprogramming system.

6. When nn is 23: More than 16 holds have been issued for the same track.

7. When nn is 24: Free a non - DASD or a track that is not held.

8. When nn is 26: A subtask issued attach, or the save area is not on a doubleword boundary.

9. When nn is 27: A main task issued detach without SAVE = parameter, or

   A main task issued detach, but the ID of the subtask in the save area passed is not valid, or

   If a main task attempts to detach an already terminating subtask.

10. When nn is 29: A DEQ is issued by a task that did not ENQ the resource. (This is valid in an AB routine.)

11. When nn is 2A: A subtask (without an ECB=parameter) has issued an ENQ macro, or

    A subtask has issued an ENQ macro to a resource that has not been dequeued by another task that has been terminated, or

    A task has issued two ENQ macros to the same resource without an intervening DEQ.

12. When nn is 2D: Emulator execution was attempted, but the EU parameter of the SUPVR macro was omitted or incorrectly specified during system generation.

13. When nn is 32: For LIOCS:

    a. An imperative macro (such as WRITE or PUT) was issued to a module that does not contain the requested function, or

    b. A PUT was issued for an ISAM retrieve module without a preceding GET, or

    c. An invalid ASA first character for the printer was used, or

    d. A wrong length record indication occurred while processing 1287 documents when RECFORM=UNDEF, or

    e. The 1287 program erroneously contained a CCW(s) with the SLI flag bit 'OFF', or

    For COBOL, a wrong length record was detected in the object program.

14. When nn is any other value: The supervisor function requested by the operand of the SVC is not defined for the supervisor being used.

Figure 4.7. Causes for Message 0S04I (Cancel Code X'21')

Cancel Code (Hexadecimal): 28

Message Code: None

Description, Action or Condition: (QTAM cancel in progress)

Cancel Code (Hexadecimal): 30

Message Code: 0P72I

Description, Action or Condition: This is a result of a program ignoring the reading of the /& statement on SYSRDR or SYSIPT.

Cancel Code (Hexadecimal): 31

Message Code: 0P75I

Description, Action or Condition: This is a result of the number of pending I/O errors exceeding supervisor capacity.

Cancel Code (Hexadecimal): 32

Message Code: 0P76I

Description, Action or Condition: This is caused by DASD file-protect limits being exceeded or by an incorrect record reference for system files on disk. It will also be posted for unrecoverable I/O errors on tape.

Cancel Code (Hexadecimal): 33

Message Code: 0P79I

Description, Action or Condition: This occurs when a DASD command chain in a file-protected environment does not start with a command code of X'07'. This code indicates a long seek and must be the first command in the chain.

Cancel Code (Hexadecimal): 34

Message Code: 0P84I

Description, Action or Condition: This is caused by an unrecoverable I/O error during a FETCH of a non-$ phase, thus resulting in the job being canceled.

Cancel Code (Hexadecimal): FF

Message Code: 0P78I

Description, Action or Condition: This occurred when an IBM-supplied component failed to post a valid cancel code.

All of these cancel codes cancel the program, task, or subtask when they occur. If multitasking is being used and a main task is canceled, all of the subtasks attached are detached and canceled as a result of the main task being canceled, with the exception of cancel code X'23'. If a dump option was specified at system generation time, the contents of the supervisor and the partition in which the cancel condition occurred is written on SYSLST.

The linkage editor map can be a great help in locating programs and subroutines that are included in the programs at object time. Common areas, load address, relocation factors, low-core and high-core addresses are also shown. In addition, the PHASE card is displayed to show where the phase was loaded (i.e., directly following the supervisor or at some other location). This map is also helpful when working with multiphase programs.

The system dump of main storage used with these items allows the programmer to relate all the information he has gathered to the contents of main storage at the time the error occurred. By using the dump and the listing, the programmer can see how his program appeared in main storage at the time of the error. By using the values found in the PIK and PIB table in the dump, he can see partition save areas, registers, and instructions to determine what actually caused the error.

There are times when a system dump is not available to the programmer, such as hard waits and unending loops. When one of these conditions occurs, the only way to get a dump of main storage is to use a stand-alone dump. Remember that the address of the communication region (COMRG) is lost when a stand-alone dump is taken. Therefore, bytes X'16'-X'17' should be displayed before taking a dump of main storage to ensure that the programmer has the correct communication region address to use when he is analyzing the dump. If bytes X'16'-X'17' are not displayed, the communications region start address can still be found by scanning the dump for the date in the form MM/DD/YY or DD/MM/YY (this indicates the start of COMRG). Although the register values in a stand-alone dump (register print area of the dump) may not be valid, the partition save area values most likely will be valid.

Wait States

The system is said to be in a wait state when the "wait" light is continuously lit and the "system" light is off. Wait states are divided into hard waits and soft waits.

If the system is in a hard wait, the wait bit in the current PSW (bit 14) is set to one and the system mask is set to zeros, thus disabling all interrupts. Because no interrupts are allowed, a PSW swap cannot occur and the system must be re-IPLed to continue processing.

A soft wait occurs when the DOS supervisor finds no in-core programs ready to run and loads a PSW with the wait bit set to one and the system mask set to all ones. The first interrupt returns control to the supervisor and processing may continue.

A wait can easily be determined as hard or soft by causing an interrupt. If the system responds with some action, the wait is soft; if not, the wait is hard. The most convenient way for the operator to cause an interrupt is to press the 1052, 3210, or 3215 request key. If the wait is soft, the attention routine responds with the "READY FOR COMMUNICATIONS" message.

SOFT WAITS

If the system is in a continuous soft wait, it is waiting for an interrupt to signal the completion of an event. Although the expected interrupt may be from the timer or external interrupt key, a missing device-end caused by hardware is the most frequent cause. The operator can make each device not-ready, then ready, to generate a device-end interrupt from each address. The system light flashes briefly as the supervisor examines and discards interrupts for which it was not waiting. The interrupt from the device waited for causes normal processing to continue. (The occurrence should be brought to the attention of the customer engineer as a possible hardware failure.) If this technique does not end the wait, take a stand-alone dump to find what the system was waiting for.

HARD WAITS

The DOS supervisor loads a hard-wait PSW when a failure occurs that puts the integrity of the control program or system data in doubt. The supervisor attempts to place a message in low core bytes 0-4. Figure 4.8 shows the explanation for each error.

If a hard wait occurs, it is imperative that this message be retrieved and recorded. Effective diagnosis is extremely difficult if this step is neglected.

If byte one of main storage contains an S (X'E2'), the following information can be obtained easily:

Check byte X'73' for a X'0F'. This indicates either a channel control check or an interface control check. Bytes X'3A'-X'3B' contain the device address. If byte X'73' does not contain a X'0F', a machine check must have occurred.

Byte one may have a W. If a W (X'E6') is found, a hard stop on SYSRES is indicated.

If the CPU detects an error in its own circuitry, or (in the System/360, model 50 or smaller) in the channel or interface control circuits, it forces a machine check interrupt. The system places an S in byte 1 and enters a hard wait. The S is a request to run the SEREP (System Environmental Recording, Editing, and Printing) dump to format and display the contents of the CPU's hardware registers and log-out area for use by the customer engineer. (A SEREP dump configured for the system should be available to the operator. A copy can be obtained from the customer engineer responsible for the CPU.)

Byte 0	Byte 1	Byte 2	Byte 3	Explanation
SYSTEM/360 SEREP Codes:				
X'00'	X'E2'	Not used	Not used	Machine check. Load SEREP. Re-IPL system.
X'01'	X'E2'	Reserved	Reserved	Channel failure: interface or channel control check. Load SEREP. Re-IPL system.
SYSTEM/370 SEREP Codes:				
X'C1'	X'E2'	A, I, S*	Not used	Unrecoverable machine check.
X'C2'	X'E2'	Not used	Not used	Unrecoverable channel failure during RMS fetch.
X'C3'	X'E2'	A, I, S*	Not used	Channel failure on SYSLOG when RMS message scheduled.
X'C4'	X'E2'	A, I, S*	Not used	Reserved (should not occur)
X'C5'	X'E2'	A, I, S*	Not used	Channel failure: ERPIBs exhausted.
X'C6'	X'E2'	A, I, S*	Not used	Channel failure; two channels damaged or a damaged channel situation occurred while RMS was executing an I/O operation.
X'C7'	X'E2'	A, I, S*	Not used	Channel failure; system reset was presented by a channel.
X'C8'	X'E2'	A, I, S*	Not used	Channel failure; system codes in ECSW are invalid.
X'C9'	X'E2'	A, I, S*	Not used	Channel failure; channel address invalid.
SYSTEM/360 and SYSTEM/370 WAIT Codes:				
X'03'	X'E6'	Channel	Unit	DOS unrecoverable disk error during program fetch. The first six sense bytes are placed in hex bytes 5-A. Re-IPL system.
X'04'	X'E6'	Not used	Not used	Cancel condition has occurred while performing a Supervisor fundtion (not a Supervisor detected problem-program error). Normally a Program Check while in Supervisor State. This condition also occurs if a fetch has been issued for and IBM-supplied transient which is not in the system core image library. IBM-supplied $$A, $$B, and $$R transients cannot be placed in a private core image library. Take a stand-alone dump; the name of the transient involved is in the first 8 bytes of the appropriate transient area. Place the transient in the system core image library. Re-IPL system.
X'05'	X'E6'	Channel	Unit	I/O Error Queue has overflowed as the result of an I/O error on a program fetch channel program. Re-IPL system.
X'06'	Not used	Not used	Not used	Reserved (should not occur).
X'07'	X'E6'	Channel	Unit	IPL I/O error. Channel can unit indicate whether SYSRES or communication device. Re-IPL system.
X'08' to X'60'		Channel	Unit	Error recovery messages. Refer to OP messages in DOS Messages, found in Preface.

*Note: A (X'C1') = SYSREC error recording unsuccessful.
I (X'C9') = SYSREC error recording incomplete.
S (X'E2') = SYSREC error recording successful.

Figure 4.8. Low Core Error Bytes

If a program check interrupt occurs while the DOS supervisor is in control of the system, the integrity of the control program itself is in doubt. System response is to put a message of 04W (X'04E6') in bytes 0 and 1 and enter a hard wait. Note that many programs may run in the supervisor state and hence cause this type of hard wait. These programs include BTMOD (in its channel appendage routine), SPOOLing programs such as POWER (which alters the address of the SVC new PSW to point to the POWER partition), most $$A and some $$B transients.

After the 04W message has been noted, a stand-alone dump should be taken. The first diagnostic steps, as with any program check, are to locate the failing instruction and determine the program in error. Use the supervisor assembly listing to determine if the program check address (location X'2D'-X'2F') falls within the supervisor nucleus (address less than label NUCEND), within the logical transient area (label LTA to LTA+X'4B0'), the physical transient area (label PTA to PTA+X'228'), or outside the supervisor (address greater than label PPBEG).

In the first case, use the supervisor listing to find what routine was being executed and what function the supervisor was attempting to perform. Use the I/O old PSW to find the device involved in the last I/O interrupt received and the SVC old PSW for the last SVC executed.

In the second and third cases, the name of the transient involved appears at the beginning of the transient area involved. Use the <u>DOS System Generation</u> listed in the <u>Preface</u> to find the function of that transient.

When the program check address is outside the supervisor, find the partition it falls in and use the program documentation to locate the failure.

If W is not present in location 1, record the communications region address contained in locations X'16'-X'17' and take a stand-alone dump. Check the PIK (located at displacement X'2E' in the communications region) to determine the task in control. Then, locate and examine the PIB table entry for the task in control.

The All Bound PIB is usually active, indicating an I/O interrupt or event has not occurred. The program or task save areas indicate the device(s) or resource(s) being waited on. The PIB supplies information such as the cancel code and the address of the partition save area. The save area contains the PSW. The instruction address portion of the PSW should be pointing to the last executed instruction. The register values can also be helpful at this point.

Register 14 is used as a standard return from an IOCS module. Register 15 contains the address of the IOCS module. Register 1 points to an ECB, RCB, CCB, or last phase or transient fetched or loaded.

<u>Note</u>: Certain unusual hardware and software failures can cause the system to halt processing with <u>both</u> the system light and the wait light on continuously. This indicates the current PSW has its wait bit set on, but the CPU is operating (processing microprogram instructions). If possible, the system should be left in this state until a customer engineer has arrived. A stand-alone dump can show the I/O operations in progress. No low-core message will be found, and a re-IPL is necessary to continue processing.

By gathering all of the proper documentation and using some of the aids given in this section, most errors should be resolved without too much difficulty.

Debugging Assembler Programs

The proper documentation and careful interpretation of system messages is needed to ensure that the programmer understands the diagnostics provided by the system. Internal pointers found in the system communications region and PIB table allow the programmer to analyze main storage dumps, to locate programs and save areas, and to determine the cause of the error.

The linkage editor map shows where programs should be located in main storage, where overlays are loaded, and whether the program is relocatable or assembled for operation in only one partition. Remember that all relocatable programs are assembled with a load address of zero in the Disk Operating System.

See Figures 1.7 and 4.4-4.6 for the format of the system communications region and the PIB table and Figure 1.2 for the format of the partition save areas.

Debugging COBOL Programs

Debugging information for the COBOL programmer includes an example of a program named TESTRUN, consisting of a source statement listing, Data Division map, Procedure Division map, diagnostic messages, linkage editor map, and an abnormal termination dump. The IBM Disk Operating System Full American National Standard COBOL Compiler is used for the compilation job step. Figure 4.9 contains the program output in its entirety.

The Data Division map provides the internal name generated by the compiler for data names and file names defined in the program. This internal name is the same as that used in the object code listing. An address is provided for each name, in the form of a base and a displacement.

The Procedure Division map is a listing of the object code. Compiler-generated card numbers identify the COBOL statement in the source deck containing the verb that corresponds to the object code. The object code listing also contains the relative address of the object code instruction. In addition to the object code, a Task Global Table (TGT), a Program Global Table (PGT), a literal pool and register assignments are provided when a Procedure Division map is requested. The TGT is used to record and save information needed during the execution of the object program. The PGT contains literals and the addresses of procedure names and generated procedure names referenced by Procedure Division instructions.

The linkage editor map contains the load address of the program and lists the names and locations of COBOL subroutines in main storage.

HOW TO USE A DUMP

When a job is abnormally terminated due to a serious error in the problem program, a message is written on SYSLST that indicates the:

1. type of interrupt; for example, a program check
2. hexadecimal address of the instruction that caused the interrupt
3. condition code
4. reason for the interrupt; for example, a data exception.

The instruction address can be compared to the Procedure Division map, where a relative address is provided for each statement. The load address of the module (which can be obtained from the map of main storage generated by the Linkage Editor) must be subtracted from the instruction address to obtain the relative instruction address as shown in the Procedure Division map. If the interrupt occurred within the COBOL program, the programmer can use the error address and the Procedure Division map to locate the specific statement in the program that caused the dump to the taken. Examination of the statement and the fields associated with it may produce information as to the specific nature of the error.

Figure 4.9 illustrates a dump caused by a data exception. Invalid data, that is, data that does not correspond to its usage, is placed in the numeric field B as a result of redefinition.

- The program interrupt occurred at hexadecimal location 0039Bc. This is indicated in the SYSLST message printed just before the dump.
- The linkage editor map indicates that the program was loaded into address 0032A0. This is determined by examining the load point of the control section TESTRUN. TESTRUN is the name assigned to the program module by the source coding: PROGRAM-ID. TESTRUN.
- The specific instructin which caused the dump is located by subtracting the load address from the interrupt address (that is, subtracting 32A0 from 39BC). The result, 71C, is the relative interrupt address and can be found in the object code listing. In this case, the instruction in question is AP (add decimal).
- The left-hand column of the object code listing gives the compiler-generated card number associated with the instruction. It is card 69. As seen in the source listing, card 69 contains the COMPUTE statement.

LOCATING A DTF

One or more DTF's are generated by the compiler for each file opened in the COBOL program. All information about that file is found within the DTF or in the fields preceding the DTF. A particular DTF may be located in a system dump as follows:

1. Determine the order of the DTF address cells in the TGT from the DTF numbers shown for each file name in the Data Division map.

   Note: Since the order is the same as the FD's (File Description) in the Data Division, the order can be determined from the source program if the Data Division map is not requested.

2. Find the relative starting address of the block of DTF cells from the TGT.

3. Calculate the absolute starting address of the block by adding the hexadecimal relocation factor for the beginning of the object module as given in the linkage editor map.

4. Allowing one fullword per DTF cell, count off the cells from the starting address found in Step 3, using the order determined in Step 1 to locate the desired DTF cell.

5. If more than one DTF is generated for a file, the above procedure should be followed using the PGT and the SUBDTF cells rather than the TGT and the DTFADDR cells. The order in which multiple DTF's appear in main storage is dependent on the OPEN option as follows:

   (a) INPUT

   (b) OUTPUT

   (c) I-O or INPUT REVERSED

   There can be two or three SUBDTF's for each file with multiple OPEN options. The Data Division map sould be used to determine the file and the number of the SUBDTF CELL assigned to it.

LOCATING DATA

The location assigned to a given data name may similarly be found by using the BL (Base Locator) number and displacement given for that entry in the Data Division map, and then locating the appropriate fullword BL cell in the TGT. The sum of the displacement and the contents of the cell give the relative address of the desired area. This can then be converted to an absolute address as described for locating a DTF.

```
// JOB DTACHK                                                                          11.49.30
// OPTION NODECK,LINK,LIST,LISTX,SYM,ERRS
// EXEC FCOBOL




     1                                     IBM DOS AMERICAN NATIONAL STANDARD COBOL                    CBF CL3-3 07/23/71


 CBL QUOTE,SEQ
00001   000010 IDENTIFICATION DIVISION.
00002   000020 PROGRAM-ID. TESTRUN.
00003   000030     AUTHOR. PROGRAMMER NAME.
00004   000040     INSTALLATION. NEW YORK PROGRAMMING CENTER.
00005   000050     DATE-WRITTEN.  FEBRUARY 4, 1971
00006   000060 DATE-COMPILED. 07/23/71
00007   000070    REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR
00008   000080     COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK AS
00009   000090     INPUT.
00010   000100
00011   000110 ENVIRONMENT DIVISION.
00012   000120 CONFIGURATION SECTION.
00013   000130 SOURCE-COMPUTER. IBM-360-H50.
00014   000140 OBJECT-COMPUTER. IBM-360-H50.
00015   000150 INPUT-OUTPUT SECTION.
00016   000160 FILE-CONTROL.
00017   000170     SELECT FILE-1 ASSIGN TO SYS008-UT-2400-S.
00018   000180     SELECT FILE-2 ASSIGN TO SYS008-UT-2400-S.
00019   000190
00020   000200 DATA DIVISION.
00021   000210 FILE SECTION.
00022   000220 FD  FILE-1
00023   000230     LABEL RECORDS ARE OMITTED
00024   000240     BLOCK CONTAINS 5 RECORDS
00025   000250     RECORDING MODE IS F
00026   000255     RECORD CONTAINS 20 CHARACTERS
00027   000260     DATA RECORD IS RECORD-1.
00028   000270 01  RECORD-1.
00029   000280     05 FIELD-A PIC X(20).
00030   000290 FD  FILE-2
00031   000300     LABEL RECORDS ARE OMITTED
00032   000310     BLOCK CONTAINS 5 RECORDS
00033   000320     RECORD CONTAINS 20 CHARACTERS
00034   000330     RECORDING MODE IS F
00035   000340     DATA RECORD IS RECORD-2.
00036   000350 01  RECORD-2.
00037   000360     05 FIELD-A PIC X(20).
00038   000370 WORKING-STORAGE SECTION.
00039   000380 01  FILLER.
00040   000390     02 COUNT PIC S99 COMP SYNC.
00041   000400     02 ALPHABET PIC X(26) VALUE IS "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
00042   000410     02 ALPHA REDEFINES ALPHABET PIC X OCCURS 26 TIMES.
00043   000420     02 NUMBR PIC S99 COMP SYNC.
00044   000430     02 DEPENDENTS PIC X(26) VALUE "01234012340123401234012340".
00045   000440     02 DEPEND REDEFINES DEPENDENTS PIC X OCCURS 26 TIMES.
00046   000450 01  WORK-RECORD.
00047   000460     05 NAME-FIELD PIC X.
00048   000470     05 FILLER PIC X.
00049   000480     05 RECORD-NO PIC 9999.
00050   000490     05 FILLER PIC X VALUE IS SPACE.
00051   000500     05 LOCATION PIC AAA VALUE IS "NYC".
00052   000510     05 FILLER PIC X VALUE IS SPACE.
00053   000520     05 NO-OF-DEPENDENTS PIC XX.
00054   000530     05 FILLER PIC X(7) VALUE IS SPACES.
00055   000534 01  RECORDA.
00056   000535     02 A PICTURE S9(4) VALUE 1234.
```

Figure 4.9. COBOL Sample Program (Part 1 of 27)

```
          2

  00057   000536     02 B REDEFINES A PICTURE S9(7) COMPUTATIONAL-3.
  00058   000540
  00059   000550 PROCEDURE DIVISION.
  00060   000560 BEGIN. READY TRACE.
  00061   000570     NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
  00062   000580     AND INITIALIZES COUNTERS.
  00063   000590 STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO COUNT, NUMBR.
  00064   000600     NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE
  00065   000610     CONTAINED IN THE FILE, WRITES THEM ON TAPE, AND DISPLAYS
  00066   000620     THEM ON THE CONSOLE.
  00067   000630 STEP-2. ADD 1 TO COUNT, NUMBR. MOVE ALPHA (COUNT) TO
  00068   000640     NAME-FIELD.
  00069   000645         COMPUTE B = B + 1.
  00070   000650     MOVE DEPEND (COUNT) TO NO-OF-DEPENDENTS.
  00071   000660     MOVE NUMBR TO RECORD-NO.
  00072   000670 STEP-3. DISPLAY WORK-RECORD UPON CONSOLE. WRITE RECORD-1 FROM
  00073   000680     WORK-RECORD.
  00074   000690 STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL COUNT IS EQUAL TO 26.
  00075   000700     NOTE THAT THE FOLLOWING CLOSES THE OUTPUT FILE AND REOPENS
  00076   000710     IT AS INPUT.
  00077   000720 STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
  00078   000730     NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES
  00079   000740     OUT EMPLOYEES WITH NO DEPENDENTS.
  00080   000750 STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
  00081   000760 STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
  00082   000770     NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO STEP-6.
  00083   000780 STEP-8. CLOSE FILE-2.
  00084   000790     STOP RUN.
```

Figure 4.9. COBOL Sample Program (Part 2 of 27)

```
3

INTRNL NAME   LVL SOURCE NAME                           BASE    DISPL   INTRNL NAME   DEFINITION    USAGE       R  O  Q  M

DNM=1-148     FD  FILE-1                               DTF=01           DNM=1-148                   DTFMT                F
DNM=1-178     01  RECORD-1                              BL=1     000    DNM=1-178     DS 0CL20      GROUP
DNM=1-199     02  FIELD-A                               BL=1     000    DNM=1-199     DS 20C        DISP
DNM=1-216     FD  FILE-2                               DTF=02           DNM=1-216                   DTFMT                F
DNM=1-246     01  RECORD-2                              BL=2     000    DNM=1-246     DS 0CL20      GROUP
DNM=1-267     02  FIELD-A                               BL=2     000    DNM=1-267     DS 20C        DISP
DNM=1-287     01  FILLER                                BL=3     000    DNM=1-287     DS 0CL56      GROUP
DNM=1-306     02  COUNT                                 BL=3     000    DNM=1-306     DS 1H         COMP
DNM=1-321     02  ALPHABET                              BL=3     002    DNM=1-321     DS 26C        DISP
DNM=1-339     02  ALPHA                                 BL=3     002    DNM=1-339     DS 1C         DISP        R  O
DNM=1-357     02  NUMBR                                 BL=3     01C    DNM=1-357     DS 1H         COMP
DNM=1-372     02  DEPENDENTS                            BL=3     01E    DNM=1-372     DS 26C        DISP
DNM=1-392     02  DEPEND                                BL=3     01E    DNM=1-392     DS 1C         DISP        R  O
DNM=1-408     01  WORK-RECORD                           BL=3     038    DNM=1-408     DS 0CL20      GROUP
DNM=1-432     02  NAME-FIELD                            BL=3     038    DNM=1-432     DS 1C         DISP
DNM=1-452     02  FILLER                                BL=3     039    DNM=1-452     DS 1C         DISP
DNM=1-471     02  RECORD-NO                             BL=3     03A    DNM=1-471     DS 4C         DISP-NM
DNM=1-490     02  FILLER                                BL=3     03E    DNM=1-490     DS 1C         DISP
DNM=2-000     02  LOCATION                              BL=3     03F    DNM=2-000     DS 3C         DISP
DNM=2-018     02  FILLER                                BL=3     042    DNM=2-018     DS 1C         DISP
DNM=2-037     02  NO-OF-DEPENDENTS                      BL=3     043    DNM=2-037     DS 2C         DISP
DNM=2-063     02  FILLER                                BL=3     045    DNM=2-063     DS 7C         DISP
DNM=2-082     01  RECORDA                               BL=3     050    DNM=2-082     DS 0CL4       GROUP
DNM=2-102     02  A                                     BL=3     050    DNM=2-102     DS 4C         DISP-NM
DNM=2-113     02  B                                     BL=3     050    DNM=2-113     DS 4P         COMP-3      R
```

Figure 4.9. COBOL Sample Program (Part 3 of 27)

```
  4



                    MEMORY MAP

                         TGT                    003E8

                    SAVE AREA                   003E8
                    SWITCH                      00430
                    TALLY                       00434
                    SORT SAVE                   00438
                    ENTRY-SAVE                  0043C
                    SORT CORE SIZE              00440
                    NSTD-REELS                  00444
                    SORT RET                    00446
                    WORKING CELLS               00448
                    SORT FILE SIZE              00578
                    SORT MODE SIZE              0057C
                    PGT-VN TBL                  00580
                    TGT-VN TBL                  00584
                    SORTAB ADDRESS              00588
                    LENGTH OF VN TBL            0058C
                    LNGTH OF SORTAB             0058E
                    PGM ID                      00590
                    A(INIT1)                    00598
                    UPSI SWITCHES               0059C
                    OVERFLOW CELLS              005A4
                    BL CELLS                    005A4
                    DTFADR CELLS                005B0
                    TEMP STORAGE                005B8
                    TEMP STORAGE-2              005C0
                    TEMP STORAGE-3              005C0
                    TEMP STORAGE-4              005C0
                    BLL CELLS                   005C0
                    VLC CELLS                   005C4
                    SBL CELLS                   005C4
                    INDEX CELLS                 005C4
                    SUBADR CELLS                005C4
                    ONCTL CELLS                 005CC
                    PFMCTL CELLS                005CC
                    PFMSAV CELLS                005CC
                    VN CELLS                    005D0
                    SAVE AREA =2                005D4
                    XSASW CELLS                 005D4
                    XSA CELLS                   005D4
                    PARAM CELLS                 005D4
                    RPTSAV AREA                 005D8
                    CHECKPT CTR                 005D8
                    IOPTR CELLS                 005D8

LITERAL POOL (HEX)

00618 (LIT+0)      00000001  1C00001A  5B5BC2D6  D7C5D540  5B5BC2C3  D3D6E2C5
00630 (LIT+24)     5B5BC2C6  C3D4E4D3  F0E90000  C0000000

     DISPLAY LITERALS (BCD)

00640 (LTL+40)    'WORK-RECORD'
```

Figure 4.9. COBOL Sample Program (Part 4 of 27)

Figure 4.9. COBOL Sample Program (Part 5 of 27)

```
    5



                   PGT                        005E0

              OVERFLOW CELLS                  005E0
              VIRTUAL CELLS                   005E0
              PROCEDURE NAME CELLS            005EC
              GENERATED NAME CELLS            00600
              SUBDTF ADDRESS CELLS            00610
              VNI CELLS                       00610
              LITERALS                        00618
              DISPLAY LITERALS                00640


 REGISTER ASSIGNMENT

  REG 6   BL =3
  REG 7   BL =1
  REG 8   BL =2

60                 00064C                    START    EQU   *
                   00064C  58 F0 C 004                L     15,004(0,12)          V(ILBDDSP0)
                   000650  05 1F                      BALR  1,15
                   000652  000140                     DC    X'000140'
                   000655  04F6F0404040               DC    X'04F6F0404040'
60                 00065C  96 40 D 048                OI    048(13),X'40'         SWT+0
63                 000660  58 F0 C 004                L     15,004(0,12)          V(ILBDDSP0)
                   000664  05 1F                      BALR  1,15
                   000666  000140                     DC    X'000140'
                   000669  04F6F3404040               DC    X'04F6F3404040'
63                 000670  41 10 C 040                LA    1,040(0,12)           LIT+8
                   000674  58 00 D 1C8                L     0,1C8(0,13)           DTF=1
                   000678  18 40                      LR    4,0
                   00067A  05 F0                      BALR  15,0
                   00067C  50 00 F 008                ST    0,008(0,15)
                   000680  45 00 F 00C                BAL   0,00C(0,15)
                   000684  00000000                   DC    X'00000000'
                   000688  0A 02                      SVC   2
                   00068A  41 00 D 1C8                LA    0,1C8(0,13)           DTF=1
                   00068E  58 F0 C 008                L     15,008(0,12)          V(ILBDIML0)
                   000692  05 EF                      BALR  14,15
                   000694  58 10 D 1C8                L     1,1C8(0,13)           DTF=1
                   000698  96 10 1 020                OI    020(1),X'10'
                   00069C  50 20 D 1BC                ST    2,1BC(0,13)           BL =1
                   0006A0  58 70 D 1BC                L     7,1BC(0,13)           BL =1
63                 0006A4  D2 01 6 000 C 038          MVC   000(2,6),038(12)      DNM=1-306       LIT+0
                   0006AA  D2 01 6 01C C 038          MVC   01C(2,6),038(12)      DNM=1-357       LIT+0
67                 0006B0                    PN=01    EQU   *
                   0006B0  58 F0 C 004                L     15,004(0,12)          V(ILBDDSP0)
                   0006B4  05 1F                      BALR  1,15
                   0006B6  000140                     DC    X'000140'
                   0006B9  04F6F7404040               DC    X'04F6F7404040'
67                 0006C0  48 30 C 03A                LH    3,03A(0,12)           LIT+2
                   0006C4  4A 30 6 000                AH    3,000(0,6)            DNM=1-306
                   0006C8  4E 30 D 1D0                CVD   3,1D0(0,13)           TS=01
                   0006CC  D7 05 D 1D0 D 1D0          XC    1D0(6,13),1D0(13)     TS=01           TS=01
                   0006D2  94 0F D 1D6                NI    1D6(13),X'0F'         TS=01+6
```

```
      6

               0006D6  4F 30 D 1D0                     CVB   3,1D0(0,13)          TS=01
               0006DA  40 30 6 000                     STH   3,000(0,6)           DNM=1-306
               0006DE  48 30 C 03A                     LH    3,03A(0,12)          LIT+2
               0006E2  4A 30 6 01C                     AH    3,01C(0,6)           DNM=1-357
               0006E6  4E 30 D 1D0                     CVD   3,1D0(0,13)          TS=01
               0006EA  D7 05 D 1D0 D 1D0               XC    1D0(6,13),1D0(13)    TS=01          TS=01
               0006F0  94 0F D 1D6                     NI    1D6(13),X'0F'        TS=01+6
               0006F4  4F 30 D 1D0                     CVB   3,1D0(0,13)          TS=01
               0006F8  40 30 6 01C                     STH   3,01C(0,6)           DNM=1-357
    67         0006FC  41 40 6 002                     LA    4,002(0,6)           DNM=1-339
               000700  48 20 6 000                     LH    2,000(0,6)           DNM=1-306
               000704  4C 20 C 03A                     MH    2,03A(0,12)          LIT+2
               000708  1A 42                           AR    4,2
               00070A  5B 40 C 038                     S     4,038(0,12)          LIT+0
               00070E  50 40 D 1DC                     ST    4,1DC(0,13)          SBS=1
               000712  58 E0 D 1DC                     L     14,1DC(0,13)         SBS=1
               000716  D2 00 6 038 E 000               MVC   038(1,6),000(14)     DNM=1-432      DNM=1-339
    69         00071C  FA 30 6 050 C 03C               AP    050(4,6),03C(1,12)   DNM=2-113      LIT+4
    70         000722  41 40 6 01E                     LA    4,01E(0,6)           DNM=1-392
               000726  48 20 6 000                     LH    2,000(0,6)           DNM=1-306
               00072A  4C 20 C 03A                     MH    2,03A(0,12)          LIT+2
               00072E  1A 42                           AR    4,2
               000730  5B 40 C 038                     S     4,038(0,12)          LIT+0
               000734  50 40 D 1E0                     ST    4,1E0(0,13)          SBS=2
               000738  58 E0 D 1E0                     L     14,1E0(0,13)         SBS=2
               00073C  D2 00 6 043 E 000               MVC   043(1,6),000(14)     DNM=2-37       DNM=1-392
               000742  92 40 6 044                     MVI   044(6),X'40'         DNM=2-37+1
    71         000746  48 30 6 01C                     LH    3,01C(0,6)           DNM=1-357
               00074A  4E 30 D 1D0                     CVD   3,1D0(0,13)          TS=01
               00074E  F3 31 6 03A D 1D6               UNPK  03A(4,6),1D6(2,13)   DNM=1-471      TS=07
               000754  96 F0 6 03D                     OI    03D(6),X'F0'         DNM=1-471+3
    72         000758  58 F0 C 004                     L     15,004(0,12)         V(ILBDDSP0)
               00075C  05 1F                           BALR  1,15
               00075E  000140                          DC    X'000140'
               000761  04F7F2404040                    DC    X'04F7F2404040'
    72         000768  58 F0 C 004                     L     15,004(0,12)         V(ILBDDSP0)
               00076C  05 1F                           BALR  1,15
               00076E  0002                            DC    X'0002'
               000770  00                              DC    X'00'
               000771  000014                          DC    X'000014'
               000774  0D0001C4                        DC    X'0D0001C4'          BL =3
               000778  0038                            DC    X'0038'
               00077A  FFFF                            DC    X'FFFF'
    72         00077C  D2 13 7 000 6 038               MVC   000(20,7),038(6)     DNM=1-178      DNM=1-408
               000782  58 10 D 1C8                     L     1,1C8(0,13)          DTF=1
               000786  18 41                           LR    4,1
               000788  58 F0 1 010                     L     15,010(0,1)
               00078C  45 E0 F 00C                     BAL   14,00C(0,15)
               000790  50 20 D 1BC                     ST    2,1BC(0,13)          BL =1
               000794  58 70 D 1BC                     L     7,1BC(0,13)          BL =1
               000798  58 10 D 1E8                     L     1,1E8(0,13)          VN=01
               00079C  07 F1                           BCR   15,1
    74         00079E                          PN=02   EQU   *
               00079E  58 F0 C 004                     L     15,004(0,12)         V(ILBDDSP0)
               0007A2  05 1F                           BALR  1,15
               0007A4  000140                          DC    X'000140'
               0007A7  04F7F4404040                    DC    X'04F7F4404040'
```

Figure 4.9. COBOL Sample Program (Part 6 of 27)

```
    7

74              0007AE  58 00 D 1E8                  L      0,1E8(0,13)          VN=01
                0007B2  50 00 D 1E4                  ST     0,1E4(0,13)          PSV=1
                0007B6  58 00 C 020                  L      0,020(0,12)          GN=01
                0007BA  50 00 D 1E8                  ST     0,1E8(0,13)          VN=01
                0007BE                   GN=01     EQU    *
                0007BE  48 30 6 000                  LH     3,000(0,6)           DNM=1-306
                0007C2  49 30 C 03E                  CH     3,03E(0,12)          LIT+6
                0007C6  58 F0 C 024                  L      15,024(0,12)         GN=02
                0007CA  07 8F                        BCR    8,15
                0007CC  58 10 C 00C                  L      1,00C(0,12)          PN=01
                0007D0  07 F1                        BCR    15,1
                0007D2                   GN=02     EQU    *
                0007D2  58 00 D 1E4                  L      0,1E4(0,13)          PSV=1
                0007D6  50 00 D 1E8                  ST     0,1E8(0,13)          VN=01
77              0007DA  58 F0 C 004                  L      15,004(0,12)         V(ILBDDSP0)
                0007DE  05 1F                        BALR   1,15
                0007E0  000140                       DC     X'000140'
                0007E3  04F7F7404040                 DC     X'04F7F7404040'
77              0007EA  58 10 D 1C8                  L      1,1C8(0,13)          DTF=1
                0007EE  94 EF 1 020                  NI     020(1),X'EF'
                0007F2  18 01                        LR     0,1
                0007F4  18 40                        LR     4,0
                0007F6  41 10 C 048                  LA     1,048(0,12)          LIT+16
                0007FA  05 F0                        BALR   15,0
                0007FC  50 00 F 008                  ST     0,008(0,15)
                000800  45 00 F 00C                  BAL    0,00C(0,15)
                000804  00000000                     DC     X'00000000'
                000808  0A 02                        SVC    2
                00080A  58 00 D 1C8                  L      0,1C8(0,13)          DTF=1
                00080E  41 10 C 050                  LA     1,050(0,12)          LIT+24
                000812  0A 02                        SVC    2
77              000814  41 10 C 040                  LA     1,040(0,12)          LIT+8
                000818  58 00 D 1CC                  L      0,1CC(0,13)          DTF=2
                00081C  18 40                        LR     4,0
                00081E  05 F0                        BALR   15,0
                000820  50 00 F 008                  ST     0,008(0,15)
                000824  45 00 F 00C                  BAL    0,00C(0,15)
                000828  00000000                     DC     X'00000000'
                00082C  0A 02                        SVC    2
                00082E  41 00 D 1CC                  LA     0,1CC(0,13)          DTF=2
                000832  58 F0 C 008                  L      15,008(0,12)         V(ILBDIML0)
                000836  05 EF                        BALR   14,15
                000838  58 10 D 1CC                  L      1,1CC(0,13)          DTF=2
                00083C  96 10 1 020                  OI     020(1),X'10'
80              000840                   PN=03     EQU    *
                000840  58 F0 C 004                  L      15,004(0,12)         V(ILBDDSP0)
                000844  05 1F                        BALR   1,15
                000846  000140                       DC     X'000140'
                000849  04F8F0404040                 DC     X'04F8F0404040'
80              000850  58 10 D 1CC                  L      1,1CC(0,13)          DTF=2
                000854  58 F0 C 028                  L      15,028(0,12)         GN=03
                000858  91 20 1 010                  TM     010(1),X'20'
                00085C  07 1F                        BCR    1,15
                00085E  18 41                        LR     4,1
                000860  41 F0 C 028                  LA     15,028(0,12)         GN=03
                000864  D2 02 1 025 F 001            MVC    025(3,1),001(15)
                00086A  58 F0 1 010                  L      15,010(0,1)
```

Figure 4.9. COBOL Sample Program (Part 7 of 27)

```
    8

                000886E  45 E0 F 008                BAL   14,008(0,15)
                000872   50 20 D 1C0                ST    2,1C0(0,13)        BL =2
                000876   58 80 D 1C0                L     8,1C0(0,13)        BL =2
                00087A   D2 13 6 038 8 000          MVC   038(20,6),000(8)   DNM=1-408        DNM=1-246
                000880   58 F0 C 018                L     15,018(0,12)       PN=04
                000884   07 FF                      BCR   15,15
   80           000886                     GN=03    EQU   *
                000886   58 10 C 01C                L     1,01C(0,12)        PN=05
                00088A   07 F1                      BCR   15,1
   81           00088C                     PN=04    EQU   *
                00088C   58 F0 C 004                L     15,004(0,12)       V(ILBDDSP0)
                000890   05 1F                      BALR  1,15
                000892   000140                     DC    X'000140'
                000895   04F8F1404040               DC    X'04F8F1404040'
   81           00089C   58 20 C 02C                L     2,02C(0,12)        GN=04
                0008A0   D5 00 C 058 6 043          CLC   058(1,12),043(6)   LIT+32           DNM=2-37
                0008A6   07 72                      BCR   7,2
                0008A8   95 40 6 044                CLI   044(6),X'40'       DNM=2-37+1
                0008AC   07 72                      BCR   7,2
   81           0008AE   D2 00 6 043 C 059          MVC   043(1,6),059(12)   DNM=2-37         LIT+33
                0008B4   92 40 6 044                MVI   044(6),X'40'       DNM=2-37+1
   82           0008B8                     GN=04    EQU   *
                0008B8   58 10 C 05C                L     1,05C(0,12)        LIT+36
                0008BC   50 10 D 1EC                ST    1,1EC(0,13)        PRM=1
                0008C0   41 20 D 1EC                LA    2,1EC(0,13)        PRM=1
                0008C4   58 F0 C 004                L     15,004(0,12)       V(ILBDDSP0)
                0008C8   05 1F                      BALR  1,15
                0008CA   8001                       DC    X'8001'
                0008CC   10                         DC    X'10'
                0008CD   00000B                     DC    X'00000B'
                0008D0   0C000060                   DC    X'0C000060'        LIT+40
                0008D4   0000                       DC    X'0000'
                0008D6   00                         DC    X'00'
                0008D7   000014                     DC    X'000014'
                0008DA   0D0001C4                   DC    X'0D0001C4'        BL =3
                0008DE   0038                       DC    X'0038'
                0008E0   FFFF                       DC    X'FFFF'
   82           0008E2   58 10 C 014                L     1,014(0,12)        PN=03
                0008E6   07 F1                      BCR   15,1
   83           0008E8                     PN=05    EQU   *
                0008E8   58 F0 C 004                L     15,004(0,12)       V(ILBDDSP0)
                0008EC   05 1F                      BALR  1,15
                0008EE   000140                     DC    X'000140'
                0008F1   04F8F3404040               DC    X'04F8F3404040'
   83           0008F8   58 10 D 1CC                L     1,1CC(0,13)        DTF=2
                0008FC   94 EF 1 020                NI    020(1),X'EF'
                000900   18 01                      LR    0,1
                000902   18 40                      LR    4,0
                000904   41 10 C 048                LA    1,048(0,12)        LIT+16
                000908   07 00                      BCR   0,0
                00090A   05 F0                      BALR  15,0
                00090C   50 00 F 008                ST    0,008(0,15)
                000910   45 00 F 00C                BAL   0,00C(0,15)
                000914   00000000                   DC    X'00000000'
                000918   0A 02                      SVC   2
                00091A   58 00 D 1CC                L     0,1CC(0,13)        DTF=2
                00091E   41 10 C 050                LA    1,050(0,12)        LIT+24
```

Figure 4.9. COBOL Sample Program (Part 8 of 27)

```
    9

                   000922  0A 02                       SVC   2
84                 000924  0A 0E                       SVC   14
                   000926  0A 0E                       SVC   14
                   000928  50 D0 5 008        INIT2    ST    13,008(0,5)
                   00092C  50 50 D 004                 ST    5,004(0,13)
                   000930  58 20 C 000                 L     2,000(0,12)           VIR=1
                   000934  95 00 2 000                 CLI   000(2),X'00'
                   000938  07 79                       BCR   7,9
                   00093A  92 FF 2 000                 MVI   000(2),X'FF'
                   00093E  96 10 D 048                 OI    048(13),X'10'         SWT+0
                   000942  50 E0 D 054        INIT3    ST    14,054(0,13)
                   000946  05 F0                       BALR  15,0
                   000948  91 20 D 048                 TM    048(13),X'20'         SWT+0
                   00094C  47 E0 F 016                 BC    14,016(0,15)
                   000950  58 00 B 048                 L     0,048(0,11)
                   000954  98 2D B 050                 LM    2,13,050(11)
                   000958  58 E0 D 054                 L     14,054(0,13)
                   00095C  07 FE                       BCR   15,14
                   00095E  96 20 D 048                 OI    048(13),X'20'         SWT+0
                   000962  41 60 0 004                 LA    6,004(0,0)
                   000966  41 10 C 00C                 LA    1,00C(0,12)           PN=01
                   00096A  41 70 C 038                 LA    7,038(0,12)           LIT+0
                   00096E  06 70                       BCTR  7,0
                   000970  05 50                       BALR  5,0
                   000972  58 40 1 000                 L     4,000(0,1)
                   000976  1E 4B                       ALR   4,11
                   000978  50 40 1 000                 ST    4,000(0,1)
                   00097C  87 16 5 000                 BXLE  1,6,000(5)
                   000980  41 80 D 1BC                 LA    8,1BC(0,13)           OVF=1
                   000984  41 70 D 1CF                 LA    7,1CF(0,13)           TS=01-1
                   000988  05 10                       BALR  1,0
                   00098A  58 00 8 000                 L     0,000(0,8)
                   00098E  1E 0B                       ALR   0,11
                   000990  50 00 8 000                 ST    0,000(0,8)
                   000994  87 86 1 000                 BXLE  8,6,000(1)
                   000998  D2 03 D 1E8 C 030           MVC   1E8(4,13),030(12)     VN=01           VNI=1
                   00099E  58 60 D 1C4                 L     6,1C4(0,13)           BL =3
                   0009A2  58 70 D 1BC                 L     7,1BC(0,13)           BL =1
                   0009A6  58 80 D 1C0                 L     8,1C0(0,13)           BL =2
                   0009AA  58 E0 D 054                 L     14,054(0,13)
                   0009AE  07 FE                       BCR   15,14
                   000000  05 F0              INIT1    BALR  15,0
                   000002  07 00                       BCR   0,0
                   000004  90 0E F 00A                 STM   0,14,00A(15)
                   000008  47 F0 F 082                 BC    15,082(0,15)
                   00000C                              DS    30F
                   000084  58 C0 F 0C6                 L     12,0C6(0,15)
                   000088  58 E0 C 000                 L     14,000(0,12)          VIR=1
                   00008C  58 D0 F 0CA                 L     13,0CA(0,15)
                   000090  95 00 E 000                 CLI   000(14),X'00'
                   000094  47 70 F 0A2                 BC    7,0A2(0,15)
                   000098  96 10 D 048                 OI    048(13),X'10'         SWT+0
                   00009C  92 FF E 000                 MVI   000(14),X'FF'
                   0000A0  47 F0 F 0AC                 BC    15,0AC(0,15)
                   0000A4  98 CE F 03A                 LM    12,14,03A(15)
                   0000A8  90 EC D 00C                 STM   14,12,00C(13)
                   0000AC  18 5D                       LR    5,13
```

Figure 4.9. COBOL Sample Program (Part 9 of 27)

```
    10

              0000AE  98 9F F 0BA                   LM     9,15,0BA(15)
              0000B2  91 10 D 048                   TM     048(13),X'10'          SWT+0
              0000B6  07 19                         BCR    1,9
              0000B8  07 FF                         BCR    15,15
              0000BA  07 00                         BCR    0,0
              0000BC  00000942                      ADCON L4(INIT3)
              0000C0  00000000                      ADCON L4(INIT1)
              0000C4  00000000                      ADCON L4(INIT1)
              0000C8  000005E0                      ADCON L4(PGT)
              0000CC  000003E8                      ADCON L4(TGT)
              0000D0  0000064C                      ADCON L4(START)
              0000D4  00000928                      ADCON L4(INIT2)
              0000D8  C3D6C2C6F0F0F0F1              DC     X'C3D6C2C6F0F0F0F1'
              0000E0  E3C5E2E3D9E4D540              DC     X'E3C5E2E3D9E4D540'
```

Figure 4.9. COBOL Sample Program (Part 10 of 27)

```
   11

                                 CROSS-REFERENCE DICTIONARY

DATA NAMES                        DEFN       REFERENCE

FILE-1                            00017      00063  00063  00072  00077
RECORD-1                          00028      00072  00072
FILE-2                            00018      00077  00077  00080  00080  00083
RECORD-2                          00036      00080
COUNT                             00040      00063  00067  00067  00067  00070  00074
ALPHA                             00042      00067  00067
NUMBR                             00043      00063  00067  00067  00071
DEPEND                            00045      00070  00070
WORK-RECORD                       00046      00072  00072  00072  00080  00082
NAME-FIELD                        00047      00067
RECORD-NO                         00049      00071  00071
NO-OF-DEPENDENTS                  00053      00070  00070  00081  00081  00081  00081
B                                 00057      00069

PROCEDURE NAMES                   DEFN       REFERENCE

STEP-2                            00067      00074
STEP-3                            00072      00074
STEP-6                            00080      00082
STEP-8                            00083      00080
```

Figure 4.9. COBOL Sample Program (Part 11 of 27)

```
       12

  CARD   ERROR MESSAGE

  56     ILA2190I-W     PICTURE CLAUSE IS SIGNED, VALUE CLAUSE UNSIGNED. ASSUMED POSITIVE.
  67     ILA5011I-W     HIGH ORDER TRUNCATION MIGHT OCCUR.
  67     ILA5011I-W     HIGH ORDER TRUNCATION MIGHT OCCUR.
```

Figure 4.9. COBOL Sample Program (Part 12 of 27)

Figure 4.9. COBOL Sample Program (Part 13 of 27)

```
// EXEC LNKEDT
```

```
JOB  DTACHK    07/23/71   DISK LINKAGE EDITOR DIAGNOSTIC OF INPUT

ACTION TAKEN    MAP
LIST   AUTOLINK   IJFFBZZN
LIST   AUTOLINK   ILBDDSP0
LIST   INCLUDE IJJCPD1                                                            0001
LIST   AUTOLINK   ILBDIML0
LIST   AUTOLINK   ILBDMNS0
LIST   AUTOLINK   ILBDSAE0
LIST   ENTRY
```

```
07/23/71   PHASE   XFR-AD   LOCORE   HICORE   DSK-AD     ESD TYPE   LABEL      LOADED   REL-FR

        PHASE***   0032A0   0032A0   004ADB   63 07 2    CSECT      TESTRUN    0032A0   0032A0

                                                         CSECT      IJFFBZZN   003C50   003C50
                                                       *    ENTRY   IJFFZZZN   003C50
                                                       *    ENTRY   IJFFBZZZ   003C50
                                                       *    ENTRY   IJFFZZZZ   003C50

                                                         CSECT      ILBDSAE0   0049F0   0049F0
                                                            ENTRY   ILBDSAE1   004A06

                                                         CSECT      ILBDMNS0   0049E8   0049E8

                                                         CSECT      ILBDDSP0   0041B8   0041B8
                                                       *    ENTRY   ILBDDSP1   004708
                                                       *    ENTRY   ILBDDSP2   0047A0
                                                       *    ENTRY   ILBDDSP3   004958

                                                         CSECT      ILBDIML0   004990   004990

                                                         CSECT      IJJCPD1    003FC0   003FC0
                                                            ENTRY   IJJCPD1N   003FC0
                                                       *    ENTRY   IJJCPD3    003FC0
```

Figure 4.9. COBOL Sample Program (Part 14 of 27)

```
// ASSGN SYS008,X'182'
// EXEC

63
67
```

Figure 4.9. COBOL Sample Program (Part 15 of 27)

```
0S03I PROGRAM CHECK INTERRUPTION - HEX LOCATION 0039BC - CONDITION CODE 2 - DATA EXCEPTION
0S00I JOB DTACHK   CANCELED
```

Figure 4.9. COBOL Sample Program (Part 16 of 27)

```
         DTACHK          07/23/71                                                               11.52.27              PAGE   1

GR 0-7   00003850 00003960 00000001 00000001   0000338A 50003C12 00003388 00003550
GR 8-F   000035B8 00003BE2 000032A0 000032A0   00003880 00003688 0000338A 000041B8
FP REG   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000
COMREG   BG ADDR IS 000208

000000   00000000 00000000 00000000 00000000   00000000 00000208 FF050000 00000000   ................   ................
000020   FF050007 40002E06 FF150007 E00039C2   5B5BC2C5 D6D1F440 FF050184 80002E00   .... ...........B   $$BEOJ4 ........
000040   00002F28 0C000000 00001D38 00000000   F75A1C00 01BB0994 00040000 0F0014BA   ................   7...............
000060   00040000 00000336 00040000 0000147A   00000000 00000BBC 00040000 000002D4   ................   ...............M
000080   00000000 00000000 00000000 00000003   00050003 06B006B0 06B041BB 00734570   ................   ................
0000A0   0146940F B47B41A0 C0544570 0B8418A8   41900156 4180B2CE 47F000DA 06B006B0   ................   ..........0.....
0000C0   06B006B0 06B006B0 06B041BB 001741BB   00504570 01464180 01569640 A0019120   ................   .&......... ....
0000E0   A00C4710 00EA9260 A00195E2 A0024780   0DC695C1 A0024780 0DC69561 A0024780   .......-...S....   .F.A.....F./....
000100   010E9104 A0004780 010E9203 008F9281   A0004BA0 0262487A C00049A0 027641AA   ................   ................
000120   C0440778 94F9703B D7017058 70589283   A0009680 A0014400 04080788 947FA001   .....9..P.......   ................
000140   4570B218 07F842B0 00E748B0 02C847F0   BC704570 BC70D205 BEEEBEF5 DC05BEEE   .....8...X...H.0   ......K....5....
000160   C0441BAA DD06BEEE 000C43A1 000742A0   023741AA C0444400 A0045890 A0044220   ................   ................
000180   A0009140 A0014710 BB64D207 01F09008   68009058 68209060 68409068 68609070   ... ......K..0..   .......-. ...-..
0001A0   4BA00262 41AAC000 D2010016 A0009898   90108200 01F04400 A0045890 A0049818   ........K.......   .....0..........
0001C0   9030989D 01F08200 00389284 C0A4D207   01F0BF50 9890BF58 820001F0 9680A000   .....0........K.   .0.&.......0....
0001E0   41100030 47F0B166 96030039 82000038   FF050007 40002E06 00001000 00002000   .....0..........   .... ...........
000200   00003000 80001048 F0F761F2 F361F7F1   32A03000 00000000 00000000 00000000   ........07/23/71   ................
000220   C4E3C1C3 C8D24040 0007AFFF 00004ADB   00004ADB 00000010 0007FFFF F875ECD1   DTACHK  ........   ............8..J
000240   A8A07CD0 00C62171 21782269 226A0000   25102514 25183CF0 F7F2F3F7 F1F2F0F4   ..@..F..........   .......072371204
000260   00002044 0000000C 22E21E4E 1EF41F04   1F140020 214C0010 5B5BC2D6 00130001   .........S...4..   .....<..$$BO....
000280   01001F98 20000000 00000000 02080000   00000294 00000000 000025AC 00000044   ................   ................
0002A0   00001F2C 00000000 00000000 00000000   00000000 00000000 00000000 00000000   ................   ................
0002C0   0000289C 00003228 100020CE 00001DC8   00002A9C 923801C9 909D01F0 4190086C   ...............H   .......I...0....
0002E0   48A00236 4AA00262 9180A000 47100306   58B0A004 9018B030 48B002C8 41CBB000   ................   ...........H....
000300   41DCB000 07F99601 A00048B0 02C841CB   B00041DC B00095FF A00F0789 90E0BF6C   .....9.......H..   ................
000320   48E001C8 D207BF50 E00094FD BF51D213   BF5801F0 07F9909D 01F09220 01C94590   ...HK..&......K.   ...0.9...0...I..
000340   02E04190 01B69500 00234780 03F49526   002347B0 00BE4860 00221A66 487002CA   .............4..   .......-........
000360   48667000 07F6181F 1B664121 000F4570   BCAC4860 BE5C1B22 43201007 4130001F   .....6..........   ...-.*..........
000380   1B234740 03904130 00151B23 47B00392   1A234220 04D94320 100747F0 04584720   ... ............   .....R.....0....
0003A0   00CA4230 04D94820 02364322 C0031A23   950B1007 47F00454 472000C8 96801002   .....R..........   .....0.....H....
0003C0   960C1004 07F91858 41430002 43540000   41455000 1A444A40 BE5495FF 40004770   .....9..........   ..&.... .... ...
0003E0   03CC4284 000007F9 95FF04B1 07891B00   5000BF74 95FF04B1 4780B238 48600236   .......9........   &............-..
000400   95600237 47800366 D502A005 02814770   04204111 00004910 BE3A47B0 04201B66   .-......N.......   ................
000420   4121000F 4570BCAC D5021009 023947B0   00C61B33 43301007 95011006 4770039E   ........N.......   .F..............
000440   92FF04D9 D200044D A00D4123 000BD500   1007A00E 47B000CA D4031002 C0B04182   ...RK.........N.   ........M.......
000460   20004870 02544338 70004930 BE3847B0   03B88930 00034A30 024891F0 30044780   ................   ...........0....
000480   B60ED501 0022BB2C 4780B8E4 D501BE4C   BE5A4720 04AE4930 BE744770 04AE950F   ..N........UN..<   ................
0004A0   00234780 04AE9110 30064780 B2384180   00014148 80001A44 4A40BE54 18584A50   ................   ......... .....&
0004C0   0274D200 04B14000 50104000 92FF4000   42205000 4260500C 92035018 91F03004   ..K... .&. ... .   ..&..-&...&..0..
0004E0   4780B62E 47F004EC 5880BF8C 44000CB8   9560C09C 47700566 D20202CD 10095860   .....0..........   .-......K......-
000500   02CC9507 60004770 0566D202 02CD6001   587002CC 1B444340 30054C40 BE485A40   ....-.....K...-.   ....... ..< ...
000520   02D04144 0000D503 70014000 47800566   9120100C 47100560 91051002 47700560   ......N... .....   .......-.......-
000540   91406004 47800560 94BF6004 91101002   47800558 96401002 96141002 9601100C   . -....-..-.....   ..... ..........
000560   D2034000 700195FF 30024770 03C64280   30029198 30060779 43203000 4322C09D   K. ..........F..   ................
000580   48603000 95003000 47800592 9F006000   07694060 05E29550 30044780 080C9504   .-............-.   .. -.S.&........
0005A0   50184780 08409101 100C4710 05D0940F   06FF91F0 500C4780 05D0D300 C09C3004   &.... ..........   ...0&.....L.....
0005C0   95035018 47D00634 9560C09C 4780067C   D2020049 1009940F 0703D300 0048500C   ..&......-.....@   K.........L...&.
0005E0   9C000184 477005F2 4032C0B4 96803006   07F94730 0BC69106 00454770 0E9C913F   .......2 .......   .9...F..........
000600   00454770 060C91AF 00440789 D201003A   05E29550 30044770 0620D202 00491009   ............K...   .S.&......K.....
000620   58600048 4A60BDDC 50600040 4032C0B4   47F0089A 95015018 4720065C 9560C09C   .-...-..&-.  ...   .0....&....*.-..
000640   47700654 45700B84 9120800F 47100094   47F0065C 91203006 47100094 9560C09C   ................   .0.*.........-..
000660   477005D0 45700B84 4B800262 4878C000   91407038 471005D0 96F006FF 95003003   ................   . ........0.....
```

Figure 4.9. COBOL Sample Program (Part 17 of 27)

```
          DTACHK            07/23/71                                                                                PAGE    2

000680    477005D0 96F0B47B 45700B84 9180800C   471006D4 1B444340 05E24C40 BE404A40    .....0..........  ...M... .S< . .
0006A0    BE3ED202 00491009 58600048 4870BDAC   4070400A 50400048 95065018 477006CC    ..K......-......   . .& ....&.....
0006C0    4870BDAE 4070400A 96F00703 91406004   471006DC 940FB47B 47F005D0 95076000    .... . ..0... -.  .........0....-.
0006E0    4770009A D2074000 60009508 60084770   07445870 60085074 00109208 40104700    ....K. .-...-...  ....-.&..... ...
000700    07704700 08045876 00001B66 41870003   43605000 89600001 4A600254 95FF6001    ................  .-&..-...-....-.
000720    47800098 45400758 D5008000 70004740   0098D500 80007001 47D00804 4540074C    ..... ..N......   ..N.......... .<
000740    47F00728 41706008 47F006F6 95FF7003   47800098 41670002 1B774376 00018970    .0....-..0.6....  ................
000760    00024A70 024C9130 700247A0 074C07F4   4B800262 4878C000 416007AA 48770060    .....<.......<.4  .........-.....-
000780    95025018 478007A0 4720079C 4188C044   9120800F 47100094 47F007A4 41770018    ..&.............  .........0......
0007A0    41770018 41870018 07F6D506 8000800A   47200098 D2020049 10095860 00485876    .........6N.....  ....K......-....
0007C0    0008D204 BE427000 95025018 474007F8   4340BE46 41440001 4240BE46 D504BE42    ..K.......&.. .8  . ........ ..N...
0007E0    80024770 00985876 0018D507 80027000   47700098 47F00804 440007DC 47700098    ..........N.....  .....0..........
000800    47F006D4 940FB47B 47F005D6 95015018   4720081C 91203006 47100094 D200BD38    .0.M.....0.O..&.  .............K...
000820    3005D203 BEDC1008 5870BEDC 95047000   50700048 4170BD38 47800862 47F0085E    ..K.............  &............0..
000840    9501500C 478005D0 9101100C 471005D0   45700B84 D201BD35 80024170 BD285070    ..&.............  ....K.........&.
000860    0048D202 70091009 47F005DA 419001B6   91060045 47700EA2 94FD0039 4170089A    ..K......0......  ................
000880    1B224320 003A4322 C09D4832 C0B4D500   3001003B 078747F0 0B3A4570 0A28D501    ..............N.  ........0......N.
0008A0    0044BD20 47800A42 D5010044 BD224780   0A429102 00444780 08CA9180 30064710    .........N......  ................
0008C0    0C1495FF 30024770 0C149139 00454770   0C084160 08F89180 00440786 96040044    ................  ...-.8..........
0008E0    910A0241 07769500 30040776 9610C08C   9603C084 41900152 91100044 47100910    ................  ................
000900    95000044 0789D602 00410041 47700A42   91040044 47700926 95500044 47700A00    ......O.........  ..........&......
000920    94773006 07F99477 30069104 30064780   09E8D601 10040044 94733006 9180100C    .....9..........  .YO.............
000940    47100948 92003003 96801002 9101100C   4710096A 45700B84 198A47D0 09624190    ................  ................
000960    01529601 800047F0 09A894FE 100C91F0   30044770 09844370 BF900670 4270BF90    .......0.......0  ................
000980    47F009A8 944B1002 440004F8 586002CC   96406004 91401002 478009A0 96101002    .0.........8.-..  . -.. ..........
0009A0    94BF1002 47F00A24 92FF500C 43703002   D2003002 4000D200 400004B1 427004B1    .....0....&.....  K... .K. .......
0009C0    91103006 478009E8 4870BE5A 49307008   477009D8 9602700A 4B70C0AE 4970BE4C    .......Y........  ...Q...........<
0009E0    474009E8 47F009CC 95003000 47700A00   95FF3002 07899198 30064780 0A2407F9    . .Y.0..........  ...............9
000A00    4832C0B4 D2000A11 003A0570 41330008   95013000 47200B66 95FF3002 07879198    ....K...........  ................
000A20    30060777 41700580 1B554350 30024145   50001A44 4A40BE54 4A500274 58104000    ...........&....  &.... ...&.... .
000A40    07F795FF 30024780 09E8D201 10000046   D202100D 0041D601 10040044 95035018    .7.......YK.....  K.....O........&.
000A60    47200AC6 D300C09C 30049560 C09C4770   0AC69101 100C4710 0AC64570 0B844B80    ...FL......-....  .F.......F......
000A80    02624878 C0009140 70384710 0AC64560   077C4160 8006D500 80068010 47700ABA    ....... .....F.-  .@.-..N.........
000AA0    92018006 0660D500 80058011 47700ABA   D2008005 80124160 80034370 60004170    .....-N.........  K......-.....-...
000AC0    70014270 60009501 50184720 0B185870   00404B70 BDDCD502 70010239 47200B18    ....-...&.......  . ....N.........
000AE0    58770000 D5017000 BEE04780 0B10D501   7000C096 47700B18 9560C09C 47700B0C    ....N.........N.  ..........-......
000B00    45700B84 9620800F 47F00B10 96203006   96401002 96011004 91270044 47700938    ..........0......  . ..............
000B20    96803006 91041002 4780093C 96043006   95003000 47700A00 07F9D200 0B53003B    ................  ..........9K.....
000B40    D2000B5D 003A4832 C0B695FF 30000789   05609582 30010787 41330008 95013000    K...............  .-..............
000B60    07D647F0 0A004842 C0B44832 C0B64570   0B7A4133 00081934 072995FF 30020787    .O.0............  ................
000B80    47F00A1E 48800276 91F0500C 47800B9A   4380500C 4188C044 07F79180 100C4710    .0........0&.....  ..&.......7......
000BA0    0BAA910F 500C4780 0B944180 006047F0   0B940700 07000700 00E201E2 D2010000    ....&.........-.0  ..........S.SK...
000BC0    0BB88200 0BB80729 D2010044 BEE2D202   0041BDD9 9560C09C 47700C08 9101100C    .........K....SK.  ...R.-..........
000BE0    47800C08 440004F8 588002CC 95078000   47700C08 944B1002 91401002 47800C04    .......8........  ......... ......
000C00    96101002 94BF1002 92040C83 91090045   47800C42 92040C83 58800040 12884770    ................  ........... ....
000C20    0CF49212 0C839104 30064780 0C5294FD   0C839120 100C4710 0C429101 10024710    .4..............  ................
000C40    0CEC95FF 30024780 0C529110 100C4710   0CEC4860 BE5A4960 BE5847B0 0CD84A60    ................  ...-...-.....Q.-
000C60    C0AE4060 BE5AD715 60006000 D2076000   00404030 600895FF 30024770 0C829602    .. -..P.-.-.K.-.  . .-..........
000C80    0C839204 600A9690 30064186 000C4080   BD4A4140 0D4A9102 00444710 0CA49130    ....-.......... .  ... ...........
000CA0    00450774 91020C83 47800D0E 4180BD48   45700E54 47F00D26 D300C09C 30049540    ................  .....0..L......
000CC0    C09C4780 0DD29560 30044780 B73E9562   30044780 B73E07F4 91020C83 471008D2    .....K.-........  ........4.......K
000CE0    9130500C 47800DBC 47F00096 96201002   47F008D2 91011003 47800C32 4B80BDDC    ..&.......0......  .0.K............
000D00    50800048 D2021009 004947F0 0C329120   100C4780 0CAC4181 00104570 0E5447F0    &...K......0....  ...............0
000D20    0D2647F0 0DF0940E 00449506 00444780   0E549220 600B4180 BD485080 00489C00    ...0.0..........  ....-......&.....
000D40    20009D00 20004760 0D424580 0E449102   600A4710 0DA69502 BE5D4770 0D6E9502    .......-........  -...............
000D60    500C4770 0D6E9520 600D4770 0DB295FF   02414780 0DB29560 C09C4770 0DA69101    &.......-.......  ........-.......
000D80    100C4780 0DA69101 600C4710 0DF09130   00454780 0DA694FE 100C4400 04F85860    ........-....0..  ..............8.-
000DA0    02CC9640 60044190 01529287 C09447F0   09E8D205 0005600C 97060DC7 97010DC7    ... -..........0  .YK...-....G...G
```

Figure 4.9. COBOL Sample Program (Part 18 of 27)

```
          DTACHK          07/23/71                                                                        PAGE   3

000DC0   D2010002 30009204 000092E6 00018200   0BB89541 30044770 0DDE94F3 600C9501   K..........W....   ...........3-...
000DE0   600C0774 91020C83 47100DF0 96021003   D2070040 60004580 0E1E47F0 08D24400   -..........0....   K.. -......0.K..
000E00   0DF04570 0E5C47F0 0E0A4580 0E1E47F0   08B29104 600A4780 0DF04180 09E847F0   .0...*.0.......0   ....-....0...Y.0
000E20   0E3496F0 0E1F4830 BE74D257 BE6CBE82   4400B2C8 4870BE5A 4B70C0AE 4070BE5A   ...0......K.....   ...H........ ...
000E40   946F3006 1B224320 003A4322 C09D4570   0A2807F8 18284320 02360922 50800048   ................   ...8........&...
000E60   42200048 4820003A 9C002000 47400E7C   9D002000 47200E70 47100D32 91060045   ............. .@   ................
000E80   47700E9E 91390045 07779102 00440717   91080044 47170004 47F00E68 18264020   ................   .........0.... .
000EA0   003AD201 00000BBA 920F0073 82000BB8   D501BE4C BE5A4720 0ED092C4 BE6B48F0   ..K.............   N..<.......D.,.0
000EC0   BE749550 F0044780 0ECE92C1 BE6B0A03   9284C094 47F00156 48F0C0AC 8000BE39   ...&0......A.,..   .....0...0......
000EE0   9118F006 077E95FF F0024770 0EE007FE   48600236 4570BC70 9287A000 95000093   ..0.....0.......   .-..............
000F00   4780B25C 94BFA001 9202BE5D 92600237   4260B101 41201007 4570BCAC 5040B120   ...*.........-..   .-..........& ..
000F20   4890BDFE D2079000 10001B44 D5039000   02784780 0F6ED501 00309000 47800F72   ....K.......N...   ......N.........
000F40   955B9000 47800F6A D5029000 BE304780   0F669510 B10147D0 0F625840 BE3447F0   .$......N.......   ........... ...0
000F60   0F764144 01004144 01004144 01004144   01004144 05015040 BE24D207 BDE0BDF0   ................   ......& ..K....0
000F80   4580B08A 18E94180 9168955C E00841F0   01564780 0FAED507 9000E008 47800FBC   .....Z.....*...0   ......N.........
000FA0   41E0E014 15E84740 0F8A47F0 0F80D503   B120B124 477000C0 47F00FCE D503B120   .....Y. ...0..N.   .........0..N...
000FC0   B1244770 0FD41840 D2024001 E0109200   BE5D07FF 5880E014 88800008 5870E010   .....M. K. .....   ................
000FE0   88700008 12004780 0FF41810 1B871A08   47F00FF8 18081817 D202BE25 E017D200   .........4......   .0.8....K.....K.
001000   BE24BE25 9200BE25 D201BDE6 BD961B99   4390E013 D201BD94 E01A4880 BD964690   ........K..W....   ....K...........
001020   B02A4880 BD944080 BDE61821 4570BCA8   4880BDE6 06201A28 4570BCA8 5010BDE0   ...... ..W......   ...W........&...
001040   9286BDE0 4580B08A 5810BDE0 41101000   4A10BDE6 12994720 B01A9500 B1014780   ................   ...W............
001060   B07C4190 C0004A90 B1004899 00000610   49190008 4740B07C 50190024 5890A004   .@..............   ..... .@&.......
001080   5840B120 9200BE5D 07F45080 020445E0   0ED89190 F0064710 B0A29180 F0064710   . .......4&.....   .Q..0.......0...
0010A0   0EDC95FF 04B14780 B0A245E0 B10441F0   BE0092F0 04930A00 92200493 9180BE02   ...............0   ...0............
0010C0   4710B0C6 0A0748F0 C0AC95FF F0024770   B0D645E0 B1045880 02049203 00939120   ...F...0....0...   .O..............
0010E0   BE034780 B0F84370 BE244177 00014270   027DD201 027FBD56 D204BE23 027C4160   .....8..........   ..K.....K....@.-
001100   000007F8 80000058 48F0C0AC D704F002   BE1CD704 BE1CF002 4400B10C 07FE0000   ...8.....0..P.0.   ..P...0.........
001120   800011E4 00000000 4570BC70 1B004540   0EF05860 902C1266 4780B13E 18064170   ...U...........    .0.-............
001140   01565860 BF141266 4750B152 186947F0   B15A5860 BF184170 01525000 600CD300   ...-.....&.....0   ...-......&.-.L.
001160   6009B101 07F74400 04084780 B1AA4111   00005510 023847B0 00C6D507 1000B18C   -....7..........   .........FN.....
001180   4770B194 9604A00F 47F0B194 5B5BC2C4   E4D4D740 95000093 47800EF4 95000277   .........0..$$BD   UMP .......4....
0011A0   4770B22A D200B1F5 02379500 00934780   0EF44570 BC7048F0 C09290F1 BFC0180F   ....K..5........   .4.....0...1....
0011C0   41FF0008 50F0BFA4 1B664111 00005510   023847B0 00C6D502 10000030 477000BE   ....&0..........   .....FN.........
0011E0   45400EF4 9140A001 47100152 95000277   47700156 92100277 47F0B210 44000408   . .4. ..........   .........0......
001200   477000BE 9680A000 92000277 92FFBFA0   9703008D 41700156 D702C08D A005D702   ................   ........P.....P.
001220   A005C08D 4400B218 07F7D500 02370277   478000BE 94FDA000 58800024 06800680   .........7N.....   ................
001240   50800024 9502BE5D 47700152 9180A000   47800152 5080BF54 47F00152 9140A001   &...............   ....&....0... ..
001260   47100156 D501A004 B27E4780 01565880   00240680 06805890 A0045080 900C47F0   ....N...........   ..........&....0
001280   01569101 00214710 00BE9203 00939501   BE6B4720 B2DE9284 C0A4078F 92D9BE6A   ................   .,...........R..
0012A0   41900156 4160BE6C 940F0E1F 07FF9200   013358A0 01F44580 00EA41A0 C0A49280   .....-..........   .....4..........
0012C0   013307F9 9601C0A4 D201003A 30004580   0E4447F0 093892F1 BE6B4570 BC704110   ...9....K.......   ...0...1.,......
0012E0   BE644800 BDFE1B66 45400EF4 18B09200   009347FB 00084860 02364570 BC704400   ......... .4....   .......-........
001300   04084770 B3081B66 45400EF4 50009030   47F00156 58800024 4B80BDDE 50800024   ......... .4&...   .0..........&...
001320   95600237 4770B33E 5080BF54 9180F002   07199284 A0009200 009347F0 01569180   .-......&.....0.   ...........0....
001340   10020719 94FEA000 47F00152 D2030024   BFC44570 BC7058E0 BFA45880 C08C9001   .........0..K...   .D..............
001360   802C5800 800C50E0 800C4570 BC7047F0   B21492D4 B38247F0 B37E92D6 B3825860   ......&........0   ...M...0...O...-
001380   0014D600 60391000 07F991F0 00214770   00BE1200 47A0B39E D3000021 02379706   ..O.-....9.0....   ........L.......
0013A0   A0009703 008F4200 002007F9 91F00021   477000BE 4570BC70 1B444400 04084770   ...........9.0..   ................
0013C0   0EF01864 47F00EF4 4570B424 18105870   A0045000 70304121 00034570 BCA4947F   .0...0.4........   ..&.............
0013E0   10021321 9012BF14 58200050 50300050   5520BEE4 47D0B40C 92F0B4E3 13721B22   ...........&&..&   ...U.....0.T....
001400   88700008 5E700054 50700054 88300008   88200008 5E300054 1F325030 005407F9   ........&.......   ..........&....9
001420   4170B3E8 1831D500 02370273 078747F0   00BE91F0 00214770 00BE4111 00005510   ...Y..N........0   ...0............
001440   023847B0 00C64570 BCA407F9 58610000   95001000 4770B468 91806002 4710B470   .....F.....9./..   ..........-.....
001460   41110004 47F0B44C 94FEA000 47F0B238   5810A004 50601030 07F94700 06D4909D   .....0.<.....0..   ....&-...9...M..
001480   01F09228 01C94590 02E09530 02374720   00BC9500 02374780 00BC4140 BEF44570   .0...I..........   ........... .4..
0014A0   B5FC4880 02364188 C0444570 B55647F0   00BC47F0 00BC47F0 00BC9218 01C9909D   ...............0   ...0...0.....I..
0014C0   01F04590 02E094FD 00199140 001B4780   B4DA9601 C0849601 C08C9180 001B4780   .0......... ....   ................
0014E0   01524700 B4FE9856 0050D203 0050BEE4   8A500008 1B655E60 BEE85060 00549200   .........&K..&.U   .&.....-.Y&-....
```

Figure 4.9. COBOL Sample Program (Part 19 of 27)

```
           DTACHK          07/23/71                                                                                     PAGE    4

 001500   B4E3940F BD014830 02724183 C0449510   800147B0 01529856 BF141266 4740B53C   .T..............  ............. ..
 001520   4140BF14 4570B556 47F00152 9607C084   9601C084 96F0BD01 47F00152 95808000   . .......0......  .....0...0......
 001540   4780B54C 96805002 96018000 D707BF14   BF1447F0 01521255 07D79601 80004930   ...<..&.....P...  ...0.....P......
 001560   02764787 00089528 01C94780 B5809502   BE5D4770 B580D500 B1010273 47870004   .........I......  ......N.........
 001580   4570BC70 5893C048 D2076000 9008D223   6008902C D21B602C 90105050 900C1135   ........K.-...K.  -...K.-...&&....
 0015A0   47F0B5C6 4570B426 1B444570 B5F44570   BC70D203 900C6004 D223902C 6008D21B   .0.F.........4..  ..K...-.K...-.K.
 0015C0   9010602C 10355034 000047F0 01524570   B4264140 00014570 B5F41330 4780B5C6   ..-...&....0....  ... .....4.....F
 0015E0   12554740 B5C64121 00474570 BCA49001   400007F9 4A400022 4840425B 48300236   ... .F..........   ..9. ... .$....
 001600   18538850 00011A45 98564000 07F79056   BF944550 B6E69856 BF9447F0 04829856   ...&...... ..7..  ...&.W.....0....
 001620   BF949503 BF904780 B23847F0 04829056   BF944550 B6E69856 BF9447F0 04E85850   ...........0....  ...&.W.....0.Y.&
 001640   BF945870 BF845970 BF7C4740 B6525870   BF781887 4A80BF88 5080BF84 4380BF90   .........@. ....  ........&.......
 001660   41880001 4280BF90 D20F7008 10004180   70004080 70129200 70119601 70144180   .........K......  .. .............
 001680   70085080 400092FF 4000D207 70006000   43860007 06804280 B6B34188 00044280   ..&. ... .K...-.  ................
 0016A0   70075886 000095FF B6B34770 B6B29200   B6B3D211 70188000 41807015 40807002   ................  ..K......... ...
 0016C0   92007001 45700B84 5870BF84 4B70BF88   D2017015 80029240 70179680 10024110   ................  K...... ........
 0016E0   700847F0 04E8D202 BF8D1009 5860BF8C   5080BF8C 18764A70 BDDC5970 02380725   ...0.YK......-..  &...............
 001700   95096000 4780B70E 95016000 077591C0   60040775 91151002 07759120 100C0775   ..-........-....  -...............
 001720   D5016006 BF910725 58760000 41770000   4A760006 59700238 072547F5 00084370   N.-.............  ...........5....
 001740   30034177 00014270 30039220 BD5C9102   600A4710 B76AD202 BE191009 5850BE18   .............*..  -.....K......&..
 001760   58550000 D2036012 50024170 B82C9108   60050717 9110600C 07179108 600D4710   ....K.-.&.......  -.....-.....-...
 001780   B7C69101 600C4710 B7F09140 600C0717   9108600C 4710B80C 4170B822 9124600C   .F..-....0. -...  ..-...........-.
 0017A0   07779102 600D0717 9180600C 07149102   600C4710 B8369160 600D4770 B8CE9101   ....-.....-.....  -......--.......
 0017C0   60050717 07F49102 600D4710 B7F64550   B8B2D501 BE186012 4770B7F6 9680600A   -....4..-....6.&  ..N...-....6..-.
 0017E0   91041003 4710B822 96081003 47F00DF0   9180600C 0714950A 30030724 4180BD50   .............0.0  ..-............&
 001800   45700E54 47F00D32 47F00E12 4550B8B6   95FF3003 0784910F 30034780 B7FC47F0   .....0...0...&..  ...............0
 001820   0E12950A 30030724 47F00E12 95013003   072447F0 0E124400 0DF05870 00404B70   .........0......  ...0.....0... ..
 001840   BDDC5070 BD909208 BD904550 B8B29101   BE104780 B87A9562 30044780 B8669509   ..&........&....  ................
 001860   BE1B47B0 B8CA9513 BE1B47B0 B8CA5870   BE184177 00015070 BE184570 0A2891F0   ................  ......&........0
 001880   500C4780 B8A29210 BD859506 50184770   B8A64570 0B849240 BD859180 800C4780   &...........&...  ....... ........
 0018A0   B8A69200 BD854180 BD684320 600047F0   0DFE9240 BD5C4180 BD584570 0E5447F0   ............-..0  ... .*.........0
 0018C0   0D32D203 6012BE11 07F59620 1003D600   1003600D 47F00DF0 4140B8F4 92FF0022   ..K.-....5....O.  ..-..0.0. .4....
 0018E0   47F003F4 92000022 D300BB2E 30049560   BB2E07F4 4770048C D202BB25 10095840   .0.4....L......-  ...4....K......
 001900   BB249507 40004770 048C5844 00001B55   43503005 4C50BE48 5A5002D0 95FF5004   .... ...........  .&..<&...&....&.
 001920   4780B958 D200BB29 50044870 BB284550   BAF647F0 B98ED500 700A0237 4770B9B2   ....K...&......&  .6.0..N.........
 001940   910F700B 471000BE 4340700B 41440001   4240700B 47F0048C 95FFB963 4780B9B6   ......... ......  . ...0..........
 001960   41700000 42750004 4C70BB2A 5A700298   D200B963 70005010 700092FF 7000D205   .........<......  K.....&.......K.
 001980   70044000 D200700A 023747F0 048C95FF   B9634780 B9B6D200 7000B963 4870B962   .. .K......0....  ......K.........
 0019A0   4C70BB2A 5A700298 D200B963 700047F0   B9769680 700B94FE A00092FF 002247F0   <.......K......0  ...............0
 0019C0   B2389224 BB2DD203 BB241002 4540B8DC   D2031002 BB249223 BB2D4770 00BE9180   ......K...... ..  K...............
 0019E0   BB2F4780 B9EE4140 BB3047F0 B9FCD202   BB251009 5840BB24 58440000 1B771B22   ....... ...0..K.  ..... ..........
 001A00   43203005 4C20BE48 5A2002D0 95FF2004   478000BE 43720004 9640BB2F 4550BAF6   ....<...........  ......... ...&.6
 001A20   47F000BE D500700A 02374770 00BE910F   700B4780 BA424360 700B0660 4260700B   .0..N...........  .......-...-.-..
 001A40   07F99140 BB2F4710 BA7C4340 B963D200   B963BB29 5860BB24 D2006000 70004240   .9. .....@. ..K.  .....-..K.-....
 001A60   70009180 700B9200 700A9200 700B4710   BAB095FF 70004780 BAB007F9 95FF7000   ................  ...........9....
 001A80   4780BA9C 43407000 D2007000 B963D200   B9632004 42420004 47F0BA62 D2007000   ..... ..K.....K.  .........0..K...
 001AA0   B963D200 B9632004 92FF2004 47F0BA62   4130BEF7 41330001 95003000 07891B44   ..K..........0..  ...7............
 001AC0   43430000 4A400262 95804000 4780BAB4   91014000 4710BAB4 58504004 D501500A   ..... .... .....  .. ......& .N.&.
 001AE0   BB2C4770 BAB49200 500A9601 40004190   015247F0 BAB44C70 BB2A5A70 0298D505   ........&... ...  ...0..<.......N.
 001B00   40007004 47850004 94BFBB2F 95FF7000   0785D200 BB297000 5070BB24 4870BB28    ...............  ..K.....&.......
 001B20   47F0BAF6 00000000 0000000C FF230000   00000000 00001831 12004780 BB4E4121   .0.6............  ................
 001B40   00474570 BCA41810 18204570 BCA44840   02368840 00015A40 02A05004 00005034   ...............   ... ... ..&...&.
 001B60   000407F9 9680A000 9101A00F 4710BC28   9550A001 478001DC 44000408 4770BB88   ...9............  .&..............
 001B80   5840C08C 47F0BB94 9563A001 478001DC   584A0004 48200236 88200001 5A2002A0   . ...0..........  ................
 001BA0   58320000 12334780 01DC5852 0004D207   50004008 D2235008 402CD21B 502C4010   ..............K.  &. .K.&. .K.&. .
 001BC0   D202400D 2001D703 20002000 D703402C   402CD200 402FA001 943F402F D2034030   K. ...P.....P. .   .K. ..... .K. .
 001BE0   2004D200 2004A001 9200A001 9601A00F   48600236 88600001 4A60026C D7076000   ..K.............  .-...-...-..P.-.
 001C00   60004860 02368860 00014A60 0270D707   60006000 D5010272 02364770 BC24D707   -..-...-...-..P.  -.-.N.........P.
 001C20   BF14BF14 47F00152 94FEA00F 48200236   88200001 5A2002A0 D200A001 20049640   .....0..........  ........K......
```

Figure 4.9. COBOL Sample Program (Part 20 of 27)

```
         DTACHK           07/23/71                                                                                  PAGE    5

001C40   A0015832 00044400 04084770 BC565840   C08C47F0 BC5A584A 0004D207 40083000     ...............    ...0......K. ...
001C60   D21B4010 302CD223 402C3008 47F001DC   9180A000 07175890 A0044880 01C890E0     K. ...K. ....0..   .............H..
001C80   90246000 90586020 90606040 90686060   9070D213 901001F0 D2079008 80009680     ..-...-..-- ..--   ..K....0K.......
001CA0   A00007F7 48600236 41220000 59200238   472000C6 12660787 41110000 59100238     ...7.-..........   ...F............
001CC0   472000C6 4910BE3A 474000C6 1B881851   4350C0B2 09851968 477000C6 18524350     ...F..... .F....   .&.........F...&
001CE0   C0B20985 19684770 00C607F7 9510A001   47B0017E 92E9BE6B 9110A008 4710B2DE     .........F.7....   .....Z.,........
001D00   4700B502 4110BE28 9283A000 9103A008   4770B166 9280A000 9200A008 47F00156     ................   .............0..
001D20   0C000800 00000000 01001D35 60000003   08002DF0 60C2C740 C3001D38 60000001     ............-...   ...0-BG C...-...
001D40   08002F20 60000001 04001E78 20000006   13001E10 20000001 1A001E10 20000005     ....-...........   ................
001D60   16001E18 20000004 07001E16 40000006   39001E18 60000004 08001D70 00000000     ............ ...   ....-...........
001D80   1F001D85 60000001 1A001E10 50000005   08001D90 03D806C0 07003B51 60000006     ....-.......&...   .....Q......-...
001DA0   1F001E3C 40000001 08003B38 1E3C1E3D   07001DB0 40000006 1F001E3C 40000001     .... ...........   .... ....... ...
001DC0   08001DC0 00000000 07001E21 40000006   31001E23 40000005 08001DD0 0008000A     ............ ...   .... ...........
001DE0   86002B50 600003D8 9200027C 20000005   86003008 60000168 00000000 1E003000     ...&-..Q...@....   ....-...........
001E00   00008000 0C000006 00001DC8 00001DF0   00000400 00000000 00001DC8 FF006000     ...........H...0   ...........H.-..
001E20   00000000 13000101 5B5BC2C1 E3E3D5C1   C6C7D700 01000001 00FE32A0 10401D80     ........$$BATTNA   FGP.......... ..
001E40   00180000 00000000 00050000 1E6C0E12   0DF00DFE 211C12AE 1EC41E56 000012B2     ................   .0.......D......
001E60   12C40156 5B5BC1D5 C5D9D900 00001DF0   0E000005 22100400 002000C8 00000000     .D..$$ANERR....0   ...........H....
001E80   00090000 00000000 00000000 00000000   00000000 00000000 00000000 00000000     ................   ................
001EA0   00000000 --SAME--                                                             ....
001EC0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00002F20     ................   ................
001EE0   615C44F0 FF000000 00FF0000 01008484   80808083 85605040 30201000 00000000     /*.0............   .....-& ........
001F00   00000000 --SAME--                                                             ....
001F40   00000000 00000000 00000000 00000000   00040007 400010BC 00000000 00002054     ................   .... ...........
001F60   00001000 00002000 00003000 900010AE   00001E00 00002B50 000023D8 0000250F     ................   .......&...Q....
001F80   00030000 000024A8 00680000 00000001   00005000 0000214D 00000000 00000000     ................   ..&.............
001FA0   FF050007 40002E06 00003228 00002054   00000208 00000001 80002C4C 90002CFE     .... ...........   ...........<....
001FC0   00002B50 00000007 00002F08 00000020   00002BD8 00000008 00001FA0 00002B6A     ...&............   ...Q............
001FE0   00000005 000002E4 00000000 00000000   00000000 00000000 00000000 00000000     .......U........   ................
002000   00000000 00000000 00000000 00000000   02080000 00000000 00000000 00000000     ................   ................
002020   2720001F 00000000 00000000 00000000   27B00034 00000000 00000000 00000000     ................   ................
002040   02080000 8500E2D7 47F001E8 00000000   000000FF 0310C2C7 19001F98 0A0032A0     ......SP.0.Y....   ......BG........
002060   240B1480 8000C6F2 1907B000 0507B000   012A0A00 8000C6F1 1907D800 0507D800     ......F2........   ......F1..Q...Q.
002080   023F0A00 8000C1D9 47F0BCEC 00003228   07002B50 84006150 47F00EB0 50000408     ......AR.0......   ...&../&.0..&...
0020A0   0C101418 8400E2D7 47F001CA 22100016   1F050000 21782178 220821A8 22382238     ......SP.0......   ................
0020C0   22682268 22682268 22682268 22682268   11281166 128212F6 12D600C2 1314135A     ................   .......6.O.B....
0020E0   134C1420 11FC1372 137A00CE 03E815D2   15A815CE 15A415D2 15A8138A 13AC13C8     .<...........Y.K   .......K.......H
002100   00BE1432 00BE00BE 144C00BE 00BE00BE   01B601B6 18D819C2 1B360000 02004478     .........<......   .....Q.B........
002120   00002F08 030045F8 04000000 05000000   06000000 07000000 08000000 09000000     .......8........   ................
002140   0A000000 0B000000 FF000000 03030500   00000000 00000000 FFFFFFFF FFFFFFFF     ................   ................
002160   FFFFFFFF 030305FF FFFFFFFF FFFFFFFF   00000618 FFFFFFFF 0009FF00 000000F8     ................   ...............8
002180   000CFF00 110000FC 000DFF00 210000FC   000EFF00 420000FC 000FFF00 420000F8     ................   ...............8
0021A0   001FFF00 000000F8 0130FF00 620000F8   0131FF00 620100F8 0132FF00 620200F8     .......8.......8   .......8.......8
0021C0   0133FF00 620300F8 0134FF00 620400F8   0135FF00 620500F8 0136FF00 620600F8     .......8.......8   .......8.......8
0021E0   0137FF00 620700F8 0180FF00 50930190   0181FF00 50C300C0 0182FF00 50C300C4     .......8....&...   ....&C......&C.D
002200   0183FF00 50C300C0 0184FF00 50C300C4   0190FF00 600800FC 0191FF00 600900FC     ....&C......&C.D   ....-.......-...
002220   0192FF00 600A00FC 0193FF00 600B00F8   0194FF00 600C00F8 0281FF00 50C300C0     ....-.......-..8   ....-..8....&C..
002240   0282FF00 50C300C0 0283FF00 50C300C0   FF000000 00000000 FF000000 00000000     ....&C......&C..   ................
002260   FF000000 00000000 FF020000 0005FFFF   80FF0000 00030000 00000265 28FF0000     ................   ................
002280   00060000 00070000 00080000 00090000   000A0000 000B0000 000C0000 000D0000     ................   ................
0022A0   000E0000 000F0000 00100000 00110000   00120000 00130000 00140000 00150000     ................   ................
0022C0   00160000 00170000 00180000 00190000   001A0000 001B0000 001C0000 001D0000     ................   ................
0022E0   00FF0000 00000000 00000000 00000000   00000000 00000000 00000000 00000000     ................   ................
002300   00000050 00000000 00001900 00000000   00000000 00000000 00000051 00000000     ...&............   ................
002320   00001900 000001F4 00000000 00000000   00000079 00000000 00001300 000007D0     .......4........   ................
002340   00000000 00000000 00000050 00000000   00001900 00000000 00000000 00000000     ...........&....   ................
002360   00000051 00000000 00001900 000001F4   00000000 00000000 00000079 00000000     ...............4   ................
002380   00001300 000007D0 00000000 00000000   00000050 00000000 00001900 00000000     ................   ...&............
```

Figure 4.9. COBOL Sample Program (Part 21 of 27)

```
         DTACHK           07/23/71                                                                                    PAGE    6

0023A0   00000000 00000000 00000051 00000000    00001900 000001F4 00000000 00000000    ................  .......4........
0023C0   00000079 00000000 00001300 000007D0    00000000 00000000 090023ED 20000023    ................  ................
0023E0   00008000 08000004 000023D8 000023E0    F1C9F0F0 C14040D9 C5C1C4E8 40C6D6D9    ...........Q....  1I00A  READY FOR
002400   40C3D6D4 D4E4D5C9 C3C1E3C9 D6D5E24B    00000000 00000000 00000000 00000000     COMMUNICATIONS.  ................
002420   00000000 --SAME--                                                             ....
002440   09002455 20000015 00008000 08000004    00002440 00002448 F1C3F0F0 C14040C1    ................  ... ....1C00A  A
002460   E3E3D54B 40F040F0 C34B0000 00000000    00000000 00000000 00000000 00000000    TTN. 0 0C.......  ................
002480   00000000 --SAME--                                                             ....
002500   00000000 00000000 00000000 00000000    000B2A3F 0B140A0A 01FF01FF 02FF12FF    ................  ................
002520   00FF1504 13FFFFFF FFFFFFFF FFFFFFFF    14FF15FF 14FF15FF FFFF03FF FFFF1001    ................  ................
002540   FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF    FFFFFFFF FFFFFFFF FFFFFFFF FFFF00FF    ................  ................
002560   FFFF13FF FFFFFFFF FFFFFFFF FFFFFFFF    FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF    ................  ................
002580   FFFFFFFF FFFFFFFF 00FFFFFF 13FFFFFF    FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF    ................  ................
0025A0   FFFFFFFF FFFFFFFF FFFF0014 01000000    00000000 00000000 02000000 00000000    ................  ................
0025C0   00000000 03000000 00000000 00000000    04000000 00000000 00000000 05000000    ................  ................
0025E0   00000000 00000000 06000000 00000000    00000000 07000000 00000000 00000000    ................  ................
002600   08000000 00000000 00000000 09000000    00000000 00000000 0A000000 00000000    ................  ................
002620   00000000 0B000000 00000000 00000000    0C000000 00000000 00000000 0D000000    ................  ................
002640   00000000 00000000 0E000000 00000000    00000000 0F000000 00000000 00000000    ................  ................
002660   10000000 00000000 00000000 11000000    00000000 00000000 12000000 00000000    ................  ................
002680   00000000 13000000 00000000 00000000    FF000000 00000000 00000000 00000000    ................  ................
0026A0   00000000 --SAME--                                                             ....
002720   F0F761F2 F361F7F1 32A03000 00000000    00000000 00000000 D5D640D5 C1D4C540    07/23/71........  ........NO NAME
002740   0007D7FF 00003F10 00003F10 00000020    0007FFFF F875ECD1 8000EC50 00002171    ..P.............  ....8..J...&....
002760   21782269 226A0000 25102514 25183CF0    F7F2F3F7 F1F2F0F4 00002044 0000000C    ...............0  72371204........
002780   232A1E4E 1EF41F04 1F140020 214C0020    00000000 00000000 00000000 20000000    .....4.......<..  ................
0027A0   00000000 02080000 0000283C 00000000    F0F761F2 F361F7F1 32A03000 00000000    ................  07/23/71........
0027C0   00000000 00000000 D5D640D5 C1D4C540    0007FFFF 00003F10 00003F10 00000030    ........NO NAME   ................
0027E0   0007FFFF F875ECD1 8000EC50 00002171    21782269 226A0000 25102514 25183CF0    ....8..J...&....  ...............0
002800   F7F2F3F7 F1F2F0F4 00002044 0000000C    23721E4E 1EF41F04 1F140020 214C0030    72371204........  .....4.......<..
002820   00000000 00000000 00000000 20000000    00000000 02080000 0000286C 00000000    ................  ................
002840   000025AC 00000044 00001F2C 00000000    00000000 00000000 00000000 00000000    ................  ................
002860   00000000 00000000 0000289C 00000000    000025AC 00000044 00001F2C 00000000    ................  ................
002880   00000000 00000000 00000000 00000000    00000000 00000000 0000289C 00010203    ................  ................
0028A0   372D2E2F 1605250B 0C0D0E0F 10111213    3C3D3226 18193F27 1C1D1E1F 404F7F7B    ................  ............ ...
0028C0   5B6C507D 4D5D5C4E 6B604B61 F0F1F2F3    F4F5F6F7 F8F97A5E 4C7E6E6F 7CC1C2C3    $.&...*.,-./0123  456789..<...@ABC
0028E0   C4C5C6C7 C8C9D1D2 D3D4D5D6 D7D8D9E2    E3E4E5E6 E7E8E94A E05A5F6D 79818283    DEFGHIJKLMNOPQRS  TUVWXYZ.........
002900   84858687 88899192 93949596 979899A2    A3A4A5A6 A7A8A9C0 6AD0A107 3F3F3F3F    ................  ................
002920   3F3F3F3F --SAME--                                                             ....
002980   3F3F3F3F 3F3F3F3F 3F3F3F3F 3F3F3F3F    3F3F3F3F 3F3F3F3F 3F3F3F3F 00010203    ................  ................
0029A0   1A091A7F 1A1A1A0B 0C0D0E0F 10111213    1A1A081A 18191A1A 1C1D1E1F 1A1A1A1A    ................  ................
0029C0   1A0A171B 1A1A1A1A 1A050607 1A1A161A    1A1A1A04 1A1A1A1A 14151A1A 201A1A1A    ................  ................
0029E0   1A1A1A1A 1A1A5B2E 3C282B21 261A1A1A    1A1A1A1A 1A1A5D24 2A293B5E 2D2F1A1A    ......$.........  ................
002A00   1A1A1A1A 1A1A7C2C 255F3E3F 1A1A1A1A    1A1A1A1A 1A603A23 40273D22 1A616263    ......@.........  .....-.. ..../..
002A20   64656667 68691A1A 1A1A1A1A 1A6A6B6C    6D6E6F70 71721A1A 1A1A1A1A 1A7E7374    ..............,.  ................
002A40   75767778 797A1A1A 1A1A1A1A 1A1A1A1A    1A1A1A1A 1A1A1A1A 1A1A1A1A 7B414243    ................  ................
002A60   44454647 48491A1A 1A1A1A1A 7D4A4B4C    4D4E4F50 51521A1A 1A1A1A1A 5C1A5354    ...............<  ...&........*...
002A80   55565758 595A1A1A 1A1A1A1A 30313233    34353637 38391A1A 1A1A1A1A 00001300    ................  ................
002AA0   FF000000 00FF0000 0000FF00 000000FF    00000000 FF000000 00FF0000 0000FF00    ................  ................
002AC0   000000FF 00000000 FF000000 00FF0000    0000FF00 000000FF 00000000 FFFF0000    ................  ................
002AE0   00000000 --SAME--                                                             ....
002B40   00000000 00000000 00000000 00000000                                           ................

LBLTYP   HEX LENGTH IS 0000
--BG--
003220                     D5D640D5 C1D4C540    FF150007 E00039C2 00003BE2 000032A0            NO NAME   .......B...S....
003240   000032A0 00003880 00003688 0000338A    000041B8 00003850 00003960 00000001    ................  .......&...-....
003260   00000001 0000338A 50003C12 00003388    00003550 000035B8 000050B1 00C2DF17    ........&.......  ...&......&..B..
```

Figure 4.9. COBOL Sample Program (Part 22 of 27)

```
          DTACHK          07/23/71                                                                      PAGE   7

003280   00000000 --SAME--                                                         ....
0032A0   05F00700 900EF00A 47F0F082 000032A0   000032A0 0007AFFF 000032A0 0007AF24   .0....0..00.....  ................
0032C0   FFFFFFDC 000032A0 00003238 00004DB6   0A0407F1 000032B0 000032B0 00004218   ................  ...1............
0032E0   00005218 00000208 00000000 00000000   00000000 00000000 00000000 00000000   ................  ................
003300   00000000 --SAME--                                                         ....
003320   00000000 58C0F0C6 58E0C000 58D0F0CA   9500E000 4770F0A2 9610D048 92FFE000   ......0F......0.  ......0.........
003340   47F0F0AC 98CEF03A 90ECD00C 185D989F   F0BA9110 D0480719 07FF0700 00003BE2   .00...0.........  0..............S
003360   000032A0 000032A0 00003880 00003688   000038EC 00003BC8 C3D6C2C6 F0F0F0F1   ................  .......HCOBF0001
003380   E3C5E2E3 D9E4D540 0001C1C2 C3C4C5C6   C7C8C9D1 D2D3D4D5 D6D7D8D9 E2E3E4E5   TESTRUN ..ABCDEF  GHIJKLMNOPQRSTUV
0033A0   E6E7E8E9 0001F0F1 F2F3F4F0 F1F2F3F4   F0F1F2F3 F4F0F1F2 F3F4F0F1 F2F3F4F0   WXYZ..0123401234  0123401234012340
0033C0   C1000000 000040D5 E8C34000 00404040   40404040 00000000 F1F2F3C4 00000000   A..... NYC ..         ....123D....
0033E0   01010014 00000000 00000000 00000000   10000000 04000000 00009200 00000108   ................  ................
003400   00003430 00000000 10003C50 1160E2E8   E2F0F0F8 40400162 10000000 04000000   ...........&.-SY  S008  ..........
003420   00000000 86BCF018 41E0E001 58201044   010034E8 20000064 00003550 00003550   ......0.........  ...Y.......&...&
003440   00000014 000035B3 00640063 00000000   00000000 000049F0 01010014 00000000   ................  .......0........
003460   00000000 00000000 00000000 04000000   00008200 00000108 000034A8 00000000   ................  ................
003480   10003C50 1168E2E8 E2F0F0F8 40400272   00000000 20000000 00000000 86BCF018   ...&..SYS008  ..  ..............0.
0034A0   41E0E001 58201044 020035B8 00000064   00003620 00000000 00000014 00000000   ................  ................
0034C0   00640063 00000000 00004A06 000049F0   00000000 00000000 00000000 00000000   ...............0  ................
0034E0   00000000 --SAME--                                                         ....
0036C0   00000000 00000000 00000000 00000000   7000004B 00000000 00000000 000038EC   ................  ................
0036E0   00000000 00000000 000033F8 00003550   000032A0 000033F8 50003C12 00000000   ...........8...&  .......8&.......
003700   00000000 --SAME--                                                         ....
003780   00000000 00003958 00003550 01000000   70003934 000041B8 00003850 00003960   ...........&....  ...........&...-
0037A0   00003550 000032A0 000033F8 50003C12   00003388 00003550 000035B8 00003BE2   ...&.......8&...  .......&.......S
0037C0   000032A0 000032A0 00003880 00003960   000041B8 00003850 00004708 00003550   ...............-  .......&.......&
0037E0   00010000 00003958 00000000 00000000   00000000 00000000 00000000 00000000   ................  ................
003800   00000000 --SAME--                                                         ....
003820   00000000 00000000 00000000 00000000   00000000 00000000 000032A0 00000000   ................  ................
003840   00000000 00003550 000035B8 00003388   000033F8 00003470 00000000 0000001C   .......&........  ...8............
003860   00000000 0000338A 00000000 00000000   00003A3E 00000000 00000000 00000000   ................  ................
003880   000049E8 000041B8 00004990 00003950   00003A3E 00003AE0 00003B2C 00003B88   ...Y...........&  ................
0038A0   00003A5E 00003A72 00003B26 00003B58   00003A3E 000032A0 00000001 1C00001A   ................  ................
0038C0   5B5BC2D6 D7C5D540 5B5BC2C3 D3D6E2C5   5B5BC2C6 C3D4E4D3 F0E90000 C0000000   $$BOPEN $$BCLOSE  $$BFCMUL0Z......
0038E0   E6D6D9D2 60D9C5C3 D6D9C400 58F0C004   051F0001 4004F6F0 40404000 9640D048   WORK-RECORD..0..  .... .60   .. ..
003900   58F0C004 051F0001 4004F6F3 40404000   4110C040 5800D1C8 184005F0 5000F008   .0...... .63   .  ... ..JH. .0&.0.
003920   4500F00C 000033F8 0A024100 D1C858F0   C00805EF 5810D1C8 96101020 5020D1BC   ..0....8....JH.0  ......JH....&.J.
003940   5870D1BC D2016000 C038D201 601CC038   58F0C004 051F0001 4004F6F7 40404000   ..J.K.-...K.-...  .0...... .67   .
003960   4830C03A 4A306000 4E30D1D0 D705D1D0   D1D0940F D1D64F30 D1D04030 60004830   ......-...J.P.J.  J...JO..J. .-...
003980   C03A4A30 601C4E30 D1D0D705 D1D0D1D0   940FD1D6 4F30D1D0 4030601C 41406002   ....-...J.P.J.J.  ..JO..J. .-.. -.
0039A0   48206000 4C20C03A 1A425B40 C0385040   D1DC58E0 D1DCD200 6038E000 FA306050   ..-.<.....$ ..&   J...J.K.-.....-&
0039C0   C03C4140 601E4820 60004C20 C03A1A42   5B40C038 5040D1E0 58E0D1E0 D2006043   ... -...-.<.....  $ ..& J...J.K.-.
0039E0   E0009240 60444830 601C4E30 D1D0F331   603AD1D6 96F0603D 58F0C004 051F0001   ... -...-...J.3.  -.JO.0-..0......
003A00   4004F7F2 40404000 58F0C004 051F0002   00000014 0D0001C4 0038FFFF D2137000    .72   ..0......  .......D....K...
003A20   60385810 D1C81841 58F01010 45E0F00C   5020D1BC 5870D1BC 5810D1E8 07F158F0   -...JH...0....0.  &.J...J...JY.1.0
003A40   C004051F 00014004 F7F44040 40005800   D1E85000 D1E45800 C0205000 D1E84830   ...... .74   ...  JY&.JU....&.JY..
003A60   60004930 C03E58F0 C024078F 5810C00C   07F15800 D1E45000 D1E858F0 C004051F   -......0........  .1..JU&.JY.0....
003A80   00014004 F7F74040 40005810 D1C894EF   10201801 18404110 C04805F0 5000F008   .. .77   ...JH..  ..... .....0&.0.
003AA0   4500F00C 00000000 0A025800 D1C84110   C0500A02 4110C040 5800D1CC 184005F0   ..0.........JH..  .&..... ..J.. .0
003AC0   5000F008 4500F00C 00000000 0A024100   D1CC58F0 C00805EF 5810D1CC 96101020   &.0...0.........  J..0......J.....
003AE0   58F0C004 051F0001 4004F8F0 40404000   5810D1CC 58F0C028 91201010 071F1841   .0...... .80   .  ..J..0..........
003B00   41F0C028 D2021025 F00158F0 101045E0   F0085020 D1C05880 D1C0D213 60388000   .0..K...0..0....  0.&.J...J.K.-...
003B20   58F0C018 07FF5810 C01C07F1 58F0C004   051F0001 4004F8F1 40404000 5820C02C   .0.........1.0..  .... .81   .....
003B40   D500C058 60430772 95406044 0772D200   6043C059 92406044 5810C05C 5010D1EC   N...-.... -...K.  -.... -....*&.J.
003B60   4120D1EC 58F0C004 051F8001 1000000B   0C000060 00000000 00140D00 01C40038   ..J..0..........  ...-.........D..
003B80   FFFF5810 C01407F1 58F0C004 051F0001   4004F8F3 40404000 5810D1CC 94EF1020   .......1.0......   .83   ...J.....
003BA0   18011840 4110C048 070005F0 5000F008   4500F00C 00000000 0A025800 D1CC4110   ... .......0&.0.  ..0.........J...
003BC0   C0500A02 0A0E0A0E 50D05008 5050D004   5820C000 95002000 077992FF 20009610   .&......&.&.&&..  ................
```

Figure 4.9. COBOL Sample Program (Part 23 of 27)

```
            DTACHK              07/23/71                                                                                        PAGE    8

003BE0   D04850E0 D05405F0 9120D048 47E0F016    5800B048 982DB050 58E0D054 07FE9620    ..&....0......0.  .......&........
003C00   D0484160 00044110 C00C4170 C0380670    05505840 10001E4B 50401000 87165000    ...-............  .&. ....& ....&.
003C20   4180D1BC 4170D1CF 05105800 80001E0B    50008000 87861000 D203D1E8 C0305860    ..J...J.........  &.......K.JY...-
003C40   D1C45870 D1BC5880 D1C058E0 D05407FE    47F0F088 47F0F0B2 47F0F046 47F0F02A    JD..J...J.......  .00..00..00..00.
003C60   47F0F27A 47F0F052 47F0F260 47F0F174    C9D1C6C6 C2E9E9D5 F3F89071 F32C9620    .02..00..02-.01.  IJFFBZZN38..3...
003C80   103C4570 F0CA9871 F32C4400 10349101    1015078E 0A099071 F32C4570 F0CA47F0    ....0...3.......  ........3...0..0
003CA0   F0369071 F32C58C0 10445BC0 10404780    F0369180 10024710 F06C0A07 40C0103E    0...3.....$.. ..  0.......0... ...
003CC0   4570F260 91801002 4710F07E 0A07D201    103E1050 47F0F036 91801002 4710F092    ..2-......0...K.  ...&.00.......0.
003CE0   0A074200 10384800 10000A00 91801002    4710F0A6 0A074000 1000D200 1038101E    ................  ..0... ...K.....
003D00   07FE9104 10154710 F0C2D703 104C104C    07FED703 10441044 07FE989B 10449104    ........0BP..<.<  ..P.............
003D20   101547E0 F0DE879A F01847F0 F0E2869A    F0185090 104407F7 91201015 4710F0F2    ....0...0..00S..  0.&....7......02
003D40   0A009180 10024710 F0FC0A07 91011004    4710F2DC 58E01028 44001030 50E01028    ........0.......  ..2.........&...
003D60   58B01038 41BB0000 91041015 4710F194    50B01044 4AB01052 4BB01000 50B0104C    ...............1.  &...........&..<
003D80   48B01000 58A01048 10AA12BB 4780F154    96401005 12BB4740 F154948F 10051BBA    ...............1.  . ..... 1.......
003DA0   4770F140 41E0105C 91201002 4710F1AA    41E01058 947F1015 91401005 4710F1AA    ..1 ...*......1.  ......... ....1.
003DC0   96801015 948F1005 58B01038 41BB0000    D2021039 104150B0 10409120 10150787    ................  K.....&.. ......
003DE0   0A0007F7 189B4B90 10545090 10444BB0    10504AB0 100047F0 F12C9547 E0004770    ...7......&.....  .&.....01.......
003E00   F1B89108 1015071E 947F1015 58EE0000    5070F328 987DF32C 91041015 4780F1E4    1...............  &.3...3...... 1U
003E20   5810104C 41110001 5010F358 5810F354    47F0F1EA D203F358 10449110 10034780    ...<....&.3...3.  .01.K.3.........
003E40   F1F694DF 10029108 10154710 F21A9512    10144770 F20A58E0 10684110 F35405EE    16..........2...  ....2.......3...
003E60   0A320000 0A320000 0A329110 10024780    F24E4110 F35405EE 47F0E00A 47F0E012    ................  2...3....0...0..
003E80   0A3298F1 E12447F0 E03098F1 E124907D    F32C5870 F3289680 101547F0 F1745810    ...1...0...1....  3...3......01...
003EA0   F35805EE 98F1E0F8 907DF32C 5870F328    91801015 4710F0E8 96801015 91281015    3....1.8..3...3.  .......0Y.......
003EC0   47E0F0E8 0A0047F0 F0E89610 10249001    F35050E0 F35C94F7 10241801 4110F31E    ..0Y...00Y......  3&&.3*.7......3.
003EE0   41E0F296 0A029801 F3509110 10244710    F2BE907C F32C5870 F3289108 10154710    ..2.....3&......  2..@3...3.......
003F00   F2609120 10154710 F10447F0 F110910C    101547B0 F2D2D703 10441044 D703104C    2-......1..01...  ....2KP.....P..<
003F20   104C58E0 F35C94EF 102407FE 91011015    4710F104 5070F328 987CF32C 91401002    .<..3*..........  ..1.&.3..@3.. ..
003F40   4780F2FA 58E01024 07FE9608 10249108    10154710 F28A9120 10154710 F28658E0    ..2.............  ....2.......2...
003F60   10284400 103050E0 102847F0 F2865B5B    C2C3C5D6 E5F10000 00000000 00000000    ......&....02.$$  BCEOV1..........
003F80   00000000 --SAME--                                                             ....
003FC0   5B5BC2C5 D9D9E3D5 47F0F02A 47F0F02A    D7011052 10529026 F1CC4530 F13C9826    $$BERRTN.00..00.  P.......1...1...
003FE0   F1CC9104 1010071E 0A099026 F1CC5860    100895FF 101E4780 F03E4A60 104A9103    1............1..-  ........0..-....
004000   102B4710 F13C4740 F0E85846 00000640    18304304 00004140 00094334 F1DF1930    ....1.. 0Y.....   ....... ....1...
004020   4780F068 4640F05A 4940F1CA 47B0F078    920B6000 4530F1A8 9140102C 4780F088    ..0.. 0.. 1...0.  ..-...1.. ....0.
004040   94BF102C 4530F1A8 4144F1E9 D2006000    4000D600 60001028 9108102C 4780F0E8    ......1...1ZK.-.   .O.-.........0Y
004060   91801002 4710F0AA 0A079110 10034780    F0CCD501 1052F1C8 4770F0C2 96F01049    ......0.........  0.N...1H..0B.0..
004080   07F31831 4110F000 0A021813 D2001040    1038D200 10381030 D24F1098 10485830    .3....0.....K..   ..K.....K.......
0040A0   1030D24F 10483000 4530F1A8 9108102A    4780F0FC 92016000 4530F1A8 91011004    ..K.......1.....  ..0...-...1.....
0040C0   4710F11E 9106102A 4750F132 58401070    41440000 D5014000 F1F34770 F1329102    ..1......&1.. ..  ....N. .13..1...
0040E0   102C4710 F1B69108 102A4710 F13258E0    10409826 F1CC9620 102C07FE D2001040    ....1.......1...  . ..1.......K..
004100   1050D500 10401048 4770F192 18534800    10344120 00031843 4342103C 43321044    .&N.. ....1.....  ................
004120   19434770 F1724202 103C8800 00084620    F1584144 00014242 103C1835 D503103C    ....1...........  1...........N...
004140   10364720 F0B2D200 10401049 D203104C    103C4340 10404144 00014240 1050D501    ....0.K.. ..K..<  ... . ..... .&N.
004160   1052F1C8 4770F0E8 0A009180 10024710    F1B40A07 07F31821 48020006 41101016    ..1H..0Y........  1....3..........
004180   0A021812 47F0F132 00000004 00003550    000032A0 0000395A 00000004 000048E0    .....01........&  ................
0041A0   F9C3F14E E5E660F0 400BCBE3 8B030141    1B130B61 5C000000 47F0F00E C9D3C2C4    9C1.VW-0 ..T....  .../*....00.ILBD
0041C0   C4E2D7F0 F3F390EC D10896FF F7C9D201    D1581000 4580F200 47000004 47F0F030    DSP033..J...7IK.  J.....2.......00.
0041E0   416D007C 5870F7BC 9400D104 91801000    41110002 4710F426 9400D105 95401000    ...@..7...J.....  ......4...J.. ..
004200   4770F08A 1BEE43E1 00011AE1 41EE0003    91011001 4710F064 41EE0001 50ED0114    ..0.............  ......0......&...
004220   9140D048 4780F1FA 1B554351 00014141    00021B75 06704450 F1CA9601 D10447F0    . ....1.........  ........&1...J..0
004240   F126D200 D0601004 9640D060 438D0060    4480F7CA 4A410006 91101000 4710F0AC    1.K..-... .-...-  ..7...........0.
004260   58440000 4A410008 910F1000 4740F35C    91201000 4710F0D6 D203D060 1000585D    .............. 3*  ......0OK..-....
004280   00608950 00088850 000847F0 F0F0D200    D0601004 9650D060 438D0060 4480F7CA    .-.&...&...000K.  .-...&.-...-..7.
0042A0   4A510002 48550000 9102D105 071E1957    47C0F1A8 910F1000 4710F116 9101D15A    ...........J.....  ..1.......1...J.
0042C0   4710F116 12774780 F12647F0 F1A25950    F7BC4720 F10C91FF F7C94710 F10C9012    ..1.....1..01..&  7...1...7I..1...
0042E0   D0FC4580 F2004700 000247F0 F192411D    00785830 F7BC1B37 418D007B 1A83509D    ....2......01...  ....7.........&.
004300   015C419D 007C1989 4780F162 95408000    4770F162 06804630 F14E589D 015C5030    .*...@....1.. ..  ..1.....1....*&.
004320   F53C9200 F53C501D 015C4110 F5400A00    91801002 4710F182 0A07581D 015C58FD    5...5.&..*..5 ..  ......1......*..
```

Figure 4.9. COBOL Sample Program (Part 24 of 27)

```
         DTACHK            07/23/71                                                                                   PAGE    9

004340   010C416D 007C5870 F7BC9101 D1044710   F1FA9812 D0FC47F0 F0F01887 47F0F1AA     .....@..7...J...  1......000...01.
004360   18851288 47C0F1BA 06804480 F1CA4188   00011A68 1B781B58 4780F1D0 1A4847F0     ......1.....1...  ..........1....0
004380   F126D200 60004000 94FED15A 9101D105   071E4111 000AD501 1000F7C4 4770F08A     1.K.-. ...J...J.  ......N...7D..0.
0043A0   18E141EE 000250ED 01149601 D10447F0   F12698EC D10807F1 501D015C 9103D159     ......&.....J..0  1...J..1&..*..J.
0043C0   4710F2E2 9102D159 4710F33E D203F7A4   F7A8D201 F6D0F6D2 D203F7B0 F7B49104     ..2S..J...3.K.7.  7.K.6.6KK.7.7...
0043E0   80034770 F2444810 F6D00610 1B171211   4770F244 9104D105 4770F336 5810F7B0     ....2...6.......  ..2...J...3...7.
004400   9101F7A0 4710F26E 9601F7A0 90E2D144   4110F7D0 4500F264 00004708 0A0298E2     ..7...2...7..SJ.  ..7...2........S
004420   D14447F0 F2844870 F6D00670 91048003   4780F284 4160F728 47F0F336 90E2D144     J..02...6.......  ..2..-7..03..SJ.
004440   4870F6D0 9103D159 47E0F2A2 5860F7A4   582D0118 D2076048 20009104 80034710     ..6...J...2..-7.  ....K.-.........
004460   F2B258F1 001045EF 000C98E2 D1445860   F7A40670 06709240 60004470 F3569103     2..1.......SJ..-  7...... -...3...
004480   D1594710 F2DA4166 00014177 000147F0   F3364170 004847F0 F336D203 F7A4F7AC     J...2..........0  3......03.K.7.7.
0044A0   D201F6D0 F6D4D203 F7B0F7B8 5810F7B0   9101F7A1 4710F322 9601F7A1 90E2D144     K.6.6MK.7.7...7.  ..7...3...7..SJ.
0044C0   4110F7D0 4500F314 000047A0 0A0298E2   D1444870 F6D047F0 F2A29104 80034780     ..7...3.........S  J...6..02.......
0044E0   F2845861 002C4166 00004170 0048581D   015C47F8 0004411D 00784111 00045010     2../............  .*.8...........&.
004500   F5389209 F538581D 015C47F8 0008D200   60016000 91021000 4710F3CE 95021001     5...5....*.8..K.  -.-.......3.....
004520   4770F37C 1B00D201 D0604000 480D0060   47F0F38E 95041001 4770F39C D203D060     ..3@..K..- ....-  .03.......3.K..-
004540   4000580D 00604E0D 0060F395 D068D062   47F0F3FC 501D00FC D207D060 40009801      ....-...-3.....  .03.&...K..-  ..
004560   D0605D00 F7C04E1D 00684E0D 0070F384   D060D06B 96F0D068 F384D069 D073581D     .-..7.........3.  .-.,.0..3.......
004580   00FC47F0 F3FC1B88 43810001 D709D060   D060415D 006A1B58 06804480 F420F384     ...03........P..-  .-...........4.3.
0045A0   D069D065 F154D060 D060F384 D060D061   96F0D068 414D0071 91304000 4740F40C     ....1..-.-3..-./  .0........ .. 4.
0045C0   96F04000 41440001 1B554851 00021B45   9601D15A 47F0F0F0 D2005000 40009201     .0 ..............  ..J..000K.&. ...
0045E0   D1059400 F7C91233 4780F43C 9400D106   47F0F440 9601D106 58A20000 41220004     J...7I....4...J.  .04 ..J.........
004600   41900020 1EAA0690 4730F458 47F0F512   91101000 4780F47E 910F1000 4770F47E     ...........4..05.  ......4.......4.
004620   9604D105 1EAA4730 F4909101 D1064710   F49047F0 F4BC1EAA 4730F4C4 9101D106     ..J.....4...J...  4..04.....4D..J.
004640   4710F4C4 47F0F4C8 45E0F08A 4140F7C6   41500003 45E0F0F0 4111000A 45E0F08A     ..4D.04H..0.. 7F  .&....00......0.
004660   12774780 F51A9240 60004166 00010670   47F0F51A 4111000A 47F0F51A 47F0F4A4     ....5.. -.......  .05......05..04.
004680   9602D105 45E0F08A 195747C0 F4EA1277   4770F4E4 45E0F126 47F0F4D0 188747F0     ..J...0.....4...  ..4U..1..04....0
0046A0   F4EC1885 06809240 60004480 F50C45E0   F1B61255 4780F504 47F0F4D0 9401D105     4...... -...5...  1.....5..04...J.
0046C0   47F0F4A8 D2006001 600089A0 000147F0   F4A44111 000AD501 1000F7C4 4770F52C     .04.K.-.-.......0  4.....N...7D..5.
0046E0   47F0F1E8 4690F44C 47F0F440 00000000   090046F0 00000001 00000000 00000004     .01Y..4<.04 ....  ...0............
004700   000046F0 00000000 00008000 0C000003   00004778 00004780 00003FC0 32005B5B     ...0............  ..............$$
004720   C2C3D4E3 F0F70000 00000000 00000800   002010F1 260048DF 80000000 00000000     BCMT07..........  ...1............
004740   00000000 0000FF00 00000000 00000000   13000000 00000000 00000079 47000000     ................  ................
004760   07004742 40000006 31004744 40000005   08004768 20000001 010048DF 20000079     .... ....... ...  ................
004780   050048DF 60000079 31004744 40000005   08004788 20000001 1E004798 30000081     ....-....... ...  ................
0047A0   00008400 08000002 000047F8 00000000   00003FC0 3200D7C3 C8C4E3C6 40400000     ............8...  ......PCHDTF  ..
0047C0   00000000 00000800 002020F3 2400488E   80000000 00000000 00000000 0000FF00     ............3...  ................
0047E0   00000000 00000000 19000020 00000000   00000051 47000000 070047DA 40000006     ................  ............ ...
004800   310047DC 40000005 08004800 20000001   1D0047EC A0000008 0500488E 60000051     .... ...........  .............-...
004820   310047DC 40000005 08004820 20000001   1E004830 30000059 01004848 20000050     .... ...........  ...............&
004840   01004898 20000050 00000000 00000000   00000000 00000000 00000000 00000000     ........&........  ................
004860   00000000 --SAME--                                                             ....
004880   00000000 00000000 00790079 0050E540   40404040 40404040 40404040 40404040     ..............&V
0048A0   40404040 --SAME--
004940   40404040 40404040 40404040 40404040   40404040 40404040 01000000 000048DF                         ........
004960   000048DF 0000488F 00004708 00004708   000047A0 00000064 3B9ACA00 FFFF407E     ................  .............. .
004980   40FF1800 00000000 5B5BC2D6 D7C5D540   47F0F00E C9D3C2C4 C9D4D3F0 F3F39015      ........$$BOPEN   .00.ILBDIML033..
0049A0   D0605810 00140A21 18305823 00001832   4B20F056 1B444343 00074830 102E8830     .-..............  ..0.............
0049C0   00044A30 10481B55 43530000 89500001   4A50104C 89400001 1A54D200 20005000     .............&..  .&.<. ....K...&.
0049E0   9815D060 07FE0008 FF000000 00000000   05F047F0 F00EC9D3 C2C4E2C1 C5F0F3F3     ...-............  .0.00.ILBDSAE033
004A00   1B0047F0 F0184100 000105F0 9005F0A8   41440000 91104014 4780F022 91084015     ...00.......0..0.  ...... ....0... .
004A20   4710F022 91204002 071E5811 00044154   00004B50 F0CC5855 00004155 00001255     ..0... .........  ...&0...........
004A40   4780F08A 1835D501 F0A23000 4780F04C   41330002 47F0F03A 5050F0A4 D5033006     ..0...N.0.....0<  .....00.&&0.N...
004A60   F09E4780 F0804403 00064403 000A1852   92F05000 D2065001 50001200 4780F07C     0...0...........  .0&.K.&.&.....0@
004A80   96015001 47F0F080 96015000 9805F0A8   58F0F0A4 07FF8900 00184B40 F0CE1604     ..&..00...&...0.  .00........ 0...
004AA0   4110F0C4 0A021B00 0A060000 000005F0   00000000 00000000 00000000 00000000     ..0D............0  ................
004AC0   00000000 00000000 00000000 00000000   5B5BC2C3 D6C2C5D9 00040008 00000000     ................  $$BCOBER........
004AE0   00000000 --SAME--                                                             ....
```

Figure 4.9. COBOL Sample Program (Part 25 of 27)

```
          DTACHK            07/23/71                                                                                     PAGE  10
   07AFE0 000000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 0000................  ................
```

Figure 4.9. COBOL Sample Program (Part 26 of 27)

```
EOJ DTACHK                                                       11.52.38,DURATION 00.03.07
```

Figure 4.9. COBOL Sample Program (Part 27 of 27)

Debugging FORTRAN Programs

When debugging FORTRAN programs, gather the program listing, linkage editor map, main storage dump, and any system messages. The linkage editor map is the most useful documentation. Using this map together with the main storage dump, the programmer can locate such information as DTF tables, I/O areas, logic modules and FORTRAN subroutines used at object time.

A sample FORTRAN program, linkage editor map, and storage map are included to show the relationship between the linkage editor map and the contents of main storage, and to indicate where the DTF tables and I/O areas are located (see Figure 4.10). The DTF tables are built as they are opened, starting from the end of main storage or the end of the partition in which the program is loaded. The I/O areas directly follow the end of the problem program. The FORTRAN program itself builds only one I/O area.

User-written code is always the first item in main storage followed by the object time FORTRAN subroutines to perform the functions required by the problem program. Only one logic module is accessed by all FORTRAN I/O units (except console typewriter) for each FORTRAN program. The console typewriter uses its own DTFCN that includes a logic module to handle all console typewriter functions. Each FORTRAN program includes an I/O unit table to convert a FORTRAN unit to a DOS symbolic unit.

FORTRAN uses the STXIT macro to handle its own program checks. It also uses the normal DOS register conventions. In addition, register 13 is the base register for the FORTRAN in-line code and points to the register save area. Register 15 is the base register for FORTRAN object time subprograms.

```
// JOB ANYNAME
// OPTION DUMP,LINK
// EXEC FFORTRAN

// FTC LISTX,NAME=SAMP2 ←——Refer to hex '18' in LISTX Map.

    MAP     YES

    LOAD    =4

    DECK    NO

    LIST    YES  ←——  Output A Listing

    LISTX   YES  ←——  Output Object Module Listing

    EBCDIC
```

Figure 4.10. FORTRAN Sample Program (Part 1 of 8)

```
DOS FORTRAN IV 360N-FO-479 3-0        SAMP2           DATE   11/11/68     TIME    00.00.06       PAGE 0001

              C PRIME NUMBER GENERATOR
0001                WRITE (3,1)
0002              1 FORMAT ('1FOLLOWING IS A LIST OF PRIME NUMBERS FROM 2 TO 1000'/
                   119X,1H2/19X,1H3)
0003                DO 4 I=5,1000,2
0004                K=SQRT(FLOAT(I))
0005                DO 2 J=3,K,2
0006                IF (MOD(I,J) .EQ. 0) GO TO 4
0007              2 CONTINUE
0008                WRITE (3,3) I
0009              3 FORMAT (I20)
0010              4 CONTINUE
0011                WRITE (3,5)
0012              5 FORMAT (' THIS IS THE END OF THE PROGRAM')
0013                STOP
0014                END
```

Internal
Statement
Number
assigned by
FORTRAN
Compiler.

Figure 4.10. FORTRAN Sample Program (Part 2 of 8)

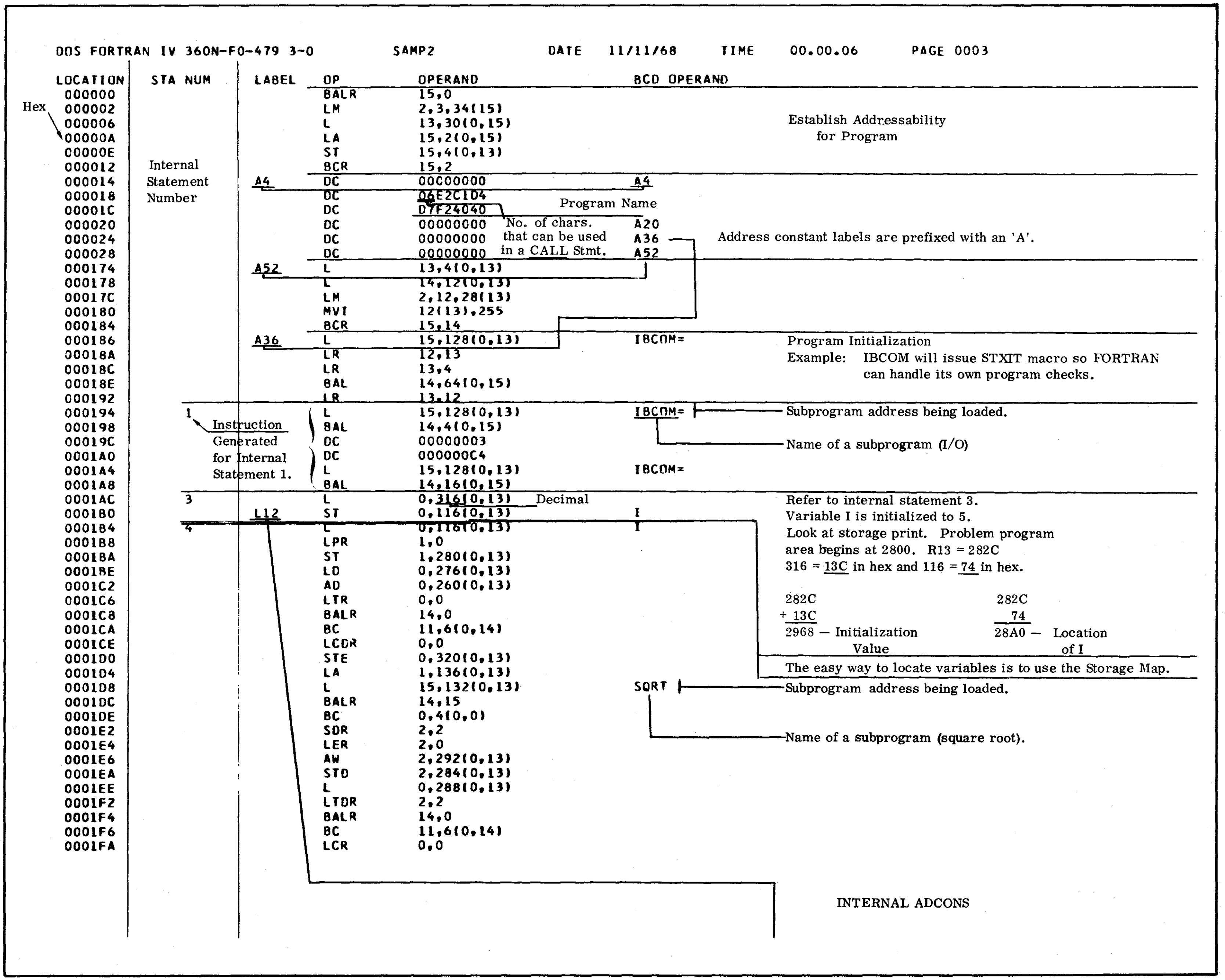
```
 DOS FORTRAN IV 360N-FO-479 3-0          SAMP2             DATE   11/11/68      TIME     00.00.06        PAGE 0002

                                      SCALAR MAP
SYMBOL       LOCATION        SYMBOL     LOCATION        SYMBOL       LOCATION        SYMBOL     LOCATION        SYMBOL     LOCATION
I                  A0        K                A4        J                  A8

                                      SUBPROGRAMS CALLED
SYMBOL       LOCATION        SYMBOL     LOCATION        SYMBOL       LOCATION        SYMBOL     LOCATION        SYMBOL     LOCATION
IBCOM=           AC          SQRT             B0

                                      FORMAT STATEMENT MAP
SYMBOL       LOCATION        SYMBOL     LOCATION        SYMBOL       LOCATION        SYMBOL     LOCATION        SYMBOL     LOCATION
     1           C4               3          108             5          10C
```

Locate these values in storage print.
P. P. begin = 2800 Variable = 2800 + DSP.

Displacement to an adcon containing the address of subprograms. Check in storage print and use Linkage Editor Map.

Displacement to symbols used on FORMAT STATEMENTS. Locate the beginning of Symbol 1 in dump and cross-reference to LISTX Map. Check the LISTX Map for the internal statement numbers 2, 9, and 12. They are not there because a Format Source Statement causes a constant to be assembled rather than machine instructions.

Figure 4.10. FORTRAN Sample Program (Part 3 of 8)



```
DOS FORTRAN IV 360N-FO-479 3-0          SAMP2          DATE   11/11/68     TIME     00.00.06      PAGE 0003

LOCATION   STA NUM   LABEL  OP      OPERAND                 BCD OPERAND
 000000                     BALR    15,0
 000002                     LM      2,3,34(15)
 000006                     L       13,30(0,15)
 00000A                     LA      15,2(0,15)
 00000E                     ST      15,4(0,13)
 000012                     BCR     15,2
 000014              A4     DC      00C00000                A4
 000018                     DC      06E2C1D4
 00001C                     DC      D7F24040
 000020                     DC      00000000                A20
 000024                     DC      00000000                A36
 000028                     DC      00000000                A52
 000174              A52    L       13,4(0,13)
 000178                     L       14,12(0,13)
 00017C                     LM      2,12,28(13)
 000180                     MVI     12(13),255
 000184                     BCR     15,14
 000186              A36    L       15,128(0,13)            IBCOM=
 00018A                     LR      12,13
 00018C                     LR      13,4
 00018E                     BAL     14,64(0,15)
 000192                     LR      13,12
 000194    1                L       15,128(0,13)            IBCOM=
 000198                     BAL     14,4(0,15)
 00019C                     DC      00000003
 0001A0                     DC      000000C4
 0001A4                     L       15,128(0,13)            IBCOM=
 0001A8                     BAL     14,16(0,15)
 0001AC    3                L       0,316(0,13)
 0001B0              L12    ST      0,116(0,13)             I
 0001B4    4                L       0,116(0,13)             I
 0001B8                     LPR     1,0
 0001BA                     ST      1,280(0,13)
 0001BE                     LD      0,276(0,13)
 0001C2                     AD      0,260(0,13)
 0001C6                     LTR     0,0
 0001C8                     BALR    14,0
 0001CA                     BC      11,6(0,14)
 0001CE                     LCDR    0,0
 0001D0                     STE     0,320(0,13)
 0001D4                     LA      1,136(0,13)
 0001D8                     L       15,132(0,13)            SQRT
 0001DC                     BALR    14,15
 0001DE                     BC      0,4(0,0)
 0001E2                     SDR     2,2
 0001E4                     LER     2,0
 0001E6                     AW      2,292(0,13)
 0001EA                     STD     2,284(0,13)
 0001EE                     L       0,288(0,13)
 0001F2                     LTDR    2,2
 0001F4                     BALR    14,0
 0001F6                     BC      11,6(0,14)
 0001FA                     LCR     0,0
```

Hex

Internal Statement Number

Establish Addressability for Program

Program Name

No. of chars. that can be used in a CALL Stmt.

Address constant labels are prefixed with an 'A'.

Program Initialization
Example: IBCOM will issue STXIT macro so FORTRAN can handle its own program checks.

Instruction Generated for Internal Statement 1.

Subprogram address being loaded.

Name of a subprogram (I/O)

Decimal

Refer to internal statement 3.
Variable I is initialized to 5.
Look at storage print. Problem program area begins at 2800. R13 = 282C
316 = 13C in hex and 116 = 74 in hex.

```
 282C                          282C
+ 13C                            74
 2968 — Initialization         28A0 — Location
         Value                          of I
```

The easy way to locate variables is to use the Storage Map.

Subprogram address being loaded.

Name of a subprogram (square root).

INTERNAL ADCONS

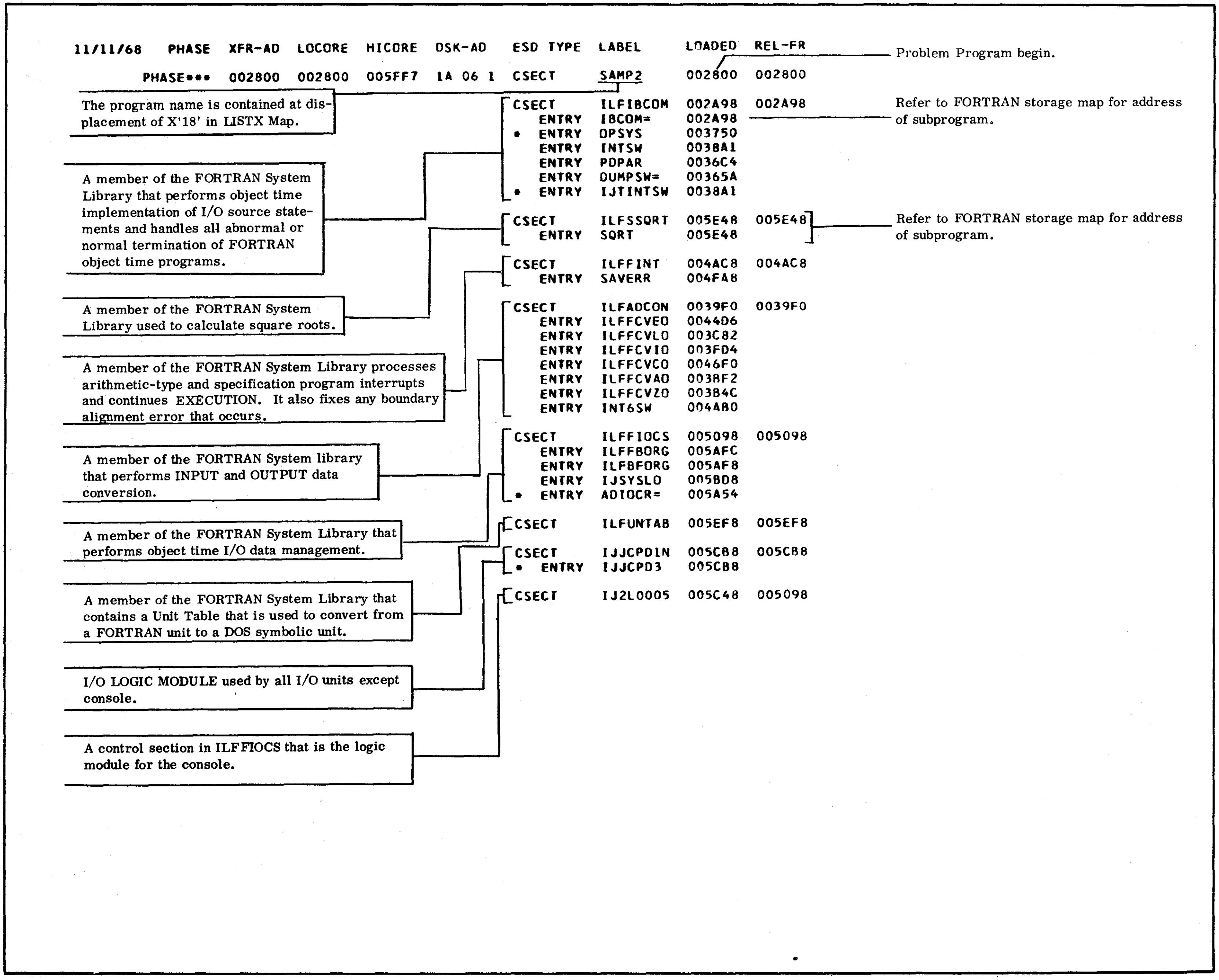
Figure 4.10. FORTRAN Sample Program (Part 4 of 8)

```
DOS FORTRAN IV 360N-FO-479 3-0          SAMP2          DATE   11/11/68      TIME     00.00.06      PAGE 0004

0001FC                 ST      0,120(0,13)       K
000200    5            L       0,324(0,13)
000204         L20     ST      0,124(0,13)       J
000208    6            L       0,116(0,13)       I
00020C                 SRDA    0,32(0)
000210                 D       0,124(0,13)       J
000214                 M       0,124(0,13)       J
000218                 S       1,116(0,13)       I
00021C                 LCR     1,1
00021E                 C       1,260(0,13)
000222                 L       14,96(0,13)       4
000226                 BCR     8,14
000228    7    2       L       0,124(0,13)       J
00022C                 L       1,104(0,13)       L20
000230                 LA      2,2(0,0)
000234                 L       3,120(0,13)       K
000238                 BXLE    0,2,0(1)
00023C    8            L       15,128(0,13)      IBCOM=
000240                 BAL     14,4(0,15)
000244                 DC      00000003
000248                 DC      00000108
00024C                 L       15,128(0,13)      IBCOM=
000250                 BAL     14,8(0,15)
000254                 DC      0450D074
000258                 BAL     14,16(0,15)
00025C    10   4       L       0,116(0,13)       I
000260                 L       1,100(0,13)       L12
000264                 LA      2,2(0,0)
000268                 LA      3,1000(0,0)
00026C                 BXLE    0,2,0(1)
000270    11           L       15,128(0,13)      IBCOM=
000274                 BAL     14,4(0,15)
000278                 DC      00000003
00027C                 DC      0000010C
000280                 BAL     14,16(0,15)
000284    13           L       15,128(0,13)      IBCOM=
000288                 BAL     14,52(0,15)
00028C                 DC      05404040
000290                 DC      40F0
                       END
 TOTAL MEMORY REQUIREMENTS 000292 BYTES
 HIGHEST SEVERITY LEVEL OF ERRORS FOR THIS MODULE WAS 0
```

Internal Adcon used in this case for the limits of a DO LOOP.

Hex number of bytes.

Figure 4.10. FORTRAN Sample Program (Part 5 of 8)



```
11/11/68    PHASE   XFR-AD   LOCORE   HICORE   DSK-AD    ESD TYPE   LABEL       LOADED   REL-FR

        PHASE***   002800   002800   005FF7   1A 06 1   CSECT      SAMP2       002800   002800

                                                         CSECT      ILFIBCOM    002A98   002A98
                                                            ENTRY   IBCOM=      002A98
                                                        *   ENTRY   OPSYS       003750
                                                            ENTRY   INTSW       0038A1
                                                            ENTRY   PDPAR       0036C4
                                                            ENTRY   DUMPSW=     00365A
                                                        *   ENTRY   IJTINTSW    0038A1

                                                         CSECT      ILFSSQRT    005E48   005E48
                                                            ENTRY   SQRT        005E48

                                                         CSECT      ILFFINT     004AC8   004AC8
                                                            ENTRY   SAVERR      004FA8

                                                         CSECT      ILFADCON    0039F0   0039F0
                                                            ENTRY   ILFFCVEO    0044D6
                                                            ENTRY   ILFFCVLO    003C82
                                                            ENTRY   ILFFCVIO    003FD4
                                                            ENTRY   ILFFCVCO    0046F0
                                                            ENTRY   ILFFCVAO    003BF2
                                                            ENTRY   ILFFCVZO    003B4C
                                                            ENTRY   INT6SW      004AB0

                                                         CSECT      ILFFIOCS    005098   005098
                                                            ENTRY   ILFFBORG    005AFC
                                                            ENTRY   ILFBFORG    005AF8
                                                            ENTRY   IJSYSLO     005BD8
                                                        *   ENTRY   ADIOCR=     005A54

                                                         CSECT      ILFUNTAB    005EF8   005EF8

                                                         CSECT      IJJCPD1N    005CB8   005CB8
                                                        *   ENTRY   IJJCPD3     005CB8

                                                         CSECT      IJ2L0005    005C48   005098
```

Problem Program begin.

The program name is contained at displacement of X'18' in LISTX Map.

Refer to FORTRAN storage map for address of subprogram.

A member of the FORTRAN System Library that performs object time implementation of I/O source statements and handles all abnormal or normal termination of FORTRAN object time programs.

Refer to FORTRAN storage map for address of subprogram.

A member of the FORTRAN System Library used to calculate square roots.

A member of the FORTRAN System Library processes arithmetic-type and specification program interrupts and continues EXECUTION. It also fixes any boundary alignment error that occurs.

A member of the FORTRAN System library that performs INPUT and OUTPUT data conversion.

A member of the FORTRAN System Library that performs object time I/O data management.

A member of the FORTRAN System Library that contains a Unit Table that is used to convert from a FORTRAN unit to a DOS symbolic unit.

I/O LOGIC MODULE used by all I/O units except console.

A control section in ILFFIOCS that is the logic module for the console.

Figure 4.10. FORTRAN Sample Program (Part 6 of 8)

```
          QWERT                11/11/68            PAGE    1

GR 0-7    000036C4 00004FA0 000029B6 00002974    0000FF74 FFFFFF8C 00002800 00002798
GR 8-F    000040E8 0A0407F1 00002810 00002810    0000282C 0000282C 620029DE 00000000
FP REG    41500000 00000000 00000000 00000000    BE1A2000 00000000 4817DF25 20000000
000000    00000000 00000000 00000000 00000000    00000000 000001E0 FF070000 00000000    FF050007 400020A8 FF150001 62000002
000030    5B5BC2C5 D6D1F340 FF05000E 80000000    00002120 08400000 0000211B 00000000    749D77FF 007FE8B7 00040000 0F000F54
000060    00040000 000002DE 00040000 00000F12    00000000 000009F8 00040000 00000278    00000000 00000000 00000000 00002508
000090    00000000 00030005 000306B0 41BB0071    45700128 41A0B8BC 45700780 18A84190    01384180 0DBC47F0 00D806B0 06B006B0
0000C0    06B006B0 06B006B0 41BB0017 41BB0050    45700128 41800138 9640A001 9120A00C    471000E8 9250A001 9281A000 92030095
0000F0    95E2A002 4780090C 4BA0023A 49A0024E    41AAB8AC 077894F9 021BD701 02380238    9680A001 9283A000 440003C8 0788947F
000120    A0014570 0D1A07F8 42B000E5 48B00270    47F0B31A 4570B31A D205B56F B575DC05    B56EB8AC 1BAADD06 B56E0012 43A10007
000150    42A0020F 41AAB8AC 4400A004 5890A004    4220A000 9140A001 471001B0 D20701C8    90086800 90586820 90606840 90686860
000180    90709898 90108200 01C84400 A0045890    A0049818 9030989C 01C88200 00389284    B90CD207 01C8B5B8 9890B5C0 820001C8
0001B0    9680A000 41100030 47F00CA2 96020019    82000018 00000000 FF050007 400020A8    00001000 00002000 80000BBC 00000000
0001E0    F1F161F1 F161F6F8 28002800 00000000    00000000 00000000 D8E6C5D9 E3404040    0000FFFF 00005F47 00005F47 00000010
000210    0000FFFF E8407E90 A0A07ED0 00871987    198E19EF 19F01A04 1A221A26 1A2A38F1    F1F1F1F6 F8F3F1F6 000018AC 0000000B
000240    1448148E 1574158C 15940010 19640010    5B5BC2D6 00020000 01001800 18680000    00000000 01E00000 00001AC8 00002788
000270    10000000 00000000 9238019D 909C01C8    47F0028C 5890008C 45A90060 4190054C    48A0020E 4AA0023A 9180A000 471002B2
0002A0    58B0A004 9018B030 48B00270 41CBB000    07F99601 A00048B0 027041CB 800095FF    A00F0789 90D0B5D0 48D0019C D207B5B8
0002D0    D00094FD B5B9D20F B5C001C8 07F9909C    01C847F0 03089107 002347B0 03089500    00994780 03085890 008CD201 0090020E
000300    41B001C8 45A90064 9220019D 45900290    4190018A 95000023 478003B4 95230023    47B000BC 48600022 1A664866 B51807F6
000330    181F4860 B49C1B22 43201007 4130001A    1B234740 03504130 00101B23 47B00352    1A234220 04634320 100747F0 040E4720
000360    00C84230 04634820 020E4322 B86B1A23    950B1007 47F0040A 472000C6 96801002    960C1004 07F91858 41430002 43540000
000390    41455000 1A444144 B93895FF 40004770    038C4284 000007F9 95FF043B 07891B00    5000B5DC 95FF043B 47800D3A 4860020E
0003C0    9560020F 47800330 D502A005 02594770    03E04111 00004910 B48A47B0 03E01B66    4121000F 4570B356 1B334330 10079501
0003F0    10064770 035E92FF 0463D200 0403A00D    4123000B D5001007 A00E47B0 00C8D403    1002B918 41822000 4870022C 43387000
000420    4930B488 47B00378 89300003 4A300220    91F03004 4780B076 41800001 41488000    1A444144 B9381858 4A50024C D200043B
000450    40005010 400092FF 40004220 50004260    500B9203 501691F0 30044780 B09647F0    04765880 B7F895FF 30024770 03864280
000480    30029198 30060779 43203000 4322B905    48603000 95003000 478004A7 9F006000    07694060 04C49120 30064710 009A9504
0004B0    50164780 0520D202 00491009 D3000048    500B9C00 000E4770 04D44032 B91C9680    300607F9 473007AE 47F004F2 5880008C
0004E0    909B8050 18984860 300045A9 005C989B    90509106 00454770 09E4913F 00454770    050891AF 00440789 4060003A 58600048
000510    4A60B424 50600040 4032B91C 47F00574    9501500B 478004B6 9101100C 471004B6    45700780 D201B3D5 80024170 B3C85070
000540    0048D202 70091009 47F004BC 4190018A    91060045 477009EA 94FD0039 1B224320    003A4322 B9054832 B91CD500 3001003B
000570    47700722 457006A0 91020044 471007C8    91390045 477007BC 91800044 478005B0    96040044 910A0219 477005B0 95003004
0005A0    477005B0 9610B8F4 9603B8EC 41900134    91100044 471005CC 95000044 477005C2    07F9D602 00410041 477006BA 91040044
0005D0    477005E2 95500044 47700678 94773006    07F99477 30069104 30064780 0660D601    10040044 94733006 9180100C 4710060C
000600    95503004 4780076A 92003003 96801002    9101100C 4710062E 45700780 198A47D0    06264190 01349601 800047F0 064894FE
000630    100C91F0 30044770 06484370 B7FC0670    4270B7FC 47F00648 92FF500B 43703002    D2003002 4000D200 4000043B 4270043B
000660    95003000 47700678 95FF3002 07899198    30064780 069C07F9 4832B91C D2000689    003A0570 41330008 95013000 4720074C
000690    95FF3002 07879198 30060777 41700490    1B554350 30024145 50001A44 4144B938    4A50024C 58104000 07F7D201 10000046
0006C0    D202100D 0041D601 10040044 95015016    47200700 58700040 4B70B424 58770000    D5017000 B55E4780 06F8D501 70008BFE
0006F0    47700700 96203006 96401002 96011004    91270044 477005F4 96803006 91041002    478005F8 96043006 95003000 47700678
000720    07F9D200 0739003B D200073F 003A4832    B91E4170 05740560 950C3001 07879500    30004133 000807D6 47F00678 4842B91C
000750    4832B91E 45700760 41330008 19340729    95FF3002 078747F0 06961B77 43703003    4C70B3DE 4A700226 92007000 47F0060C
000780    4880024E 91F0500B 47800796 4380500B    4188B8AC 07F79180 100C4710 07A6910F    500B4780 07904180 006047F0 07900729
0007B0    D2010044 B560D202 0041B421 92040837    91090045 478007F6 92040837 58800040    12884770 07E69212 08379104 30064780
0007E0    080694FD 08379120 100C4710 07F69101    10024710 08A095FF 30024780 08069110    100C4710 08A04860 B49A4960 B49847B0
000810    088C4A60 B9164060 B49AD715 60006000    D2076000 00404030 600895FF 30024770    08369602 08379204 600A9690 30069102
000840    00444710 084E9130 00454770 08D09102    08374780 08A84186 000C4080 B3DA4180    B3D84570 099447F0 08C0D300 B9043004
000870    9540B904 47800918 95603004 4780B19A    95623004 4780B19A 47F008D0 91020837    47100588 91010039 47800902 47F0009C
0008A0    96201002 47F00588 9120100C 47800856    41810010 45700994 47F008C0 47F00930    940E0044 95060044 47800994 9220600B
0008D0    45800984 9102600A 471008EC 9502500B    478008F8 95FF0219 478008F8 41900134    9287B8FC 47F00660 D2050005 600C9706
000900    090D9701 090DD201 00023000 92040000    92E60001 820009E0 94F3600C 9501600C    477008D0 91020837 47100930 96021003
000930    D2070040 60004580 095E47F0 05884400    09304570 099C47F0 094A4580 095E47F0    05789104 600A4780 09304180 066047F0
000960    097496F0 095F4830 B4B4D257 B4ACB4C2    44000DB6 4870B49A 4B70B916 4070B49A    946F3006 1B224320 003A4322 B9054570
```

Figure 4.10. FORTRAN Sample Program (Part 7 of 8)

```
002360   5B5BC2D6   C4E2D7E5   00000001   00000002     080101B2   5B5BC2D6   C6D3D7E3   00000001
002380   00000002   08020350   5B5BC2D6   C9E2F0F5     00000001   00000002   090103D0   5B5BC2D6
0023A0   D4E3F0F2   00000001   00000002   09020451     5B5BC2D6   D4E3F0F5   00000001   00000003
0023C0   00010451   5B5BC2D6   D4E3F0F6   00000001     00000003   00020451   5B5BC2D6   D6D9F0F1
0023E0   00000001   00000003   010100F0   5B5BC2D6     D7C5D5C3   00000001   00000003   010201E3
002400   5B5BC2D6   D7C5D5F2   00000001   00000003     020101EC   5B5BC2D6   E2C4C9F3   00000001
002420   00000003   02020190   5B5BC2D6   E2C4D6F1     00000001   00000003   03010308   5B5BC2D6
002440   E2C4D6F2   00000001   00000003   03020268     00000000   00000000   00000000   00000000
002460 TO THE NEXT LINE ADDRESS CONTAINS 00000000
002500   00000000   00000000   00002787   00001E30     00000280   000004D8   0000008C   00000F18
002520   000002E2   00000856   000008CE   00000000     00000000   00000000   00000000   00000000
002540 TO THE NEXT LINE ADDRESS CONTAINS 00000000
002780   00000000   00000000   00000000   00000000     FF150007   42005DF2   620050A2   00005F28
0027A0   00005EF8   0000FF08   00005A68   920053D2     00005C88   62002D40   0000FF08   0000001F
0027C0   B2005D5E   00005E42   00000008   0000FF78     0000001E   00006089   00005870   0000032A
0027E0   421F9B61   00000000   4E000000   0000001F     BE158000   00000000   92FB4264   45504206
002800   05F09823   F02258D0   F01E41F0   F00250F0     D00407F2   00002814   06E2C1D4   D7F24040
002820   0000282C   00002986   00002974   C9C7D5D6     00002804   E340D3C1   FF0029DE   00005E48
002840   D9C540C5   D5E3D9E8   40E2E3C1   E3C5D4C5     D5E340D4   C9E2E2C9   D5C74B40   D3C1C2C5
002860   D340C240   C9D5E2C5   D9E3C5C4   4B404040     E340C6C9   0000282C   00003824   0000481C
002880   00005814   D6C3C5C4   E4D9C540   00002A5C     00002980   00002A04   E6C9D5C7   00002A28
                  -I = 999   -K         -J         -IBCOM       -SQRT
0028A0   000003E7   0000001F   00000003   00002A98     00005E48   8000296C   80002974   800028E0
0028C0   80002898   021A34F1   C6D6D3D3   D6E6C9D5     C740C9E2   40C140D3   C9E2E340   D6C640D7
0028E0   D9C9D4C5   40D5E4D4   C2C5D9E2   40C6D9D6     D440F240   E3D640F1   F0F0F01E   18131A01
002900   F21E1813   1A01F322   02101422   021A1F40     E3C8C9E2   40C9E240   E3C8C540   C5D5C440
002920   D6C640E3   C8C540D7   D9D6C7D9   C1D422D9     00000000   00000000   00000000   00000000
002940   4E000000   000003E7   4E000000   0000001F     4E000000   00000000   00000000   00000000
                                         -Initial Value I
002960   00000000   00000000   00000005   433E7000     00000003   58D0D004   58E0D00C   982CD01C
002980   92FFD00C   07FE58F0   D08018CD   18D445E0     F04018DC   58F0D080   45E0F004   00000003
0029A0   000028C4   58F0D080   45E0F010   5800D13C     5000D074   5800D074   10105010   D1186800
0029C0   D1146A00   D1041200   05E047B0   E0062300     7000D140   4110D088   58F0D084   05EF4700
0029E0   00042822   38206E20   D1246020   D11C5800     D1202222   05E047B0   E0061300   5000D078
002A00   5800D144   5000D07C   5800D074   8E000020     5D00D07C   5C00D07C   5B10D074   13115910
```

Figure 4.10. FORTRAN Sample Program (Part 8 of 8)

Debugging RPG Programs

The documentation for debugging RPG problem programs is the source statement listing, the linkage editor map and a core dump. The memory map of the source statement listing gives the displacement of a given routine or pointer from the beginning of the RPG program and can be used with the linkage editor map to locate the DTF tables, I/O areas, etc., in main storage.

A discussion of Halt Analysis is included to help determine the cause of an H0 (internal halt) condition. In addition, Figure 4.11 shows how to find the DTF pointer containing the address of the DTF table of address pointers. Each address pointer is four bytes and points to a DTF table. Within the DTF table, the programmer can locate the CCB that points to the CCW chain, that in turn, points to the I/O areas.

A sample source listing, linkage editor map and a dump of main storage that identifies all program areas can be found in Figure 4.13.

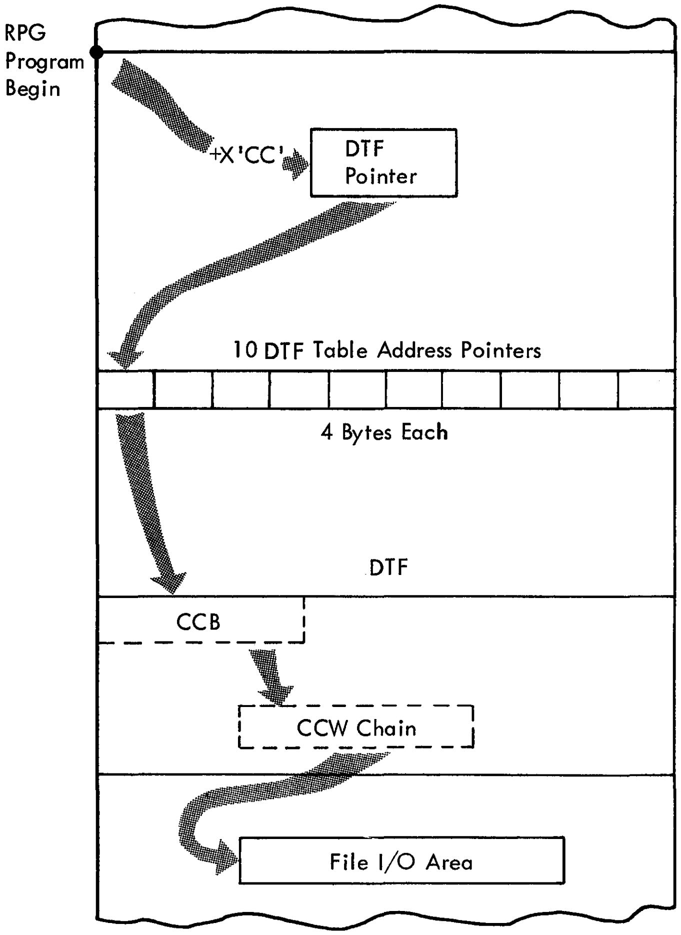


Figure 4.11. Using RPG Pointers to DTF

HALT ANALYSIS

During the execution of an RPG object program, the job is canceled if a halt indicator is turned on and it is not turned off before reading the next input record. Before the job is canceled, RPG checks the dump option switch in the communications region. If the switch is on, a PDUMP of the problem program area is given before job cancelation. The message "JOB CANCELED DUE TO PROGRAM REQUEST" is written on SYSLOG whenever a halt indicator (H0-H9) is left on.

When a RPG object program is canceled with the preceding message, it must be determined which of the halt indicators is on. If one or more of the indicators H1-H9 are on, the programmer must determine the reason. For further information on the halt indicators H1-H9, refer to RPG Specifications listed in the Preface.

If H0 (RPG's internal halt indicator) is on, RPG determines the reason. When H0 is on, a 7-byte area in main storage is initialized by RPG to indicate what caused the halt zero. This area is located at a displacement of X'11C' from the beginning of the RPG object program. Figure 4.12 shows the possible contents of the 7-byte halt zero analysis area.

CAUSES OF A HALT ZERO CONDITION

The object program:

- Read an input record that was not defined on the Input Specifications sheet (columns 21-41).

Displacements in Hexadecimal From Register 3	11C	120	121	122
Condition That Turned H0 On	*	Resulting Byte Combinations Set (Hexadecimal)		
Initialized on or is on due to programmer request	N/A	00	00	00
Invalid chaining request	(A)	02	N/A	N/A
Undefined record type	(B)	10	N/A	N/A
Collating sequence error (matching records)	N/A	04	N/A	N/A
Record sequence error (predetermined sequence)	N/A	08	N/A	N/A
DAM (record not found)	(C)	N/A	80	N/A
DAM (data check)	(C)	N/A	40	N/A
DAM (wrong length record)	(C)	N/A	20	N/A
ISAM (invalid key length)	(B)	N/A	N/A	FF
ISAM (DASD error)	(C)	N/A	N/A	80
ISAM (wrong length record)	(C)	N/A	N/A	40
ISAM (illegal EOF--within limits)	(C)	N/A	N/A	20
ISAM (duplicate record)	(C)	N/A	N/A	04
ISAM (no record found)	(C)	N/A	N/A	10

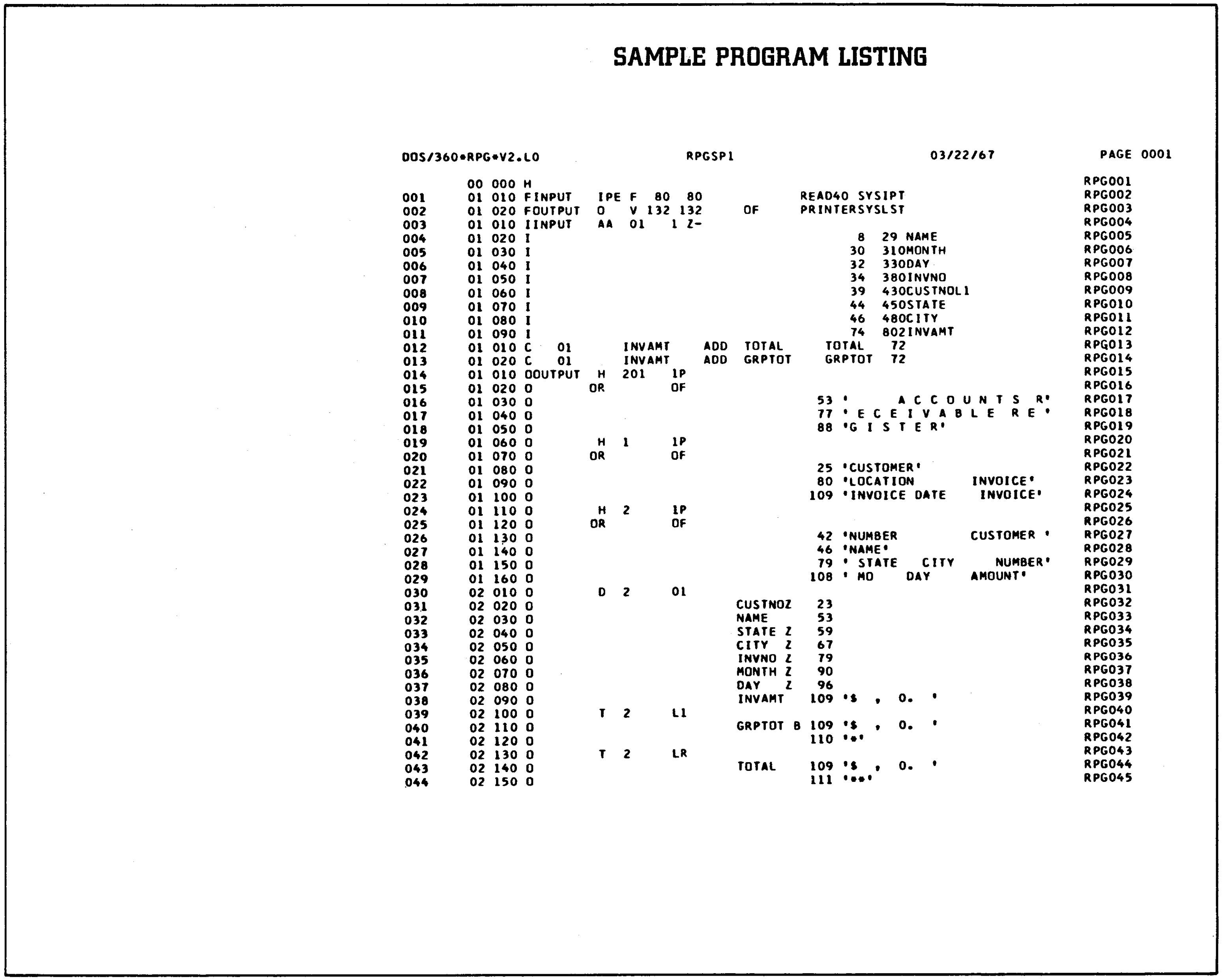
*
(A) = Chaining identifier
(B) = Address of IORB
(C) = Address of DTF table
N/A = Not applicable

All Files (120)
DAM Files Only (121)
ISFMS Files Only (122)

Figure 4.12. Halt Indicator (H0) Analysis Aid

- Found an input record out of the predetermined sequence of card type specified by the entry in Sequence (columns 15-16) on the Input Specifications sheet.
- Found an input record out of sequence when the entry in Matching Fields (columns 61-62) on the Input Specifications sheet was used for sequence checking a single input file.
- Encountered a chaining field in the chaining file that does not appear in the chained file during random processing of multiple input files.
- Did not find a record with the correct key at the designated track address during random processing by record key of a DAM file.
- Did not find the record key that designates the lower limit (obtained from the RAF) during sequential processing between limits of an indexed sequential file.
- Found a wrong length record during processing of an indexed sequential file.
- Found an invalid length record (zero or too long) during random processing by record identification of a file on a DASD.
- Found a difference between the key length of a DASD record in an indexed sequential file and the length as specified in Length of Record Address Field (columns 29-30) on the File Description Specifications sheet during processing with RAF support (random, ADDROUT, or between limits).
- Found a difference between the key length in the chained indexed sequential file and the length as specified (columns 44-51) on the Input Specifications sheet during chaining of multiple input files.
- Encountered a data check on the DASD during random processing of a DAM file.
- Encountered a DASD error during sequential or random processing of an indexed sequential file.

Note: Unless the H0 indicator is turned off by a SETOF Operation entry on the Calculation Specifications sheet, the program terminates before the next input record is read.



SAMPLE PROGRAM LISTING

```
DOS/360*RPG*V2.LO                  RPGSP1                         03/22/67             PAGE 0001

        00 000 H                                                                     RPG001
001     01 010 FINPUT   IPE F  80  80             READ40 SYSIPT                      RPG002
002     01 020 FOUTPUT  O   V 132 132     OF      PRINTERSYSLST                      RPG003
003     01 010 IINPUT   AA  01    1 Z-                                               RPG004
004     01 020 I                                         8  29 NAME                  RPG005
005     01 030 I                                        30  310MONTH                 RPG006
006     01 040 I                                        32  330DAY                   RPG007
007     01 050 I                                        34  380INVNO                 RPG008
008     01 060 I                                        39  430CUSTNOL1              RPG009
009     01 070 I                                        44  450STATE                 RPG010
010     01 080 I                                        46  480CITY                  RPG011
011     01 090 I                                        74  802INVAMT                RPG012
012     01 010 C   01      INVAMT    ADD  TOTAL     TOTAL    72                      RPG013
013     01 020 C   01      INVAMT    ADD  GRPTOT    GRPTOT   72                      RPG014
014     01 010 OOUTPUT  H  201   1P                                                  RPG015
015     01 020 O       OR        OF                                                  RPG016
016     01 030 O                                    53 '      A C C O U N T S  R'    RPG017
017     01 040 O                                    77 ' E C E I V A B L E  R E '    RPG018
018     01 050 O                                    88 'G I S T E R'                 RPG019
019     01 060 O        H  1     1P                                                  RPG020
020     01 070 O       OR        OF                                                  RPG021
021     01 080 O                                    25 'CUSTOMER'                    RPG022
022     01 090 O                                    80 'LOCATION       INVOICE'      RPG023
023     01 100 O                                   109 'INVOICE DATE    INVOICE'     RPG024
024     01 110 O        H  2     1P                                                  RPG025
025     01 120 O       OR        OF                                                  RPG026
026     01 130 O                                    42 'NUMBER         CUSTOMER '    RPG027
027     01 140 O                                    46 'NAME'                        RPG028
028     01 150 O                                    79 ' STATE   CITY     NUMBER'    RPG029
029     01 160 O                                   108 ' MO     DAY     AMOUNT'      RPG030
030     02 010 O        D  2     01                                                  RPG031
031     02 020 O                          CUSTNOZ   23                               RPG032
032     02 030 O                          NAME      53                               RPG033
033     02 040 O                          STATE Z   59                               RPG034
034     02 050 O                          CITY  Z   67                               RPG035
035     02 060 O                          INVNO Z   79                               RPG036
036     02 070 O                          MONTH Z   90                               RPG037
037     02 080 O                          DAY   Z   96                               RPG038
038     02 090 O                          INVAMT   109 '$  ,  0.  '                  RPG039
039     02 100 O        T  2     L1                                                  RPG040
040     02 110 O                          GRPTOT B 109 '$  ,  0.  '                  RPG041
041     02 120 O                                   110 '*'                           RPG042
042     02 130 O        T  2     LR                                                  RPG043
043     02 140 O                          TOTAL    109 '$  ,  0.  '                  RPG044
044     02 150 O                                   111 '**'                          RPG045
```

Figure 4.13. RPG Sample Program (Part 1 of 10)

SAMPLE PROGRAM LISTING (CONTINUED)

```
        DOS/360*RPG*V2.LO                    RPGSP1                        03/22/67            PAGE 0002
                                                          SYMBOL  TABLES
RESULTING  INDICATORS

ADDRESS RI       ADDRESS RI       ADDRESS RI       ADDRESS RI       ADDRESS RI       ADDRESS RI       ADDRESS RI

 000011 OF        000014 1P        000015 LR        000016 00        000017 01        00007A L0        00007B L1
 000085 H0        000086 H1        000087 H2        000088 H3        000089 H4        00008A H5        00008B H6
 00008C H7        00008D H8        00008E H9
FIELD  NAMES

ADDRESS FIELD          ADDRESS FIELD          ADDRESS FIELD          ADDRESS FIELD          ADDRESS FIELD

000123  NAME           000139  MONTH          00013B  DAY            00013D  INVNO          000140  CUSTNO
000143  STATE          000145  CITY           000147  INVAMT         00014B  TOTAL          00014F  GRPTOT

LITERALS

ADDRESS LITERAL                          ADDRESS LITERAL                          ADDRESS LITERAL

000153        A C C O U N T S  R         000168   E C E I V A B L E  R E          000183  G I S T E R
00018E  CUSTOMER                         000196  LOCATION       INVOICE
LITERALS

ADDRESS LITERAL                          ADDRESS LITERAL                          ADDRESS LITERAL

0001AC  INVOICE DATE    INVOICE          0001C3  NUMBER         CUSTOMER          0001DB  NAME
0001DF   STATE    CITY     NUMBER        0001F7   MO     DAY    AMOUNT
LITERALS

ADDRESS LITERAL                          ADDRESS LITERAL                          ADDRESS LITERAL

00020C    --,--/.--                      000217  *                                000218  **


                                                MEMORY MAP

INPUT/OUTPUT INTERCEPT                                          000220
TABLE (INPUT AND OUTPUT)                                        00021C
DETERMINE RECORD TYPE                                           000464
DATA SPECIFICATION                                              000248
GET INPUT RECORD                                                00078C
DETAIL CALCULATIONS                                             000904
TOTAL CALCULATIONS                                              000958
DETAIL LINES                                                    000AD2
TOTAL LINES                                                     00096C
INPUT/OUTPUT REQUEST BLOCKS POINTER                             00126C
LOCATION OF DTF TABLE POINTERS                                  000D18
```

Figure 4.13. RPG Sample Program (Part 2 of 10)

SAMPLE PROGRAM LISTING (CONTINUED)

```
     DOS/360*RPG*V2.LO                 RPGSP1                         03/22/67          PAGE 0003

INPUT/OUTPUT INTERFACE ROUTINES                          000DE0
WORK AREA POINTER                                        001574
OVERFLOW BYPASS                                          000ACA
CONTROL LEVEL                                            00063C
TABLE(ASSEMBLE 4)                                        000BAC
TEST ZONE (BCD)                                          0012AC
OVERFLOW LINES                                           0009EA
LINKAGE PROGRAM                                          0013E4


PROGRAM LENGTH  001699

'END OF COMPILATION'
```

Figure 4.13. RPG Sample Program (Part 3 of 10)

SAMPLE PROGRAM LINKAGE EDITOR MAP

```
JOB  RPGSP1    03/22/67   DISK LINKAGE EDITOR DIAGNOSTIC OF INPUT

ACTION TAKEN     MAP
LIST           INCLUDE       IJCFCIZO                                          RPG00114
LIST           INCLUDE       IJDFYPZZ                                          RPG00115
LIST   ENTRY




03/22/67   PHASE   XFR-AD  LOCORE  HICORE  DSK-AD   ESD TYPE  LABEL     LOADED  REL-FR

        PHASE***   0033E4  002000  00378B  25 6 1   CSECT     RPG001    002000  002000

                                                    CSECT     IJCFCIZO  0036A0  0036A0
                                                    *  ENTRY  IJCFZIZO  0036A0

                                                    CSECT     IJDFYPZZ  003710  003710
                                                    *  ENTRY  IJDFYZZZ  003710
```

Figure 4.13. RPG Sample Program (Part 4 of 10)

SAMPLE PROGRAM'S OUTPUT

```
                      A C C O U N T S   R E C E I V A B L E   R E G I S T E R

CUSTOMER                                LOCATION     INVOICE       INVOICE  DATE      INVOICE
 NUMBER       CUSTOMER NAME            STATE   CITY   NUMBER          MO     DAY      AMOUNT

 10712       AMALGAMATED CORP           33      61     11603          11     10   $    389.25
                                                                                  $    389.25*

 11315       BROWN WHOLESALE            30     231     12324          12     28   $    802.08
 11315       BROWN WHOLESALE            30     231     99588          12     14   $    261.17
                                                                                  $ 1,063.25*

 11897       FARM IMPLEMENTS            47      77     10901          10     18   $     27.63
                                                                                  $     27.63*

 18530       BLACK OIL                  16      67     11509          11      8   $    592.95
```

DUMP TAKEN HERE

Figure 4.13. RPG Sample Program (Part 5 of 10)

CORE DUMP

```
          RPGSP1               03/22/67             PAGE    1

GR 0-7    0000354C 00003544 000001E0 00002000   00003000 00004000 00005000 00006000
GR 8-F    50002B90 00003574 00000001 0000208E   00002085 400033E6 50003452 00002AD2
002000    058058D8 000607FD 000033E4 00000000   00000000 0000F000 00000000 00000000   00000000 00000000 00000000 00000000
002030    00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000
002060    00000000 00000000 00000000 00000000   00000000 00000000 0000F000 00000000   00000000 00F00000 00000000 00000000
002090    00000000 00002220 00002000 00002000   0000221C 00002000 00002000 00002464   00002248 0000278C 00002904 00002958
0020C0    00002AD2 0000296C 0000326C 00002D18   00002DE0 00002000 00002000 00002000   00002000 00006000 00003574 00007000
0020F0    00008000 00009000 0000A000 00002ACA   00002000 00002000 0000263C 00002BAC   000032AC 000029EA 00000000 0000326C
002120    100000C4 C1E3C140 C3C1D9C4 40404040   40404040 40404040 40000C00 0C00000C   00000C00 0C000C00 00000C02 07308C00
002150    59295C40 40404040 40C140C3 40C340D6   40E440D5 40E340E2 4040D940 C540C340   C540C940 E540C140 C240D340 C54040D9
002180    40C540C7 40C940E2 40E340C5 40D9C3E4   E2E3D6D4 C5D9D3D6 C3C1E3C9 D6D54040   40404040 40C9D5E5 D6C9C3C5 C9D5E5D6
0021B0    C9C3C540 C4C1E3C5 40404040 C9D5E5D6   C9C3C5D5 E4D4C2C5 D9404040 40404040   4040C3E4 E2E3D6D4 C5D940D5 C1D4C540
0021E0    E2E3C1E3 C5404040 C3C9E3E8 40404040   40D5E4D4 C2C5D940 D4D64040 4040C4C1   E8404040 4040C1D4 D6E4D5E3 40402020
002210    6B202021 4B20205C 5C5C0000 07FE0000   05E050C0 F01E18C0 D500F022 C00D58C0   F01E58F3 0098078F 58F300D0 07FF0000
002240    000032AC 4000022C 05805810 803E1211   078E5881 00041288 078E1A83 589300C8   4A910002 58290000 41900FFF 41989001
002270    05F847F0 F09647F0 F0C647F0 F0DE47F0   F11847F0 F1240700 80002720 41AC0002   41900003 9500A000 4780F034 D201F032
0022A0    A0009200 000041AA 00024690 F0229500   C0064780 F05444B0 F0904770 F05441CC   000647F0 F07C41CC 00029500 C0004780
0022D0    F06844B0 F0904720 F07C9500 C0024780   F08644B0 F09047A0 F08641CC 0002D201   F084C000 96F00000 41980FFF 41990001
002300    07FAF800 10001000 44B0F0C0 189B8890   00041A19 910D1000 4710F082 94F01000   960C1000 1B1918A9 41980FFF 41990001
002330    07FCF200 1000A000 180C45C0 F096189A   18C088B0 000489B0 000416B9 47F0F01A   189B8890 00041AA9 D200F105 A000910D
002360    A0004710 F0FC960F A00094FC A0001BA9   44B0F112 1AA99200 A0004198 0FFF4199   000107FC F8001000 A000180C 45C0F0DE
002390    18C047F0 F01AD201 F12CC002 92000000   95401000 4770C004 41AC0004 12BB4780   F07806B0 44B0F154 4780F078 41980FFF
0023C0    41990001 07FAD500 10001001 D2153123   20074110 313941A0 201D41B0 001145CF   00004110 313B41A0 201F41B0 001145CF
0023F0    00004110 313D41A0 202141B0 002445CF   00004110 314041A0 202641B0 002445CF   00004110 314341A0 202B41B0 001145CF
002420    00004110 314541A0 202D41B0 001245CF   00004110 314741A0 204941B0 003645CF   000007FE 18530C40 18530C40 F2249000
002450    202694FC 900245BC 000007FE 92003017   07FE4110 58D300E8 50E00004 508D0008   58A300B4 582A0008 1A2341B0 201A4210
002480    F027DC00 F0272000 41A00000 581AB000   1A135893 00C84A91 000641F0 F06605EF   58F1000C 12FF4780 E0181AF3 58C30108
0024B0    58AC00DC 1AA305EF 58A300B0 501A0040   58ED0004 588D0008 07FE50ED 000C5829   0000D705 F01AF01A D7031010 10109047
```

Figure 4.13. RPG Sample Program (Part 6 of 10)

CORE DUMP (CONTINUED)

```
         RPGSP1             03/22/67          PAGE    2

0024E0  D02805E1 00000000 01009847 D028188E   18909500 80044780 8026D200 900F8004   9204900E 58F30094 05EF5890 80481A93
002510  47F0804C 41800FFF 41548001 41658001   41768001 41878001 07F4B006 00000448   58F08000 12FF4780 806ED703 80608060
002540  1AF305EF B0044040 D6028060 80604740   8076960F 100047F0 81369045 D0284140   00034150 00441BFF 180943F4 805F12FF
002570  478080C8 06F04450 81484790 80C0182F   41220001 88200001 89200004 16F244F0   814288F0 0004419F 900094FC 90001B9F
0025A0  47F080C4 44F0813C 419F9001 44508148   478080DA 069094FC 90004199 00018850   000118CB 18BA4640 80869845 D0281B90
0025D0  06901880 47F08100 181B58F3 011805EF   581D0024 91F01000 47C0812C 91041014   4780811C 4490814C 47B0812C 47F08124
002600  4490814C 47D0812C 92F03085 92043120   92F01000 44908152 189058ED 000C07FE   D2009000 C000F200 9000C000 91008063
002630  D5008000 1018D200 1018B000 D502A000   90004780 B000D202 A0009000 D200307B   811407FB D5FFA003 90034780 B000D2FF
002660  A0039003 D201307B 811407FB D5FFA003   90034780 B000D2FF A0039003 D202307B   811407FB D5FFA003 90034780 B000D2FF
002690  A0039003 D203307B 811407FB D5FFA003   90034780 B000D2FF A0039003 D204307B   811407FB D5FFA003 90034780 B000D2FF
0026C0  A0039003 D205307B 811407FB D5FFA003   90034780 B000D2FF A0039003 D206307B   811407FB D5FFA003 90034780 B000D2FF
0026F0  A0039003 D207307B 811407FB D5FFA003   90034780 B000D2FF A0039003 D208307B   811407FB B0080000 00000444 4511001C
002720  BF180000 000003CC 00002085 00000000   E2C9C7D5 0D000100 454E0030 58C30110   43902000 41A00060 05BC4770 404041A0
002750  30179201 E00496F0 A00050A1 000847F0   4030B00A 000003CC 0000044C D2031004   4028D203 100C402C 47F0E006 41A30085
002780  50A10008 9201E004 47F0E006 058047F0   800EB00A E7D7D9C5 0000045C 181D58D3   00E8501D 000850ED 0004D708 307B307B
0027B0  47F08126 58C300C8 4AC10002 582C0000   5890807A 1A9358A1 000896F0 A0009585   100B4770 805050C3 011C9210 312058A3
0027E0  00B0501A 004058F1 000C12FF 47808070   1AF358C3 010858AC 00DC1AA3 05EF58ED   000458DD 000807FE 00000448 580300C8
002810  4A010006 189050ED 001858F0 800A1AF3   05FF9200 900E58F3 009405EF 95F03085   477080B4 58A300B0 D703A040 A04047F0
002840  80709522 90134770 80EA58ED 00184111   000492F0 10019108 1014078F 48C0816A   46C080E4 96F03015 D208307B 811447F0
002870  807040C0 816A07FE 508D0014 58F300AC   41FF0066 05EF588D 001458A1 00089200   A00058ED 001847F0 80264770 802607FE
0028A0  800CF0F0 F0F0F0F0 F0F0F0F0 0700B006   0000071C 58108122 1A1345E0 807E47F0   80D658ED 001C95F0 1001077E 92F08164
0028D0  95F08165 478080D6 41F000AA 40F0817E   95B08112 078E5810 816647F0 82300700   B00C0000 00000000 00010004 41F08180
002900  92A305EF 05805893 00E890DE 90F498AD   30EC95F0 30174770 802CD732 90009000   F8739008 3147FA73 9008314B F837314B
002930  900895F0 30174770 804CD732 90009000   F8739008 3147FA73 9008314F F837314F   900898DE 90F407FE 05805893 00E890DE
002960  90F498AD 30EC98DE 90F407FE 582030C8   58A030E8 90DEA000 921CA00D 9240A00E   9240A00F 9240A065 58D03094 58C3010C
002990  058095F0 307B4770 80289202 20325890   C0101A93 05F94002 00289202 202E4100   202005ED D2003011 20330580 95F03015
0029C0  47708028 92022032 5890C014 1A9305E9   40020028 9202202E 41002020 05EDD200   30112033 98DEA000 07FE5820 30C858A0
0029F0  30E890DE A000921C A00D9240 A00E9240   A00F9240 A06558D0 309458C3 010C58B0   30FC9200 A00CD600 A00C300C D600A00C
```

Figure 4.13. RPG Sample Program (Part 7 of 10)

CORE DUMP (CONTINUED)

```
          RPGSP1              03/22/67           PAGE    3

002A20   300DD600 A00C300E D600A00C 300FD600   A00C3010 D600A00C 3011D600 A00C3012   D600A00C 3013078B 058095F0 30114770
002A50   802C9201 202F9202 20325890 C0001A93   05E94002 00289202 202E4100 202005ED   D2003011 20330580 95F03011 47708028
002A80   92012032 5890C004 1A9305E9 40020028   9202202E 41002020 05EDD200 30112033   058095F0 30114770 80289202 20325890
002AB0   C0081A93 05E94002 00289202 202E4100   202005ED D2003011 20330580 98DEA000   07FE5820 30C858A0 30E890DE A000921C
002AE0   A00D9240 A00E9240 A00F9240 A06558D0   309458C3 010C9200 20330580 95F03014   47708026 9201202F 92022032 5890C000
002B10   1A9305E9 40020028 9202202E 41002020   05ED0580 95F03014 47708022 92012032   5890C004 1A9305E9 40020028 9202202E
002B40   41002020 05ED0580 95F03014 47708022   92022032 5890C008 1A9305E9 40020028   9202202E 41002020 05ED0580 95F03017
002B70   47708022 92022032 5890C00C 1A9305E9   40020028 9202202E 41002020 05ED0580   47F0800E 96F08001 98DEA000 07FED200
002BA0   30112033 98DEA000 07FE800A 000008C4   00000BE2 00000C00 00000C24 00000CB8   00000CE6 059058B0 2020D217 B01D3153
002BD0   D217B035 316BD20A B04D3183 41000058   07FE0590 58B02020 D207B011 318ED215   B03A3196 D216B056 31AC4100 006D07FE
002C00   059058B0 2020D217 B01231C3 D203B02A   31DBD217 B03731DF D214B057 31F74100   006C07FE 059058B0 2020D205 A010A100
002C30   DE05A010 3140D204 B017A011 D215B01F   3123D203 A010A100 DE03A010 3143D201   B039A012 D203A010 A100DE03 A0103145
002C60   D202B040 A011D205 A010A100 DE05A010   313DD204 B04AA011 D203A010 A100DE03   A0103139 D201B058 A012D203 A010A100
002C90   DE03A010 313BD201 B05EA012 D20AA020   320CDE0A A0203147 925BA021 D209B063   A0214100 006D07FE 059058B0 2020D20A
002CC0   A020320C DE0AA020 314F925B A021D209   B063A021 FB33314F 314FD200 B06D3217   4100006E 07FE0590 58B02020 D20AA020
002CF0   320CDE0A A020314B 925BA021 D209B063   A021D201 B06D3218 58C0310C 4100006F   07FE0004 41F08180 00002D70 00002DA8
002D20   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   00000000 00003014 00003084 00002000
002D50   00002000 00002000 00002000 00002000   00002000 00003188 00002000 95F03017   00008000 08000001 00002D90 00002D98
002D80   000036A0 0281C2C2 00003219 00003188   23003219 20000050 415E0000 0700544A   00009200 301A9200 00008400 0C000003
002DB0   00002DD0 00002DD8 00003710 08B00000   00003195 00000000 0700415E 00000000   11003195 20000084 01003195 30000001
002DE0   900FF1F0 05805500 81CA4780 80A65500   81CE4780 815A5823 00CC1840 1B664360   400D0660 89600002 1A265812 000058F1
002E10   00105823 00CC4160 81BA4172 002CD500   40186000 47808058 D5006000 81C24780   81B65A60 81C65A70 81DA47F0 80385877
002E40   000007F7 980F81EA 07FE41D0 00FE06C0   95C84004 47708076 41550050 92405000   59C081DE 47408092 44D080A0 5BC081DE
002E70   41550100 47F08076 12CC4780 809E06C0   44C080A0 07FED200 50015000 45E08180   45E0818C 5A1081F6 50120000 18C14110
002EA0   81A25030 81B245E0 81925830 81B25812   00005851 00085855 00009508 40184770   80E0585C 006C9522 40184770 80EC585C
002ED0   00C09520 40184770 80F8585C 00589510   40184770 8104585C 00709540 40184770   8110585C 00844155 00005050 400048C0
002F00   400845E0 80644122 0004D503 200081D2   47808136 41440020 46F080AA 4100000A   582300CC 4122002C 58120000 5A1081F6
002F30   50120000 41220004 46008142 47F0805E   45E08180 45E0818C 18C14110 81AA45E0   81924122 0004440C    44780 817C46F0
```

Figure 4.13. RPG Sample Program (Part 8 of 10)

CORE DUMP (CONTINUED)

```
          RPGSP1               03/22/67            PAGE   4
002F60    815E47F0 805E5843 00C85823 00CC41F0    000A5812 000007FE 50C0819A 4500819E    00002DA8 0A0207FE 5B5BC2D6 D7C5D540
002F90    5B5BC2C3 D3D6E2C5 00002000 47F081B6    01020408 10202240 FF000000 00000001    D6D7C5D5 C3D3D6E2 00000000 00000004
002FC0    00000004 00000100 00FFFFFF FFFFFFFF    0000326C 80002720 00003219 00002000    00003000 00004000 00005000 00006000
002FF0    400024E4 0000326C 00002085 4000274A    000032AC 00003574 6000250A 00002DE0    9202200E 9500400E 47807038 41E04018
003020    1B004300 400F1AE0 430E0000 9502400E    47807040 95CB4004 4780805E 9503E000    4780805E 45EF0000 47F0805E 45EF0008
003050    47F0805E 58504000 95CB4004 47807066    06504205 000045EF 000C5850 400048C0    400845E0 806447F0 805E45EF 000047F0
003080    7052A020 419070F0 41B0400F 50F070EC    95FF400F 47807024 95E34010 47807024    45A0709A 45A070BC 45A070B2 95FF4010
0030B0    47807040 95E34010 47807040 45A0709A    45A070BC D6004013 40149200 401445A0    70B295FF 40114780 705E45A0 709A45A0
0030E0    70BC45A0 70B295FF 40124780 707A95FF    40114770 709245A0 709A45A0 70BCD203    400F81E6 48C04008 065041CC 000145E0
003110    806447F0 805E4190 70F447F0 70721B00    430B0000 89000003 56090000 950CB000    478070E4 07FA4199 000441BB 000107FA
003140    58504000 06504205 00004155 000148C0    400841CC 000158F0 70EC45EF 000C4100    70E445EF 000407FA 92F04014 07FA0000
003170    00003710 00000083 00000003 00000081    00000001 8B28D21A 05709222 401347F0    805E91D0 40404040 40404040 40404040
0031A0    40404040 40404040 40404040 40404040    40404040 40404040 40404040 40404040    40404040 40404040 40404040 40404040
          --SAME--
003200    40404040 40404040 40404040 40404040    40404040 40404040 40C2E4C7 40F2F540    C4C1E3C1 40C3C1D9 C4404040 40404040
003230    40404040 40404040 40404040 40404040    40404040 40404040 40404040 40404040    40404040 40404040 40404040 40404040
003260    40404040 40404040 40C6C9D3 00003219    00000000 00500000 00010401 FFFFFF00    00000011 01014181 C0000000 00003195
003290    00000000 006D0000 000202FF FFFFFF00    00110011 02000000 00000000 9202C030    4290C02E 42A0C02F CC01C02E C0351B99
0032C0    4390C030 4199C031 D401C02E 9000D500    C02EC02F 9200C030 07FB1020 000F0FF0    F00A0A0A 0A0A0A0A 0A0A0A0A 0A0A0A0A
0032F0    0A0A0A0A 0A0A0A0A 0A0A0A0A 0A0A0A0A    0A0A0A0A 0A0A0A0A 0A0A0A0A 0A0A0A0A    0A0A0A0A 0A0A0A0A 0A0A0A0A 0A0A0A0A
003320    0A4A0A0A 0A0A0A0A 0A0A0A0A 0A0A0A0A    0A1A0A0A 0A0A0A0A 0A0A0A0A 0A0A0A0A    0A2A0A0A 0A0A0A0A 0A0A0A0A 0A0A0A0A
003350    0A0A0A0A 0A0A0A0A 0A0A0A0A 0A0A0A0A    0A0A0A0A 0A0A0A0A 0A0A0A0A 0A0A0A0A    0A0A0A0A 0A0A0A0A 0A0A0A0A 0A0A0A0A
003380    0A0A0A0A 0A0A0A0A 0A0A0A0A 0A0A0A0A    0A0A0A0A 0A0A0A0A 0A0A0A0A 0A0A0A0A    0A101112 13141516 1718190A 0A0A0A0A
0033B0    0A202122 23242526 2728290A 0A0A0A0A    0A0A0A32 33343536 3738390A 0A0A0A0A    0A404142 43444546 4748490A 0A0A0A0A
0033E0    0A003AE9 05D0185D 48200016 D201D168    2008D203 D16A2028 5820D16F 89500008    88500008 41220002 1B521825 18324140
003410    00944160 00044170 01145854 30001A52    50543000 8746D034 9847D172 1A421A52    1A621A72 5800D182 58F300D0 05EF58F3
003440    00A05873 00E405EF 587300E4 58F300C0    05EF9400 301441C3 008541A0 000141B3    008E95F0 C0004780 D13887CA D07C5873
003470    00E4D200 D1583084 58F30084 05EFD200    01593084 D2003084 D15895F0 30154780    D0C495F0 307B4770 D0D89101 D1574710
```

Figure 4.13. RPG Sample Program (Part 9 of 10)

CORE DUMP (CONTINUED)

```
          RPGSP1            03/22/67           PAGE   5

0034A0   D0C49601 D15747F0 D0D85873 00E458F3   00BC05EF 587300E4 58F300C4 05EF95F0   30154780 D1149102 D1574780 D0F258F3
0034D0   01145873 00E405EF 58F300B0 9602D157   587300E4 05EFD200 3084D159 58F300B8   587300E4 05EF47F0 D06291F0 30144710
003500   D10658F3 00A05873 00E405EF 5800D186   58F300D0 587300E4 05EF1800 0A0E4820   00169140 203B4780 D14E4100 D1664110
003530   D15E0A02 9206D137 47F0D114 03030000   8D9A4020 5B5BC2D7 C4E4D4D7 00002000   0000378C 000013E4 00001000 00002000
003560   00003000 00004000 D6D7C5D5 C3D3D6E2   00709540 400033E6 50003452 400033E6   001C4040 404040F8 5000278E A00028BE
003590   00000000 405B4040 40F5F9F2 00003000   00004000 00005000 00006000 4122002C   58120000 5A1081F6 50120000 41220004
0035C0   46008142 47F0805E 45E08180 45E0818C   18C14110 81AA45E0 81404122 00044400   81244780 817C46F0 815E47F0 805E5843
0035F0   00C85823 00CC41F0 000A5812 00C007FE   50C0819A 4500819E 00000000 0A0207FE   5B5BC2D6 D7C5D540 5B5BC2C3 D3D6E2C5
003620   00000000 47F081B6 01020408 10202240   FF000000 00000001 D6D7C5D5 C3D3D6E2   00000000 00000004 00000004 00000100
003650   00FFFFFF FFFFFFFF 00000000 00002018   50003AB2 00004299 400033E6 600034F6   00002010 40202020 20202020 20202020
003680   20202020 20202020 20202020 20222120   22212020 20202040 4002FFFF 9500400E   47F0F04E 0A320000 47F0F010 0A320000
0036B0   D2001020 1017D200 10171016 0A009180   10024710 F0280A07 50E0F048 58E01020   D501F046 E0004770 F04058E0 101C07FE
0036E0   58E0F048 07FE615C 80003050 00224A00   F04C9180 10024710 F05C0A07 42001020   0A009180 10024710 F06C0A07 07FE70F0
003710   0A320000 47F0F02E 0A320000 90CEF070   58E01018 06E0D200 1028E000 98CEF070   0A009180 10024710 F02C0A07 07FE9180
003740   10024710 F0380A07 91011015 4780F04E   94FE1015 91021003 078E47F0 F0549101   1004078E 12004780 F05E18F0 07FF928B
003770   10280A00 91801002 4710F06E 0A0707FE   0000006E 00002220 9000315E
```

Figure 4.13. RPG Sample Program (Part 10 of 10)

Debugging PL/I Programs

PL/I is made up of subroutines, a control program, and a mainline routine. The mainline routine manipulates the data to develop information for a particular application. The control program initializes the mainline routine. It also provides linkage to subroutines to get the data from a file for use by the mainline routine and to create new files of updated information. The control program uses transients much the same as the supervisor, and it has its own transient area (Figure 4.14).

The PL/I storage areas important for debugging purposes are: static storage, dynamic storage area (DSA), and library work space. Figure 4.15 shows the contents of these areas, when assigned, pointers to these areas, and where each area is located. Static storage contains all of the information needed by a PL/I program; literals, address constants, control blocks and block descriptions. Figure 4.16 shows a list of subroutines that are moved into the control program to provide needed functions to the mainline.

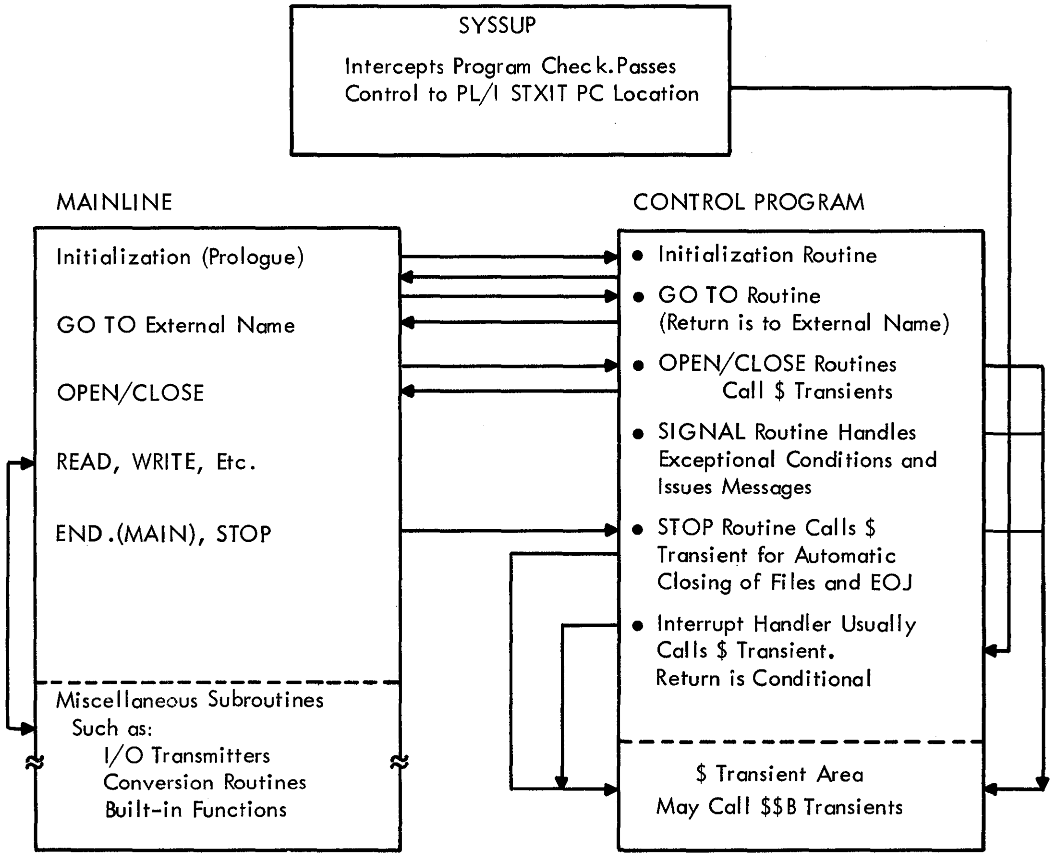


Figure 4.14. PL/I Program Structure

	Static	Dynamic	Library Work Space
Contents	Literals, Address Constants, Control Blocks, Block Descriptions.	Linkage register and AUTOMATIC data save areas.	Register save area and work area for PL/I subroutines.
When Assigned	At Compile Time - Determine size, contents and location.	At Compile Time - Determine size and location of data within DSA. At Object Time - Determine actual location when procedure is activated.	At Linkage Editor Time - Part of PL/I control program is AUTOLINKed regardless of need.
How Pointed to	Register 12 when procedure is active.	Register 13 when procedure is active. Register 11 Used with Register 10 nested procedures.	Register 13 when subroutine is active.
Where located	Doubleword boundary immediately following procedure.	Doubleword boundary immediately following highest core location (taken from COMRG).	IJKZWSA from linkage editor map.

Figure 4.15. PL/I Storage Areas

Part of the PL/I prologue moves a list of subroutine address constants from static storage to an entry point table within the PL/I control program.

This table contains 16 positional entries as follows:

Displacement	Position	Subroutine	Description
Source			
0	0	IJKVBCM (28)	Converts fixed binary to intermediate.
4	1	IJKVTCM (29)	Converts float to intermediate.
8	2	IJKVPCM (2A)	Converts fixed decimal to intermediate.
C	3	IJKVFCM (2B)	Converts numeric field float to intermediate.
10	4	IJKVECM (2C)	Converts 'E' or 'F' format to intermediate.
14	5	IJKVGIM (2D)	Converts character string to bit string.
18	6	IJKVIGM (2E)	Converts bit string to character string.
1C	7	IJKTSTR (2F)	X format item, PAGE, SKIP,
20	8	IJKTLCM (30)	LINE, COLUMN
Target			
0	9	IJKVCBM (31)	Converts intermediate to fixed binary.
4	A	IJKVCTM (32)	Converts intermediate to float.
8	B	IJKVCPM (33)	Converts intermediate to fixed decimal.
C	C	IJKVCFM (34)	Converts intermediate to numeric field float.
10	D	IJKVCEM (35)	Converts intermediate to 'E' or 'F' format.
Sysfiles			
14	E	IJKSYSI (36)	SYSIN
18	F	IJKSYSA (37)	SYSPRINT

Figure 4.16. Entry Point Table

Figure 4.17 shows the layout of core storage at object time. The PL/I problem procedure (mainline coding), the control program, and the library subroutines are shown. These areas can be located by using the linkage editor map.

Figure 4.17 also shows the dynamic storage areas that are loaded directly behind the rest of the PL/I program. Because they are not on the linkage editor map, the programmer must look in bytes 40-43 (decimal) of the system communications region and find the high-core address of the program to locate the first DSA. Remember that the communication region has two ending address entries (one entry for the last phase loaded and another entry for the ending address of the program). The ending address of the program is the entry the programmer must use.

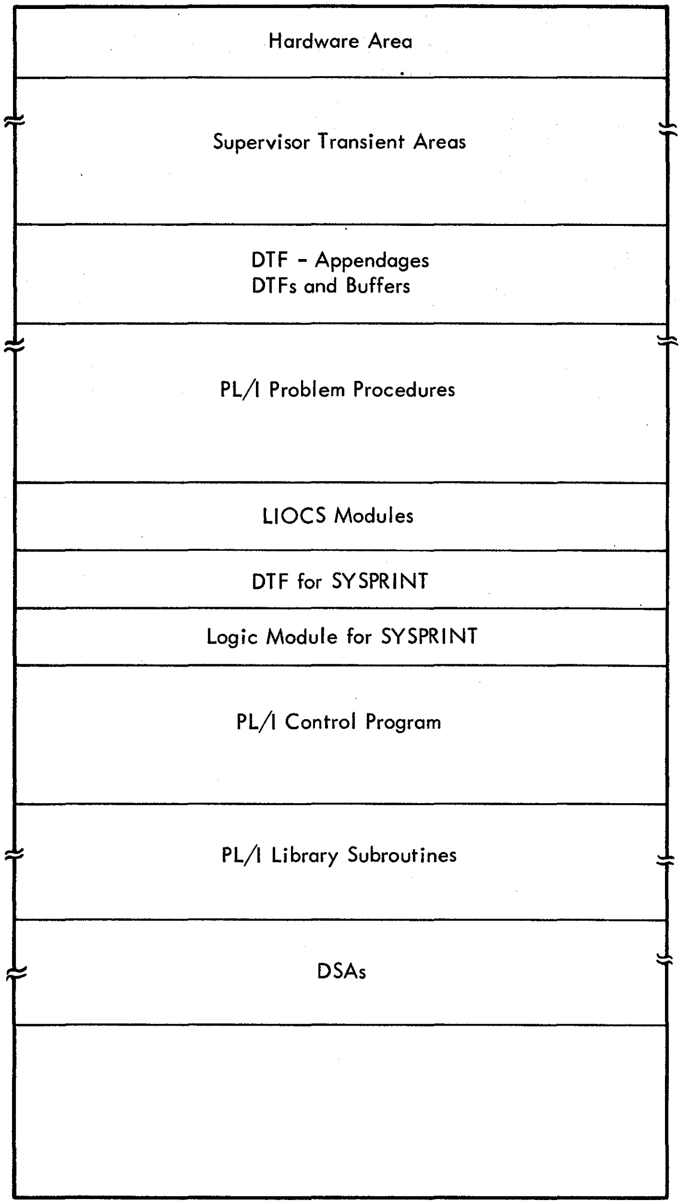


Figure 4.17. Object Time Core Usage

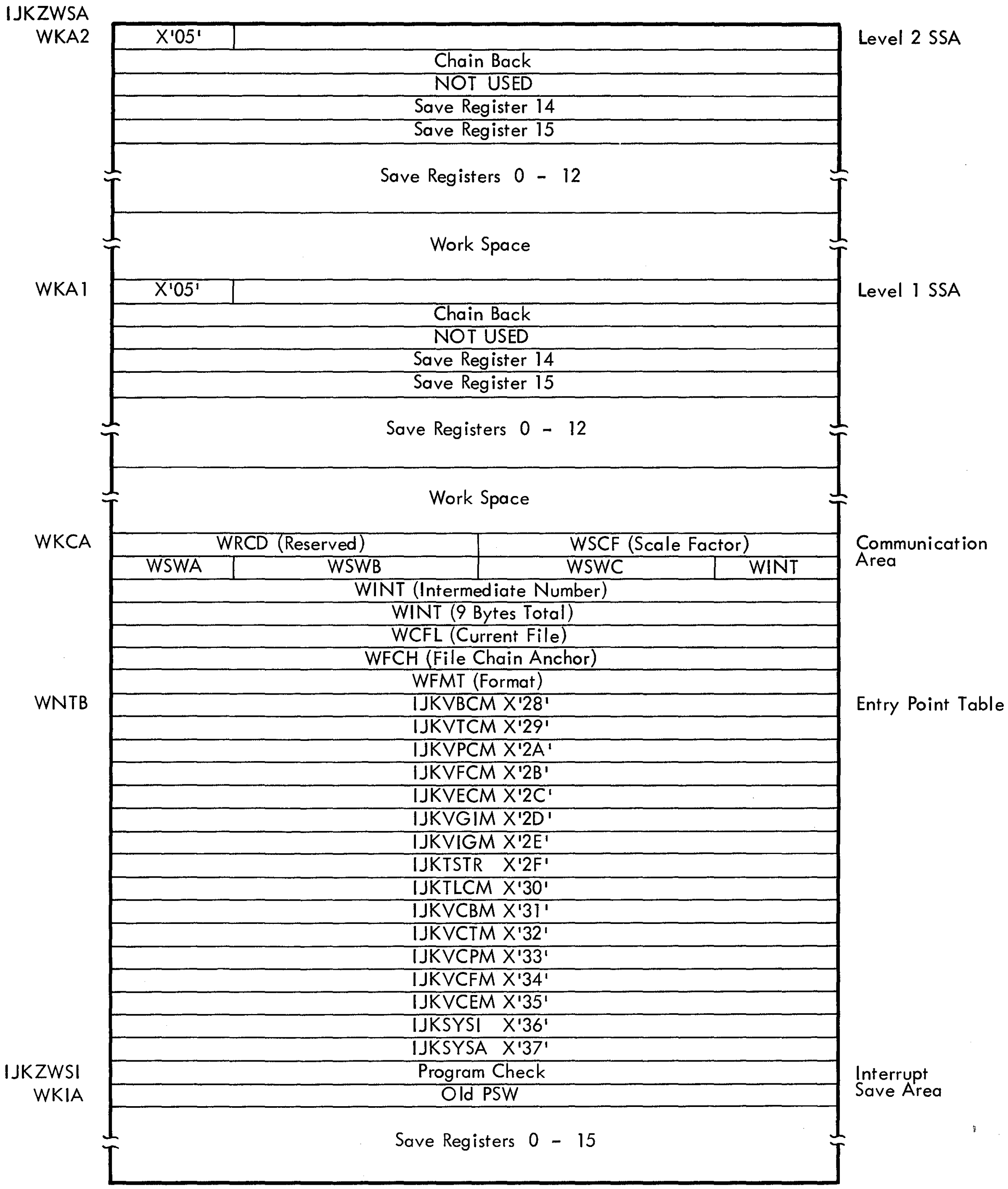


Figure 4.18. Library Work Space

Figure 4.18 shows an example of library storage that contains work space, save areas, and pointers to the DSA (chain back). A communications region follows the work areas. Figure 4.19 shows some of the information stored in this area.

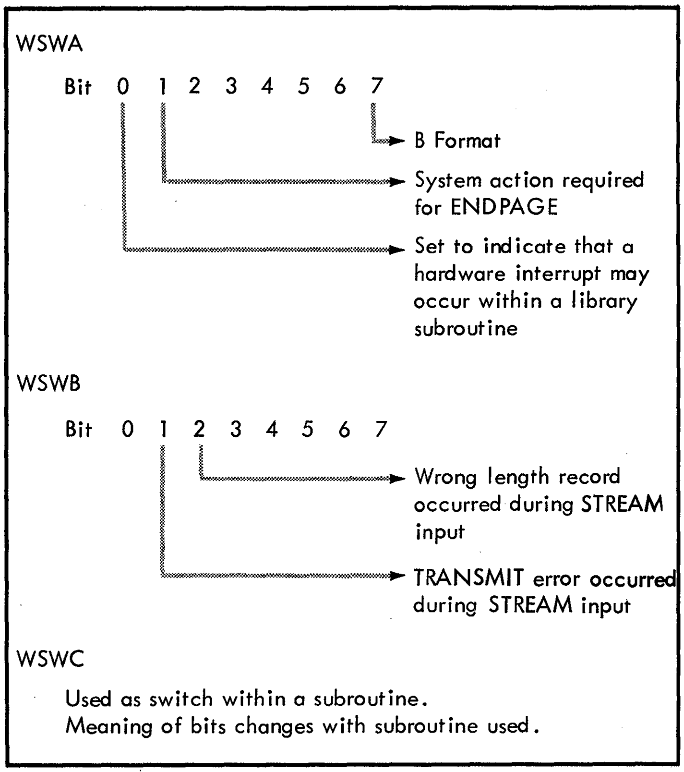


Figure 4.19. Communications Area Switches

Figure 4.20 shows the layout of the DSA. Each procedure has a DSA containing the following information:

1. The condition of the DSA, indicated by the flag bytes.

2. A block description in static storage pointed to by the last 3 bytes of the first word (see Figure 4.21).

3. The chain forward and back addresses that allow the programmer to follow the program flow from procedure to procedure.

A dummy DSA is built to indicate the beginning of a chain of DSAs when backward chaining. Figure 4.20 shows the dummy DSA. Figure 4.22 gives an example of DSA chaining.

Dummy DSA

Word	Content	
1	Flags X'00'	Invocation Count
2	A(End of Core)	
3	Chain Forward A(DSAMAIN)	
4	Bit 0 PL/I Dump	AL3(STOP Routine)
5	Program Mask Default X'0E000000'	

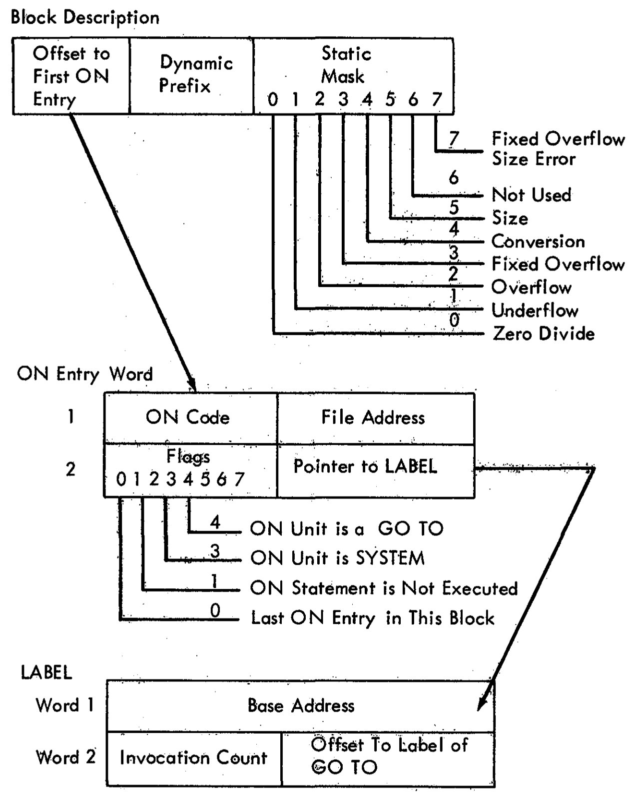
DSA Layout

Word	Content	
1	Flags *	AL3(Block Description)
2	Chain Back Address	
3	Chain Forward Address	
4	Return Address (R14)	
5	Entry Point (R15)	
6	Save Area For	
18	Registers 0 - 12	
19	Invocation Count	
20	Register 0 of Calling Block	
	Dynamic Storage For AUTOMATIC Data	

* Flag Byte

X'00' - Dummy DSA
X'01' or X'81' - ON Entries
X'03' or X'83' - No ON Entries
X'05' - Library Work Space

Figure 4.20. Dummy DSA and DSA Layout



ON Code	Meaning	Program Interrupt
X'01'	Overflow	Exponent Overflow
X'02'	Underflow	Exponent Underflow
X'03'	Zero Divide	Floating Point Divide
X'04'	Fixed Overflow	Decimal/Fixed Point Overflow
X'05'	Size	
X'06'	Conversion	
X'09'	Error	
X'0A'	Endfile	
X'0B'	Endpage	
X'0C'	Transmit	
X'0D'	Key	
X'0E'	Record	

Figure 4.21. Block Description

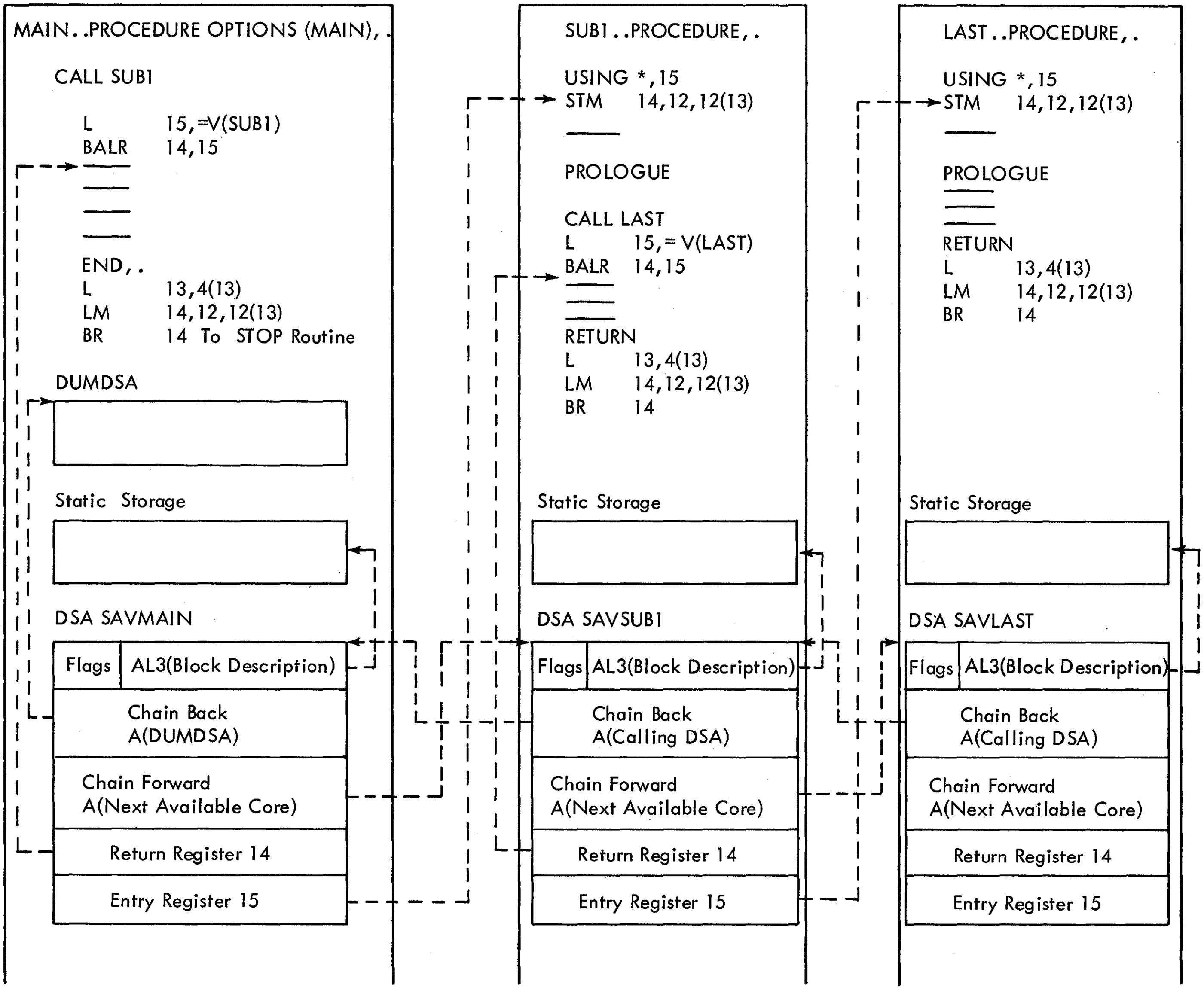


Figure 4.22. DSA Chaining

When debugging PL/I problem programs, the programmer must know how to locate the DTF tables. In PL/I, the DTF table has an appendage that precedes the DTF proper and contains a pointer to the DTF (Figure 4.23). The appendage can easily be found because it is listed as the linkage editor map by filename. Figure 4.24 shows the format of the DTF used by PL/I. A sample program, linkage editor map and core dump are included in Figure 4.25 to aid the programmer in debugging his PL/I problem program.

Word	Contents		
1	Open Mask	AL3(DTF-T)	
2	Flag 1	Chain Address	
3	Flag 2	Communications Byte	Record Length
4	Maintenance Word		
5	Buffer Address		
6		Relative Pointer	
7	Page Size	Current Line	Print Files

Bit	TOPM Open Mask	TFL1 Flag Byte 1	TFL2 Flag Byte 2	TXRR Communications Byte
0	File Closed		Stream	First Time
1	Input	Clear Open Mask	Consecutive	Transmit Error
2	Output	00 - Fixed 01 - Variable 11 - Undefined	Regional	Wrong Length Record
3	Update		0	Still to Write
4	Update Next Must Be a Read	Print File	Direct	Still to Wait For A Regional File
5	File Has Keys	Allow Page Size	Sequential	
6	File Has No Keys	Backwards	Unbuffered	EOF
7	No Buffers	Not End of Chain Address Table	Buffered	System File

Note: Meaning if Bit = 1

Figure 4.23. PL/I Consecutive File DTF-A Appendage

Word				
1	Open Mask	AL3(DTF-T)		
2	Flag 1	Chain Address		
3	Flag 2	Communica-tions Byte	Record Length	
4	Maintenance Word			
5	Record Address			
6	00-Regional 1 08-Regional 3	AL3(Disk Address Routine)		
7	Address of Key Field - Regional 3			
8	Number of Records/Track - Regional 1			
9	Logical Unit Set at Open		Error Bytes for LIOCS	
10	Key Length 00 - Regional 1 Length - Regional 3		▨	M Module
11	B Bin	B Bin	C Cylinder	C Cylinder
12	Head	Head	R Record	▨
13	C	C	H	H
14	Number of Tracks		▨	
15	C	C	H	H
16	Number of Tracks		▨	
17	C	C	H	H
18	Number of Tracks		▨	

Figure 4.24. PL/I Regional File DTF-A

SUMMARY OF PL/I DEBUGGING AIDS

1. Register 13 (at the start of the dump) points to a library work space or dynamic storage area.

2. Always chain back from the library work space or DSA to find the active DSAs.

3. The DSA (main) can be located by using the value contained in the COMREG at a decimal displacement 40-43. Adjust this value to a doubleword boundary.

4. If the PL/I error signaled is a program check, use the PSW and registers located in IJKZWSI. This area has a PSW followed by the registers stored R0 through R15.

5. The PL/I control program intercepts and tries to handle all errors except machine checks. The control program issues a PL/I error message, and a main storage dump is executed from the control program. Following the dump, PL/I closes all files. If a printed output is produced at the time of error, PL/I prints the last item after the dump is taken. This gives the programmer a starting point to the correct area of the failure.

6. PL/I program check error message codes 11 through 1E are the same as the interrupt codes 01 through 0E on the IBM S/360 Reference Data card, GX20-1703.

7. PL/I error messages point to the approximate location where an error occurred.

8. The name of the last PL/I transient fetched is at the label 'CIJKS'. To find the labels CIJKS and DISPL (item 9), use register 13 and the chain back fields to locate the dummy DSA (identified by its first byte being zero). Scan backward approximately four fullwords and locate X'0A04' (load instruction). Following the load is X'47F1xxxx', which is labeled DISPL. Immediately following this is CIJKS. (PL/I (D) transients are of the format 'IJKSxx '. DISPL and CIJKS are in the routine IJKSZCA.)

9. The branch instruction that activates a transient just fetched is at the label 'DISPL'.

10. To locate the current file, look in 'WCFL' in the library work space. This contains a request code and pointer to a DTF-A. To find WCFL, use the label IJKZWSI in the linkage editor map. Start in the core dump at that location and scan backward until you locate a fullword beginning with X'FE'. The fullword immediately preceding this is the current file address, labeled WCFL. (WCFL is in the routine IJKSZCA.)

11. The first word of any DTF-A points to a logical IOCS DTF.

12. The linkage editor map has a CSECT for each DTF-A.

13. PL/I register usage is as follows:

    a. The instruction flow base registers in a procedure are 13 (11 and 10) for dynamic storage and 12 for static storage.

    b. The data accessing base registers in a procedure are 13 (11 and 10) for dynamic storage and 12 for static storage.

    c. The instruction flow base registers in a PL/I subroutine are 15 and 12.

    d. The data accessing base registers in a PL/I subroutine are 13 and 11.

    e. The parameter passing registers are 0 through 5. (Register 1 is the most commonly used.)

14. The following language aids are useful in debugging both source and object problems.

    a. Dyndump

    b. Display

    c. Display using the reply option (to halt a program)

    d. Null labels

    e. Signal statement (to force dump)

Note: In multiphase programs, the first four characters of the phase names must be identical and unique to the program. If they are not, either the DSA is overlaid or the DSA may not fit into the available core.

HANDLING COMPILE TIME ABORTS

The following pointers within the control program indicate how far the compiler progressed before the abort condition occurred:

1. Register 12 points to the start of the control program and is used as the base register.

2. The KSAVE1 area contains return registers in the following order:

   R 14 - points to last active routine
   R 15
   R 0
   R 1
   R 2

   To locate KSAVE1, add X'D8' to the contents of register 12.

3. The K5PH area (8 bytes) contains the name of the phase now in storage. The phase name is constructed in the following manner:

   PL/Ixxxx

   The last four bytes xxxx contain the actual phase identifier such as D75. To locate K5PH, add X'284' to the contents of register 12.

Exceptions: D00, D05, D10. During these three phases, the phase name can be located by adding X'108' to the contents of register 12. If K5PH contains phase C95, the actual phase may be either C95 or D11.

4. IJKZWSI is valid only if a PSW has been stored there.

```
        DOS PL/I COMPILER  360N-PL-464 CL2-0           CHAIN               06/06/66            PAGE 001

/*   EXAMPLE OF THREE EXTERNAL PROCEDURES PASSING PARAMETER */

                    /*   EXAMPLE OF THREE EXTERNAL PROCEDURES PASSING PARAMETER */
                    /* THIS IS THE MAIN PROCEDURE IT PASSES (CTR) TO PROC SUB1 */
    1               STAHT..        PROCEDURE OPTIONS (MAIN),.
    2                       DECLARE
                                  INCARD FILE INPUT RECORD ENVIRONMENT (F(75) MEDIUM
                                              (SYSIPT,2540)),
                                  OTCARD FILE OUTPUT RECORD ENVIRONMENT (F(80) MEDIUM
                                              (SYSPCH,2540)),
                                  CARDIN CHARACTER (75),
                                   1 CARDOUT,
                                     2 FIRST75 CHARACTER (75),
                                     2 SEQCTR PICTURE '99999',.
    3                       DECLARE CTR PICTURE '99999',.
    4               OPEN..         OPEN FILE (INCARD),.
    5               OPN..         OPEN FILE (OTCARD),.
    6               C..           CALL SUB1(CTR),.
    7               READ..         READ FILE (INCARD) INTO (CARDIN),.
    8               ONE..         ON ENDFILE (INCARD)  GO TO END,.
    9               FIR..          FIRST75=CARDIN,.
   10               SEQ..         SEQCTR=CTR,.
   11               WRITE..       WRITE FILE (OTCARD) FROM (CARDOUT),.
   12               CC..          CALL REPEAT (CTR),.
   13               GOTO..        GO TO READ,.
   14               END..         CLOSE FILE (INCARD),.
   15               END1..        CLOSE FILE (OTCARD),.
   16                       END,.
```

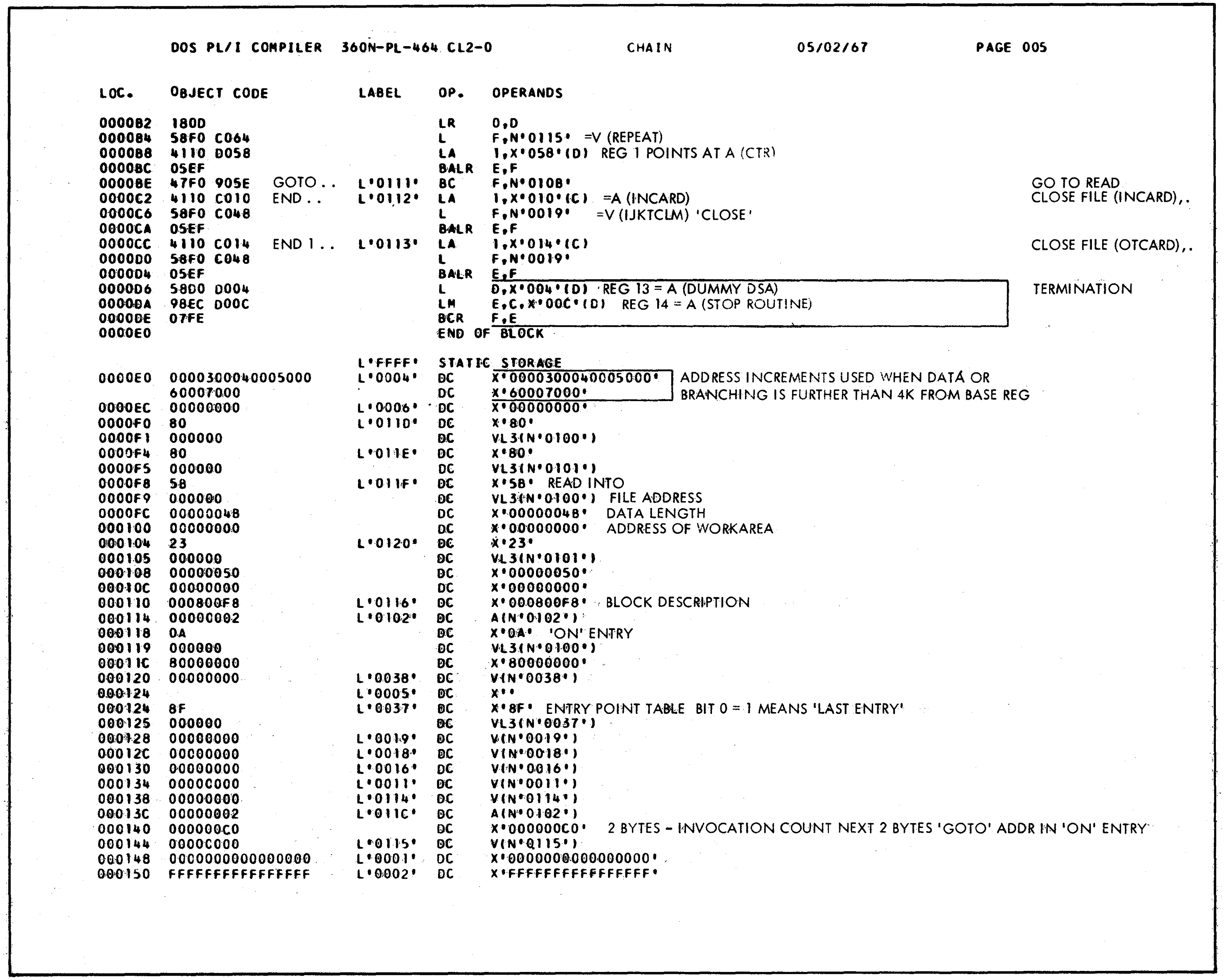
Figure 4.25. PL/I Sample Program (Part 1 of 18)

```
     DOS PL/I COMPILER  360N-PL-464 CL2-0            CHAIN            06/06/66           PAGE 002

                      S Y M B O L   T A B L E   L I S T I N G

  STAHT                       0102 00 0 ENTRY      ARITHM. DECIMAL FLOAT   6             EXT
  INCARD                      0100 01 1            FILE                                  EXT
  OTCARD                      0101 01 1            FILE                                  EXT
  CARDIN                      0106 01 1            STRING  ALIGNED CHAR.  75    AUTOM    INT
  CARDOUT                     0105 01 1 STRUCT.  1         PACKED               AUTOM.   INT
  FIRST75                     0104 01 1 STRUCT.  2 STRING          CHAR.  75             INT
  SEQCTR                      0103 01 1 STRUCT.  2 PICTURE DECIMAL FIXED  5,0            INT
  CTR                         0107 01 1            PICTURE DECIMAL FIXED  5,0   AUTOM.   INT
  OPEN                        0108 01 1            LABEL   CONST.                        INT
  OPN                         0109 01 1            LABEL   CONST.                        INT
  C                           010A 01 1            LABEL   CONST.                        INT
  READ                        010B 01 1            LABEL   CONST.                        INT
  ONE                         010C 01 1            LABEL   CONST.                        INT
  FIR                         010D 01 1            LABEL   CONST.                        INT
  SEQ                         010E 01 1            LABEL   CONST.                        INT
  WRITE                       010F 01 1            LABEL   CONST.                        INT
  CC                          0110 01 1            LABEL   CONST.                        INT
  GOTO                        0111 01 1            LABEL   CONST.                        INT
  END                         0112 01 1            LABEL   CONST.                        INT
  END1                        0113 01 1            LABEL   CONST.                        INT
  SUB1                        0114 01 1 ENTRY      ARITHM. DECIMAL FLOAT   6             EXT
  REPEAT                      0115 01 1 ENTRY      ARITHM. DECIMAL FLOAT   6             EXT




     DOS PL/I COMPILER  360N-PL-464 CL2-0            CHAIN            06/06/66           PAGE 003

      INTERNAL NAME  OFFSET          TYPE          MODULE OFFSET         OFFSET TABLE

      0102           0034            STATIC        000114
      0105           0148            AUTOMATIC
      0106           00F4            AUTOMATIC
      0107           013F            AUTOMATIC
      0114           0058            STATIC        000138
      0115           0064            STATIC        000144
```

Figure 4.25. PL/I Sample Program (Part 2 of 18)

```
      DOS PL/I COMPILER  360N-PL-464 CL2-0               CHAIN            05/02/67           PAGE 004

LOC.    OBJECT CODE            LABEL       OP.    OPERANDS

000000  05F0                               BALR   F,0
000002                 STAHT..  L'0102'    BEGIN  OF BLOCK  01                                       PROLOGUE
000002  0700                               BCR    0,0
000004  45E0 F00A                          BAL    E,X'00A'(F)
000008  000000E0                           DC     A(N'FFFF')  A (STATIC STORAGE)
00000C  58C0 E000                          L      C,X'000'(E)
000010  189F                               LR     9,F   REG 9 ALSO USED AS BASE REG
000012  1831                               LR     3,1   NO MEANING IN A 'MAIN' PROCEDURE
000014  58F0 C054                          L      F,N'0011'  =V (IJKSZCM)                            DSA INITIALIZE
000018  05E0                               BALR   E,0
00001A  41E0 E00E                          LA     E,X'00E'(E)
00001E  051F                               BALR   1,F
000020  01                                 DC     X'01'  DSA 'FLAG BYTE'
000021  000110                             DC     AL3(N'0116')  AL3 (BLOCK DESCRIPTION)
000024  00000198                           DC     LENGTH OF DSA OF BLOCK 01
000028  4110 C044                          LA     1,X'044'(C)  =A (ENTRY POINT TABLE)                ENTRY POINT MOVE
00002C  58F0 C050                          L      F,N'0016'     =V (IJKSZLM)
000030  05EF                               BALR   E,F
000032  D203 D050 3000                     MVC    X'050'(04,D),X'000'(3)  NOT USEFUL IN A 'MAIN' PROCEDURE  IN LINE PROLOGUE
000038  4110 C010      OPEN.. L'0108'      LA     1,X'010'(C)  A (FILE NAME)                         OPEN FILE (INCARD),.
00003C  58F0 C04C                          L      F,N'0018'  =V (IJKTOPM) 'OPEN'
000040  05EF                               BALR   E,F
000042  4110 C014      OPN..  L'0109'      LA     1,X'014'(C)  A (FILE NAME)                         OPEN FILE (OTCARD),.
000046  58F0 C04C                          L      F,N'0018'  =V (IJKTOPM) 'OPEN'
00004A  05EF                               BALR   E,F
00004C  41E0 D13F      C..    L'010A'      LA     E,N'0107'  REG 14 = ADDRESS OF 'CTR'               CALL SUB1 (CTR),.
000050  90EE D058                          STM    E,E,X'058'(D)
000054  180D                               LR     0,D
000056  58F0 C058                          L      F,N'0114'  =V (SUB1)
00005A  4110 D058                          LA     1,X'058'(D)
00005E  05EF                               BALR   E,F
000060  4110 D0F4      READ.. L'010B'      LA     1,N'0106'  REG 1 = ADDRESS OF 'CARDIN'             READ FILE (INCARD) INTO (CARDIN)
000064  5010 C020                          ST     1,X'020'(C)  STORE 1 TO 'WORKAREA ADDRESS' OF 'CONTROL BLOCK'
000068  4110 C018                          LA     1,X'018'(C)  REG 1 = ADDRESS OF 'CONTROL BLOCK'
00006C  58F0 C040                          L      F,N'0038'  =V (IJKTCBM) 'TRANSMITTER'
000070  05EF                               BALR   E,F
000072  4120 C030      ONE..  L'010C'      LA     2,N'0116'  REG 2 = ADDRESS OF BLOCK DESCRIPTION    ON ENDFILE (INCARD) GOTO END,.
000076  4A20 2000                          AH     2,X'000'(2)  INCREMENT REG 2 TO POINT AT 'ON' ENTRY
00007A  D201 C060 D048                     MVC    X'060'(02,C),X'048'(D)  MOVE DSA INVOCATION COUNT TO STATIC STORAGE
000080  4150 C05C                          LA     5,X'05C'(C)  LA 5, =A (STAHT)
000084  5050 2004                          ST     5,X'004'(2)  STORE REG 5 TO 'ON' ENTRY
000088  9288 2004                          MVI    X'004'(2),X'88'  RESTORE FLAG BYTE IN 'ON' ENTRY
00008C  D24A D148 D0F4 FIR..  L'010D'      MVC    N'0105'(4B),N'0106'                                FIRST 75 = CARDIN,.
000092  D204 D193 D13F SEQ..  L'010E'      MVC    N'0105'+4B(05),N'0107'                             SEQCTR = CTR,.
000098  4110 D148      WRITE.. L'010F'     LA     1,N'0105'  REG 1 = ADDRESS OF 'CARDOUT'            WRITE FILE (OTCARD) FROM (CARDOUT),.
00009C  5010 C02C                          ST     1,X'02C'(C)  STORE REG 1 TO 'CONTROL BLOCK'
0000A0  4110 C024                          LA     1,X'024'(C)  REG 1 = ADDRESS OF 'CONTROL BLOCK'
0000A4  58F0 C040                          L      F,N'0038'  =V (IJKTCBM) 'TRANSMITTER'
0000A8  05EF                               BALR   E,F
0000AA  41E0 D13F      CC..   L'0110'      LA     E,N'0107'  REG 14 = ADDRESS OF 'CTR'               CALL REPEAT (CTR),.
0000AE  90EE D058                          STM    E,E,X'058'(D)
```

Figure 4.25. PL/I Sample Program (Part 3 of 18)



```
         DOS PL/I COMPILER  360N-PL-464 CL2-0               CHAIN              05/02/67            PAGE 005

LOC.     OBJECT CODE                LABEL     OP.    OPERANDS

000082   180D                                 LR     0,D
000084   58F0 C064                            L      F,N'0115'  =V (REPEAT)
000088   4110 D058                            LA     1,X'058'(D)  REG 1 POINTS AT A (CTR)
00008C   05EF                                 BALR   E,F
00008E   47F0 905E   GOTO . .     L'0111'     BC     F,N'0108'                                         GO TO READ
0000C2   4110 C010   END . .      L'0112'     LA     1,X'010'(C)  =A (INCARD)                          CLOSE FILE (INCARD),.
0000C6   58F0 C048                            L      F,N'0019'    =V (IJKTCLM) 'CLOSE'
0000CA   05EF                                 BALR   E,F
0000CC   4110 C014   END 1 . .    L'0113'     LA     1,X'014'(C)                                       CLOSE FILE (OTCARD),.
0000D0   58F0 C048                            L      F,N'0019'
0000D4   05EF                                 BALR   E,F
0000D6   58D0 D004                            L      D,X'004'(D)  REG 13 = A (DUMMY DSA)               TERMINATION
0000DA   98EC D00C                            LM     E,C,X'00C'(D)   REG 14 = A (STOP ROUTINE)
0000DE   07FE                                 BCR    F,E
0000E0                                        END OF BLOCK

                                  L'FFFF'     STATIC STORAGE
0000E0   0000300040005000         L'0004'     DC     X'0000300040005000'  ADDRESS INCREMENTS USED WHEN DATA OR
         60007000                             DC     X'60007000'          BRANCHING IS FURTHER THAN 4K FROM BASE REG
0000EC   00000000                 L'0006'     DC     X'00000000'
0000F0   80                       L'011D'     DC     X'80'
0000F1   000000                               DC     VL3(N'0100')
0000F4   80                       L'011E'     DC     X'80'
0000F5   000000                               DC     VL3(N'0101')
0000F8   58                       L'011F'     DC     X'58'  READ INTO
0000F9   000000                               DC     VL3(N'0100')  FILE ADDRESS
0000FC   00000048                             DC     X'00000048'   DATA LENGTH
000100   00000000                             DC     X'00000000'   ADDRESS OF WORKAREA
000104   23                       L'0120'     DC     X'23'
000105   000000                               DC     VL3(N'0101')
000108   00000050                             DC     X'00000050'
00010C   00000000                             DC     X'00000000'
000110   000800F8                 L'0116'     DC     X'000800F8'   BLOCK DESCRIPTION
000114   00000002                 L'0102'     DC     A(N'0102')
000118   0A                                   DC     X'0A'   'ON' ENTRY
000119   000000                               DC     VL3(N'0100')
00011C   80000000                             DC     X'80000000'
000120   00000000                 L'0038'     DC     V(N'0038')
000124                            L'0005'     DC     X''
000124   8F                       L'0037'     DC     X'8F'  ENTRY POINT TABLE  BIT 0 = 1 MEANS 'LAST ENTRY'
000125   000000                               DC     VL3(N'0037')
000128   00000000                 L'0019'     DC     V(N'0019')
00012C   00000000                 L'0018'     DC     V(N'0018')
000130   00000000                 L'0016'     DC     V(N'0016')
000134   0000C000                 L'0011'     DC     V(N'0011')
000138   00000000                 L'0114'     DC     V(N'0114')
00013C   00000002                 L'011C'     DC     A(N'0102')
000140   000000C0                             DC     X'000000C0'   2 BYTES - INVOCATION COUNT NEXT 2 BYTES 'GOTO' ADDR IN 'ON' ENTRY
000144   0000C000                 L'0115'     DC     V(N'0115')
000148   0000000000000000         L'0001'     DC     X'0000000000000000'
000150   FFFFFFFFFFFFFFFF         L'0002'     DC     X'FFFFFFFFFFFFFFFF'
```

Figure 4.25. PL/I Sample Program (Part 4 of 18)

```
        DOS PL/I COMPILER  360N-PL-464 CL2-0              CHAIN              06/06/66            PAGE 006

LOC.    OBJECT CODE           LABEL    OP.    OPERANDS

000158  00FEFFFF              L'000A'  DC     X'00FEFFFF'
00015C  0C                    L'000B'  DC     X'0C'
                                       END
```

```
        DOS PL/I COMPILER  360N-PL-464 CL2-0              CHAIN              06/06/66            PAGE 007

 SYMBOL    TYPE    ESID     ADDR     LENGTH  ESID       EXTERNAL SYMBOL TABLE

INCARD      SD     0001    000000    00009B
IJCFZIZO    ER     0002
IJKTXCF     ER     0003
OTCARD      SD     0004    000000    0000F0
IJCFZOZ4    ER     0005
STAHT       SD     0001    000000    000160
INCARD      ER     0002
OTCARD      ER     0003
SUB1        ER     0004
REPEAT      ER     0005
IJKSZCA     ER     0006
IJKSZCM     ER     0007
IJKSZLM     ER     0008
IJKTOPM     ER     0009
IJKTCLM     ER     000A
IJKSYSA     ER     000B
IJKTCBM     ER     000C
```

```
        DOS PL/I COMPILER  360N-PL-464 CL2-0              CHAIN              06/06/66            PAGE 008

 BLOCK     LENGTH OF DSA      BLOCKTABLE

  01         0198
```

Figure 4.25. PL/I Sample Program (Part 5 of 18)

```
          DOS PL/I COMPILER  360N-PL-464 CL2-0              CHAIN              06/06/66              PAGE 001

/*   EXAMPLE OF THREE EXTERNAL PROCEDURES PASSING PARAMETER */

                         /*   EXAMPLE OF THREE EXTERNAL PROCEDURES PASSING PARAMETER */
                         /* THIS PROCEDURE ACCEPTS (CTR)   ZEROES IT ON THE FIRST ENTRY */
                         /* THEN PASSES IT TO PROCEDURE 'LAST' FOR INCREMENTING */
                         /* ON SUBSEQUENT ENTRIES THIS PROCEDURE ONLY PASSES (CTR) TO 'LAST */
    1                    SUB1..  PROCEDURE (CTR),.
    2                             DECLARE
                                        CTR PICTURE '99999',.
    3                    CT..          CTR=0,.
    4                    CL..          CALL LAST (CTR),.
    5                    RET..         RETURN,.
    6                    REPEAT..       ENTRY (CTR),.
    7                    CC..          CALL LAST (CTR),.
    8                    R..           RETURN,.
    9                    END..        END,.
```

```
          DOS PL/I COMPILER  360N-PL-464 CL2-0              CHAIN              06/06/66              PAGE 002

                         S Y M B O L    T A B L E    L I S T I N G

SUB1                               0100 00 0 ENTRY      ARITHM. DECIMAL FLOAT   6                EXT
CTR                                0101 01 1            PICTURE DECIMAL FIXED  5,0  PARAM.     INT
CT                                 0102 01 1            LABEL   CONST.                           INT
CL                                 0103 01 1            LABEL   CONST.                           INT
RET                                0104 01 1            LABEL   CONST.                           INT
REPEAT                             0105 01 0 ENTRY      ARITHM. DECIMAL FLOAT   6                EXT
CC                                 0106 01 1            LABEL   CONST.                           INT
R                                  0107 01 1            LABEL   CONST.                           INT
END                                0108 01 1            LABEL   CONST.                           INT
LAST                               0109 01 1 ENTRY      ARITHM. BINARY  FIXED  15                EXT
```

```
          DOS PL/I COMPILER  360N-PL-464 CL2-0              CHAIN              06/06/66              PAGE 003

           INTERNAL NAME  OFFSET         TYPE          MODULE OFFSET        OFFSET TABLE

           0100           0014           STATIC        0000DC
           0101           0058           AUTOMATIC
           0105           0018           STATIC        0000E0
           0109           0020           STATIC        0000E8
```

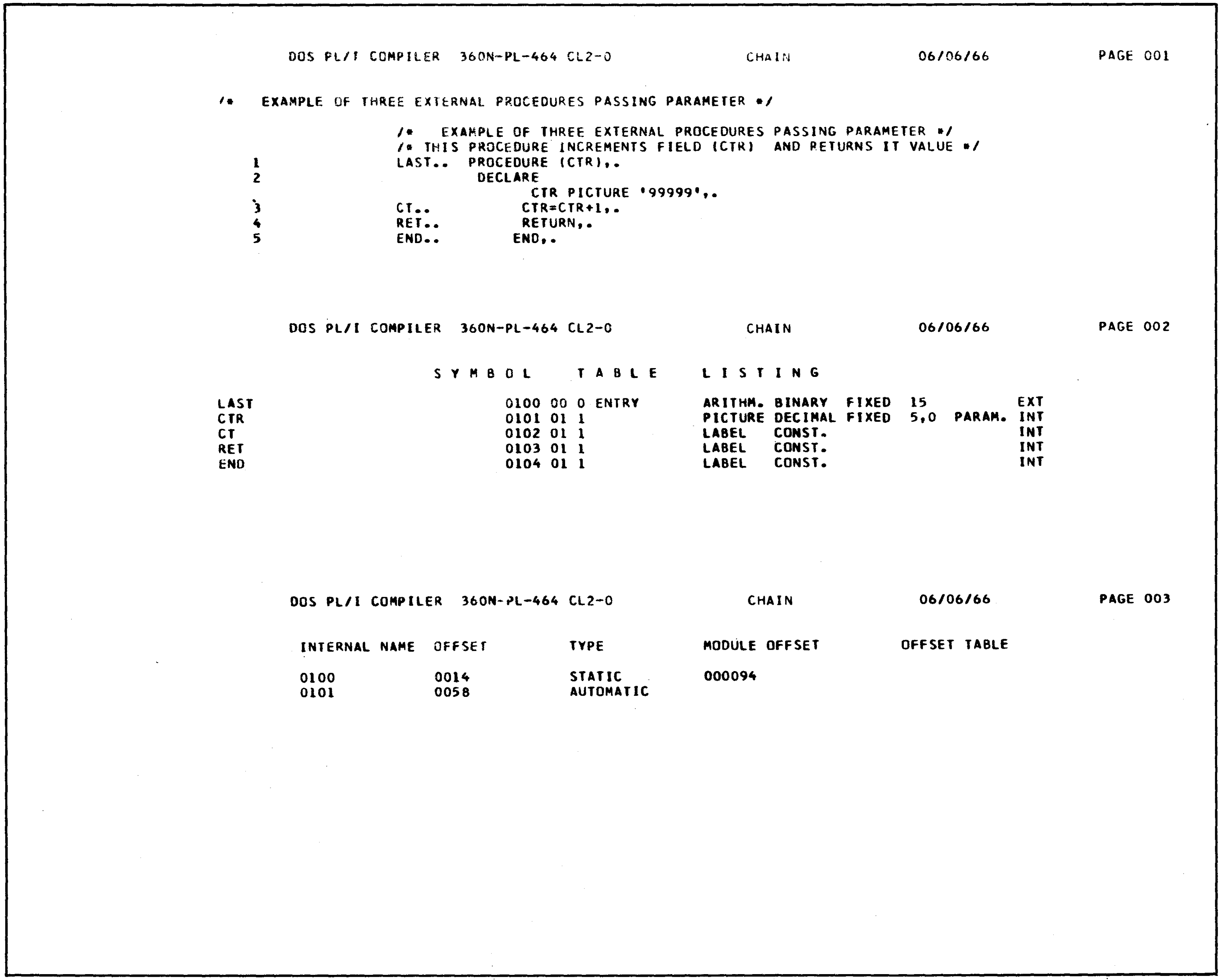
Figure 4.25. PL/I Sample Program (Part 6 of 18)

```
           DOS PL/I COMPILER  360N-PL-464 CL2-0              CHAIN                 06/06/66              PAGE 004

LOC.     OBJECT CODE          LABEL     OP.    OPERANDS

000000                        L'0100'   BEGIN  OF BLOCK  01
000000   90EC D00C                      STM    E,C,X'00C'(D)
000004   45E0 F00C                      BAL    E,X'00C'(F)
000008   000000C8                       DC     A(N'FFFF')
00000C   58C0 E000                      L      C,X'000'(E)
000010   189F                           LR     9,F
000012   1831                           LR     3,1
000014   58F0 C01C                      L      F,N'0012'
000018   05E0                           BALR   E,0
00001A   41E0 E00E                      LA     E,X'00E'(E)
00001E   051F                           BALR   1,F
000020   03                             DC     X'03'
000021   0000D8                         DC     AL3(N'010A')
000024   00000100                       DC     LENGTH OF DSA OF BLOCK 01
000028   D203 D058 3000                 MVC    N'0101'(04),X'000'(3)
00002E   D203 D050 3004                 MVC    X'050'(04,D),X'004'(3)
000034   5860 D058            L'0102'   L      6,N'0101'
000038   D204 6000 C03D                 MVC    X'000'(05,6),X'03D'(C)
00003E   5850 D058            L'0103'   L      5,N'0101'
000042   41E0 5000                      LA     E,X'000'(5)
000046   90EE D060                      STM    E,E,X'060'(D)
00004A   180D                           LR     0,D
00004C   58F0 C020                      L      F,N'0109'
000050   4110 D060                      LA     1,X'060'(D)
000054   05EF                           BALR   E,F
000056   58D0 D004            L'0104'   L      D,X'004'(D)
00005A   98EC D00C                      LM     E,C,X'00C'(D)
00005E   07FE                           BCR    F,E
000060   47F0 909C            L'0112'   BC     F,N'0111'
000064   90EC D00C            L'0105'   STM    E,C,X'00C'(D)
000068   45E0 F00C                      BAL    E,X'00C'(F)
00006C   000000C8                       DC     A(N'FFFF')
000070   58C0 E000                      L      C,X'000'(E)
000074   5890 C024            L'0114'   L      9,X'024'(C)
000078   1831                           LR     3,1
00007A   58F0 C01C                      L      F,N'0012'
00007E   0700                           BCR    0,0
000080   05E0                           BALR   E,0
000082   41E0 E00E                      LA     E,X'00E'(E)
000086   051F                           BALR   1,F
000088   03                             DC     X'03'
000089   0000D8                         DC     AL3(N'010A')
00008C   00000100                       DC     LENGTH OF DSA OF BLOCK 01
000090   D203 D058 3000                 MVC    N'0101'(04),X'000'(3)
000096   D203 D050 3004                 MVC    X'050'(04,D),X'004'(3)
00009C   5860 D058            L'0106'   L      6,N'0101'
0000A0   41E0 6000                      LA     E,X'000'(6)
0000A4   90EE D060                      STM    E,E,X'060'(D)
0000A8   180D                           LR     0,D
0000AA   58F0 C020                      L      F,N'0109'
0000AE   4110 D060                      LA     1,X'060'(D)
```

Figure 4.25. PL/I Sample Program (Part 7 of 18)

```
        DOS PL/I COMPILER  360N-PL-464 CL2-0          CHAIN          06/06/66          PAGE 005

LOC.    OBJECT CODE          LABEL    OP.   OPERANDS

0000B2  05EF                          BALR  E,F
0000B4  58D0 D004            L'0107'  L     D,X'004'(D)
0000B8  98EC D00C                     LM    E,C,X'00C'(D)
0000BC  07FE                          BCR   F,E
0000BE  58D0 D004            L'0108'  L     D,X'004'(D)
0000C2  98EC D00C                     LM    E,C,X'00C'(D)
0000C6  07FE                          BCR   F,E
0000C8                                END OF BLOCK

                             L'FFFF'  STATIC STORAGE
0000C8  0000300040005000     L'0004'  DC    X'0000300040005000'
        60007000                      DC    X'60007000'
0000D4  00000000             L'0006'  DC    X'00000000'
0000D8  000C00F8             L'010A'  DC    X'000C00F8'
0000DC  00000000             L'0100'  DC    A(N'0100')
0000E0  00000064             L'0105'  DC    A(N'0105')
0000E4                       L'0005'  DC    X''
0000E4  00000000             L'0012'  DC    V(N'0012')
0000E8  00000000             L'0109'  DC    V(N'0109')
0000EC  00000000             L'0113'  DC    A(N'0100')
0000F0  0000000000000000     L'0001'  DC    X'0000000000000000'
0000F8  FFFFFFFFFFFFFFFF     L'0002'  DC    X'FFFFFFFFFFFFFFFF'
000100  00FEFFFF             L'000A'  DC    X'00FEFFFF'
000104  0C                   L'0008'  DC    X'0C'
000105  F0F0F0F0F0           L'010C'  DC    X'F0F0F0F0F0'
                                      END


        DOS PL/I COMPILER  360N-PL-464 CL2-0          CHAIN          06/06/66          PAGE 006

 SYMBOL    TYPE   ESID    ADDR    LENGTH  ESID       EXTERNAL SYMBOL TABLE

SUB1        SD    0001   000000   000110
REPEAT      LD           000064           0001
LAST        ER    0002
IJKSZCN     ER    0003


        DOS PL/I COMPILER  360N-PL-464 CL2-0          CHAIN          06/06/66          PAGE 007

 BLOCK      LENGTH OF DSA      BLOCKTABLE

  01          0100
```

Figure 4.25. PL/I Sample Program (Part 8 of 18)



```
          DOS PL/I COMPILER  360N-PL-464 CL2-0              CHAIN               06/06/66            PAGE 001

/*   EXAMPLE OF THREE EXTERNAL PROCEDURES PASSING PARAMETER */

                         /*   EXAMPLE OF THREE EXTERNAL PROCEDURES PASSING PARAMETER */
                         /* THIS PROCEDURE INCREMENTS FIELD (CTR)  AND RETURNS IT VALUE */
     1                   LAST..  PROCEDURE (CTR),.
     2                            DECLARE
                                        CTR PICTURE '99999',.
     3                   CT..           CTR=CTR+1,.
     4                   RET..          RETURN,.
     5                   END..         END,.


          DOS PL/I COMPILER  360N-PL-464 CL2-0              CHAIN               06/06/66            PAGE 002

                              S Y M B O L    T A B L E    L I S T I N G

LAST                               0100 00 0 ENTRY         ARITHM. BINARY  FIXED  15              EXT
CTR                                0101 01 1               PICTURE DECIMAL FIXED  5,0  PARAM.  INT
CT                                 0102 01 1               LABEL   CONST.                      INT
RET                                0103 01 1               LABEL   CONST.                      INT
END                                0104 01 1               LABEL   CONST.                      INT


          DOS PL/I COMPILER  360N-PL-464 CL2-0              CHAIN               06/06/66            PAGE 003

           INTERNAL NAME  OFFSET         TYPE          MODULE OFFSET        OFFSET TABLE

           0100           0014           STATIC        000094
           0101           0058           AUTOMATIC
```

Figure 4.25. PL/I Sample Program (Part 9 of 18)

```
        DOS PL/I COMPILER  360N-PL-464 CL2-0                 CHAIN               06/06/66             PAGE 004

LOC.    OBJECT CODE           LABEL    OP.   OPERANDS

000000                        L'0100'  BEGIN OF BLOCK  01
000000  90EC D00C                      STM   E,C,X'00C'(D)
000004  45E0 F00C                      BAL   E,X'00C'(F)
000008  00000080                       DC    A(N'FFFF')
00000C  58C0 E000                      L     C,X'000'(E)
000010  189F                           LR    9,F
000012  1831                           LR    3,1
000014  58F0 C018                      L     F,N'0012'
000018  05E0                           BALR  E,0
00001A  41E0 E00E                      LA    E,X'00E'(E)
00001E  051F                           BALR  1,F
000020  03                             DC    X'03'
000021  000090                         DC    AL3(N'0105')
000024  00000110                       DC    LENGTH OF DSA OF BLOCK 01
000028  D203 D058 3000                 MVC   N'0101'(04),X'000'(3)
00002E  D203 D050 3004                 MVC   X'050'(04,D),X'004'(3)
000034  5860 D058             L'0102'  L     6,N'0101'
000038  F224 D0E0 6000                 PACK  X'0E0'(3,D),X'000'(5,6)
00003E  F8F0 D060 C035                 ZAP   X'060'(0,D),X'035'(1,C)
000044  FAF2 D060 D0E0                 AP    X'060'(0,D),X'0E0'(3,D)
00004A  F83F D0E8 D060                 ZAP   X'0E8'(4,D),X'060'(0,D)
000050  92F9 C012                      MVI   N'0105'+2,X'F9'
000054  F823 D070 D0E8                 ZAP   X'070'(3,D),X'0E8'(4,D)
00005A  92F8 C012                      MVI   N'0105'+2,X'F8'
00005E  5860 D058                      L     6,N'0101'
000062  F342 6000 D070                 UNPK  X'000'(5,6),X'070'(3,D)
000068  96F0 6004                      OI    X'004'(6),X'F0'
00006C  58D0 D004             L'0103'  L     D,X'004'(D)
000070  98EC D00C                      LM    E,C,X'00C'(D)
000074  07FE                           BCR   F,E
000076  58D0 D004             L'0104'  L     D,X'004'(D)
00007A  98EC D00C                      LM    E,C,X'00C'(D)
00007E  07FE                           BCR   F,E
000080                                 END OF BLOCK

                              L'FFFF'  STATIC STORAGE
000080  0000300040005000      L'0004'  DC    X'0000300040005000'
        60007000                       DC    X'60007000'
00008C  00000000              L'0006'  DC    X'00000000'
000090  000800F8              L'0105'  DC    X'000800F8'
000094  00000000              L'0100'  DC    A(N'0100')
000098                        L'0005'  DC    X''
000098  00000000              L'0012'  DC    V(N'0012')
00009C                                 DS    CL0004
0000A0  0000000000000000      L'0001'  DC    X'0000000000000000'
0000A8  FFFFFFFFFFFFFFFF      L'0002'  DC    X'FFFFFFFFFFFFFFFF'
0000B0  00FEFFFF              L'000A'  DC    X'00FEFFFF'
0000B4  0C                    L'000B'  DC    X'0C'
0000B5  1C                    L'0107'  DC    X'1C'
                                       END
```

Figure 4.25. PL/I Sample Program (Part 10 of 18)

```
          DOS PL/I COMPILER  360N-PL-464 CL2-0              CHAIN              06/06/66          PAGE 005

  SYMBOL    TYPE   ESIC    ADDR     LENGTH  ESID      EXTERNAL SYMBOL TABLE

 LAST        SD    0001   000000   000088
 IJKSZCN     ER    0002




          DOS PL/I COMPILER  360N-PL-464 CL2-0              CHAIN              06/06/66          PAGE 006

  BLOCK      LENGTH OF DSA      BLOCKTABLE

   01          0110




JOB  COMPILE   06/06/66   DISK LINKAGE EDITOR DIAGNOSTIC OF INPUT

ACTION TAKEN    MAP
LIST   PHASE CHAIN,S
LIST   AUTOLINK   IJCFZIZO
LIST   AUTOLINK   IJCFZOZ4
LIST   AUTOLINK   IJKSYSA
LIST   AUTOLINK   IJJCPIN
LIST   AUTOLINK   IJKSZCA
LIST   AUTOLINK   IJKSZLM
LIST   AUTOLINK   IJKTCBM
LIST   AUTOLINK   IJKTXCF
LIST   ENTRY
```

Figure 4.25. PL/I Sample Program (Part 11 of 18)

```
06/06/66   PHASE   XFR-AD   LOCORE   HICORE   DSK-AD    ESD TYPE   LABEL      LOADED   REL-FR

           CHAIN   002190   002000   00307F   30 1 1    CSECT      INCARD     002000   002000

                                                        CSECT      IJCFZIZ0   0024B8   0024B8

                                                        CSECT      IJKTXCF    003008   003008
                                                        *  ENTRY   IJKTXCR    003030
                                                        *  ENTRY   IJKTXCW    003042

                                                        CSECT      OTCARD     0020A0   0020A0

                                                        CSECT      IJCFZ0Z4   002508   002508

                                                        CSECT      STAHT      002190   002190

                                                        CSECT      SUB1       0022F0   0022F0
                                                           ENTRY   REPEAT     002354

                                                        CSECT      IJKSZLM    002DA0   002DA0

                                                        CSECT      IJKSYSA    0025B0   0025B0

                                                        CSECT      IJKTCBM    002DE0   002DE0

                                                        CSECT      LAST       002400   002400

                                                        CSECT      IJKSZCN    002758   002758
                                                           ENTRY   IJKSZCA    002988
                                                           ENTRY   IJKSZCM    002990
                                                           ENTRY   IJKTOPM    0029BA
                                                           ENTRY   IJKTCLM    0029C6
                                                        *  ENTRY   IJKSZCI    0027E6
                                                        *  ENTRY   IJKSZCP    00291A
                                                           ENTRY   IJKSZCS    002816
                                                        *  ENTRY   IJKSZCT    0029AC
                                                           ENTRY   IJKZWSA    002A20
                                                        *  ENTRY   IJKZWSI    002B44

                                                        CSECT      IJJCPIN    002680   002680
                                                        *  ENTRY   IJJCP3     002680
```

Figure 4.25. PL/I Sample Program (Part 12 of 18)

```
FLOATING POINT REGISTERS    0                   2                   4                   6
                            0000000000000000    0000000000000000    0000000000000000    0000000000000000

REG0     00003218   7E002420   000022A8   00003278     00002270   000022CC   0000318F   00001000
REG8     00002000   00002400   00002000   00002000     00002480   00003318   00002428   00002758

40-CSW  KEY-00  ADDR-002040  STATUS-0000100000000000  COUNT-000000     48-CAW  KEY-00  ADDR-002038

PSW CONTENTS        EXTERNAL INTERRUPT  SUPERVISOR CALL     PROGRAM CHECK       MACHINE CHECK       INPUT/OUTPUT
FIELD        FORMAT-OLD 18   -NEW 58   -OLD 20   -NEW 60   -OLD 28   -NEW 68   -OLD 30   -NEW 70   -OLD 38   -NEW 78
SYSTEM MASK     BIT-11111111 -00000000 -11111111 -00000000 -11111111 -00000000 -01011011 -00000000 -11111111 -00000000
PROTECTION KEY  HEX-0        -0        -0        -0        -0        -0        -5        -0        -0        -0
AMWP            BIT-0101     -0100     -0101     -0100     -0101     -0100     -1011     -0000     -0101     -0100
INTERRUPT CODE  HEX-0000     -0000     -0000     -0000     -0007     -0000     -C2C5     -0000     -000C     -0000
INSTR LENGTH    DEC-0        -0        -1        -0        -3        -0        -3        -0        -2        -0
CONDITION CODE  DEC-0        -0        -0        -0        -2        -0        -1        -0        -0        -0
PROGRAM MASK    BIT-0000     -1111     -1110     -0000     -1110     -0000     -0110     -0000     -1110     -0000
INSTR ADDRESS   HEX-000000   -000GC0   -0024CA   -000B02   -00248A   -000C94   -D1F340   -0008F4   -0024CA   -000206

4C-UNUSED-00000000     50-TIMER-FE9912FF     54-UNUSED-00FFB342
```

Figure 4.25. PL/I Sample Program (Part 13 of 18)

```
                                                                                  REMAIN
 DTF-A    MASK        FLAG      CHAIN              RCD       MAINT                IN
'INCARD'      A(DTF-T)          ADDR     FLAGS    LNGTH      WORD     A(BUFFER)   BUFR LNGTH   START OF DTF TABLE
002000    42 002018    01 002AF8    4500 004B    10000000     00002050    004B 004B    00008000    08000001

002020    00002038    00002040    000024B8    02810202     00002050    00003008    02002050    2000004B
                                                           BUFFER
002040    47000000    4700004B    00000000    00000000     C940C1D4    40C140C4    C1E3C140    C3C1D9C4

002060 TO THE NEXT LINE ADDRESS CONTAINS 5C5C5C5C

002080    5C5C5C5C    5C5C5C5C    5C5C5C5C    5C5C5C5C     5C5C5C5C    5C5C5C5C    5C5C5CD7    C1434780
          DTF-A 'OTCARD'
0020A0    220020B8    01002000    45100050    10000000     00002140    00500050    00008400    04000002

0020C0    000020E0    00000000    00002508    04904141     00002140    40404040    07004700    00000040

0020E0    41002140    20000050    010020F0    20000050     40404040    40404040    40404040    40404040

002100 TO THE NEXT LINE ADDRESS CONTAINS 40404040

002140    C940C1D4    40C140C4    C1E3C140    C3C1D9C4     5C5C5C5C    5C5C5C5C    5C5C5C5C    5C5C5C5C

002160 TO THE NEXT LINE ADDRESS CONTAINS 5C5C5C5C
                                                           STAHT
002180    5C5C5C5C    5C5C5C5C    5C5C5CF0    F0F0F0F1     05F00700    45E0F00A    00002270    58C0E000

0021A0    189F1831    58F0C054    05E041E0    E00E051F     010022A0    00000198    4110C044    58F0C050

0021C0    05EFD203    D0503000    4110C010    58F0C04C     05EF4110    C01458F0    C04C05EF    41E0D13F

0021E0    90EED058    180D58F0    C0584110    D05805EF     4110D0F4    5010C020    4110C018    58F0C040

002200    05EF4120    C0304A20    2000D201    C060D048     4150C05C    50502004    92882004    D24AD148

002220    D0F4D204    D193D13F    4110D148    5010C02C     4110C024    58F0C040    05EF41E0    D13F90EE

002240    D058180D    58F0C064    4110D058    05EF47F0     905E4110    C01058F0    C04805EF    4110C014

002260    58F0C048    05EF58D0    D00498EC    D00C07FE     00003000    40005000    60007000    00000000

002280    80002000    800020A0    58002000    0000004B     00003174    230020A0    00000050    000031C8

0022A0    0008F8F8    00002192    0A002000    880022CC     00002DE0    8F0025B0    000029C6    000029BA

0022C0    00002DA0    00002990    000022F0    00002192     000100C0    00002354    00000000    00000000
                                                           SUB 1
0022E0    FFFFFFFF    FFFFFFFF    00FEFFFF    0C000000     90ECD00C    45E0F00C    000023B8    58C0E000

002300    189F1831    58F0C01C    05E041E0    E00E051F     030023C8    00000100    D203D058    3000D203

002320    D0503C04    5860D058    D2046000    C03D5850     D05841E0    500090EE    D058180D    58F0C020
                                                                       REPEAT
002340    4110D060    05EF58D0    D00498EC    D00C07FE     47F0909C    90ECD00C    45E0F00C    000023B8

002360    58C0EC00    5890C024    183158F0    C01C0700     05E041E0    E00E051F    030023C8    00000100

002380    D203D058    3000D203    D0503004    5860D058     41E06000    90EED060    180D58F0    C0204110

0023A0    D06005EF    58D0D004    98ECD00C    07FE58D0     D00498EC    D00C07FE    00003000    40005000

0023C0    60007000    00000000    000CF8F8    000022F0     00002354    00002758    00002400    000022F0

0023E0    0000CC00    00000000    FFFFFFFF    FFFFFFFF     00FEFFFF    0CF0F0F0    F0F002FF    03FF09FF
          LAST
002400    90ECDC0C    45E0F00C    00002480    58C0E000     189F1831    58F0C018    05E041E0    E00E051F

002420    03002490    00000110    D203D058    3000D203     D0503004    5860D058    F224D0E0    6000F8F0
```

Figure 4.25. PL/I Sample Program (Part 14 of 18)

```
002440    D060C035    FAF2D060    D0E0F83F    D0E8D060     92F9C012    F823D070    D0E892F8    C0125860
002460    D058F342    60000070    96F06004    58D0D004     98ECD00C    07FE58D0    D00498EC    D00C07FE
002480    00003000    40005000    60007000    00000000     0008F8F8    00002400    00002758    00000000
                                                                                   IJCFZIZO
0024A0    00000C00    00000000    FFFFFFFF    FFFFFFFF     00FEFFFF    0C1C0000    0A320000    0A320000
0024C0    47F0F010    0A320000    0A009180    10024710     F01C0A07    91011004    4780F02C    91401002
0024E0    4780F010    50E0F04C    58E01020    D501F04A     E0004770    F04458E0    101C07FE    58E0F04C
                                  IJCFZOZ4
002500    07FE615C    00002F40    0A320000    0A320000     0A320000    50E0F0A4    0A009180    10024710
002520    F01C0A07    91101003    4710F040    D2001030     10280201    1036102E    58E01028    D24F1038
002540    E00058E0    F0A407FE    41E10030    50E01008     0A009180    10024710    F0540A07    18E14110
002560    F0780A00    9180F07A    4710F066    0A07181E     41E10028    50E01008    47F0F010    00000000
002580    00000000    00000004    00002590    00000000     09002598    0000000B    F4F1F5F0    C940D9C5
                                                           IJKSYSA
0025A0    E3D9E800    00000000    00000000    00000000     A00025D0    0D000000    81010078    00000000
0025C0    00002600    00000000    00000000    07000700     00008400    08000003    000025F8    00000000
0025E0    03002680    3100C9D1    E2E8E2D3    E2400040     90002600    47000000    01002600    20000079
002600 TO THE NEXT LINE ADDRESS CONTAINS 40404040
002660    40404040    40404040    40404040    40404040     40404040    40404040    40000000    00000000
002680    00000000    00000000    47F0F00C    9023F000     91031015    4770F066    58301028    06304120
0026A0    00091803    43030J00    4332F0CB    19034780     F0364620    F0284920    F0CA47B0    F046920B
0026C0    10284530    F0A09180    10154780    F056947F     10154530    F0A04122    F0C0D200    10282000
0026E0    D6001028    101E0A00    91041015    4780F07A     4530F0A2    92011028    0A004530    F0A29101
002700    10044780    F09A9104    10154710    F09A9108     101F4710    F0AE58E0    10189823    F00007FE
002720    0A009180    10024710    F0AC0A07    07F31821     48020006    41101016    0A021812    47F0F09A
                                                                                   IJKSZCN
002740    0BC8E38B    0301411B    130B0004    F9C3F14E     E5E660F0    4010C128    905CF408    58C0F2B4
002760    585D0008    50D50004    18AD18D5    5A510004     5550C29C    47B0C07E    505D0008    D200C28A
002780    C2B84850    C29A1255    4780C038    485A0048     41550001    4050C29A    405D0048    D203D04C
0027A0    A0145851    0000505D    0000D200    50025003     9103D000    4710C078    4A550000    96405004
0027C0    91805004    4710C078    41550008    47F0C064     985CC408    07FE41F0    C08E985C    C4089203
                          IJKSZCI
0027E0    D000051F    210005F0    58C0F224    9680C3EE     58D0C428    D2330014    C3F4950D    C3EF4110
                                                                           IJKSZCS
002800    C2B89202    C2B84780    C0E892F1    C2959200     C28F47F0    C27A90EC    D00C58C0    F1F69200
002820    C2B8947F    C3EE947F    C39492FE    C119185D     91055000    4740C0F0    58550004    47F0C0D8
002840    920EC119    47F0C0D6    58450000    95061000     4720C122    D200C109    1000DC00    C109C118
002860    91004002    4710C122    947FC3EE    982CD01C     07FE0A11    20408010    04081865    91036000
002880    4780C198    4710C152    58460000    4A440000     05004000    10004780    C15A9180    40044710
```

Figure 4.25. PL/I Sample Program (Part 15 of 18)

```
0028A0  C1524144  000847F0  C1385866  000447F0   C124950A  10004740  C174950F  10004720
0028C0  C174D502  4001C3A1  4770C142  91404004   4710C142  91104004  4710C198  91084004
0028E0  4780C110  947FC394  58140004  47F0C1CA   95081000  4770C1A8  9640C394  47F0C110
                                                                     IJKSZCP
002900  18619107  C2BA92F4  C2954710  C18A92F2   C2959200  C28F47F0  C27A90EC  D00C58C0
002920  F0F2185D  91055000  4780C22A  4740C1E0   58550004  47F0C1CC  D5011004  50484770
002940  C1D818D5  58610000  5060D038  4A610006   5060D010  58550000  D2005002  50039180
002960  C3EE4770  C21498EC  D00C07FF  947FC3EE   D233C3F4  D014D202  C3F1D011  50D0C428
                            IJKSZCA              IJKSZCM
002980  0A114510  C0C62200  9207F08A  45F0F00C   9208F082  184C58C0  F07C4700  C24C4510
                                      IJKSZCT                        IJKTOPM
0029A0  C0BE2300  9200C243  47F0C438  05F092F3   F03F9200  F03947F0  F02492F0  F0339200
                     IJKTCLM
0029C0  F02D45F0  F01492F3  F0279204  F02190EC   D00C05F0  18514140  F01C4100  F1BC4110
                                          FLAG   DUMMY DSA ENDCORE  CHAIN FRWD A (IJKSZCT)
0029E0  F0140A04  47F10000  5BC9D1D2  E2F0F040   00000003  00007FF0  00003080  000029AC
         PROG MASK                                                     INV CT
                                         MSG SW
002A00  0E000000  00002844  000027E6  00002758   00000808  00002840  00002868  00002830
         SSA2                                             SAVE
002A20  05000000  00000000  00000000  00000000   00000000  00000000  00000000  00000000
002A40 TO THE NEXT LINE ADDRESS CONTAINS 00000000
                                                 SSA1
002A60  00000000  00000000  00000000  00000000   05000000  00003080  00000000 ,00002F40
002A80  000024B8  00003080  00002018  00002050   00002000  0000004B  400021B0  5B002000
002AA0  0000004B  00003174  40002192  00002000   00002A20  00002DE0  00000000  00000000
002AC0 TO THE NEXT LINE ADDRESS CONTAINS 00000000                    CURRENT   FILE CHAIN
                    SCALE FACTOR           SW      DECIMAL INTEGER        FILE      ANCHOR
002AE0  00000000  00000000  00000000  00000000   00000000  00000000  230020A0  FE0020A0
         FORMAT    ENTRY POINT TABLE
002B00 TO THE NEXT LINE ADDRESS CONTAINS 00000000
002B20  00000000  00000000  00000000  00000000   00000000  00000000  00000000  C0002B3C
                  IJKZWSI - INTERRUPT SAVE AREA REG 0    REG 1     REG 2     REG 3     REG 4
002B40  A0002580  00000000PSW00000000  00000000   00000000  00000000  00000000  00000000
         REG 5     REG 6     REG 7     REG 8      REG 9     REG 10    REG 11    REG 12
002B60  000022CC  000031BF  00001000  00002000   00002400  00002000  00002000  00002480
         REG 13    REG 14    REG 15              START OF 65 DOUBLE WORD PL/I TRANSIENT AREA
002B80  00000000  00000000  00000000  00000000   18C118BD  41D04080  50B0D004  1895D202
002BA0  D0899001  5880D088  91808000  4780C0B4   91808008  4780C03E  91088004  4780C03E
002BC0  96048004  48100016  D2008019  104E9140   80044780  C04A9407  8000D600  80009000
002BE0  94098009  91428008  4710C0BC  58108000   D7011000  10005820  80104110  C150D202
002C00  C0798001  4500C07C  00002088  0A025020   80109680  80099101  80094710  C09AD202
002C20  8005D08D  D202D08D  D089947F  80009180   90004199  00044780  C00E58DD  000498EC
002C40  D00C07FE  94FB8004  47F0C09E  91208000   4710C05C  58108000  41110000  95101014
002C60  4770C05C  D208C144  1000D203  1000C140   91801015  4710C0F0  9207C138  45E0C11E
002C80  91101015  923FC138  4780C110  9227C138   45E0C11E  91011004  4710C114  9237C138
002CA0  45E0C11E  D2081000  C14447F0  C0824120   C1385021  00080A00  91801002  4710C132
```

Figure 4.25. PL/I Sample Program (Part 16 of 18)

```
002CC0   0A0707FE   00000000   07000000   20000001    00000400   00000100   00000138   00000000
002CE0   5B5BC2D6   D7C5D540   00008000   00000000    00000000   00000000   00000000   00000000
002D00 TO THE NEXT LINE ADDRESS CONTAINS 00000000
002D80   18C19834   D0145820   50109180   50004780    C0484510   C01E585B   00001000   D540D202
         IJKSZLM
002DA0   90E4D00C   5840F038   413040E4   43210000    89200019   88200017   1A23D202   20011001
002DC0   91801000   41110004   4780F00C   98E4D00C    07FE45A0   C108D501   00002A20   4770C078
         IJKTCBM
002DE0   90ECD00C   18CF58B0   C21850D0   B05441D0    B0509868   10005060   B0D8949F   600943F0
002E00   600044F0   C20C4710   C03258F0   C21C051F    65009140   B0D85820   60104710   C1309180
002E20   60094770   C0AE9111   B0D847C0   C0744970    601647D0   C05E4870   60169620   600958A0
002E40   60109130   600447B0   C06E41AA   00041898    45E0C1EA   58106000   41110000   58F10010
002E60   910C6004   4780C08E   1B0045EF   00044840    601641E0   C09E90EC   D00C47FF   000C9110
002E80   B0D84780   C0BA9608   600047F0   C1C6947F    60094830   600A4133   00049610   60094970
002EA0   600A4780   C0D24740   C0CE4870   600A9620    60099130   60044740   C1004070   60169101
002EC0   B0D84710   C0F45020   B098D202   80018099    47F0C1C6   189818A2   45E0C1EA   47F0C1E0
002EE0   41970004   199347D0   C11A5810   60004111    000058F1   001045EF   00148990   00105090
002F00   B098D203   2000B098   41220004   47F0C0DE    947F6009   91026009   4780C144   58F0C21C
002F20   051F0A00   58106000   58F10010   41110000    4840600A   41E0C160   90ECD00C   47FF0008
002F40   50206010   94F76D00   91306004   47B0C182    D201B098   20004840   B0984840   C2204122
002F60   00044040   60169101   B0D84710   C19C5020    B098D202   70018099   47F0C1B4   19744780
002F80   C1AC47D0   C1A81874   96206009   189218A8    45E0C1EA   58F0C21C   91406009   4780C1C6
002FA0   4110C222   05EF58F0   C21C9130   60044770    C1E09120   60094780   C1E04110   C22405EF
002FC0   58DD0004   98ECD00C   07FE41F0   0100197F    47D0C204   1B7FD2FF   A0009000   1A9F1AAF
002FE0   47F0C1EE   06704470   C21007FE   9100B0D8    D200A000   90000000   00002A20   00002816
                               IJKTXCF
003000   00040C00   0E000000   05E058B0   E0729140    B0D84710   E01A5850   B0D89602   500998EC
003020   D00C07FE   58F0E06E   980CD014   051F0A00    05E058B0   E04A5850   B0D89640   500947F0
003040   E01E05E0   58B0E038   58508008   96205009    05E09101   50084780   E01E9130   50044790
                    A(BL DESC)            FORWD
003060   E0144111   0004501D   001CD201   D026500A    98ECD00C   07FE0000   00002816   00002A20
FLAG     DSA 'STAHT' CHAIN BACK  CHAIN    REG 14      REG 15     REG 0      REG 1      REG 2
003080   01 0022A0  000029F0   00003218   5E00224E    00002354   0C003080   000030D8   000022A8
         REG 3      REG 4      REG 5      REG 6       REG 7      REG 8      REG 9      REG 10
0030A0   00C02000   00002270   000022CC   0000C000    00001000   00002000   40002192   00002000
         REG 11     REG 12                           R0 OF CALLING BLOCK    A(CTR)
0030C0   00002000   00002270   0001 0000  00002B44    C2002018   00000000   000031BF   00000000
                    INVOC CT
0030E0 TO THE NEXT LINE ADDRESS CONTAINS 00000000
                                                                 CARDIN
003160   00000000   00000000   00000000   0000C000    00000000   C940C1D4   40C140C4   C1E3C140
003180   C3C1C9C4   5C5C5C5C   5C5C5C5C   5C5C5C5C    5C5C5C5C   5C5C5C5C   5C5C5C5C   5C5C5C5C
```

Figure 4.25. PL/I Sample Program (Part 17 of 18)

```
                                                                                       CTR
0031A0   5C5C5C5C   5C5C5C5C   5C5C5C5C   5C5C5C5C    5C5C5C5C   5C5C5C5C   5C5C5C5C   5C5C5C|F0
         CTR                   CARDOUT
0031C0   F0F0F0F2|  00000000 | C940C1D4   40C140C4    C1E3C140   C3C1D9C4   5C5C5C5C   5C5C5C5C

0031E0 TO THE NEXT LINE ADDRESS CONTAINS 5C5C5C5C
                                                                            DSA 'SUB 1'
003200   5C5C5C5C   5C5C5C5C   5C5C5C5C   5C5C5C5C    5C5C5CF0   F0F0F0F1   030023C8   00C03080

003220   00003318   7E0023A4   00002400   00003218    00003278   000022A8   000030D8   00002270

003240   000022CC   000031BF   00001000   00002000    000022F0   00002000   00002000   000023B8

003260   00020000   00003080   00000000   00000000    000031BF   00000000   000031BF   00000000

003280 TO THE NEXT LINE ADDRESS CONTAINS 00000000
                                                                            DSA 'LAST'
003300   00000000   00000000   00000000   00000000    00000000   00000000   03002490   00003218

003320   00003428   00000000   00000000   00000000    00000000   00000000   00000000   00000000

003340 TO THE NEXT LINE ADDRESS CONTAINS 00000000

003360   00030000   00003218   00000000   00000000    000031BF   00000000   00000000   00000000

003380   00000000   0000002C   00002C00   00000000    00000000   00000000   00000000   00000000

0033A0 TO THE NEXT LINE ADDRESS CONTAINS 00000000

0033E0   00000000   00000000   00000000   00000000    00000000   00000000   00001F00   00000000

003400   0000002C   00000000   00000000   00000000    00000000   00000000   00000000   00000000

003420 TO THE NEXT LINE ADDRESS CONTAINS 00000000

007FE0   00000000   00000000   00000000   00000000    00000000   00000000   00000000   00000000
```

Figure 4.25. PL/I Sample Program (Part 18 of 18)

Glossary

For a more complete list of data processing terms, refer to IBM Data Processing Techniques, A Data Processing Glossary, GC20-1699.

access method: Any of the data management techniques (sequential, indexed sequential, or direct) available to the user for transferring data between main storage and an input/output device.

ASCII (American National Standard Code for Information Interchange): A 128-character, 7-bit code. The high-order bit in the System/360 8-bit environment is zero.

asynchronous: Without regular time relationship. The user's programs run asynchronously with the I/O interrupts. BTAM's channel appendage routine runs synchronously with the I/O interrupts.

background program: In multiprogramming, the background program is the program with lowest priority. Background programs execute from a stacked job input.

batched job: Programs that execute from a stacked job input. Batched jobs run under the control of job control.

block:
1. To group records physically for the purpose of conserving storage space or increasing the efficiency of access or processing.

2. A physical record on tape or DASD.

BTAM (Basic Telecommunications Access Method): A basic access method that permits a READ/WRITE communication with remote devices.

catalog: To enter a phase, module, or book into one of the system libraries.

CCB: See Command Control Block.

CCH: See Channel Check Handler.

Channel Check Handler (CCH): A feature that assesses System/370 channel errors to determine if the system can continue operations.

channel program: One or more Channel Command Words (CCWs) that control(s) a specific sequence of channel operations. Execution of the specific sequence is initiated by a single start I/O instruction.

checkpoint record: A record containing the status of the job and of the system at the time the checkpoint routine writes the record. This record provides the necessary information for restarting a job without returning to the beginning of the job.

checkpoint/restart: A means of restarting execution of a program at some point other that the beginning. When a checkpoint macro instruction is issued in a problem program, checkpoint records are created. These records contain the status of the program and the machine. When it is necessary to restart a program at a point other than the beginning, the restart procedure uses the checkpoint records to reinitialize the system.

checkpoint routine: A routine that records information for a checkpoint.

Command Control Block (CCB): A 16-byte field required for each channel program executed by physical IOCS. This field is used for communication between physical IOCS and the problem program.

communications region: An area of the supervisor set aside for interprogram and intraprogram communication. It contains information useful to both the supervisor and the problem program.

control program: A group of programs that provides functions such as the handling of input/output operations, error detection and recovery, program loading, and communication between the program and the operator. IPL, supervisor, and job control make up the control program in the Disk and Tape Operating Systems.

core storage: See main storage.

data file: A collection of related data records organized in a specific manner. For example, a payroll file (one record for each employee, showing his rate of pay, deductions, etc) or an inventory file (one record for each inventory item, showing the cost, selling price, number in stock, etc).

data protection: A safeguard invoked to prevent the loss or destruction of customer data.

data set security: A feature that provides protection for disk files. A data secured file cannot be accidentally accessed by a problem program.

Disk Operating System (DOS): A disk resident programming system that provides operating system capabilities for 16K and larger IBM System/360 and IBM System/370.

DOS: See Disk Operating System.

DOS Volume Statistics: A facility that monitors and records the number of temporary read and write errors on currently accessed tape volumes. This facility has two options, Error Statistics by Tape Volume (ESTV) and Error Volume Analysis (EVA).

DTF (define the file) macro instruction: A macro instruction that describes the characteristics of a logical input/output file, indicates the type of processing for the file, and specifies the main storage area and routines to process the file. To do this, use the appropriate entries in the keyword operands associated with the DTF macro instruction.

Environmental Recording, Editing, and Printing (EREP): A program that edits and prints data that has been stored on the system recorder file.

EREP: See Environmental Recording, Editing, and Printing.

Error Statistics by Tape Volume (ESTV): One of the two options of the DOS volume Statistics. With ESTV support, the system collects data on tape errors by volume for any tape volumes used by the system.

Error Volume Analysis (EVA): One of the two options of the DOS Volume Statistics. With this option, the system issues a message to the operator when a number of temporary read or write errors (specified by the user at system generation time) has been exceeded on a currently accessed tape volume.

ESTV: See Error Statistics by Tape Volume.

EVA: See Error Volume Analysis.

extent: The physical locations on Input/Output devices occupied by or reserved for a particular file.

fetch:

1. To bring a program phase into main storage from a core image library for immediate execution.
2. The routine that retrieves requested phases and loads them into main storage (see system loader).
3. The name of a macro instruction (FETCH) used to tranfer control to the system loader.
4. To transfer control to the system loader.

Fetch/Load Trace (F/L Trace): A program that records information about phases and transients as they are called from the core image library.

file: See data file.

fixed length record: A record having the same length as all other records with which it is logically or physically associated.

F/L Trace: See Fetch/Load Trace.

foreground initiation: A set of system routines to process operator commands for initiation of a foreground program.

foreground program: In multiprogramming, foreground programs are the highest priority programs. Foreground programs may be executed from a job stack or in an SPI environment.

Forms Control Buffer (FCB): The buffer in the IBM 3811 Printer Control Unit that stores carriage control information for the IBM 3211 Printer.

Generalized Supervisor Calls Trace (GSVC Trace): A program that records SVC interrupts as they occur. All or a selected group of SVCs can be traced.

GSVC Trace: See Generalized Supervisor Calls Trace.

IDRA: See Independent Directory Read-in Area.

Independent Directory Read-in Area (IDRA): A resident area, created by supervisor option, into which the system reads core image library directories for fetch and load operations. Using IDRA frees the physical transient area to perform error recovery procedures.

Initial Program Load (IPL): The initialization procedure that causes Disk and Tape Operating Systems to commence operation.

interrupt: A break in the normal sequence of instruction execution. It causes an automatic transfer to a preset storage location where appropriate action is taken.

IOCS (Input/Output Control System): A group of macro instruction routines provided by IBM for handling the transfer of data between main storage and external storage devices.

I/O (input/output) error logging: The process of recording OBR and SDR records or the system recorder file.

I/O Trace (Input/Output Trace): A program that records I/O device activity for all or a selected group of I/O devices.

IPL: See Initial Program Load.

Job Accounting Interface: A program that accumulates accounting information for each job step to: charge usage of the system, help plan new applications, and help supervise system operation more efficiently.

Job Accounting Table: An area in the supervisor where accounting information is accumulated for the user.

job control: A program that is called into storage to prepare each job or job step to be run. Some of its functions are to assign I/O devices to certain symbolic names, set switches for program use, log (or print) job control statements, and fetch the first program phase of each job step.

job step: The execution of a single processing program.

K: 1024.

language translators: A general term for any assembler, compiler, or other routine that accepts statements in one language and produces equivalent machine language instructions. For example, Assembler, COBOL, etc are language translators.

librarian: The set of programs that maintains, services, and organizes the system libraries.

library: An organized collection of programs, source statements, or object modules maintained on the system resident device. Three libraries are used by the Disk and Tape Operating Systems: core image library, source statement library, and relocatable library.

linkage editor: A system service program that edits the output of language translators and produces executable program phases. It relocates programs or program sections and links together separately assembled (or compiled) sections.

load: To fetch, i.e., to read a phase into main storage returning control to the calling phase.

logical IOCS: A comprehensive set of macro instruction routines provided to handle creation, retrieval, and maintenance of data files.

LSERV (label cylinder display): A program that formats a listing of the label cylinder located on SYSRES. LSERV can run in and partition any outputs the list on SYSLST, which may be assigned to disk, tape, or printer.

Machine Check Analysis and Recording (MCAR): A feature that records System/370 machine check interrupt error information on the system recorder file and then attempts to recover from the interrupt.

main storage: All addressable storage from which instructions can be executed or from which data can be loaded directly into registers.

MPS: See multiprogramming system.

multiplexer channel: A channel designed to operate with a number of I/O devices simultaneously on a byte basis. That is, several I/O devices can be transferring records over the multiplexer channel, time sharing it on a byte basis.

multiprogramming system: A system that controls more than one program simultaneously by interleaving their execution.

multitask operation: Multiprogramming; called multitask operation to express not only concurrent execution of one or more programs in a partition, but also of a single reenterable program used by many tasks.

OBR: See outboard recorder.

OLTEP: See On-Line Test Executive Program.

On-Line Test Executive Program (OLTEP): The control program of the on-line test system. OLTEP is the interface between the on-line test and the operating system.

outboard recorder (OBR): A feature that records pertinent data on the system recorder file when an unrecoverable I/O error occurs.

overlap: To do something at the same time that something else is being done; for example, to perform input/output operations while instructions are being executed by the central processing unit.

overlay: A program segment (phase) that is loaded into main storage. It replaces all or part of a previously retrieved section.

PCIL: See Private Core Image Library.

PDAID: See Problem Determination Aids.

phase: The smallest complete unit that can be referenced in a core image library. Each program overlay is a complete phase. If the program has no overlays, the program itself is a complete phase.

physical IOCS: Macro instructions and supervisor routines (Channel Scheduler) that schedule and supervise the execution of channel programs. Physical IOCS controls the actual transfer of records between the external storage medium and main storage, and provides I/O device error recovery.

Private Core Image Library (PCIL): A file reference in the same manner and for the same purpose as the system core image library, but distinct from the system core image library. PCIL increases available core image library space to enable compiling, linkage editing, and executing in the foreground partition, when a private core image library is assigned to that foreground partition.

private library: A core image, relocatable, or source statement library that is separate and distinct from the system library.

problem determination: A procedure or process (provided by IBM) that the user can follow after an error message to determine the cause of the error.

Problem Determination Aids (PDAID): Programs that trace a specified event when it occurs during the operation of a program. The traces provided are: QTAM Trace, I/O Trace, F/L Trace, and GSVC Trace.

QTAM Trace: A program that records certain supervisor and I/O activities on tape or in main storage.

RDE: See Reliability Data Extractor.

record: A general term for any unit of data that is distinct from all others when considered in a particular context.

Recovery Management Support (RMS): A feature for System/370 that consists of the MCAR (Machine Check Analysis and Recording) and CCH (Channel Check Handler) functions. RMS gathers information about System/370 hardware reliability and attempts certain error recovery operations. RMS is a part of the entire reliability, availability, and serviceability support for System/370.

Reliability Data Extractor (RDE): A function that provides hardware reliability data that is analyzed by IBM.

relocatable: A module or control section whose address constants can be modified to compensate for a change in origin.

restart: See checkpoint/restart.

RMS: See Recovery Management Support.

SDR (statistical data recorder): A feature that records the cumulative error status of an I/O device on the system recorder file.

selector channel: A channel designed to operate with only one I/O device at a time. Once the I/O device is selected, a complete record is transferred one byte at a time.

self-relocating: A programmed routine that is loaded at any doubleword boundary and can adjust its address values so as to be executed at that location.

self-relocating program: A program that is able to run in any area of storage by having an initialization routine to modify all address constants at object time.

Single Program Initiator (SPI): Under DOS, a program that is called into storage to perform job control type functions for foreground programs not executing in batch job mode.

SORTED DSERV: A program that gives you an alphamerically sorted listing of any or all of the library directories.

SPI: See Single Program Initiator.

stand-alone dump: A program that displays the contents of main storage from a minimum of 8K bytes to a maximum of 16384K bytes. It helps to determine the cause of an error.

supervisor: A component of the control program. It consists of routines to control the functions of program loading, machine interruptions, external interruptions, operator communications and physical IOCS requests and interruptions. The supervisor alone operates in the privileged (supervisor) state. It coexists in main storage with problem programs.

system residence: The external storage space allocated for storing the basic operating system. It refers to an on-line tape reel or disk pack that contains the necessary programs required for executing a job on the data processing system.

task selection: The supervisor mechanism for determining which program should gain control of CPU processing.

telecommunications: A general term expressing data transmission between remote locations.

teleprocessing: A term associated with IBM telecommunications systems expressing data transmission between a computer and remote devices.

trace:
1. To record a series of events as they occur.

2. The record of a series of events.

track hold: A function for protecting DASD tracks that are currently being processed. When track hold is specified in the DTF, a track that is being modified by a task in one partition cannot be concurrently accessed by a task or subtask in another partition.

transient area: This is a main storage area (within the supervisor area) used for temporary storage of transient routines.

transient routines: These self-relocating routines are permanently stored on the system residence device and loaded (by the supervisor) into the transient area when needed for execution.

UCS: See Universal Character Set.

UCSB: See Universal Character Set Buffer.

undefined record: A record having an unspecified or unknown length.

Universal Character Set: A printer feature that permits the use of a variety of character arrays.

Universal Character Set Buffer (UCSB): A buffer in a printer control unit that stores the code equivalents of the characters on an interchangeable print chain or train cartridge.

variable length record: A record having a length independent of the length of other records with which it is logically or physically associated. (Contrasted with fixed length record.) It contains fields specifying physical and logical record lengths.

Index

Indexes to systems reference library manuals are consolidated in the publication <u>DOS Master Index</u>, GC24-5063. For additional information about any subject listed below, refer to other publications for the same subject in the Master Index.

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

DOS System Programmer's Guide

GC24-5073-2

READER'S COMMENT FORM

This sheet is for comments and suggestions about this manual. We would appreciate *your* views, favorable or unfavorable, in order to aid us in improving *this* publication. This form will be sent directly to the author's department. Please include your name and address if you wish a reply. Contact your IBM branch office for answers to technical questions about the system or when requesting additional publications. Thank you.

Name

Address

What is your occupation?

How did you use this manual?

As a reference source

As a classroom text

As a self-study text

Your comments* and suggestions:

*** We would especially appreciate your comments on any of the following topics:**

Clarity of the text
Organization of the text
Accuracy
Cross-references
Index
Tables
Illustrations
Examples
Appearance
Printing
Paper
Binding

GC24-5073-2

YOUR COMMENTS, PLEASE . . .

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

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

CUT ALONG THIS LINE

Fold

Fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
112 East Post Road
White Plains, N.Y. 10601

Attention: Department 813 U

Fold

Fold

DOS System Programmer's Guide (S360-50) GC24-5073-2

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]