File No. S360-21
Order No. GC28-2032-6

IBM Systems Reference Library

IBM Time Sharing System
Assembler Programmer's Guide

This publication explains the use of the Time Sharing System (TSS) for assembler language programmers. It describes how to assemble, store, and execute programs in TSS, introduces the command system, and explains the basic rules of task and data management. Numerous examples are given showing typical user-system interaction. The appendixes include information on assembler options, output, and restrictions, as well as program

SEVENTH EDITION (April 1976)

This is a revision of, and makes obsolete GC28-3032-5 and Technical Newsletter GN28-3201. This new edition of the *Assembler Programmer's Guide* includes revised user-system interaction examples and editorial changes, and deletes an outdated appendix.

This edition is current with Release 2.0 of the IBM Time Sharing System/370 (TSS/370), and remains in effect for all subsequent versions or modifications of TSS unless otherwise noted. Significant changes or additions to this publication will be provided in new editions or Technical Newsletters.

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

A form is provided at the back of this publication for readers' comments. If the form has been removed, comments may be addressed to IBM Corporation, Time Sharing System – Department 80M, 1133 Westchester Avenue, White Plains, New York 10604.

This publication is a guide to the use of the assembler language facilities of TSS. It is intended for applications programmers who code in the assembler language. The publication is divided into four parts.

Part I is an overview of the Time Sharing System, outlining the major concepts of the system.

Part II describes the basic task and data management information you will need to use the system effectively: how to execute tasks in conversational and nonconversational mode, and how to name, catalog, store, and manipulate your data sets. It also explains specific system facilities available to you as an assembler programmer.

Part III comprises a series of examples that illustrate typical activities you will perform while using the system. They begin with the most straightforward necessities, such as logging on, and in succeeding examples progress to increasingly sophisticated concepts. These examples may be read for instruction or used as models for accomplishing specific tasks.

Part IV is a set of appendixes containing additional information on the use of the system. This reference material includes descriptions of assembler options, output, and restrictions, as well as explanations of program control system use, interrupt handling, and command creation.

Prerequisite Publications

You must be familiar with the basic concepts and terminology of TSS as described in *IBM Time Sharing System: Concepts and Facilities*, GC28-2003.

You should be familiar with the TSS assembler language, since this book does not describe the language but rather the use of the system. The assembler language is specified in these publications:

IBM Time Sharing System: Assembler Language, GC28-2000

IBM Time Sharing System: Assembler User Macro Instructions, GC28-2004

You will also need to refer to:

IBM Time Sharing System: Command System User's Guide, GC28-2001, for a complete description of the command system.

Associated Publications

Other publications you may wish to refer to for details not presented in this guide are:

IBM Time Sharing System: Terminal User's Guide, GC28-2017, for instructions on how to operate the various terminals supported by TSS.

IBM Time Sharing System: Linkage Editor, GC28-2005, for a description of the linkage editor program.

IBM Time Sharing System: Data Management Facilities, GC28-2056, for a description of access methods and data management facilities.

Once you begin using the system, you will note that a number of messages are issued by the system. For a detailed description of these messages and for information on any responses you may have to make to them, consult the publication *IBM Time Sharing System: System Messages*, GC28-2037.

Contents

Figures

Tables

Part I. Introduction

This is an overview, for the assembler programmer, of the major concepts of the Time Sharing System. Each concept will be described in detail later in this manual.

The System and Your Relationship To It

The Time Sharing System comprises a set of programs that make it possible for you to use system facilities concurrently with other users. Your terminal is one of many that are at users' locations. They are all connected to a computer center, where an operator manipulates the cards, tapes, disks, and listings that are required to complete the commands you issue. The system creates a separate task for each current user to make all of the system facilities available to him. You are each allocated brief time intervals during which your task is executed. Thus it appears that only you are connected to the system.

Communicating With the System

In TSS, you may run your programs conversationally or nonconversationally. When you want direct communication with the system while you are assembling, debugging, and executing your program, the conversational mode will better suit your needs. When time does not permit staying at your terminal, or your program is already checked out, you can use the nonconversational mode.

To assemble and run your program conversationally, you enter commands and data at your terminal. The system analyzes each statement as it is received. If an error is found, you are prompted to correct it. When the entire program has been entered, it is analyzed as a whole, and you are again prompted to correct errors. When your corrected program is assembled, you may execute it and monitor its progress from your terminal (see "Conversational Mode" in Part II).

To assemble and run your program as a nonconversational task, you can either:

- Enter commands and input data (including source statements) at the terminal and specify that they be stored as input for a continuing (or separate) task, or
- Submit a card deck or tape containing commands and input data to the computer center.

In nonconversational mode there is no direct communication between you and the system. Errors in your source program could prevent the assembly from being completed, since there is no way for you to correct yourself. Any system messages that develop during execution of the task will be printed out at the computer center (see "Nonconversational Mode" in Part II).

You can mix modes of operation, starting out conversationally and switching to nonconversational mode; however, you may *not* switch from nonconversational to conversational (see "Switching Modes" in Part II).

There are two additional means of communicating with the system; their availability and use will vary from installation to installation. They are described in Part II under "Multiterminal Task (MTT)" and "Remote Job Entry (RJE)."

How You Gain Access to the System

Before using TSS you must be joined to the system by your system administrator or system manager. When you are joined, information about your identification is stored:

- User Identification (userid)—code that uniquely identifies you to the system.
- Password—code that provides additional protection against unauthorized use of your user identification.
- Charge Number—account to which your use of the system is charged.
- Priority—code indicating the relative priority of your work.
- Privilege Class—code identifying you as a user (as opposed to, say, an operator).

Each time you attempt to communicate with the system, whether conversationally or nonconversationally, you must issue the LOGON command, with operands that have enough information to identify you. The system checks the information you have supplied against the information it has stored about you; when you are recognized, you can begin entering data. For a detailed description of the LOGON operands, see Example 1 in Part III.

Commands at Your Disposal

The time-sharing command system comprises a series of commands with which you tell the system what you want it to do. For example:

- Task management commands allow you to initiate, terminate, or change the system's operation for you.
- System inquiry commands request specific information from the system about your data sets.
- Data set management commands allow you to establish, manipulate, and eliminate your data sets.

See Part II for command descriptions.

How Storage is Handled

Virtual Storage

You will not be directly concerned with the installation's physical limitations on main storage. Special addressing techniques, internal to the system, will provide you with a storage capacity that is theoretically equal to the total range of addresses that can be specified in an instruction. The system's addressing techniques combine main and secondary storage to create a virtual storage area in which your task will operate. Your installation will inform you of the specific virtual storage limits available for your problem programs and data sets.

Although you have an extremely large virtual storage capacity, efficient programming is important, since performance can be degraded by excessive demands on the available storage at your installation (see "Efficient Use of Virtual Storage" in Appendix C).

Each user has his own storage space for program execution; therefore, other users cannot interfere with your programs, nor can you interfere with theirs, because neither of you can refer to the other's virtual storage space. You may share another user's programs and permit him to share yours; however, specific commands must be issued to accomplish this (see "Sharing and Protecting Your Data Sets" in Part II).

Sharing Time

There may be many users communicating with the system at the same time that you are. However, the system appears to be serving each of you exclusively, because, cyclically, it is giving each of you a time slice during which all the facilities required by your task are in fact exclusively yours. Unless the system is overloaded, its speed will allow it to do your work as well as that of other users without the intervals being apparent to you.

Catalog and Library Concepts

System Catalog

The system maintains a catalog to give you the means for recording the locations of data and, later, retrieving that data by name alone. Conceptually the system catalog is much like the catalogs in libraries; it is an index that points to items that reside elsewhere. You are therefore relieved of the responsibility of keeping track of data-location information. The structure of the catalog protects your data sets from being accessed by other users, unless you specifically permit others to share them (see "Catalog Structure" in Part II).

Program Libraries

When it is assembled, your program can consist of one or more object modules. All programs in TSS are stored, in object-module form, in program libraries. A program consisting of only one object module is stored within one library; a program that consists of several object modules may reside in different libraries, depending upon how you have stored them. During linkage editing and during execution, the system can automatically retrieve all required object modules, if you have defined the libraries that hold those modules.

There are four categories of program libraries: system library (SYSLIB), user library (USERLIB), user-defined job libraries, and linkage editor libraries. A program library list, defining the hierarchy of libraries available to you, is used to store object modules in the specified library and to search each library for object modules that must be loaded at execution time. Libraries and their uses are described in Part II under "Maintaining Program Libraries."

How to Use This Manual

Parts I and II contain the information you will need to assemble and run a program at your terminal. If you are not familiar with the basic rules of task and data management, you should read Part II before attempting a terminal session. Running Examples 1 through 4 in Part III will give you a basic understanding of how to assemble and execute a program at your terminal.

The remaining examples in Part III illustrate the use of commands and system facilities for a wide range of functions. You should scan these examples and the appendixes initially; they are primarily for reference when more detailed information on a specific facility (such as handling interruptions) is required.

Part II. How to Use TSS

Part II presents detailed coverage of the basic task and data management information that you will need to know to assemble and run programs in the time-sharing system. Included are discussions of:

- Conversational and nonconversational modes of operation
- Data set management, supplying only those facts that are essential for basic use of the system. (Those assembler programmers requiring more detailed information on data management should refer to *Data Management Facilities.*)
- The system catalog and your maintenance responsibilities
- Sharing facilities and the need for protection
- Facilities available for producing large volume output
- The macros available to assembler language programmers.

At the end of this part is a table showing sample usages of the commands and macro instructions available to you, along with a notation of which examples in Part III illustrate their use.

Task Management

TSS tasks may be executed in either of two modes: conversational or nonconversational. During conversational task execution you remain in communication with your task, obtaining intermediate results and modifying your program while it is executing. There is no communication however, between you and a nonconversational task; no task output is available until the task has been completed or is terminated by you or the system. TSS also allows you to switch a conversational task to nonconversational, when user-task communication is no longer needed.

Conversational Mode

In conversational mode you communicate with the system by means of a typewriter-like terminal. Your terminal may be one of the following:

- The IBM 2741, which is an IBM Selectric typewriter specially equipped for terminal use.
- The IBM 3277 CRT Display Terminal
- The IBM 1050 System, which can include both a typewriter and a card reader. Input can be entered into the system via the keyboard or the card reader.
- The Teletype Model 33 or 35 KSR.[1]

Your terminal may be located at the computer center or at a remote location. In either case terminal operation is the same: you enter a command directing the system to do certain work, the system responds, you enter another command, etc. The system communicates with you by printing out messages and data at your terminal. Thus you are able to solve problems which arise and make changes as you receive processing results during task execution.

Initiating Your Task

To initiate conversational task processing, you either:

- Dial up the system, the number being determined by your installation, or
- Press the attention button on the terminal, if the terminal is "hardwired" (i.e., directly connected to the computer).

You have thus begun the log-on process and set up a conversational task in the system. Since you have already been granted access to the system by being joined by your system manager or administrator, you now identify yourself by typing in the LOGON command with the parameters set up for you at join time. The system then completes initiation of your task. See Example 1 in Part III for a description of the LOGON operands and an example of their use.

Executing Your Task

After you have logged on, the system asks you to enter your next command and, in effect, enters into a conversation with you. Your portion of this dialog consists of your commands and any source language statements that you enter during execution of your task, plus your replies to the messages issued by the system. The system's contribution to this dialog consists of messages, responses to commands, and requests for the next command. The system informs you that it is ready to accept your next command by printing, at your terminal, an underscore character (_) beneath the first character position of a new line.

[1] Trademark of Teletype Corporation, Skokie, Illinois. Terminals which are equivalent to those explicitly supported may also function satisfactorily. The customer is responsible for establishing equivalency. IBM assumes no responsibility for the impact that any changes to the IBM-supplied products or programs may have on such terminals.

Entering Commands

Every command you enter from the terminal keyboard starts on a new line. It may begin above the underscore that requests its entry or at any other point on the line. Each command has an operation part specifying what is to be done (as CALL), and each may have one or more operands that qualifies the operation (as NAME= followed by the name of your object program, say PRIME. This qualifies the operation to mean "execute my object program, PRIME"). The end of the command is indicated by pressing the RETURN key.

Each command is analyzed, when it is entered, to determine if it is valid; if it is, all the actions requested by the command are performed before you are requested to enter the next command. If the command is not complete or valid as entered, the system issues a message to request you to supply additional information before the command is executed. The system issues three types of messages to the conversational user:

- Prompting messages which ask you to supply omitted operands or additional information;
- Response messages which tell you of the actions the system has taken in executing a command, and
- Diagnostic messages which inform you of command and source language errors.

SYSIN and SYSOUT

Your task's input to the system contains the sequence of commands you issue; this sequence is called SYSIN. Your system input stream can also include data to be prestored in the system, or actual input records to an executing program. When you are in the conversational mode, your terminal is your task's SYSIN device.

Your task's system output stream, called SYSOUT, is directed to the terminal. It consists basically of system messages, and may also contain output from your object programs if you so choose. Because the terminal is thus a combined SYSIN/SYSOUT device, it generates a copy mixture of the two system streams. You, and every other user, have your own unique SYSIN/SYSOUT, which are not recorded by the system in any form other than as a listing printed at your terminal.

Assembling and Running a Program

Let us suppose that you wish to assemble and run a simple program named "PRIME" conversationally. Your entry to the system is achieved by typing in the LOGON command with the appropriate parameters. (See Example 1 in Part III for a detailed description of LOGON operands.) You would then issue the ASM command (which initiates the Assembler) with the desired parameters to call for assembly of your source program (see Example 2 in Part III for a detailed description of ASM operands). Your source program may then be entered conversationally from the termnal, instruction by instruction, or may be a prestored source data set.

When your program has been analyzed and assembled, the resulting object program is stored in a program library. You may then call for its execution by issuing the CALL command followed by your program name, or by simply entering your program name. When you have completed your task for this session, you tell the system to disconnect you by issuing the LOGOFF command (no operands), which terminates that terminal session.

For example:

```
LOGON  JONES,JOHN,,ACCT30
   .                (system acknowledgment)
ASM  PRIME,N        (you request assembly of a source pro-
   .                gram that is not prestored)
   .                (program instructions and, possibly,
   .                system messages)
   .                (system indication of successful assem-
   .                bly)
PRIME               (you call for program execution)
LOGOFF              (end of session)
```

You can interrupt execution of your conversational task at any point by pressing the ATTENTION key at your termnal. This will generally result in your task being placed in the command mode, giving you the opportunity to redirect the system. However, the effects of interrupts will vary depending upon the conditions; these are described in Table 20 in Appendix D.

When assembling a program, you can also specify that various types of listings are to be created (see Example 1 Part III). In conversational mode these listings are automatically placed in a list data set unless you specify that no list data set is to be created. To have your list data set printed, you must issue the PRINT command, establishing a separate nonconversational task that will print your listings on the high speed printer at the central installation (see "Nonconversational Task Initiation"). Listings on SYSOUT are automatically put out at your terminal. Example 2 in Part III illustrates this use of the PRINT command.

Checking Out and Modifying Programs

In addition to the conversational prompting and diagnostic facilities that the assembler contains to assist you in debugging your source program, you are also provided the option of requesting an internal symbol dictionary (ISD) in your object module. An ISD allows you to make full use of the program control system (PCS) with which you may examine and modify various parts of your program during execution. You can use PCS commands and statements to perform one, or any combination, of these:

1. Request display of data fields and instruction locations within your object program, specifying these items by their symbolic names as used in the source language program.
2. Modify variables within your program, specifying these variables by their symbolic names and specifying the new value for each variable.
3. Specify the statements within your program at which execution is to be stopped or started. When program execution has been stopped, you may intervene, as described in items 1 and 2, before you direct resumption of program execution.
4. Specify the statements within your program at which the actions described in 1 and 2 are to be automatically performed.
5. Obtain the values of your program's variables at a specified point in its execution, with the variables formatted according to their types.
6. Establish logical (true or false) conditions which allow or inhibit the actions described in items 3, 4, and 5.

The use of program control facilities will greatly simplify the preparation of source programs, because many functions previously source-coded can conveniently be made available after assembly. Neither the PCS commands nor the modifications they may make in your program remain part of the stored object module; they are removed when the module is unloaded. PCS is discussed in greater detail in Appendix B.

Multiterminal Task (MTT)

In addition to the single terminal mode of operation described above, in which *you* initiate your task, TSS has a multiterminal mode under which the task is initiated by an MTT administrator. A multiterminal task is designed to permit a large number of users at different terminals to share the same task. The task must be specially prepared for execution within the MTT environment, and should, most appropriately, be an application which can be used simultaneously by many users. Logging on with the intention of connecting to such an application program requires the use of the BEGIN command in place of the LOGON command. A complete description of the MTT facility may be found in *IBM Time Sharing System: Multiterminal Task Program and Operation*, GC28-2034.

Nonconversational Mode

You will probably want to assemble and run some programs without being in direct communication with the system while they are being processed; these call for nonconversational processing. The manner in which you define what you wish done in these tasks will vary with the type of nonconversational task you are creating.

Initiating Your Task

There are several ways in which you can initiate a nonconversational task from either a conversational task or from another nonconversational task (see Figure 1).

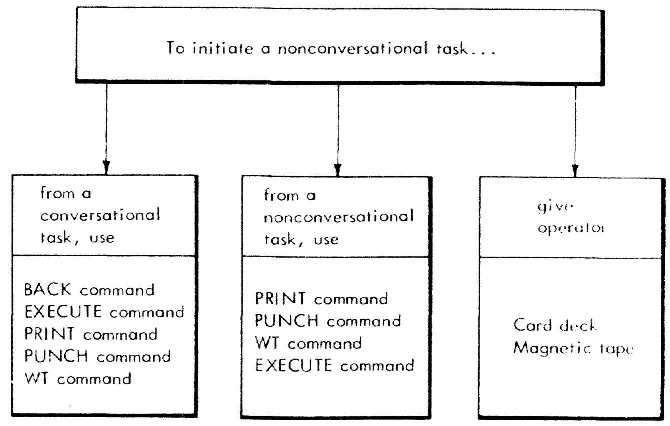


Figure 1. Nonconversational Task Initiation

PRINT, PUNCH, WT—You can issue PRINT, PUNCH and WT commands in either a conversational or nonconversational task. These commands initiate nonconversational tasks that transfer data between a direct-access device and a printer, card punch, or tape unit, respectively. Several of these commands may be issued within a single task; each will set up a separate and independent nonconversational task (see Figure 2). Example 2 in Part III illustrates the use of the PRINT command.

EXECUTE—You can issue the EXECUTE command in a conversational task to initiate a nonconversational task (see Example 2 in Part III). The EXECUTE command names a prestored sequence of commands that is to be executed as, and acts as the SYSIN data set for, a nonconversational task. This sequence must begin with a LOGON command and end with a LOGOFF command, and it must be prestored so that it can be retrieved by name. (The SYSIN data set used by the EXECUTE command, or any other nonconversational task, can contain an EXECUTE command, thus permitting initiation of additional nonconversational tasks.) The nonconversational task thus initiated is treated as a separate task, independent of the conversational (or nonconversational) one in which you set it up, and with which you may now continue (see Figure 3).

BACK—The BACK command is used when you want to switch a conversational task to nonconversational mode; it is described under "Mixed Mode."

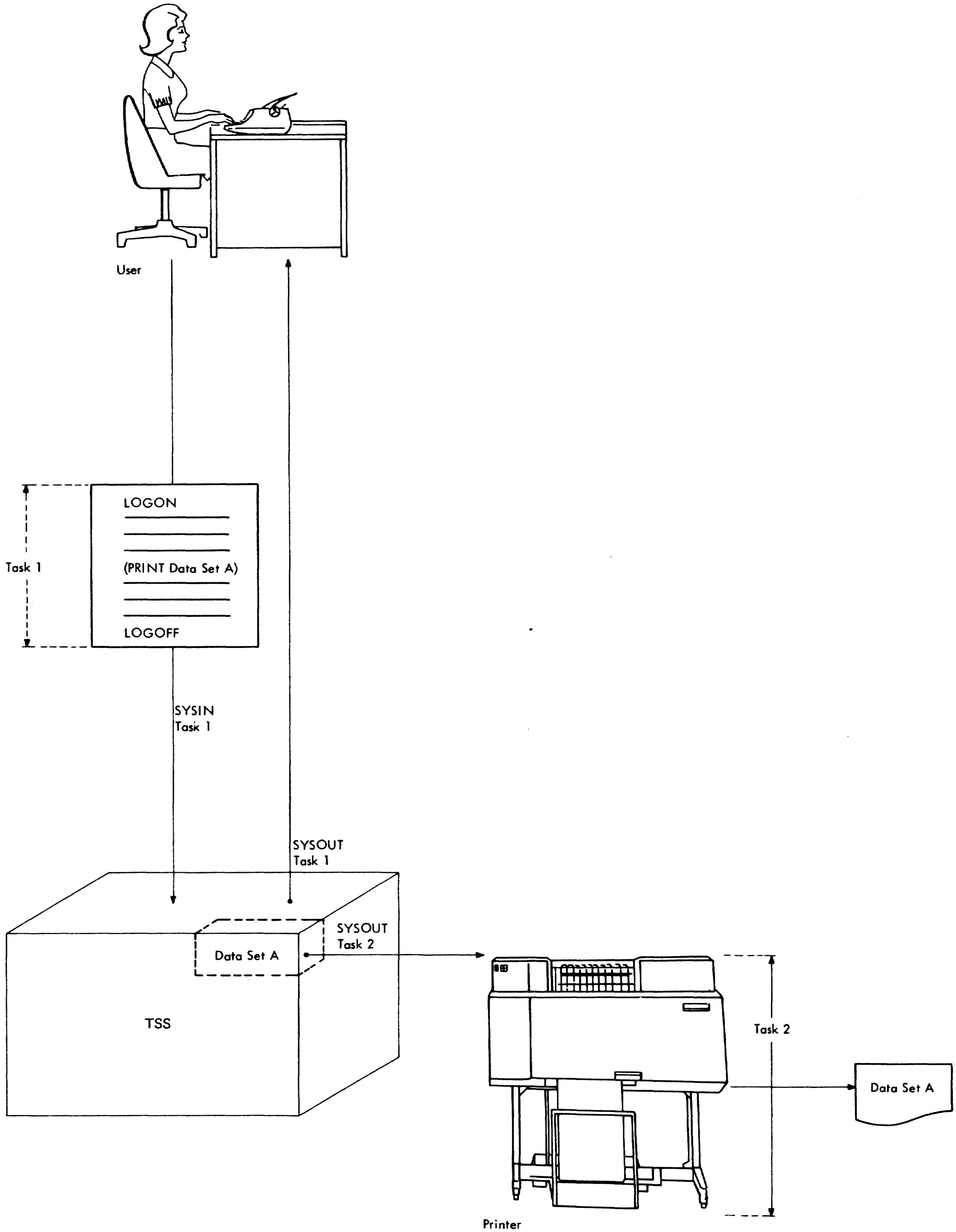


Figure 2. Nonconversational Task Initiated by PRINT Command

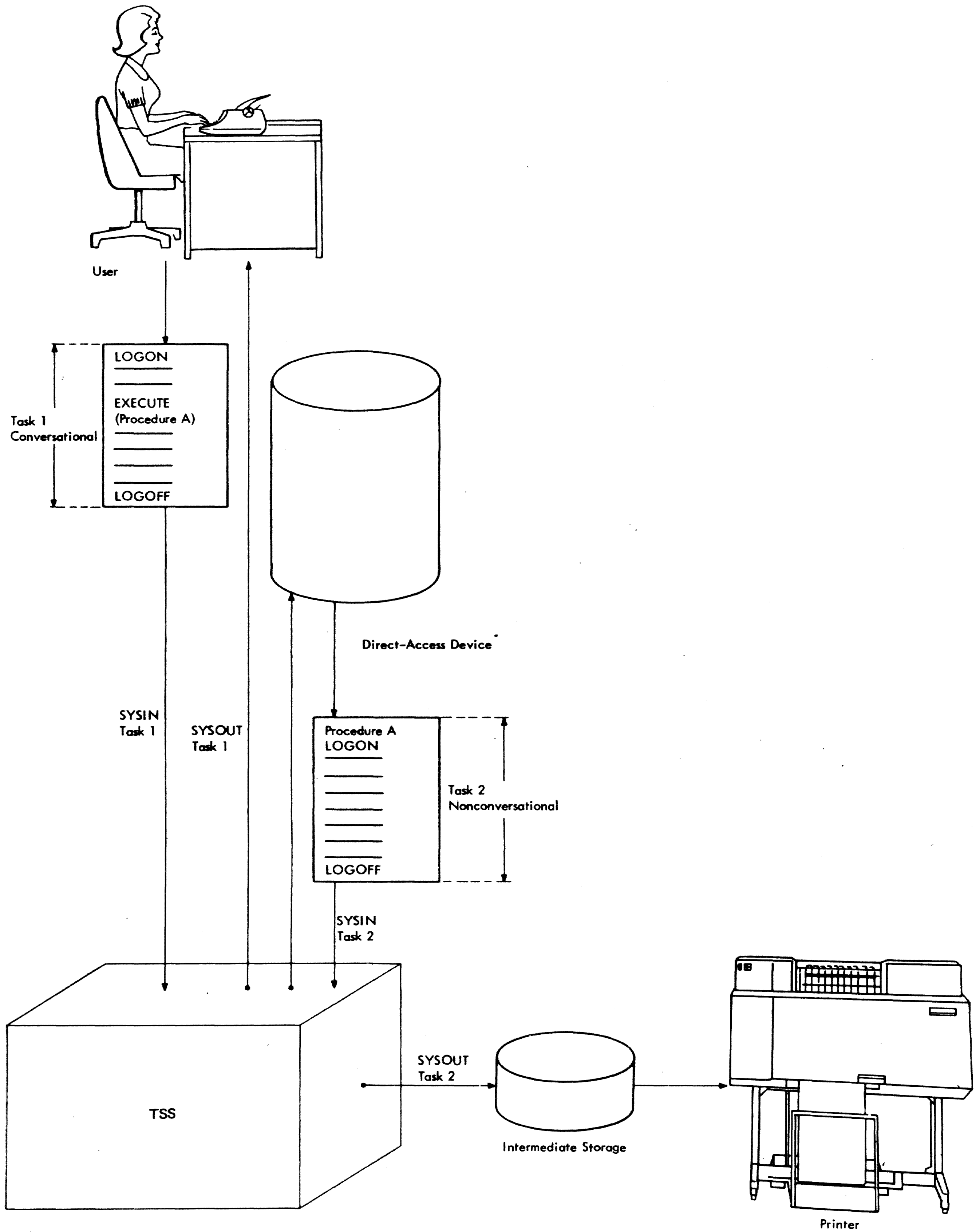


Figure 3. Nonconversational Task Initiated by EXECUTE Command

Operator-assisted—You can also have the operator initiate nonconversational tasks for you by supplying him with a card deck or magnetic tape; the contents of the deck or tape will depend upon what you want done. If you want to enter data into the system for later use, i.e., prestore it, the task set up by the operator will transfer the data from the input medium to a direct access device and catalog it so that it is later available to you by its name. If you want to enter a card deck command procedure, the task that is set up by the operator will execute the commands in the command procedure you have defined (see Example 12 in Part III).

Executing Your Task

Regardless of which method of initiation you use, the nonconversational task you create is assigned a batch sequence number and is executed as soon as the required resources are available. You can issue the EXHIBIT command to determine the status of your previously initiated nonconversational tasks.

During execution of a nonconversational task, there is no communication between you and the system. The system analyzes, in sequence, each command of the SYSIN data set and, if it is valid, executes it. If a command is invalid, the system terminates the task. When execution of a nonconversational task begins, it cannot be interrupted by pressing your ATTENTION key, as your terminal is not associated with the task.

Any listings you request are automatically written on SYSOUT (with no record kept in the system), unless you have specifically asked for a list data set. When a list data set is requested, its printing is accomplished, as in conversational processing, by the issuance of the PRINT command, which sets up another separate nonconversational task.

Terminating Your Task

The execution of noneonversational tasks (except those initiated by PRINT, PUNCH, and WT commands) is terminated when their LOGOFF command is executed. The system then automatically prints out the task's SYSOUT data set. For nonconversational tasks, the SYSOUT data set consists of the commands from SYSIN that were executed, any data that your program writes to SYSOUT, and (if no list data set was specified) diagnostic messages and whatever listings you requested. If a list data set was specified, diagnostic messages will be printed with your listings.

Tasks created by the PRINT, PUNCH, and WT commands terminate when the data transfer is complete. You may also terminate any of your nonconversational tasks by issuing a CANCEL command, identifying each task to be terminated by its batch sequence number. Your task may also be cancelled from the operator's console via its batch sequence number.

Mixing Modes

You can begin a task at your terminal, and then issue a BACK command to have the task's execution completed in the nonconversational mode. Before issuing the BACK command, you must have stored a SYSIN data set that is to function as the command procedure and, if desired, input data for the nonconversational portion of your task. The SYSIN data set must not contain a LOGON command (because you have already logged on), but it should end with a LOGOFF command.

When you issue a BACK command for a task, the system checks that it can provide sufficient resources to continue your task nonconversationally. If it cannot, the system will reject your request; you may then try to initiate the switch later.

You do not initiate a separate task when you issue the BACK command; you still have only one task in the system. This task, however, is nonconversational and has no connection with your terminal (see Figure 4). If the system accepts your BACK request, it establishes the nonconversational task, assigns a batch-sequence number to that task, and writes that number out at your terminal; after that your terminal is inactive. You must then log on again if you wish to initiate a new conversational task at your terminal.

Remote Job Entry (RJE)

An additional facility for running nonconversational tasks in TSS, available at some installations, is the remote job entry (RJE) facility. With RJE it is possible to enter batch jobs at remote terminals in the same format as that used at local, on-line card readers. Printed output is then returned to the originating station, unless another station or local high-speed printer is specified. A complete discussion of this facility may be found in *Time Sharing System: Remote Job Entry*, GC28-2057.

Task Management Commands

These commands allow you to initiate, terminate, or change the system's operation in your behalf. In conversational mode, communication takes place at your terminal. In nonconversational mode, the information is sent to the task's SYSOUT data set. The facilities provided are summarized below.

ABEND—unloads all modules in your virtual storage and returns your task to the status that existed immediately after the LOGON process.

BACK—changes the mode of your conversational task to nonconversational.

BEGIN—notifies the system that you wish to connect to an MTT application program.

CANCEL—terminates the execution of a nonconversational task prior to its normal end.

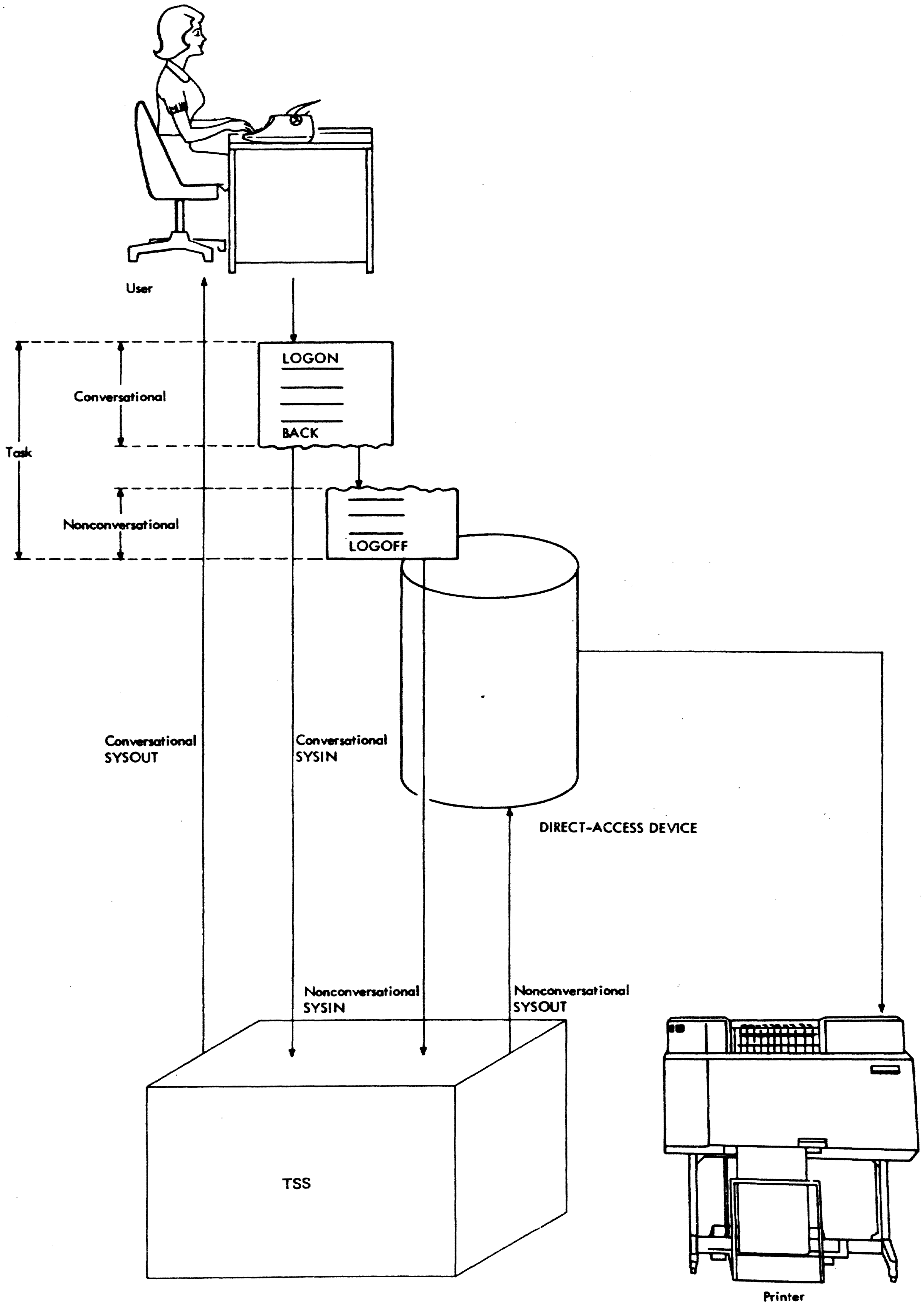


Figure 4. Converting a Conversational Task to Nonconversational Mode Using the BACK Command

CHGPASS—notifies the system that you wish to change your password.

EXECUTE—initiates an independent nonconversational task using a prestored and cataloged command stream.

LOGOFF—notifies the system that you wish to terminate your task.

LOGON—identifies you to the system so that you may begin your task.

SECURE—identifies and reserves types of input/output devices needed for private data sets (in nonconversational tasks *only*).

TIME—establishes a time limit in virtual storage for the execution of a task; it can be changed by you during your task.

USAGE—requests a summary of system resources available to you, as well as those you have used since JOIN time and since the current LOGON.

ZLOGON—performs a user-defined function immediately after LOGON operations have been completed.

Data Set Management

TSS provides you with facilities for systematic and convenient management of your data sets. These data set management facilities make it possible for you to: identify your data sets; efficiently store and retrieve them within the system; share them with other users; copy, modify, and erase them; and define their existence and use in the system.

Naming and Cataloging Your Data Sets

Naming Your Data Set

A data set name uniquely identifies a data set. It is in the form of one or more symbols separated by periods. For example, the data set name AR.TWO.DESIGN consists of three components that are delimited by periods to indicate a hierarchy of categories. Starting from the left, each symbol of the name is a category within which the next symbol is a unique subcategory. A fully qualified name identifies an individual data set. A partially qualified name identifies a group of data sets.

For example: If AR.TWO.DESIGN is a fully qualified data set name, AR and AR.TWO are partially qualified names identifying groups of data sets, one of which is AR.TWO.DESIGN. The group AR.TWO is a subgroup of AR.

These basic rules are to be observed by you in the design of data set names:

1. Each component, or simple name, can consist of from one to eight alphameric characters; the first must always be alphabetic.
2. A period must be used to separate components.
3. The maximum number of characters (including periods) in the data set name is 44. For data sets used exclusively within TSS, you are limited to 35 characters, because the system automatically prefixes each name with your eight-character user identification followed by a period. For data sets to be interchanged with OS or OS/VS, you can employ 44-character data set names. These data sets, however, cannot be cataloged in TSS without being renamed.
4. The maximum number of single-character qualification levels to a single-character basic name is 17, for data sets used in TSS. Normally, however, you will employ only a few qualification levels.
5. The fully qualified data set names in each user's data set name-structure must be unique, and each must uniquely identify one data set.

System Catalog

The system catalog is a special data set that resides on one or more direct-access devices. It is used for filing data set descriptions that must be stored within the system so that, once a data set is created and cataloged, it can subsequently be located by using only its name. To understand the structure and significance of the system catalog, you must become familiar with the basic concepts of data sets, their naming and residence.

Catalog Structure: The system catalog is organized into a hierarchy of indexes: a master index, which consists of a set of user identification codes, one for each user who has been joined to the system; and a collection of separate indexes, each of which is subordinate to one of the user identification codes in the master index. Going down the hierarchy, each of the indexes will correspond to a level of qualification in the data set name structure you have adopted. In effect, the system has *its* own catalog and you have *your* own.

When your data set is cataloged, the required indexes are established in your user catalog, in accordance with the fully qualified name of the data set (see Figure 5). An index is established for each level of qualification. The master index points to the highest-level index of your catalog. This index, and each index thereafter, points to the location of the next lower index. The lowest-level index contains a data set descriptor (DSD) which points to the data set control block (DSCB) which, in turn, points to the specific volumes and pages on which the data set is located.

At the time your user identification is placed in the master index, another special entry is created in your catalog called your USERLIB. Your USERLIB is your own private library for object programs. Except for your USERLIB, you control all entries in your catalog by the way you name your data sets and by the way you use the cataloging and uncataloging facilities of the system.

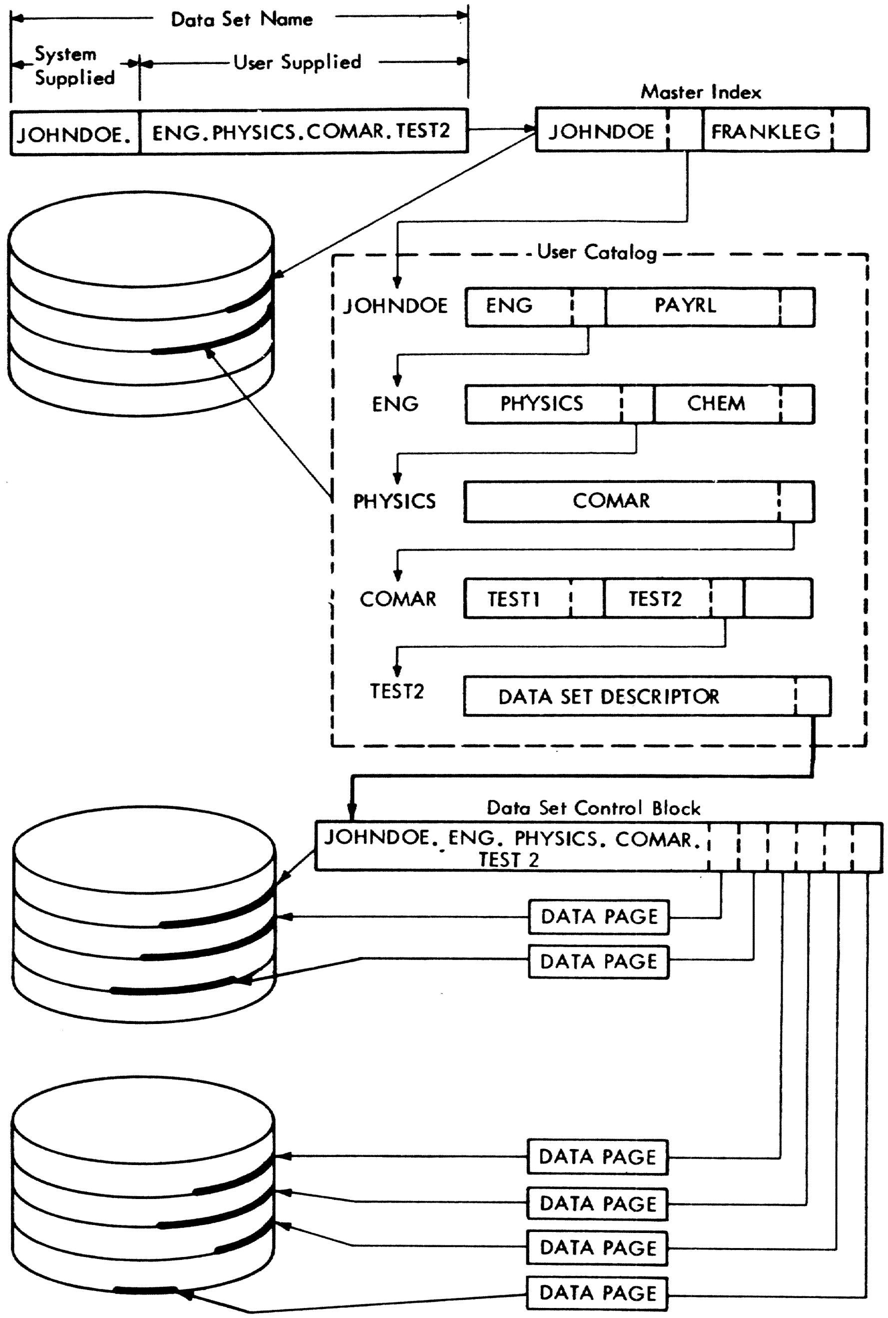


Figure 5. System Catalog Concept

Some of these facilities are for entering, removing, and renaming catalog entries; others are for indicating which data sets can be shared by others, and to what extent. These facilities are described later in this part.

Generation Data Groups: The cataloging facilities of TSS provide an option that assigns numbers to individual data sets in a sequentially ordered collection, thereby allowing you to catalog the entire collection under a single name. You can distinguish among successive data sets in the collection without assigning a new name to each data set. Because each data set is normally created from the data set created on the previous run, the new data set is called a generation, and the number associated with it is called a generation number. The entire structure of data sets of the same name is called a generation data group (GDG). You can refer to a particular generation by specifying, with the common name of the group, either the relative generation number or the absolute generation name of the data set. The use of generation data groups is illustrated in Example 22 in Part III.

Cataloging Your Data Set

You can catalog and uncatalog data sets in several ways. Sometimes cataloging is automatic; in other cases, you must issue a CATALOG command to catalog the data set. All data sets with virtual storage organization (VAM) are automatically cataloged when they are created.

The CATALOG command may be used to catalog a physical sequential (SAM) data set, or to alter the entry of a previously cataloged data set (e.g., to rename a cataloged data set or to change the version number of a generation data group member). If you employ generation data groups (GDG), you must initially use the CATALOG command to set up the structure for the GDG name, number of generations to be retained, disposition of old generations when the specified number of retentions is exceeded, etc.

When you catalog a data set, you can specify either read-only or unlimited access. You can always erase your own data set, but if you have cataloged it with read-only access, you cannot write into it, thus ensuring against accidentally overlaying data.

You can use the DELETE command to remove a catalog entry for a data set if:

1. You want to remove the catalog entry of a data set from the catalog but not erase it, and the data set resides on a private volume.

2. You want to remove the catalog entry of a data set you are sharing from your catalog (because you no longer have a need to share that data set).

The ERASE command can also be used for uncataloging. ERASE removes the catalog entry, and erases the data set as well *if* it resides on a direct-access volume. (Erasing means making the storage space of the data set available for other use.)

So that you can specify whether you want to be given one data set name at a time when you enter either the ERASE or DELETE command, provision is made to set the value of DEPROMPT (a value contained in your User Profile) to either YES or NO by using the DEFAULT command (see "User Profile" in this part). If the value was set to YES, and you specify a partially qualified data set name, the system will issue a prompting message giving you the opportunity to specify that all remaining data sets under that level name are to be erased without prompting. Otherwise, you will be given one data set name at a time for disposition. If the value was set to NO, all data sets grouped under this partially qualified name will be erased or deleted without individual presentation. If you specify a fully qualified name, the data set will be erased or deleted no matter what was specified for DEPROMPT.

You have the option in certain commands, as PRINT and PUNCH, if a cataloged data set is involved, of specifying whether it is to be erased or not after the output operation.

Data Set Organization

A data set's organization defines the overall relationships of the component records into which the data set is logically subdivided. The component records are called logical records, because each is a logical entity containing information for the problem program that is to process the data set. In TSS, there are two fundamentally different types of data set organizations: virtual storage data sets and physical data sets.

Virtual Storage Data Sets

Data sets with a virtual storage organization reside only on direct-address volumes; they are automatically cataloged by the system when they are created. You create, read, and process these data sets on the basis of the logical records they contain. The system, however, uses the page as the unit of transfer between the direct-access device and your virtual storage. Virtual storage data sets may have any of these organizations:

Virtual sequential (VS): In a virtual sequential data set, the order of logical records is determined solely by the order in which the records were created. In creating this type of data set, you provide the system with a stream of records. The system organizes the data into pages, and stores the data set on a direct-access device. After the data set has been created, you can read back the records in the order in which they were created merely by requesting one record after another.

Virtual Index sequential (VI): In a VI data set, the records are organized in sequence based on a data key associated with each record. The data key may be a control field that is part of the record (such as a part number), or it may be an arbitrary identifier (such as a line number) that is the beginning of each logical record, and is added to each record to give it a unique key.

One special form of virtual index sequential data set is the *line data set*, with a maximum of 132 bytes per record. A line data set is organized by line number, where each line is a record and is prefixed with the line number as its key. Source programs are line data sets. You can inspect and display these data sets by line number using the LINE? command. Other commands enable you to effect replacements, insertions, and deletions on line data sets.

Because records in the virtual index sequential organization have logical and physical relationships, you can request the system to perform any or all of these operations:

- Retrieve or create (in a manner similar to that for sequential organization) logical records whose keys are in ascending collating sequence.

- Retrieve or create individual records whose keys are in any order. (Processing is, of course, slower here than if it were being done in the collating sequence, because a search is required to locate each record's position.)
- Add records, with new keys, to the data set. The system automatically locates the proper position in the data set for the new control and makes all necessary adjustments for subsequent retrieval in logical sequence.
- Delete existing records from the data set. The system automatically updates the page locators (and the page directory if necessary) and makes the space used by the deleted records available for other uses.
- Update existing records in the data set, either expanding or contracting their size.

Virtual Partitioned (VP): A VP data set is used to combine individually organized groups of data into a single data set. Each group of data is called a member, and each member is identified by a unique name. Program module libraries are a good example of a VP data set. Your USERLIB is organized this way, and the compiled object modules you store in USERLIB are its members.

The partitioned organization allows you to refer to either the entire data set (via the partitioned data set's name) or to any member of that data set (via a name consisting of the name of the data set qualified by the member name in parentheses).

Example: A partitioned data set named INVENTORY whose members consist of monthly data sets such as JAN, FEB, and MAR, could be referred to in one of the following manners:

INVENTORY	Entire library of inventory data
INVENTORY(JAN)	January inventory data
INVENTORY(FEB)	February inventory data
INVENTORY(MAR)	March inventory data

The partitioned data set may be composed of VS or VI members or a mixture of both. Individual members, however, cannot be of mixed organization.

You can assign additional names, called aliases, to each member, and subsequently locate a member on the basis of either the member name or any of its aliases. The partitioned data set organization is ideally suited for storage of libraries of programs or other groups of data that are frequently referred to together.

Physical Sequential Data Sets

Data sets with a physical sequential organization can reside on either direct-access or magnetic tape volumes. The logical records in these data sets have an organization which is determined solely on the basis of their position relative to the beginning of the data set. When these records are processed in TSS, the block is used as the unit of transfer to and from the device involved. A block can consist of one or more logical records. Data sets with physical sequential organization are called PS data sets. You will use PS data sets each time you process magnetic tape in your programs. Volumes containing data sets with PS organization can be interchanged among TSS and IBM OS or OS/VS installations.

Data Set Residence

Maintaining Program Libraries

A program in TSS can consist of one or more object modules that are linked and are executable. A program consisting of only one object module is stored entirely within one library; a program that consists of several object modules may reside in different libraries, depending on how you have stored them. During linkage editing and during execution, the system can automatically retrieve all required object modules if you have defined the libraries that hold them.

There are four categories of program libraries:

- System library (SYSLIB)
- User library (USERLIB)
- User-defined job libraries
- Other user-defined libraries

System library, accessible to all users, includes TSS/360 programs and the installation's standard subroutines and functions.

User library is the private library that was assigned to you when you were joined to the system. This library is available each time you log on. If you do not employ job libraries in a task, all the object modules resulting from your use of the language processors are placed in your user library. In addition, if no special library is assigned for the output of the linkage editor, the linkage editor object modules are placed in your user library.

Job libraries are defined for use within one task when you want to restrict your user library to checked-out standard object modules that you execute frequently or that you use frequently in the buildup of other object modules; or you may want to use a special object module that will temporarily replace one you normally would use.

The program library list, a defined hierarchy of those libraries, is set up at log-on time, and consists of the user library and SYSLIB. Job libraries designated for a task are removed from the hierarchy at log-off time.

The library at the top of the list always automatically receives all object modules resulting from language processing. If no job libraries are defined, the library at the top of the list is always the user library. However, you can specify that a job library be added to the program library list to receive the output of the language processors. You do this by issuing a DDEF command that defines the job library, and contains the

JOBLIB operand (see Example 7 in Part III). When this command is executed, the name of the job library is placed at the top of the program library list. That library then receives all subsequent outputs of the language processors until another job library is defined (and it is placed at the top of the list), an existing job library is moved to the top of the list using the JOBLIBS command, or a RELEASE command is issued for the job library.

In addition to using the program library list to store object modules, the system uses this list to control its order of search when looking for object modules that must be loaded at execution time. The library at the top of the list is searched first, then the next, etc.; finally the user library and SYSLIB are searched.

The program library list can also be used, during linkage editing, to define the following for the system:

- The library that is to receive the link-edited object module.
- The sequence in which libraries are to be searched by automatic call if the system must find other object modules that will complete the link-edited object module.

For example, if no other library is specified, the output of the linkage editor is stored in the library currently at the top of the program library list. If another library is specified at the time the linkage editor run is defined, that library receives the link-edited object module. That library can be the user library, any of the current job libraries, or a special library defined by a DDEF command that has no JOBLIB operand.

Using Public and Private Volumes

When a data set is stored in the system, it resides on one or more direct-access or magnetic-tape volumes; the identification of these volumes is available in the system catalog. A volume can be a removable disk pack or a reel of tape. It should also be noted that physical sequential data sets are not cataloged when they are created (the CATALOG command must be used to accomplish this), and their residence is restricted to private volumes.

At system startup time, the system operator designates each direct-access device and its associated volumes as either public or private. A public volume is a direct-access volume that must be mounted prior to the beginning of system operation, and must remain mounted during operation; it can be used by many users concurrently. A private volume can be mounted or dismounted at any time prior to or during operation; it is restricted to use in one task at a time. For more than one of your tasks (say one conversational and one or more nonconversational) to access the same data set residing on a private volume at the same time, PERMIT and SHARE commands must be issued to give your userid sharer status, despite the fact that the userid is the same. Magnetic-tape volumes are always classified as private volumes; direct-access volumes can be either public or private. Magnetic tape volumes and physical sequential formatted direct access volumes are always classified as private volumes; VAM (Virtual Access Method)——formatted direct access volumes may be either public or private.

The system assumes that you desire storage on a public volume unless you specifically ask for storage on a private volume. Public volumes are always mounted and available for allocation to your task, thus providing the most efficient type of storage for data sets which must be retained in the system.

If you employ private volumes, you may need to wait for devices on which to mount those volumes. Each time a request is made for a device on which to mount a private volume, the system must determine whether or not it can honor the request, based on the current requirements throughout the system for that device.

Volume and Data Set Labels

All volumes used to store cataloged data sets must contain standard volume and data set labels to permit the system to locate the data sets.

All public direct-access volumes automatically contain standard volume and data set labels, which the system creates and maintains. Direct-access volumes can also contain user's data set labels, which are processed by user-written routines.

Magnetic-tape volumes may contain (1) standard volume and data set labels, or (2) standard volume and data set labels plus user's data set labels, or (3) they may be unlabeled. All labels, on magnetic-tape volumes with standard labels, are also created and maintained by the system; user's data set labels are processed by user-written routines.

Detailed explanations of standard labels and their use on direct-access and magnetic-tape volumes are given in *Data Management Facilities.*

Tailoring TSS to Meet Your Needs

You can tailor the operating environment within which your task is performed to meet your specific needs without affecting anyone else's use of the system. You can rename existing commands and keyword operands, and provide your own default values for omitted operands; these alterations can be temporary or permanent.

User Profile

The system maintains a special data set, called a user profile, which contains the defaults for operands, synonyms for command names and operand keywords, and command symbol values. The first time you log on, the prototype user profile in the system library (SYSLIB) is copied into your virtual storage. You can use this profile as it stands, or you can change it with the DEFAULT, SYNONYM, or SET commands.

- DEFAULT allows you to change system-supplied default values for command operands.
- SET allows you to define a command symbol that may be referenced or modified by other commands.
- SYNONYM allows you to rename commands, operands, expressions and values.

The changes made with these commands will affect only the current task's operating environment. If you want these changes to be included in your permanent user profile, you can issue the PROFILE command to copy this altered user profile from virtual storage into your USERLIB. Example 30 in Part III illustrates the use of these commands. For a detailed description of user profile management, see *Command System User's Guide*.

Defining Your Data Set

Before a problem program or a command can process a data set, the system requires complete information about the data set, including the manner in which it is to be processed. You can make this information available from a variety of sources; for example:

DEFINITION OF PROBLEM PROGRAM I/O DATA SETS	DEFINITION OF DATA SETS PROCESSED BY COMMANDS
1. I/O source statements	1. DDEF commands (which in turn may use information provided by your user profile)
2. DDEF commands or DDEF macro instructions (which in turn may use information provided by your user profile)	2. System catalog
3. System catalog	3. Command itself
4. Problem program	
5. Data set label	

The following paragraphs describe how to use these sources to identify data sets for the system.

Data Control Block

The information required to identify a data set to be processed by a problem program is contained in the data set's data control block (DCB), a group of contiguous fields in your program. The DCB contains these types of information:

- The name of the DDEF command (the ddname) to be associated with the data set
- Type of data set organization
- Record-format information (format type, record length, etc.)
- Device-dependent options
- Exit addresses:

  SYNAD: synchronous error exit address, for automatically transferring control to a user-supplied routine if an uncorrectable I/O error occurs.

  EODAD: end of data set address, for automatically transferring control to an end-of-data routine when end of an input data set is detected during processing.

  EXLST: address of an exit list in which (in the case of sequential data sets intended for interchange with OS or OS/VS you can define the address of routines for creating and verifying the user data set labels that can be employed on magnetic-tape and direct-access volumes; or the address of a routine to be used at OPEN TIME for modifying the data control block.
- Working storage used by the access method routines.

You request the system to begin construction of a data control block at assembly or compilation time. There are various ways in which the fields in a data control block may be filled. For example, some may be filled in at assembly/compilation time. Others may be filled in during program execution from user or system-supplied information. In any event, the fields are filled in according to a fixed priority scheme based on the source supplying the information. The sources of information and their priorities are:

1. Your program
2. DCB macro instruction
3. DDEF command (and system catalog)
4. Data set label

Not every source is valid for every field. These two general rules apply: (1) When a field has been filled by a higher priority source, it cannot be replaced by information from a lower priority source. (2) A field that has not been specified by a higher priority source, may be filled in by a lower priority source if that source is valid for that field (see Figure 6).

1. You can include one or more routines in your program, to add to or modify the contents of a data control block. Generally, these routines can be called at any time during execution. The restrictions on the use of a problem program to modify a data control block are described under the DCB macro instruction in *Assembler User Macro Instructions*. These facilities simplify problem program modification of a data control block in the assembler language:
   - A DCBD macro instruction (described later) can be used to symbolically refer to the fields of a data control block by their field names.
   - At OPEN time, the system provides a DCB exit during which the problem program can, in effect, call upon a user-written DCB modification routine that will update the DCB and return control to OPEN.

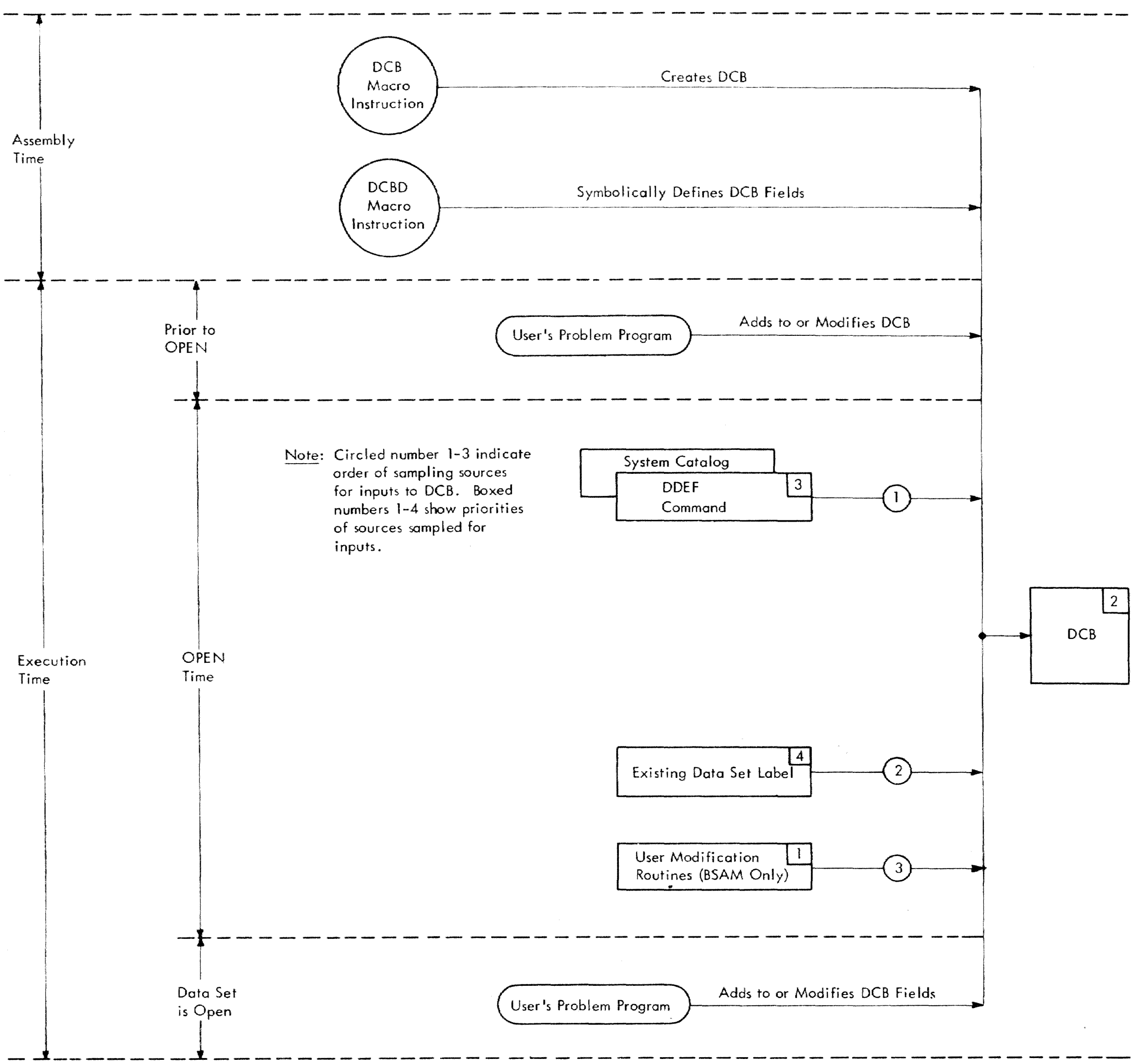


Figure 6. Flow of Information to and from a Data Control Block

2. The DCB macro instruction can be used to fill in any, or all, fields at assembly; however, once a field is specified in this way, it can be changed only by your problem program.
3. At OPEN time, information from the DCB operand of a DDEF command can be, and frequently is, used to complete the data control block. This process is shown in Figure 6. Any field that is empty at OPEN time, and for which the DDEF is a valid source, can be filled by DDEF information. If the data set to be processed is an input data set that was previously catalogued, its DDEF command will indicate this and the system will retrieve certain data control block information (e.g., the data set's location) from the system catalog.
4. Also, at OPEN time a field of the data control block for an existing data set can be filled with information from the data set's label if this field has not been specified by any other source and the data set label is a valid source for that field.

This procedure for data control block definition and modification can be greatly simplified for most applications; the flexibility is provided for the special case where data control block changes must be made between assembly time and the time a data set is actually processed. In these situations, the facility to modify allows you to change only the required fields; you do not have to restate the entire data control block each time a program is run. To facilitate data control block modification, you should include in the data control block only those fields needed for program execution—others should be left empty for possible subsequent fill-in. Once the data set is closed, the DCB is restored to its pre-OPEN state. When the data set is opened again, the system starts the fill-in procedure based on the data control block information provided by the DCB macro instruction and any problem program modification to the DCB since the last CLOSE.

Identification of Assembler Data Sets

A data set, to be processed by a problem program written in assembler language, must be identified by a DCB macro instruction in the source program. The assembler uses the DCB macro instruction to set up the data control block at assembly time, and, if you have supplied operands in the DCB macro instruction, to enter those operands into the data control block.

The number of operands that can be specified in the DCB macro instruction depends upon the organization of the associated data set.

The symbolic chain that relates the macro instructions used for data retrieval and storage (GET, PUT, READ, WRITE, etc.) to their associated data sets is shown in Figure 7. It also illustrates how information in the data control block and DDEF command identify assembler data sets to the system.

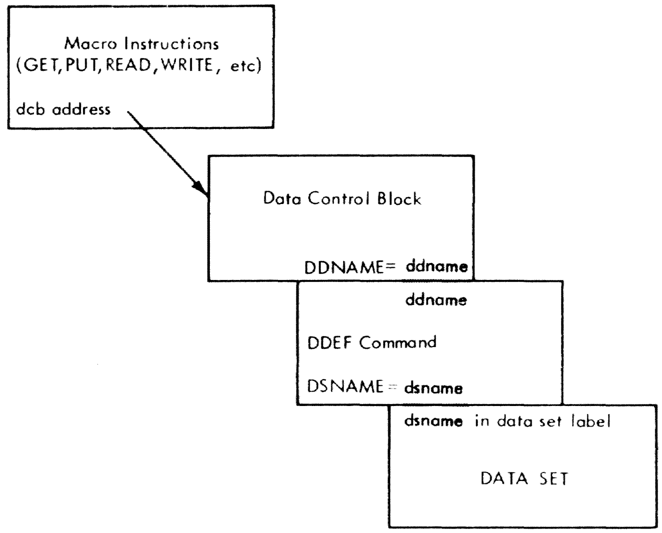


Figure 7. Data Set Identification

Data Definition Commands

The DDEF command is used to identify a data set during execution of a task and to define its requirements for system resources. It may also be used to define a job library, to define a special data set for the DUMP program control command, to complete the data control block of a program at execution time, and to concatenate input data sets (i.e., relate them so that several different data sets can be read in as if they were one).

Any DDEF command you issue during a task remains in force throughout the task, unless you enter a RELEASE command for that data set. The RELEASE command is the opposite of the DDEF command: the DDEF command sets up task control information for the data set; the RELEASE command removes that information. If the DDEF required a private volume to be mounted, RELEASE can be used to free it for assignment to another task.

The DDEF commands used in a session or in a command procedure need not be issued directly during the session or be included explicitly in the command procedure. One, or more, or all, of the DDEF commands needed can be made available by using the CDD (call data definition) command.

The CDD command is used to retrieve one or more DDEF commands from a data set; you must supply the name of the data set. If this is all you specify, the system assumes that you want to use all the DDEF commands in the data set. If you want to use only selected DDEF commands, you identify each by its ddname. You should prestore frequently used DDEF commands in a data set and call them in this fashion wherever possible. CDD can be used in either conversational or nonversational tasks.

In a conversational task, the system analyzes the data set's requirements at the time the DDEF command is issued. It will then attempt to allocate the required resources (and, for private volumes, issue any mounting messages that are required) at that time. If the required space cannot be allocated, or the specified volumes cannot be mounted, the system will inform you, thereby allowing you to proceed with other work.

The DDEF command is illustrated in the examples, and is discussed in detail in Appendix E.

System Inquiry Commands

There are several commands in TSS with which you can request specific information from the system regarding your data set, catalog, and job libraries. If you issue these commands in a conversational task, the information is displayed at your terminal; in a nonconversational task it is sent to the task's SYSOUT data set. The facilities provided by these commands are summarized below.

The PC? command is used to request the presentation of a concise listing of all or part of your cataloged data sets. You will be presented with the data set name, the access (owner access if owned by you; user access if owned by someone else), and the user identification of the owner if it is not owned by you. The PC? command can conveniently be used at regular intervals without an operand to present the entire catalog listing for help in housekeeping.

The DSS? command is used to request presentation of the status of one or more cataloged data sets. Information is given pertaining to sharing status, access status, device type and volume identification, creation and expiration dates, and data set organization. PC? should be used when a general listing of data sets is required; DSS? should be used only when more detailed information is required about the status or organization of data sets.

The POD? command is used to request a list of the member names (and, optionally, the alias names and other member-oriented data) of individual members of virtual partitioned data sets, such as your user library and your cataloged job libraries.

The DDNAME? command can be used to request a display of all DDNAMES you have defined within a task or just those for the job libraries you have defined. Used with the JOBLIBS command, it can be used to review and modify your JOBLIB chain.

The LINE? command requests presentation of a line or a series of consecutive lines of a line data set that you own or are now sharing. The data set must either be cataloged or defined by a previous DDEF in the current task.

Data Set Establishment

The Text Editor

The Text Editor is a powerful command repertoire provided for the TSS user. These commands provide for manipulating lines of information, either within an existing region or line data set, or as they are being entered dynamically into a region or line data set. With the text editing facility, you can create and edit data sets simultaneously. You can correct, insert, or delete lines; or segment a data set. You can transfer lines from one data set to another. You can also display lines of a data set at your terminal and nullify previous changes that were made by the text editor commands. *Command System User's Guide* provides a complete discussion of all the facilities of the Text Editor, including a description of the commands available.

Prestoring Data in the System

Data that is prestored in the system has been created as virtual sequential data sets, or virtual index sequential data sets, or members of partitioned data sets, and then stored on public direct-access volumes. System operator initiated commands prestore data on public volumes. Normally, the DATA command stores a data set on a public volume. However, you can cause a data set to be prestored on a *private volume* by issuing a DDEF command (with VOLUME=PRIVATE operand) prior to the DATA command. In either case, the data set is automatically cataloged at the time it is created.

Data Command

The DATA command prestores data sets entered from the terminal (conversational mode) or from the SYSIN data set (nonconversational mode). This method is particularly effective for relatively small amounts of data, such as small program input data sets, SYSIN data sets for nonconversational tasks and data sets consisting of DDEF commands.

The DATA command builds a virtual sequential or virtual index sequential data set; or adds to an existing virtual sequential or virtual index sequential data set. A detailed description of the use of this command is contained in *Data Management Facilities.*

Operator-Assisted Input

You can also enter data by means of nonconversational tasks that are initiated by the system operator.

Example: You can have a command procedure entered (and set up a nonconversational task for execution), or you can create a virtual sequential or virtual index sequential data set and store it on a public direct-access volume. The tasks are initiated by an RT command issued by the operator.

You can submit data sets on punched cards to the system operator, who can then enter the data into the system via a high-speed reader designated by the system. Two types of input data sets are permitted: command procedure data sets and data card data sets. The two types can be interspersed, one following another, in any order within a single batch of punched cards. The rules for setting up these data sets are given in the following paragraphs.

NOTE: If you want to enter a command procedure together with the data sets it refers to, you must make sure the data sets precede the command procedure. The system, generally, will try to execute the command procedure as soon as it has been read.

Command Procedure Data Set: This contains all commands needs to run a nonconversational task. Each command is punched on a new card, in exactly the format used to enter commands from a terminal. The first card of the data set must be a LOGON command; the last card, LOGOFF (see Figure 8). A SECURE command card must immediately follow LOGON if any private devices are required. Other commands are as required for the particular task.

When the command procedure data set is read in, it becomes the SYSIN data set of a nonconversational task and is executed as soon as the necessary system re-

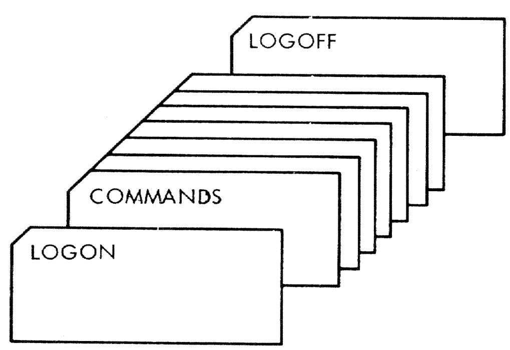


Figure 8. Organization of Command Procedure Data Set

sources are available. After execution, the SYSIN data set is eliminated. It does not remain cataloged nor does it remain in system storage.

Data Card Data Set: This contains any information you want to put into public storage as a cataloged data set. It may also include commands. You may enter a command procedure data set in this way, if you do not wish to have it set up as nonconversational task after entry; or you may prestore DDEF commands. When this type of data set is read, a virtual sequential or virtual index sequential data set is created and cataloged in public storage, where it will reside until it is erased. Unlike the command procedure data set, it is not executed upon being read.

The organization of a data card data set is shown in Figure 9. The first card of the data set must be a data descriptor card; the last one a %ENDDS card. Each data card corresponds to one logical record.

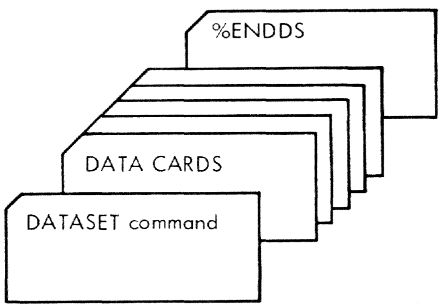


Figure 9. Organization of a Data Card Data Set

The data descriptor record identifies the data set as a data card data set. It must start in column 3 with the operand DATASET, followed by the user identification and the data set name under which it is to be cataloged. The following information can be supplied to the system on the same card:

- The code to be used in reading the cards (EBCDIC or BCD).
- The first card column to be read in creating the data set.
- The last column to be read.
- The organization of the new data set. If LINE is specified, a VISAM data set will be created, each line prefaced by a seven character line number. If left unspecified, a VSAM data set will be created. The system assigns 100 as the first line number in a VISAM data set and increments by 100. The maximum number of lines (data records) is 100,000.
- The action to be taken by the system (accept the record, skip the record, or end reading of the data set) if an uncorrectable read error occurs.

The terminating card, %ENDDS, must be punched starting in column 3.

The format of the data descriptor record is given in *Command System User's Guide*.

Sharing and Protecting Your Data Sets

You cannot gain access to any data sets other than your own unless you have system authorization to do so, or have been given authorization by another user who owns the data sets involved. In TSS, cataloged data sets may be shared or unshared.

A shared data set is cataloged and belongs to one user, but may be shared with other users on any of these bases:

1. *Read-only access:* The sharer may read the data set, but may not change it in any way.
2. *Read-and-write access:* The sharer can both read and write to the data set, but he may not erase it.
3. *Unlimited access:* The sharer, in effect, can treat the data set as his own; he may even erase it.

You issue a PERMIT command to designate the other users who may share your data sets, which data sets they may share, and the type of access those users may have. You may also use the PERMIT command to change any access authorization you may previously have given. A separate PERMIT command is required for each level of access to a data set, but any number of sharers may be authorized for the same level of access with a single PERMIT command. After issuing the PERMIT command, you must issue an ABEND or LOGOFF command to update your catalog entry regarding who may share which data sets and to what level of access. The sharer will not have access to your data set until this update has been effected.

To gain access to a data set for which he has been previously authorized, the sharer must issue a SHARE command. To see how this command is used, assume that the sharer's user identification is JMC200 and that he has been permitted to share one data set. The data set is owned by user RKP100, and is cataloged by him under the fully qualified name ENG.PHYSICS.COMAR.TEST2. Assume also that the sharer wants to name the data set ENG.CHEM.NOTAR.TEST1. He would then issue the SHARE command shown at the top of Figure 10. In response to that command, the system would search the owner's catalog to see if the prospective sharer is authorized. If he is not, the command is ignored and the user is informed that he may not share the data set; if he is authorized, the system places in the sharer's catalog a pointer to the *owner's (complete) name of the data set.* This is a sharing descriptor that bears the name by which the sharer refers to the data set. Whenever the *sharer* subsequently refers to the data set by his name, the system locates the data set by the search procedure shown on Figure 10.

The name assigned to a data set by its owner is not affected in any way by other users who assign their

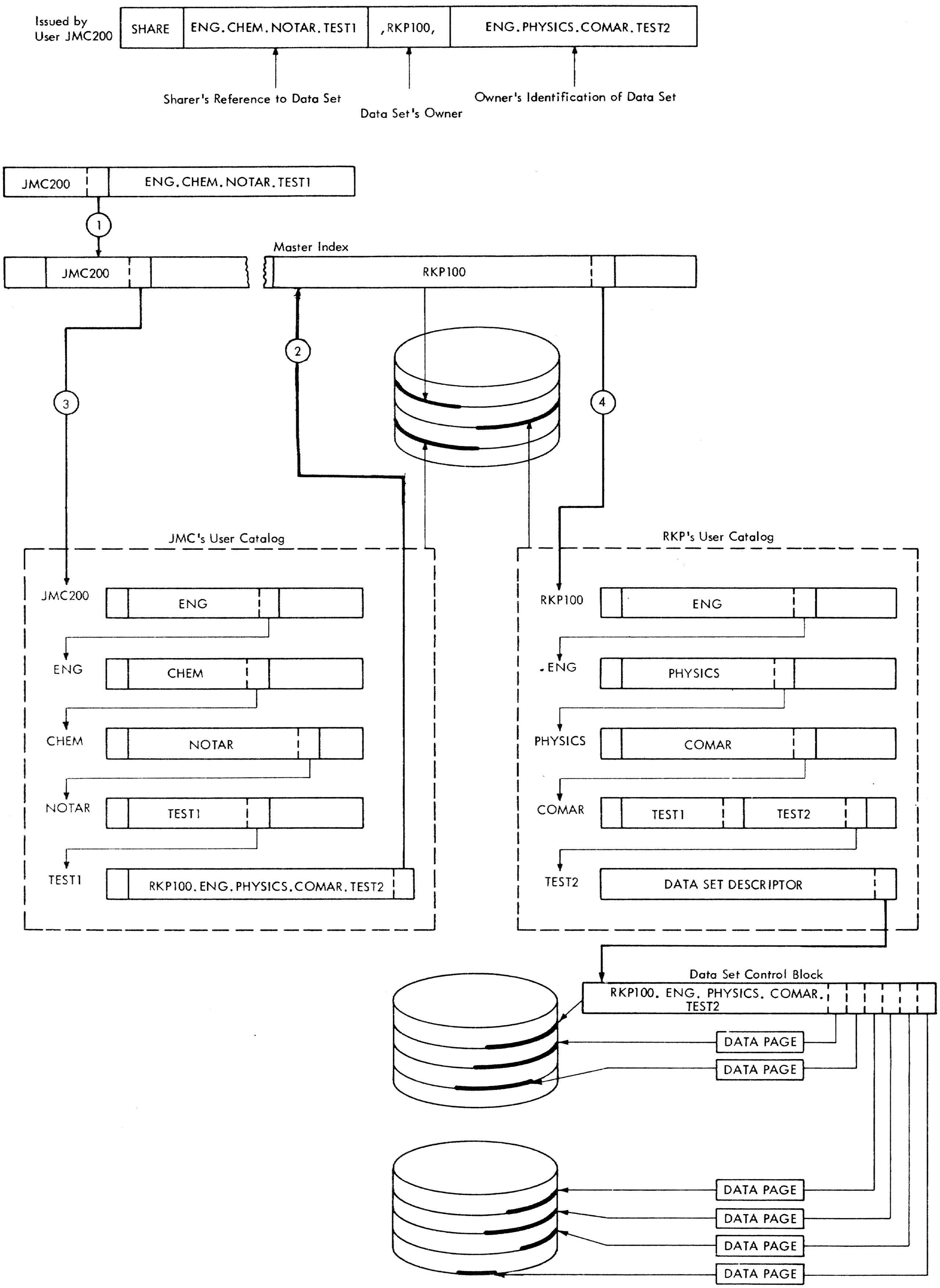


Figure 10. Sharing of Cataloged Data Sets

own names to that data set. Sharers may use the same name as the owner because user identifications are unique in the system.

A sharer's catalog entry is not removed if the owner erases or uncatalogs a shared data set. Each sharer must use the DELETE command to update his own catalog (i.e., to get rid of sharing descriptor entries). When deleting a shared data set, the user must enter the complete sharing descriptor; there is no prompting for individual data sets under each sharing descriptor.

If the owner allows another user to share all his data sets, the sharer can refer to them as a group in the SHARE command by specifying his name for the collection and then specify ALL. In this case, the system places a pointer to the owner's user identification in the sharer's catalog, thereby making all of the owner's catalog available to the sharer. Similarly, groups of data sets with names having common higher-order components can be specified by using partially qualified names for the owner's catalog.

To be concurrently accessible by more than one task, a data set must be cataloged and must be a virtual storage data set.

Data Set Manipulation

Copying, Modifying, and Erasing Data Sets

You can use the CDS command to make a copy of any data set (or any member of a partitioned data set) to which you have access except data sets whose records are in undefined format, such as program module libraries. You can also use it to renumber the lines of a line data set as it is being copied. Both the original and copy data sets must be defined in your task.

You can use the MODIFY command to insert, delete, replace or inspect records of a VI data set, or of a VI member of a VP data set. You have to identify the record to be modified (by its key or line number). You can review modifications, and play back corrected lines for confirmation of your changes.

You can use the VV, VT, and TV commands to copy your data sets depending on their origin and desired destination. The VV command causes a VAM data set to be copied into public storage. The VT command causes a VAM data set to be reproduced on 9-track magnetic tape. The TV command retrieves and writes into public storage a data set previously written on 9-track magnetic tape by the VT command.

You can use the ERASE command to erase data sets that you own. If you are sharing someone elses data set, you can remove its entry from your catalog by issuing the DELETE command, and erase it if you have unlimited access.

Transferring Data to Standard Output Devices

The three commands used to transfer data sets to specific output devices are:

PRINT—initiates transfer of a specified data set to the printer for high-speed printout.

WT—initiates transfer of a specified data set to a magnetic-tape device for recording in a format suitable for printing either off-line or via the PRINT command.

PUNCH—initiates transfer of a specified data set for card punching.

You can issue these commands in either conversational or nonconversational tasks. Each command requests the system to initiate an independent nonconversational task to perform the function of the command. Once that task is set up, the issuing task continues.

PRINT Command

The PRINT command prints data sets on the computer center's high-speed printer. It processes data sets that were created by using basic sequential, virtual sequential or virtual index sequential access methods.

The data set may or may not be cataloged; if not, you must define it by a previous DDEF command; if cataloged, you can specify in the PRINT command that the data set is to be erased after printing has been completed. The PR macro instruction may also be used to perform these functions. System programmers can also invoke the PRINT command to print data sets written in ASCII[1] code. See *IBM Time Sharing System: System Programmer's Guide* GC28-2008, for a description of this facility.

WT (Write Tape) Command

The WT command writes a data set on tape for later processing, either off-line or by the PRINT command. It can process, as input, data sets that were created by using either virtual sequential or virtual index sequential access methods. You must give the name of the input data set. If the input data set is not cataloged, you must define it by a previous DDEF command. If the input data set is cataloged, you can specify the ERASE option of the WT command to erase the input data set after the WT task is completed. The WT macro instruction may also be used to perform these functions.

PUNCH Command

The PUNCH command punches a data set on cards, using the installation's high-speed card punch. It can process data sets that were created by using either the virtual sequential or virtual index sequential access methods. The data set may or may not be cataloged. If not, you must define it by a previous DDEF command; if cataloged, you can use the erase option in the PUNCH command to specify that the data set is to be erased

[1] The American National Standard for Information Interchange, ANSI X3.4-1968, hereinafter referred to as ASCII.

after punching is completed. The PU macro instruction provides the same options and facilities as the PUNCH command.

See *Command System User's Guide* for a discussion of commands; *Assembler User Macro Instructions* for macros.

Assembler Language Facilities

Input/Output During Program Execution

TSS includes complete program I/O facilities for the conversational and nonconversational modes of operation. In both modes, conventional I/O facilities and dynamic I/O facilities are provided. Depending upon the application, these dynamic facilities can be used alone, or in conjunction with those for conventional I/O.

The principal differences between these two facilities, illustrated in Figure 11, are summarized below.

CONVENTIONAL I/O	DYNAMIC I/O
1. Source program must contain all instructions required for I/O operations. In effect, data processing must be preplanned in detail.	1. Source program need not contain instructions for conventional I/O. All I/O can be achieved via SYSIN/SYSOUT, using Dynamic I/O source statements, and/or Program control commands and statements (Conditional dynamic I/O is possible).
2. Data to be processed must be made available and defined for system prior to program execution.	2. Data to be processed can be decided upon, based on results of processing; no predefinition of data for system is required.

Dynamic I/O facilities can be used either by issuing commands and statements at the terminal, or by including special source language instructions in the source program.

- Because program control facilities are equally useful for program checkout and modification, a description of program control commands and statements is given in Appendix B.
- Dynamic I/O facilities peculiar to source language are described under the summary of the assembler language's problem program I/O facilities.

Conventional Problem Program Input/Output

Assembler language users can apply one or more of the facilities shown in Figure 12 to control conventional program I/O. The access method facilities (VSAM, VISAM, VPAM, BSAM, and QSAM) permit data sets to be created and processed, using system macro instructions that are similar to the I/O statements in higher-lever languages.

TSS also includes the resident terminal access method (RTAM) and the multiple access method (MSAM).

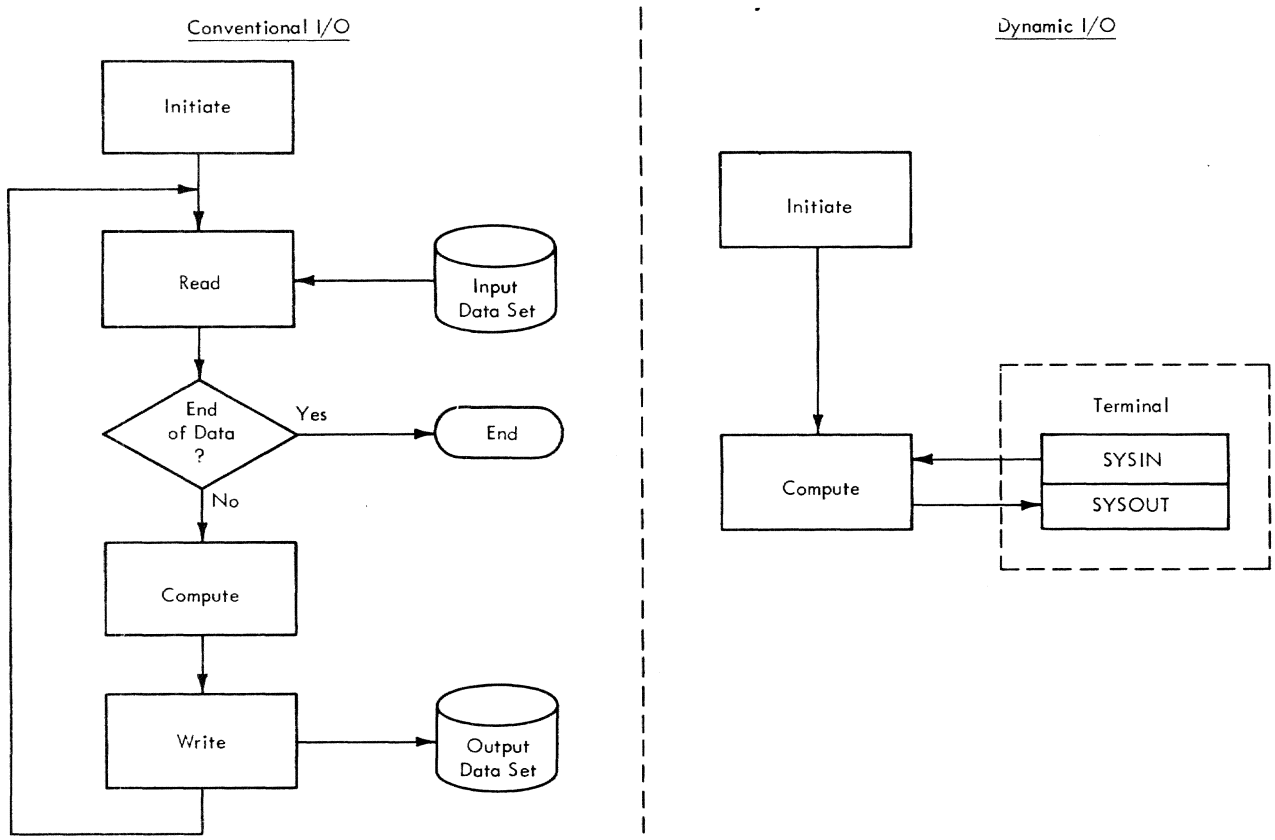


Figure 11. Conventional vs Dynamic I/O

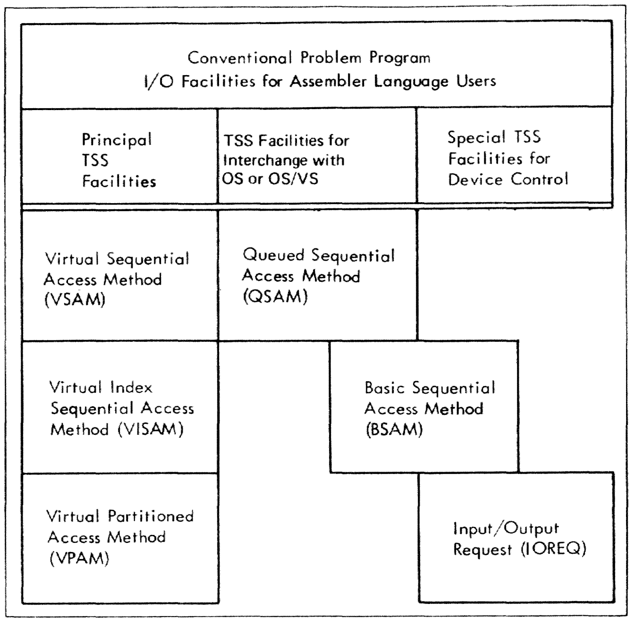


Figure 12. Conventional I/O Facilities

The applications programmer's use of the RTAM facilities is indirect, through the command language. In this publication, the RTAM facilities are considered by describing the effects of inserting commands and data, via SYSIN, and of the system's outputs via SYSOUT. MSAM, which can be used only by users in privilege-class E, processes data sets on unit record devices, such as card readers, card punches, and printers. Full descriptions of RTAM and MSAM appear in *System Programmer's Guide;* system programmers generally can use RTAM facilities directly. Depending upon the planning for a specific installation, the MSAM facilities for control of card readers and punches, and printers may be available to users; those facilities are described in *Assembler User Macro Instructions.*

The I/O request (IOREQ) facility permits the control of I/O devices, using system macro instructions similar to those used for machine-level programming (i.e., IOREQ allows you to write your own channel programs).

The virtual storage access methods (VSAM, VISAM, and VPAM) are specifically designed for the programming environment of TSS. They are simple to use, yet they provide a wide range of facilities for data storage and retrieval.

The basic sequential access method (BSAM) is intended primarily for data set interchange with OS, OS/VS, or when the data set is to be written on magnetic tape. Also, BSAM can be used for applications requiring limited device control. For special applications that call for more direct device control, the I/O request (IOREQ) facility can be used.

The relationships between the data-set organization and the data-management system macro instructions of each I/O facility are summarized in Figure 13. Complete information on these macros is available in *Assembler User Macro Instructions.* A discussion of access method facilites is contained in *Data Management Facilities.*

General Service Macro Instructions

The general service macro instructions (used to identify and prepare data sets for processing, and to terminate their processing) are essentially the same for all access methods. The mnemonics and short titles for these macro instructions are:

- DCB—Define data control block for I/O operations
- DCBD—Provide symbolic names for fields of a DCB
- OPEN—Prepare a DCB for processing
- CLOSE—Disconnect a data set from user's problem program

DCB Macro Instruction

The DCB macro instruction is included in a source program to reserve space for a data control block and, if you desire, to place in that data control block, at assembly time, information describing the characteristics and intended uses of a data set. Table 1 briefly describes each of the operands in a DCB macro instruction. Also, it indicates the access methods in which each operand can be specified, if the DCB macro instruction is the source of the information. Table 1 also gives the valid alternate sources for each operand.

A DCB macro instruction is required for each data set processed by the assembler language's conventional I/O facilities.

DCBD Macro Instruction

A DCBD macro instruction is required if you want to refer to the fields of a data control block by their symbolic names in your program. A dummy control section (DSECT) will be generated at assembly time to provide a symbolic name for each field that can be specified in any data control block. By properly initializing your base registers you can thus refer symbolically to any or all fields of the data control blocks in your program. Only one DCBD macro instruction may be issued during an assembly; if you issue more than one the instruction will be ignored and a diagnostic message will be issued.

OPEN Macro Instruction

The OPEN macro instruction completes one or more specified data control blocks so their associated data sets can be processed.

OPEN is common to all access methods; however, other aspects of the OPEN process (label processing, specification of the volume disposition when volume switching occurs, identification of the I/O access characteristics of the data sets involved and the related DUPOPEN macro instruction) differ with the access method being used and the intended processing of the data itself.

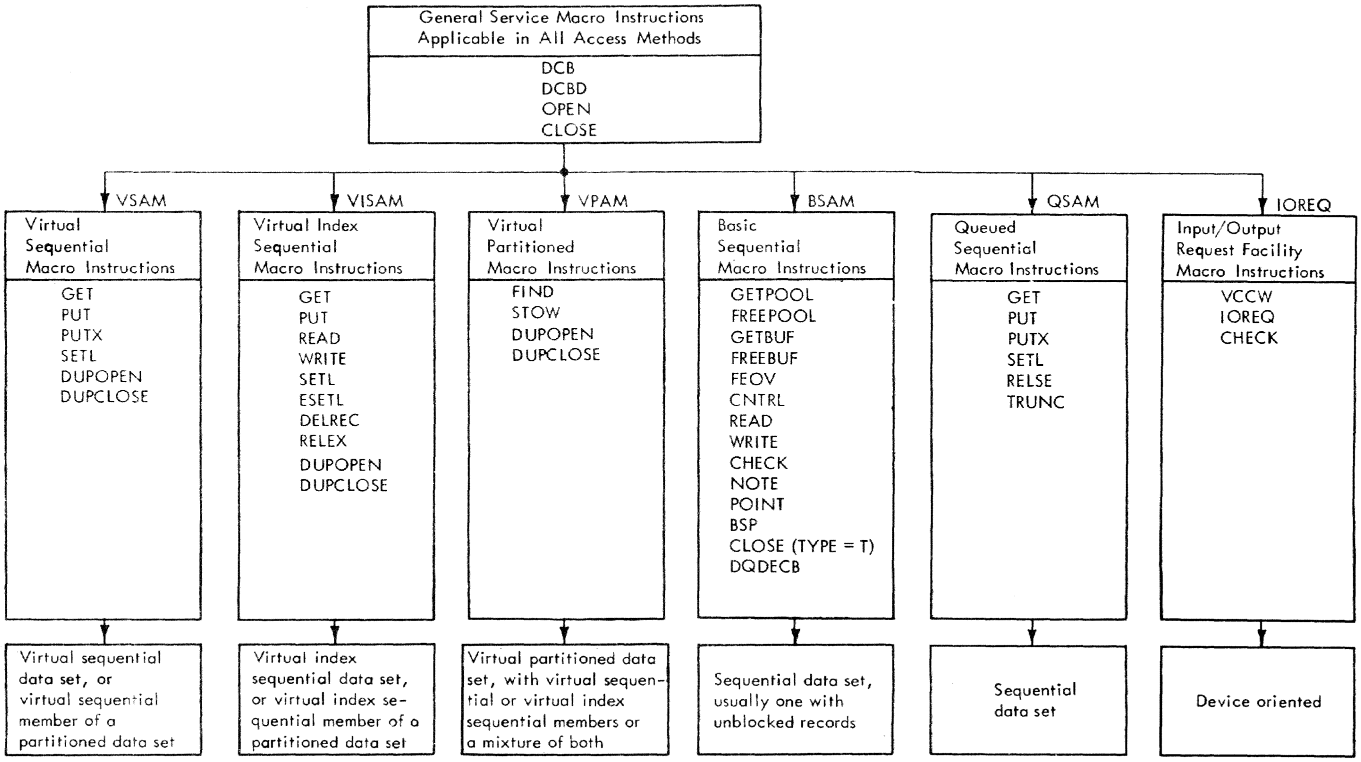


Figure 13. Summary of Data Management System Macro Instruction and Data Set Organizations

CLOSE Macro Instruction

A CLOSE macro instruction logically disconnects a data set from your program and should be issued when a data set's processing is completed.

Duplexing a Data Set

Critically important virtual storage (VAM organized) data sets on public volumes may be safeguarded against loss of data by duplexing them. The DUPOPEN macro instruction links a primary and secondary data set together such that all changes to the primary data set are immediately reflected by corresponding changes in the secondary data set. At any instant, therefore, the data sets should be exact duplicates. If read errors occur in the primary copy, the secondary copy is used for error recovery. To ensure that the two data sets are always identical, you should never perform an operation on either data set without invoking the duplexing mechanism.

You duplex a data set by issuing a DUPOPEN macro instruction instead of the OPEN macro instruction, specifying as operands the addresses of the data control blocks for the primary and secondary data sets. Whenever possible the two data sets should be allocated space on separate physical volumes. The data set properties specified in both data control blocks, and their corresponding DDEF statements, must be consistent.

The external storage required when duplexing a data set is exactly double that required by non-duplexed data sets, and the time required for data output is almost doubled. To save on time and resources, you should therefore be judicious in your duplexing requests.

Data sets that have been opened with a DUPOPEN macro instruction are closed with a DUPCLOSE macro instruction, with the address of the two data control blocks as operands.

Dynamic Input/Output for the Assembler Language

In addition to the program control facilities available to all users, the following macro instructions may be used for problem program communication with the system I/O streams:

- GATRD (read record from SYSIN)
- GATWR (write record to SYSOUT)
- GTWRC (write record to SYSOUT with ASA carriage control character)
- GTWAR (write record to SYSOUT and read response from SYSIN)
- GTWSR (write record on SYSOUT and read record from SYSIN)

Table 1. DCB Operands, Their Specification, Access Methods, and Alternate Sources

DCB OPERAND	SPECIFIES	APPLICABLE ACCESS METHOD								VALID ALTERNATE SOURCES		
		VSAM	VISAM	VPAM	BSAM	IOREQ	QSAM	MSAM	TAM	USER'S PROGRAM	DDEF COMMAND	DATA SET LABEL
DDNAME	Symbolic name identical to that used in ddname operand of DDEF command associated with data set	X	X	X	X	X	X			X		
DSORG	Data set organization	X	X	X	X	X	X			X	X	
RECFM	Record format information	X	X	X	X	X	X			X	X	X
LRECL	Logical record length	X	X	X	X		X			X	X	X
EODAD	Address of user's end-of-data routine for input data sets	X	X	X	X		X			X		
SYNAD	Address of user's synchronous error exit routine (entered when an uncorrectable error occurs in I/O operation)		X	X (only for VISAM members)	X	X	X			X		
KEYLEN	Key length		X	X (only for VISAM members)	X	X				X	X	X
RKP	Displacement of key from first byte of logical record		X	X (only for VISAM)						X	X	
PAD	Space to be left on each page of virtual index sequential data set (to allow subsequent insertions)		X	X (only for VISAM members)						X	X	X
MACRF	Type of macro instructions used in processing data set (GET, PUT, READ, WRITE, etc.)				X		X			X	X	
DEVD	Device on which data set resides plus, for some device types, device-dependent information (data code, tape density, etc.)				X	X	X			X	some device dependent information	some device dependent information
OPTCD	Optional service desired, write with validity check (for direct-access devices only)				X		X			X	X	X
BLKSIZE	Maximum block length				X		X			X	X	X
IMSK	Number code indicating what system error recovery and recording procedures (if any) are to be invoked				X		X			X	X	
EXLST	Address of user's exit list				X		X			X		
NCP	Number of consecutive READ, WRITE, or IOREQ macro instructions issued before CHECK macro instruction				X	X				X	X	
BUFNO	Number of buffers				X					X	X	
BFALN	Buffer alignment				X					X	X	
BUFL	Buffer length				X					X	X	
EROPT	Option to be executed if an error occurs						X	X			X	
BFTEK	Buffer technique				X				X	X	X	
PRTSP	Print spacing option					X		X			X	
STACK	Card stacker selection					X		X		X	X	
MODE	Mode of operation					X		X		X	X	
TRTCH	Recording technique for 7-track tape				X					X	X	
BUFCB	Buffer control block address				X					X		

- PAUSE (switch conversational task from program mode to command mode)
- COMMAND (switch conversational or nonconversational task from program mode to command mode)
- SYSIN (write and/or read a message in SYSIN/SYSOUT)

GATRD

The GATRD macro instruction reads a line data set record from your SYSIN and places it in your specified area. In conversational mode, the system prints an underscore on your terminal typewriter and unlocks your keyboard. The program containing GATRD then waits for you to insert a record. If no record is inserted, the task is terminated.

In nonconversational mode, the system refers to the SYSIN data set and reads its next record. You must arrange the records in the SYSIN data set so that the appropriate record is obtained by the system in response to each GATRD macro instruction. If the input record exceeds one line, a GATRD macro instruction is required for each line.

GATWR

The GATWR macro instruction writes a record to your SYSOUT from a user-specified area. In conversational mode, the record is printed on the terminal typewriter; in nonconversational mode, the record is stored in the task's SYSOUT data set.

GTWRC

The GTWRC macro instruction writes a message on the user's SYSOUT, with ASA carriage control character. Either Type I or Type II linkage is used, depending upon the privilege class of the user's program.

GTWAR

The GTWAR macro instruction writes a record from a user-specified output area to SYSOUT, and then reads a record from SYSIN into a user-specified input area. In conversational mode, the output record is printed at your terminal, and the program issuing GTWAR waits for you to insert the input record. If no record is inserted, the task is terminated.

In nonconversational mode, the system writes the output record to the task's SYSOUT data set, and reads the next record in the SYSIN data set. It is your responsibility to have the appropriate record available for each GTWAR macro instruction. If the input record exceeds one line, a GATRD macro instruction is required for each additional line.

GTWSR

The GTWSR macro instruction may be used only in the conversational mode; the system will terminate the task if an attempt is made to execute a GTWSR macro instruction in the nonconversational mode. The GTWSR macro instruction prints the output message on your terminal typewriter and waits for you to provide the input record, which, when entered, will be stored by the system in the area specified in the GTWSR macro instruction. If the input record exceeds one line, a GATRD macro instruction is required for each additional line.

SYSIN

The SYSIN macro instruction services the program in which it appears by providing information about the current operating task. This is accomplished by retrieving input (i.e. either a command or data) from the Source List or the SYSIN device for the task. A user can alter the action of the SYSIN routine by entering the system command prompting string (usually an underscore followed by a backspace) following the prompting string of the SYSIN macro routine produced at the terminal.

PAUSE

The PAUSE macro instruction switches a conversational task from program mode to command mode, while still retaining program control. A message specified in the PAUSE macro instruction is typed at your terminal. Control is then returned to you, who can then enter any commands. Any system output generated by a command issued after the PAUSE is typed at your terminal. After each command is executed, the system prompts you for the next command. To resume a previously interrupted CALL command (one that was executing the object module containing PAUSE) you issue another GO command. This GO can specify that the interrupted object module be resumed at the point of interruption, or at any other point, or a new object module can be called and its execution begun. If a PAUSE macro instruction is encountered in a program executing in the nonconversational mode, the message is written on SYSOUT, and program execution continues.

COMMAND

The COMMAND macro instruction has a function similar to PAUSE; however, it can be executed in either conversational or nonconversational tasks. In a conversational task, COMMAND has the same effect as PAUSE.

In a nonconversational task, the system refers to the SYSIN data set when COMMAND is executed. You can prestore any commands you want in the SYSIN data set, to subsequently control the system. Any system messages resulting from the execution of these commands are sent to the task's SYSOUT data set. The program execution is resumed by a GO command.

The LINE? command can also obtain output dynamically. One or more lines from a line data set that belongs to you, or that you are currently permitted to share, can be specified in this command. The LINE? command can be issued in either the conversational or nonconversational mode. In conversational mode, the specified line or group of lines is printed at your terminal; in nonconversational mode, the specified line or group of lines is written on the task's SYSOUT data set.

Table 2 summarizes the processing rules for input data through use of the input GATE macro instructions (GATRD, GTWAR, and GTWSR). Table 3 summarizes the processing rules for output data through use of the output GATE macro instructions (GATWR, GTWAR, and GTWSR).

Communication with the Operator

These macro instructions may be included in assembler written problem programs to communicate with the system operator:

- WTO (write-to-operator),
- WTOR (write-to-operator-with-reply).

These macro instructions should be used only in programs with specialized I/O routines to request operator intervention.

Communications with the System Log

The system log is a data set that is maintained by the system on a direct-access device. Its characteristics are established according to the needs of an installation, and are defined at the time that the system is generated.

Sources of information for the system log are:

- The operator, who may enter any noteworthy events that occurred on his shift (MESSAGE command).
- Assembler-written problem programs—WTL (write-to-log) macro instruction.

Table 2. SYSIN Records Specified with GATE Macro Instructions

	SOURCE	DESTINATION
DEVICE	RULES	RULES
Terminal keyboard	Each record is terminated by end-of-block character (EOB); this character is registered when system detects RETURN key has been pressed and not immediately preceded by hyphen; if necessary to continue record over more than one line, hyphen indicates record is being continued; hyphen does not become part of record; maximum line length: IBM 1052 and IBM 2741—130 bytes; IBM 3277—255 bytes; Teletypewriter Model 35KSR—80 bytes	These macro instructions specify expected length of input record Record is placed in area specified in macro instruction
Terminal card reader	Each record is terminated by EOB, which can be registered in one of two ways: If terminal's EOB switch is ON, EOB code is registered automatically after card is read or when EOB code is detected on card If EOB switch is OFF, an EOB code is transmitted only when detected on card or program tape Cards are 80 bytes long	
Direct-access	Each input record is a single record of a virtual sequential or virtual index sequential data set	

Table 3. SYSOUT Records Specified with GATE Macro Instructions

SOURCE		DESTINATION
RULES	DEVICE	RULES
These macro instructions specify length and location of data to be produced as output GATWR, GTWSR, GTWRC, and GTWAR are used for problem program output	Terminal	In conversational mode, output of these macro instructions appears on SYSOUT device Records longer than one line are continued Maximum line lengths: 1052 and 2741—130 bytes; IBM 3277—255 bytes; Teletype Model 35—80 bytes
	Direct-access	In nonconversational mode, output is written on SYSOUT data set for subsequent off-line printing

Commands and Macro Instructions

The following chart (Table 4) shows the commands available to you. The commands are grouped under ten general categories. The chart shows commands themselves, the corresponding macro instructions (if any), sample usages, and the examples in Part III which illustrate the command. Positional operand notation is used. The Command System User's Guide gives a fuller description of all command formats.

Table 4. Commands and Macro Instructions *(Continued)*

FUNCTION	COMMAND	MACRO INSTRUCTION	SAMPLE USAGES	EFFECT	ILLUSTRATIVE EXAMPLES
Task Management	Attention Button		(User presses button)	Gains attention of the system for logging at the very beginning of the session. Thereafter during your conversational task, halts current activity. See Appendix D for specific effect.	1-11,13-19, 21,22,24-30
	LOGON		ADUSERID,MYPASS,, ADACCT29	Identifies you to the system for initiation of your task. Here you enter your identification, password, and account number. Confirmation follows and full messages are standard.	All
	CHGPASS		CHGPASS	Notifies the system that you wish to change your password.	2
			CHGPASS NEWPASWD=WORD	Notifies the system that you wish to change your password to WORD.	
	BEGIN		BEGIN CALC	Notifies the system that you wish to connect to an MTT application program.	None
	LOGOFF		LOGOFF	Notifies system that you want to terminate your task.	All
	BACK		BACK ALPHA	Switches your conversational task to nonconversational mode. Here you specify the data set ALPHA as the source of further commands.	11
	EXECUTE		EXECUTE BETA	Requests the execution in nonconversational mode of a sequence of commands contained in data set BETA, while you continue in conversational mode at the terminal.	11
	CANCEL		CANCEL 4120	Terminates execution of nonconversational task which was assigned batch sequence number 4120.	7,19
	TIME		TIME 15	Allocates 15 minutes of processing time to the task before the user is notified by a message at the terminal if in conversational mode; by ABEND if in nonconversational mode.	2
	USAGE		USAGE	Presents a summary of system resources available to you as well as those that have been used by you since you were first joined and since the current LOGON.	21
	ZLOGON		ZLOGON	The user-defined procedure called ZLOGON is executed. After initial LOGON procedures are completed, this invocation is automatic.	1
General Data Management	CDD	CDD	CDD MYDDEFS	Causes execution of all the DDEF commands that you placed in a data set named MYDDEFS.	13
	CATALOG	CAT	CATALOG GAMMA,,R	Causes system to create an entry in your catalog for a physical sequential data set, or change an entry. Here an entry is created for the data set GAMMA. By default, the system will recognize it as a new data set, with access = R (read only).	4, 10, 22

Table 4. Commands and Macro Instructions *(Continued)*

FUNCTION	COMMAND	MACRO INSTRUCTION	SAMPLE USAGES	EFFECT	ILLUSTRATIVE EXAMPLES
			CATALOG DELTA,U,, SIGMA	Here you update (U) your catalog so that the data set SIGMA is cataloged under the name DELTA. The CATALOG command cannot be used for changing access for a VAM-organized data set and the access is therefore defaulted by a comma.	
	CDS	CDS	CDS SIGMA, SIGMA2	Copies the data set SIGMA, naming the copy SIGMA2.	18
	CLOSE	CLOSE	CLOSE IOTA,T	Causes the system to temporarily close the data set, IOTA.	
	DELETE	DEL	DELETE KAPPA	Removes the entry for the data set KAPPA from your catalog.	21
	DATA		DATA EPSILON	Requests the system to build a data set named EPSILON from the data or commands which follow.	11,13,16, 20,23
	DDEF	DDEF	DDEF MYDD,, MYDATA	Defines a data set and describes its characteristics to the system for the current task. The data set exists, or is being created. Here you define a data set named MYDATA. The second comma defaults its organization. The name of the definition is MYDD. Appendix E describes the other parameters.	3-11,13-15, 17,18,20, 22,23
General Data Management	ERASE	ERASE	ERASE DELTA	Erases data set (releases direct-access storage for other use), and if cataloged, deletes name from catalog. Here you release the storage space of the data set DELTA.	4,6,7,14,21
			ERASE USERLIB (MYPGM)	Here you erase an object program module named MYPGM from your USERLIB.	
	EVV		EVV 2311,387542	Causes a catalog entry to be created for a private VAM volume. Here you specify the device type and the volume serial number.	17
	JOBLIBS		JOBLIBS DDNAME= OMEGA	Causes your JOBLIB, OMEGA, to be moved to the logical top of your library list.	7
	MODIFY		MODIFY PHI	Permits you, with subsequent parameters, to insert, replace or delete records in a VISAM data set named PHI.	4,7
		OBEY	OBEY 'DDEF OUTDD,, OUTDS'	Causes the system to create a JFCB with a DDNAME of OUTDD and a DSNAME of OUTDS.	31
	PERMIT		PERMIT SIGMA2, ADPAL,RO	Authorizes other user to have access to your data set(s). Here you authorize read-only (RO) access to your SIGMA2 data set to a user whose identification is ADPAL.	18
	RELEASE	REL	RELEASE MYDD	Revokes the data definition established by the previously issued DDEF command named MYDD.	9,10,17
	RET		RET OMEGA,TCR	Causes modification to be made to the catalog entry for your VAM-organized data set OMEGA. It will occupy temporary virtual storage (T) and be deleted when the CLOSE macro instruction is executed (C). Access is changed to read only.	5,6

Table 4. *(Continued)*

FUNCTION	COMMAND	MACRO INSTRUCTION	SAMPLE USAGES	EFFECT	ILLUSTRATIVE EXAMPLES
General Data Management	SHARE		SHARE MINE,ADHISID, HISN	Creates entry in your catalog for data set for which owner has granted you authorization with PERMIT command. Here you cause an entry to be created in your catalog under MINE for the data set HISN belonging to a user whose identification is ADHISID.	18
	TV		TV ABC,XYZ	Retrieves data set that was written onto tape via VT command (ABC) and writes it on a VAM volume, with name XYZ.	25
	VT		VT ONE,TWO	Copies VAM data set (ONE) to magnetic tape as physical sequential data set (TWO).	25
	VV		VV FOUR,FIVE	Copies VAM data set (FOUR) onto direct access storage assigning name as FIVE.	25
Bulk Output	PUNCH	PU	PUNCH SIGMA	Causes the data set SIGMA to be punched on cards.	21
	PRINT	PR	PRINT SOURCE.PGM3	Causes the data set SOURCE.PGM3 to be printed on the high-speed printer.	5,9
	WT	WT	WT GAMMA	Causes the data set GAMMA to be written on magnetic tape for subsequent off-line printing.	21
Device Management	SECURE		SECURE (DA=3, 2311) (TA=3, 9)	Reserves devices required for private volumes during execution of nonconversational tasks. This command at the beginning of your sequence of commands for the nonconversational task secures three 2311 disk units and three 9-track tape units.	20
Program Management	LOAD	LOAD	LOAD MAIN2	Transfers the specified object module from its containing library to user's virtual storage.	9
	UNLOAD	DELETE	UNLOAD MAINB	Removes object module MAINB from user's virtual storage.	7,15,17,23
	CALL		CALL MYPG	Here you cause your object module MYPG to be loaded and executed.	5-7,9,10, 11,13,15
	GO		GO	Your program was interrupted. The GO command causes execution to begin at the point of interruption.	8,9
	BRANCH		BRANCH NEW	Your program was interrupted. You decide to branch to another entry point (NEW) in your current program.	9

Table 4. Commands and Macro Instructions *(Continued)*

FUNCTION	COMMAND	MACRO INSTRUCTION	SAMPLE USAGES	EFFECT	ILLUSTRATIVE EXAMPLES
Obtaining Information About Data Sets	DDNAME?		DDNAME?JOBLIB=N	Causes printing of all DDNAMES and DSNAMES currently defined.	7
	DSS?		DSS?	Causes printing of detailed information about each of your cataloged data sets.	21
			DSS? PHI,SIGMA	Here you ask for catalog information about PHI and SIGMA.	
	PC?		PC?	Prints abbreviated descriptions (name, access and, if shared, owner's identification) of all your cataloged data sets.	21
			PC? IOTA.KAPPA	Presents information on all data sets with the qualification IOTA.KAPPA.	
	POD?		POD?JOBLIBA,ALIAS=Y	Requests information about a specified partitioned data set. Here you ask for the names and aliases for each member of the partitioned data set JOBLIBA.	21
	LINE?		LINE? SOURCE.MAIN	Causes printing of records from the specified line data set. Here you print the contents of the line data set SOURCE.MAIN at the terminal.	8,9,16
			LINE? MU,800, (1200,1900)	Here you print line 800 and lines 1200-1900 of the line data set MU.	
	EXHIBIT		EXHIBIT UID	Causes a display of all active tasks on the system.	7
Text Editor Facilities	EDIT		EDIT DSONE	The Text Editor is invoked. DSONE is the data set name associated with the data set to be edited.	26,27
	END		END	This denotes the completion of editing initialized by an EDIT command.	26,27
	REGION		REGION XYZ	This defines region XYZ in the current data set.	26,27
	DISABLE		DISABLE	Revisions made to a data set after DISABLE are collected in a temporary area.	27
	ENABLE		ENABLE	Revisions made since the last DISABLE are permanent.	27
	CONTEXT		CONTEXT,,ABCDEF, UVWXYZ	The entire data set (current region) is searched, replacing ABCDEF with UVWXYZ.	27
	CORRECT		CORRECT 400 s t e ms 3 7 70 @* $ *%* y s@	Characters from line 400 of the current region are adjusted as requested. SYSTEM 370 is the resultant line.	27
	UPDATE		UPDATE	Lines entered following UPDATE are inserted into the current region according to the given line number.	26
	EXCERPT		EXCERPT ABC, ABC2	Region ABC2 from data set ABC will be inserted into the current data set.	27
	EXCISE		EXCISE-1	The line preceding the current line pointer location will be deleted, from the current region.	26
	INSERT		INSERT +10	Insertion in the current data set will begin 10 lines beyond the current position, with a default increment of 100.	26
	NUMBER		NUMBER 100,500, 1000,100	The range of lines (100-500) in the current region will be renumbered, beginning with 1000 (also in increments of 100).	27
	LIST		LIST 400,700	Lines 400-700 inclusive of the current region are displayed.	27
	LOCATE		LOCATE,,'ABC'	The entire current region will be searched for the character string ABC.	27

Table 4. Commands and Macro Instructions *(Continued)*

FUNCTION	COMMAND	MACRO INSTRUCTION	SAMPLE USAGES	EFFECT	ILLUSTRATIVE EXAMPLES
Text Editor Facilities	POST		POST	Makes all previous editing changes permanent.	27
Program Control	AT		AT MAIN.AB100	Causes notification to be printed on SYSOUT when program execution reaches specified location. Here you request the message when the program MAIN reaches the instruction named AB100.	9
			AT MAIN.AB100;IF MAIN.AB>MAIN.AC; STOP	Here you cause execution to stop at the instruction named AB100 if AB is greater than AC.	
	DISPLAY		DISPLAY MAIN.AB	Prints on SYSOUT the current contents of the data field AB in program MAIN.	8,9
	DUMP		DUMP MAIN.AB	Causes the name and contents of the data field AB in program MAIN to be placed in the PCSOUT data set for later printing.	9
	IF		IF M.AC > M.AD SET M.AB = 2	Defines a logical condition (true or false) that must be true to cause execution of the remainder of the IF statement. Here you cause the data field AB in program M to be set to 2 if the current value of AC is greater than AD.	9
	QUALIFY		QUALIFY MAIN	Causes subsequent names to be qualified by MAIN. You can then write .AB instead of MAIN.AB.	9
	REMOVE		REMOVE ALL	Deletes all previously issued AT commands or PCS statements that include AT commands.	
			REMOVE 10	Here you specify deletion of the statement to which the system assigned the number 10 when it was first entered.	9
	SET		SET .AB = .AC	Causes specified data field in virtual storage to be changed. Here you set the qualified field AB equal to AC.	8,9
	STOP		STOP	Stops program execution and causes printing on SYSOUT of current instruction location and program status information.	8
User Profile Management	DEFAULT		DEFAULT ACC=R	Access code for the CATALOG command will now be defaulted to read-only.	30
	SYNONYM		SYNONYM SPECIAL= ZLOGON	The ZLOGON procedure may now be invoked with either name, ZLOGON or SPECIAL.	30
	PROFILE		PROFILE	The session profile replaces the user profile in USERLIB.	30
	MCAST	MCAST	MCAST CONT=@, CP=///:	The continuation control character is changed to an @ character; the default prompt string is replaced with a series of three slashes and no carriage return.	28
	MCASTAB		MCASTAB INTRAN=Y	Allows you to replace the system's input character translation and switch table with one which you have written for your task.	

Table 4. Commands and Macro Instructions *(Continued)*

FUNCTION	COMMAND	MACHRO INSTRUC-TION	SAMPLE USAGES	EFFECT	ILLUSTRA-TIVE EXAMPLES
Command Creation	BUILTIN		BUILTIN NAME=TEST2	Defines TEST2 as the name of the object program which the user can invoke as if it were a command.	29
	KEYWORD		KEYWORD	Causes printing of all the command names, and their associated parameters, currently defined in your USERLIB.	29
			KEYWORD COMNAME=GO	Here you ask for the parameter keywords of the command GO to be printed.	
	PROCDEF		PROCDEF NAME=TEST3	Defines TEST3 as the name of a user-written command procedure.	28

Macro Instructions Used in Examples

This table (Table 5) lists the macro instructions whose use is shown in Part III, grouped by function. Detailed coding information on these macro instructions is provided in the publication *Assembler User Macro Instructions.*

Table 5. Macro Instructions Used in Examples

FUNCTION	MACRO INSTRUCTION	USE	ILLUSTRATIVE EXAMPLES
VSAM Data Management	DCB	Reserves space for a data control block (DCB) which describes a virtual sequential (VS) data set.	5,6,11,14
	OPEN	Initializes a specified DCB for processing and catalogs new data sets.	5,6,11,14
	GET	Moves a virtual sequential data set record to a virtual storage location.	6
	PUT	Includes a record from virtual storage in a virtual sequential data set.	5,6,11,14
	CLOSE	Logically disconnects a specified data set from your program.	5,6,11,14
	DUPOPEN	Initializes two specified DCBs for processing in duplex mode and causes creation of a catalog entry if data set is new.	
	DUP-CLOSE	Logically disconnects a duplexed data set from your program.	
VISAM Data Management	DCB	Reserves space for a data control block (DCB) which describes a virtual index sequential (VIS) data set.	14
	OPEN	Initializes a specified DCB for processing and catalogs new data sets.	14
	PUT	Includes the next sequential record from virtual storage in a virtual index sequential data set.	14
	READ	Transfers a virtual index sequential record to a virtual storage location.	14
	DELREC	Deletes a specified record from a virtual index sequential data set.	14
	CLOSE	Logically disconnects a specified data set from your program.	14
	DUPOPEN	Initializes two specified DCBs for processing in duplex mode and causes creation of a catalog entry if data set is new.	
	DUP-CLOSE	Logically disconnects a duplexed data set from your program.	
BSAM Data Management	DCB	Reserves space for a data control block (DCB) which describes a physical sequential data set.	10
	OPEN	Initializes a specified DCB for processing.	10
	READ	Transfers a block of data from a physical sequential data set to a virtual storage location.	10
	WRITE	Writes a block of data from virtual storage in a physical sequential data set.	10
	CHECK	Required when processing physical sequential data sets to check the I/O operation requested by a READ or WRITE macro instruction for I/O completion, errors, or exceptional conditions.	10
	CLOSE	Logically disconnects a specified data set from your program.	10
SYSIN-SYSOUT I/O	GATRD	Reads a record from SYSIN and places it in a specified area in your virtual storage.	2-4,11,12, 16, 19
	GATWR	Transfers a message from your program to SYSOUT.	2-4, 6, 7, 12, 13, 16, 19, 23
Program Linkage	SAVE	Stores the contents of specified registers in a save area.	6,7,10,11, 14, 20
	RETURN	Restores the contents of specified registers from a save area, and returns control to the location specified in register 14.	6, 7, 20, 23
	EXIT	Terminates program execution and causes the next command from SYSIN to be processed.	2-5,10,12, 14, 16, 17, 19, 20
Interrupt Management	SPEC	Specifies the entry point of an interrupt-handling routine to which control is to pass with the occurrence of a specified type of program interrupt.	20
	SIR	Informs the system of the presence of your interrupt-handling routine.	20
	DIR	Terminates possibility of your interrupt-handling routine receiving control.	20
User-Defined Procedures	BPKD	Supplies the linkage between the assembler object module and BUILTIN procedure name.	29

Part III. Examples

Part III is devoted to thirty examples showing user-system interaction. The dialog between you and the system appears (along with explanatory comments) as it would at the terminal. They are typical examples of system use. Use the examples as a learning device, and as models for designing your own work.

Commands and concepts are presented in an ordered sequence: the most necessary and basic ones appear first, and are reviewed in subsequent examples. The examples are designed so that the beginner should read them in sequence. Precise system responses are not indicated. Instead, short narratives describing system reactions to your input are given throughout. The expanded facilities of the Command System such as the Text Editor, PROCDEF, BUILTIN, and User Profile are depicted for your guidance. Those familiar with the commands and concepts can use the examples for reference.

All VSAM, VISAM, and VPAM data sets are automatically cataloged at the time they are created. DDEF time includes the specific issuance of the command (or macro) by the user, as well as the implied system issuance of DDEF by such facilities as the DATA command and the ASM command. The system creates the initial catalog entry and provides the user with unlimited access. You must deliberately update the initial catalog entry if this access is not desired. The examples stress this concept in all of its forms, using a narrative wherever the system will take such automatic action.

Assembler programs are shown where they are necessary to clarify use of the commands. Only the relevant statements are included, and usually do not show base register usage. Full program linkage and reenterable programs are shown in the later examples.

Various types of messages are issued to your terminal by the system. The three types are as follows:

Prompting Messages—Request that you supply command operands or other information. Since the system does not recognize confirmation mode as such, you will be prompted only for omitted operands that have no default option specified.

Information Messages—Either inform you of actions the system has taken in executing a command, or request additional information.

Diagnostic Messages—Inform you of errors and prompt you for corrections.

In these examples, lines typed by the system are headed SYS, lines you enter are headed YOU. Lines in which both the system and you enter something are headed S, Y. Lines printed by your program are headed PGM, and cards entered from the terminal card reader and reprinted are headed CIP, for card image printout.

The use of the PRINT command for obtaining language processor output listings is illustrated in the examples as follows:

PRINT LIST.module-name, , ,EDIT

You will automatically be given the latest generation of your list data set.

Some examples use the ERASE option so that unwanted data sets may be removed from the system. This procedure is recommended whenever practical so that public storage will not be burdened with unwanted user data. The format is as follows:

PRINT LIST.module-name, , ,EDIT,ERASE

Example 1: Initiating and Terminating a Conversational Task

In this example, you initiate a simple conversational task and then terminate it. The commentary explains the keyboard entries required to converse with the system.

To begin a conversational task, first make sure that the terminal is properly prepared (refer to instructions provided by your installation or to the *Terminal User's Guide*). When you dial up the system or press the attention button for the first time in your task, the system assumes a log-on operation and the keyboard is unlocked for you to enter the LOGON command along with the appropriate LOGON parameters. Since the system will not prompt for individual LOGON operands, all of them must be entered at the same time. You cannot begin a task until you have logged on properly.

During your dialog with the system, your commands are not entered into the system until you press the return key.

```
YOU: (press attention button or dial up the system)
```

From this point on, pressing the attention button halts current activity in most situations. Consult Appendix D for the specific action taken in each situation. The keyboard is unlocked to receive your LOGON command and operands.

```
     LOGON ADUSERID,MYPASS*,24,ADACCT30,N,5,P
                                     29,N,5,P
```

Before pressing the return key, you notice a typing error in the charge number (ADACCT30). To correct this error you backspace 8 characters, move the paper up one line to avoid overtyping, and then enter the proper characters.

```
SYS: TSS/370 RELEASE 2.0
```

This is the first message you will receive, indicating the system and level you have dialed, and informing you that your LOGON attempt has been recognized by the system. You must still wait for LOGON acknowledgement before you can begin your task.

```
SYS: TASKID=1111 LOGON AT 15:21 ON 04/12/76
```

With this message the system acknowledges your LOGON; you may now begin processing.

Explanation of LOGON Operands

ADUSERID

User Identification

User identity is the first of the LOGON parameters. You enter your full identification. It was assigned to you when you were joined to the system. Its first two characters identify the administrator who authorized your access to the system.

MYPASS*

Password

This code word (password) provides protection against unauthorized use of your user identification. Conversationally it must be used if one was assigned at JOIN time. Nonconversationally it is optional. The system will prompt you and allow you to overprint it conversationally to ensure password security.

24

Addressing

24 or 32 bit addressing may be specified. The present system addressing will be assumed. (32 bit addressing is valid for System 360 only.)

ADACCT29

Charge Number

This is the charge or account number that was assigned to you by your administrator.

N

Control Section Packing

This operand specifies the type of control section packing provided for you by the Dynamic Loader. Possible values are as follows:

- A = pack all control sections
- P = pack all prototype control sections only
- O = pack only those control sections having neither PUBLIC nor PROTOTYPE attributes
- X = pack all control sections, except prototype
- N = no control section packing will be done

5

Maximum Auxiliary Storage

You may specify the number of pages needed. If not, the system will assume the number of pages specified at SYSGEN.

P

Pristine

Permits you to log on with only system supplied defaults and, if you wish, without your USERLIB. Possible values are:

P=USERLIB is defined but session profile reflects only system defaults, etc.

X=USERLIB is not defined and session profile reflects only system defaults, etc.

SYS:

ZLOGON will now be invoked, if it has been defined by you. You can continue with other commands or procedures as soon as the system underscore appears.

S,Y: KB

After logging you on, the system prints a single underscore and then backspaces; this is the standard signal that it is ready to receive your next command on the same line. Here you specify that you want folded mode; that is that certain lower case characters (as a-z and ! "¢) be translated by the system into their upper case equivalents (A-Z and $ # @, respectively). Thus, with KB, you no longer need to perform many shifting operations.

When you initiate a conversational task, the system automatically assumes folded mode; hence in this example you need not have specificed KB. However, there are other character control commands, such as KA, which invoke EBCDIC mode at the keyboard. Thus, if you specify KA and at a later time in your session wish to return to folded mode, you must enter KB.

YOU: LOGoff
SYS:

You decide to conclude your session by logging off. The system will respond with an acceptance message, indicating the date and time your task was terminated. Note that LOGoff translates to LOGOFF.

Example 2: Assembling and Correcting from the Terminal

In this example, you type in the source statements of a short program and correct several errors while assembling the program. The assembled object module is stored in your USERLIB. The listings you selected are printed as a separate task, only if requested using the PRINT command.

```
YOU: (press attention button or dial up the system)
     LOGON ADUSERID,MYPASS*,,ADACCT29
```

You enter your identification, password, and account number. System defaults are taken for the remaining operands.

```
SYS:
```

The system will complete the LOGON procedure and invite you to enter the next command with an underscore.

```
S,Y: CHGPASS
```

At this time you decide to change your password. You thus enter the CHGPASS command.

```
SYS: ENTER CURRENT PASSWORD
```

The system will prompt you for your current password with an overprinted line. Because you have entered the correct password, the system will prompt you to enter your new password.

```
SYS: ENTER NEW PASSWORD
```

The system validates your new password and invites you to enter your next command.

```
S,Y: TIME MINS=15
```

The TIME command establishes a period of time a task will be allowed to run in virtual memory. Since your task requirements will be small, you decide to override the SYSGEN value established for "task time," and set 15 minutes as the upper limit. You will be alerted when this new interval is exhausted. TIME is useful to monitor tasks which may inadvertently loop, or otherwise take abnormal actions.

```
S,Y: ASM NAME=ATIMES2,STORED=N,ASMLIST=Y,CRLIST=Y,PMDLIST=Y,LINCR=(1300,100)
```

The system acknowledges receipt of the ASM command. Language processing commences.

Explanation of ASM Operands

NAME=ATIMES2

Object Module Name

You assign the name ATIMES2 to the object module. The object module created by the assembler is placed in the library at the top of your program library list—in this case, your USERLIB. This parameter cannot be defaulted.

STORED=N

Prestored Source Data Set

The N indicates that you are going to enter source statements rather than assemble from a prestored source data set. The system creates a data set from your source statements and automatically creates a name for it by prefixing "SOURCE." to the name you supplied as the first assembly parameter (SOURCE.ATIMES2)

MACROLIB=

Macro Library

This parameter permits you to indicate additional macro libraries on which you have stored macro definitions you created. Your default by omission of this parameter means that macro definitions are to be obtained only from the system macro library during assembly.

VERID=

Version Identification

You may assign an identification to the assembled object module to distinguish it from other assemblies of the same module name. It will appear on the PMD output listing. Your default (by omission) of this parameter will yield the current date and time.

ISD=

Internal Symbol Dictionary

For each source program symbol, the Internal Symbol Dictionary (ISD) shows its type, length, and the relative internal locations assigned by the assembler. This information is necessary for full utilization of the Program Control System (PCS) debugging capabilities.

By default (omission) an ISD is produced.

SYMLIST=

Source Listing

The listings you request with this and the next five parameters will form your listing data set. (See Appendix A for a detailed explanation of these listings.) The system creates a name for the listing data set by prefixing "LIST." to the name you supplied as the first assembly parameter (LIST.ATIMES2), using generation data group logic. The source listing will reveal the source input statements.

By default, you will not receive a source statement listing.

ASMLIST=Y

Object Listing

The Y indicates you wish to receive the object listing. This listing shows the assembled object code and the assembler-assigned displacement addresses, both in hexadecimal form. A separate source listing (SYMLIST) should be requested if continuation of source input statements is very frequent because statements listed in the object listing are the concatenated forms of continued source statements. In other words, the SYMLIST shows how the source statements were received, whereas the ASMLIST shows the form used in the assembly of the object module.

CRLIST=Y

Cross-Reference Listing

You request the cross-reference listing, which indicates the type, length, and assembled hexadecimal location for each symbol. It also indicates the hexadecimal location of each symbol reference.

STEDIT=

Edited Symbol Table

The edited symbol table is merely the cross-reference listing without the reference locations.

By default, it will not be produced.

ISDLIST=

Internal Symbol Dictionary Listing

To obtain the listing of the Internal Symbol Dictionary (ISD) you must also have requested an ISD above. You omit this parameter; it therefore will not be produced, by default.

PMDLIST=Y

Program Module Dictionary Listing

You specify a listing that shows the entries in the Program Module Dictionary (PMD). It is helpful in determining the structure of your object module and its relocation properties.

LISTDS=

List Data Set

In nonconversational mode you must specify if you want your listings placed in a list data set. By default they will be placed on SYSOUT, and no record of them retained in the system after printout. Conversationally, a list data set is automatically created unless you specify otherwise. In the latter case they will be printed out at your terminal.

```
LINCR=(1300,100)
```

Starting Line Number, Increment

The system creates a number for each line of input as the source data is formed. You specify that the first line is to be numbered 1300 and that additional line numbers are to be incremented by 100. Line numbers and increments can consist of three to seven digits, the last two of which must be zero when the data set is initially formed. Since the line numbers for a source data set are supplied at the time it is created, this parameter has no meaning when assembling from a prestored source data set.

```
SYS:
```

The user is invited to enter source statements.

```
S,Y: 0001300CST CSECT
```

You enter your source statements in free form, i.e., by separating the fields with a single space. *Notice that you must leave a space for a null name (label) field.*

This basic program reads an integer from the terminal, multiplies it by two, and writes the product at the terminal. It is limited to integers between 0 and 4.

```
S,Y: 0001400BEGIN BASR 11,0
S,Y: 0001500 USING *,11 LOCAL BASE REG
S,Y: 0001600 GATRD AREA+3,LENGTH READ FROM SYSIN
S,Y: 0001700 MVZ AREA+3(1)=X'00'CONVERT TO BINARY
SYS: 0001700 E***OPERAND FIELD IMPROPERLY DELIMITED
SYS: 0001700 MVZ AREA+3(1)=X'00'CONVERT TO BINARY
```

The assembler examines each statement for syntactical errors. It discovers that you have omitted a comma and informs you with an error message, then prints out the line requiring correction. See Appendix A for a further explanation of assembler diagnostics.

```
S,Y: #
     1700, MVZ AREA+3(1),=X'00' CONVERT TO BINARY
```

The system then prints the number sign (#) after which you enter the number of the erroneous line, a comma, and then the content of the corrected line. Then you press the return key.

At this point you can make any number of modifications, deletions, or insertions of new lines.

```
S,Y: #(press return key)
```

Instead you indicate the end of modifications by pressing the return key.

```
S,Y: 0001800 L 5,AREA1
```

The system then prompts you with the line number for the next statement, which you enter.

```
S,Y: 0001900 SLA 5,1 MULT BY 2
S,Y: 0002000* RESTRICTED TO INTEGERS FROM 0 to 4
S,Y: 0002100 ST 5,AREA
S,Y: 0002200 MVZ AREA+3(1),=X'FF' CONVERT TO EBCDIC
S,Y, 0002300 GATWR AREA+3,LENGTH WRITE ON SYSOUT
S,Y: 0002400 EXIT 'PGM FINISHED'
S,Y: 0002500AREA DC F'0' RD/WR AREA
S,Y: 0002600LENGTH DC F'1' LENGTH OF AREA
S,Y: 0002700 END
```

The END statement identifies the last source statement.

```
SYS: 0002700 E *** 'AREA1   ' UNDEFINED SYMBOL
```

After the last source statement has been entered, the assembler expands your macro instructions and searches for global errors. Here it has discovered one. The system invites you to correct your source program, if you so desire. It does *not* perform a syntactical check on the statements you modify or enter at this point. The line in which the error was located is first printed out to help you in making corrections.

```
YOU: Y
S,Y: #
     1800, L 5,AREA
```

You indicate that you wish to modify your source (response of Y), and then enter the correct statement.

```
S,Y: #(press return key to indicate end of modifications)
```

The system now reassembles your modified program, rescanning all the statements as if this were a new assembly. The assembler indicates it has found no errors.

```
YOU:
```

Your listings were automatically placed in a list data set since you are operating in conversational mode. The printing of a list data set is not an automatic function. You must therefore issue the PRINT command.

```
S,Y: PRINT LIST.ATIMES2,,,EDIT
```

The system will establish a nonconversational task to print the current generation of LIST.ATIMES2.

```
SYS:
```

The system assigns a batch sequence number (BSN) for your listing data set. It contains the listing you specified earlier (i.e., object, cross-reference, and PMD).

```
S,Y: ERASE SOURCE.ATIMES2
SYS:
```

The system confirms the erasure. Since all VAM data sets are automatically cataloged when created (in this case as a result of the ASM command), you are urged to erase data sets for which you have no further use. This allows public storage to be freed for other purposes.

The list data set is automatically cataloged by the system as the current generation of LIST.ATIMES2. The assembler has stored your object module in the library at the top of your program library list (in this case, your USERLIB, which is an automatically cataloged data set).

```
S,Y: LOGOFF
SYS:
```

The LOGOFF is accepted by the system.

Example 3: Assembling and Executing

In this example, you enter and assemble the same program you assembled in Example 2, but give it a different name. You cause the resulting object module to be stored in a temporary library. After executing the assembled program, you save the source program for use in a future session.

The terminal is used as SYSIN for your program input as well as SYSOUT for your program output.

```
YOU: (press attention button or dial up the system)
     LOGON ADUSERID,,,ADACCT29
```

You log on, defaulting your password. If you were joined with a password, the system will prompt for it with an overprinted line.

```
SYS: ENTER PASSWD
     XXXXXXXX
```

After prompting, the carriage is positioned at the first overprinted position, allowing you to overprint your password. This facility is used to ensure password security.

```
SYS:
```

The system will complete the LOGON procedure and invite you to begin your task.

```
S,Y: DDEF DDNAME=TEMPDD,DSORG=VP,DSNAME=SCRATCH,OPTION=JOBLIB
```

The DDEF command is used to describe a data set to the system. It defines a data set only during the session in which the command appears. Every data set you use must be defined for the current session, even if it has been previously cataloged. Some data sets, such as listing and source data sets, are automatically defined by the system and, thus, do not require an explicit DDEF on SYSIN.

Here you define a JOBLIB data set. All libraries require virtual partitioned (VP) organization. Since SCRATCH is the most recently defined library, the system places it at the top of your program library list. Object modules created by the assembler, therefore, will be stored in it.

```
S,Y: KA
```

With KA, you indicate you wish to use the full EBCDIC character set during input. Both upper and lower case letters will be translated as their respective equivalents.

```
S,Y: DEFAULT ASMALIGN=N
```

You do *not* want the source code in your program listing aligned in columns 1, 10, and 16. Instead you want the source code to appear exactly as you entered it.

```
S,Y: ASM AT2,N,PMDLIST=Y,LINCR=(300,100)
```

A combination of positional and keyword parameter notation is illustrated here. The format of the module name must be as indicated (see AT2). Since the name used becomes a member of a virtual partitioned data set when the object module is created, partially-qualified names and generation data group names cannot be used. Virtual partitioned data set members must be identified with simple names.

In this and in following examples you press the tab key to separate source statement fields. You have set terminal tabs at columns 17, 23, and 40, so that the typewriter listing of your input (following the seven-digit line number supplied by the system) conforms to standard coding-sheet format. When setting your terminal tab stops during your task, you will create several spurious tab characters which you want to prevent the system from interpreting. Therefore, after setting your tab stops, erase the unwanted line by backspacing and then immediately pressing the return key.

The system now prompts you by printing the number it has assigned to the first source line.

```
S,Y: 0000300CST      CSECT
S,Y: 0000400BEGIN    BASR  11,0
S,Y: 0000500         USING *,11              LOCAL BASE REG
S,Y: 0000600         GATRD AREA+3,LENGTH     READ FROM SYSIN
S,Y: 0000700         MVZ   AREA+3(1),=X'00'  CONVERT TO BINARY
S,Y: 0000800         L     5,AREA
S,Y: 0000900         SLA   5,1               MULT BY 2
S,Y: 0001000*   RESTRICTED TO INTEGERS FROM 0 TO 4
S,Y: 0001100         ST    5,AREA
S,Y: 0001200         MVZ   AREA+3(1),=X'FF'  CONVERT TO EBCDIC
S,Y: 0001300         GATWR AREA+3,LENGTH     WRITE ON SYSOUT
```

```
S,Y: 0001400          EXIT  'pgm finished'
S,Y: 0001500AREA      DC    F'0'                RD/WR AREA
S,Y: 0001600LENGTH    DC    F'1'                LENGTH OF AREA
S,Y: 0001700          END
```

SYS:

Your source input is scanned and you are told that no errors were found. The assembler completes the assembly process and your next command is solicited.

S,Y: PRINT LIST.AT2,,,EDIT

System will establish nonconversational task to print the current generation of LIST.AT2. Your program has been assembled without error. The listings you have requested (a PMD listing and, through default, an object listing) form your listing data set. It will be printed as a separate task.

S,Y: AT2

This command causes your object module to be loaded from your SCRATCH job library and executed. Execution begins at the first location in the CSECT.

SYS: (unlocks keyboard)

When the GATRD macro instruction in your program is executed, the system unlocks the keyboard.

YOU: 3

Then you enter your input data from the terminal, and press the return key.

PGM: 6

GATWR prints program output on SYSOUT (the terminal in a conversational task).

PGM: EXIT, RELEASE ALL UNNEEDED DEVICES.

The EXIT macro instruction causes this message to be printed. If you had private data sets you no longer needed, you would issue a RELEASE command to free the devices on which their volumes were mounted.

PGM: pmg finished

The EXIT macro instruction then prints the message you specified and returns control to the terminal, which is indicated by the underscore. You decide to log off.

S,Y: LOGOFF
SYS:

The LOGOFF is accepted by the system. Since all VAM data sets are automatically cataloged, SOURCE.AT2 remains cataloged for future use. You must specifically issue ERASE for data sets you no longer desire, *prior* to entering the LOGOFF command.

Example 4: Correcting and Reassembling a Prestored Source Program

In this example, you modify the source data set you cataloged in Example 3 so that, when it is assembled, the program will accept input more than once. Then you execute the program and enter the data several times.

Having completed the LOGON procedure, you enter your first command.

```
S,Y: MODIFY SETNAME=SOURCE.AT2
```

You want to modify the source program you created in the previous example. If not already defined in this current session, the data set named in a MODIFY command must be cataloged.

```
SYS:
```

The system acknowledges the MODIFY command and invites your input using a # sign.

```
S,Y: #
     R,500,600
```

The system prompts you with the number sign (#). Before you make any modifications to your source data set, you review several of its statements. You enter the R (for review), a comma, and the numbers of the source lines you want to review; then you press the return key.

```
SYS: 0000500          USING *,11              LOCAL BASE REG
SYS: 0000600          GATRD AREA+3,LENGTH     READ FROM SYSIN
```

Since you entered your source data set in tab format, statement fields are separated by a tab character. The number of spaces in the tab is not recorded. When printed on your terminal, fields appear wherever the terminal tabs are now set.

```
S,Y: #
     550,HERE          EQU   *     CHECK IF END BRANCH IF YES
```

After printing the two lines, the system again prompts you with the number sign. You insert a new statement following the number you assign to it. You can replace an existing statement with this same procedure.

```
S,Y: #
     630,             CLI     AREA+3,C'E'
S,Y: #
     670,             BE      LEAVE
```

You insert two more source statements. They will check for your selected value of E, indicating end of data.

```
S,Y: #
     R,1300,1400
```

You review another part of your program.

```
SYS: 0001300          GATWR AREA+3,LENGTH WRITE ON SYSOUT
     0001400          EXIT  'PGM FINISHED'
```

The EXIT message is presented by the system in upper case this time, since KB is the mode by default.

```
S,Y: #
     1350,            B       THERE
```

Here you insert a statement. Notice that, within the MODIFY command, no checks are made for line errors. The incorrect THERE will be discovered later, during assembly.

```
S,Y: #
     1400,LEAVE      EXIT   'PGM FINISHED'
```

You add a name field to the EXIT statement.

To remind you that you changed the source data set, you decide to rename the current source data set. Before this can be done, the csect name must be changed, even if the new module will be going into a separate library.

```
S,Y: #
     R,300
```

You review the line containing the csect name.

```
SYS: 0000300CST CSECT
S,Y: #
     300,SECT CSECT
```

You change the name of the csect.

```
S,Y: #
     %E
```

You signal the end of modifications to terminate the MODIFY command.

```
S,Y: CATALOG DSNAME=SOURCE.AT2,STATE=U,NEWNAME=SOURCE,AT2EX4
```

You use the CATALOG command to rename the current source data set SOURCE. AT2EX4. At assembly time, the associated list data set will be named LIST.AT2EX4 by the system. The U indicates the updating of an existing catalog entry. This command corresponds to the CAT macro instruction.

```
S,Y: DDEF TEMPDD,VP,SCRATCH,OPTION=JOBLIB
```

This command defines your job library established in an earlier session. The system places it at the top of your program library list. Disposition is defaulted by the system to OLD, since SCRATCH already exists in your catalog.

```
S,Y: ASM AT2EX4,Y
```

The Y specifies that the source data set is prestored.

```
SYS: 0001700 E *** 'THERE  'UNDEFINED SYMBOL
```

This is the error that was not detected during modification. The line in which the error was located is printed out to help you in making corrections. The number of the END statement is given when undefined symbols are encountered.

```
SYS:
```

Your source input is scanned and you are asked if you wish to enter any modifications.

```
YOU: Y
```

You indicate that you wish to modify your source data set (response of Y).

```
S,Y: #
     1350,           B       HERE
S,Y: #
     (press return key to signal end of modifications)
```

The assembler rescans the source input, and finds no further errors.

```
SYS:
```

The assembly process is completed, without errors. The assembled object module now resides in your job library (SCRATCH), the library at the top of your program library list.

This time you do not issue a PRINT command but the data set still exists as the current generation of LIST.AT2EX4. If the listing is later desired, you need only to issue: PRINT LIST.AT2EX4,,,EDIT,Y.

```
S,Y: AT2EX4
```

This command causes the object module which includes the external symbol BEGIN to be loaded from SCRATCH and executed.

```
SYS: (unlocks keyboard)
YOU: 4
```

You enter a number at the keyboard (SYSIN to be read by GATRD).

```
PGM: 8
```

GATWR causes the computed results to be printed at the terminal (SYSOUT).

```
SYS: (unlocks keyboard)
YOU: 1
PGM: 2
SYS: (unlocks keyboard)
YOU: E
PGM: EXIT, RELEASE ALL UNNEEDED DEVICES.

PGM: PGM FINISHED
```

Your program detects the entry in SYSIN of E and branches to the EXIT macro instruction, which prints its messages and returns control to the keyboard. The system prompts for the next command with an underscore.

```
S,Y: _ERASE USERLIB(ATIMES2)
```

Now that you know that your modified program runs correctly, you decide to erase from USERLIB the object module you assembled in Example 2. With this form of the ERASE command, only the module named within the parentheses is erased.

```
S,Y: _LOGOFF
SYS:
```

The LOGOFF is accepted by the system. The listing data set (LIST.AT2EX4) will automatically become the current generation of the pre-established generation data group. Your new object program module (AT2EX4) resides in SCRATCH, which was cataloged in a previous session. You cataloged your source data set (SOURCE.AT2EX4) when you changed its name.

Example 5: Writing a Data Set and Printing It

In this example, you execute a program that you have previously assembled and checked out. Its object module resides on your USERLIB. Your program causes a data set to be written. You request that it be printed later on the system high-speed printer as a separate task.

After the LOGON procedure is completed, you begin processing. The program which you are going to run, named PROC5, includes the following source statements:

```
CST5   CSECT
       ENTRY STRT5
STRT5  EQU   *
       .
       .
       .
       LA    2,20                                        SET FOR 20 CYCLES
       OPEN  (DCBNM,(OUTPUT))                            OPEN DCB
LABEL  EQU   *
       (create record at AREA)
       .
       .
       .
       PUT   DCBNM,AREA                                  PUT RECORD IN DATA SET
       BCT   2,LABEL                                     RECYCLE
       CLOSE (DCBNM)                                     CLOSE DCB
       EXIT
AREA   DS    80C                                         DATA AREA
DCBNM  DCB   DDNAME=OUTDD,RECFM=FA
       END
```

Your program will write a data set with 80-character records from the storage area named AREA. Notice that your DCB macro instruction includes the DDNAME that is a parameter in the DDEF command, which in turn contains the name of the data set (OUT5). The DDEF command relates the correct data set to your program because every data set name must be unique in your task.

S,Y: DDEF OUTDD,VS,OUT5,(LRECL=80)

With this command, you define for this session the data set which your program will write. Record length=80. The DISP field in the DDEF command is defaulted to existence (i.e., default is NEW if the data set is being created initially in the current task; default is OLD if data set already exists and is cataloged). Since this data is being created now, the default for disposition is NEW.

See Appendix E for further details of the DDEF parameters.

S,Y: CALL STRT5

This command causes the object module defining STRT5 to be loaded into virtual storage and executed.

PGM: EXIT, RELEASE ALL UNNEEDED DEVICES

Your program completes its work and this message is issued by the EXIT macro instruction. Control is returned to the terminal.

S,Y: RET OUT5,R

A public volume was selected for your data set because you defaulted the volume field in your DDEF command, and it was cataloged for you automatically, with access qualifier=U. You decide to protect the data set by updating the access qualifier to read-only, using the RET command. You can also use the RET command to change a temporary data set to a permanent one (or vice versa) by specifying P (or T) in the retention code (and you could have it erased automatically at CLOSE or LOGOFF by specifying C or L in the code, if it is a temporary data set).

S,Y: PRINT DSNAME=OUT5, PRTSP=EDIT

To print your newly-written data set, this command creates a separate (nonconversational) task. You could have used the PRINT macro instruction to create the task.

Explanation of PRINT Operands

DSNAME=OUT5

The data set to be printed with this command must either be defined within the current task by a DDEF command, as it is in this example, or it must be cataloged. Its records must be fixed or variable length, and must include a USASI control character (RECFM=FA or VA).

STARTNO=

You want printing to begin with the first byte of each data set record. You can enter a number consisting of one to six digits. You default this parameter by omission.

ENDNO=

This parameter specifies at which byte in each data set record printing is to end. Since your records are shorter than the default length, your printing will end at the last (80th) byte of each record. You default this parameter by omission.

PRTSP=EDIT

Since you want spacing to be controlled by the control character your program has supplied in each record, you choose EDIT. The default is 1. Since EDIT was selected, the values for header, lines per page, and page number will not prevail. However, if one of the other spacing options (1, 2, or 3) has been selected, these three values would be required.

ERASE=

This parameter is meaningful only if the data set being printed is cataloged. In that case, you can specify that the data set be erased after it is printed. By parameter omission, there will be no data set erasure.

ERROROPT=

This parameter applies only to data sets on tape. It specifies the action to be taken if an unrecoverable error is found while a data set record is being read. Since the data set is on a direct access device, the parameter is ignored by omission (default).

FORM=

Here you can specify the form number of the printer paper you desire for your output. The default (STANDARD FORM) is determined by your installation.

STATION=

This parameter applies only at installations where the user has been given the privilege of directing print jobs to an RJE station. It permits you to indicate the station at which you want your output printed.

SYS:

The system informs you that it has accepted the requested nonconversational task, and assigned it a batch sequence number (BSN).

S,Y: LOGOFF
SYS:

The LOGOFF is accepted by the system. Your conversational task is therefore terminated.

Example 6: Reading and Writing Cataloged Data Sets

In this example, you run a previously-assembled program that resides on your USERLIB. It causes records of a cataloged data set to be read. After performing several calculations, your program writes records in two new data sets. It then issues a message on the terminal indicating that the task was completed.

After the LOGON procedure is completed, you begin processing. The program you are going to run includes the following source statements:

```
PST6      PSECT
          ENTRY    STRT6
          ENTRY    EOD6
          DC       F'76'                     SAVE AREA LENGTH
          DC       18F'0'                    REMAINDER OF SAVE AREA
AREA1     DS       80C                       DATA
AREA6A    DS       80C                       AREAS
AREA6B    DS       80C
DCBIN6    DCB      DDNAME=INPDD,EODAD=EOD6
DCBOUT6A  DCB      DSORG=VS,DDNAME=OUT6A,RECFM=F,LRECL=80
DCBOUT6B  DCB      DSORG=VS,DDNAME=OUT6B,RECFM=F,LRECL=80
CST6      CSECT
STRT6     SAVE     (14,12)                   SAVE REGISTERS IN CALLER'S SAVE AREA
          L        14,72(0,13)               GET RCON FROM CALLER'S SAVE AREA
          ST       14,8(0,13)                STORE FORWARD POINTER
          ST       13,4(0,14)                STORE BACKWARD POINTER
          LR       13,14                     SET PSECT AND SAVE AREA REGISTER
          USING    PST6,13                   PSECT COVER REGISTER
          LR       12,15
          USING    STRT6,12                  CSECT COVER REGISTER
            .
            .
            .
          OPEN     (DCBIN6,,DCBOUT6A,(OUTPUT),DCBOUT6B,(OUTPUT))
ALPHA     GET      DCBIN6,AREA1              READ INPUT DATA
            .
          (calculations with results to AREA6A and AREA6B)
            .
            .
          PUT      DCBOUT6A,AREA6A           WRITE RECORD
          PUT      DCBOUT6B,AREA6B           WRITE RECORD
          B ALPHA                            LOOP
EOD6      CLOSE    (DCBIN6,,DCBOUT6A,,DCBOUT6B)
          GATWR    A,L                       MESSAGE ON SYSOUT
          L        13,4(0,13)                RELOAD CALLER'S SAVE AREA
          RETURN   (14,12)                   RETURN TO CALLER
*
A         DC       C'FINISHED WRITING TWO DATA SETS'   MESSAGE
L         DC       A(L'A)                              MESSAGE LENGTH
          END
```

This and the program in the previous example define the entry point by including an ENTRY statement for it in the PSECT. By this technique, the R-value for the entry point will be resolved as the location of the PSECT save area.

This program uses the standard SAVE-RETURN linkage. When you execute the CALL STRT6 command, control is passed to your program by the calling program (in this example, the system). The SAVE macro instruction and the four instructions that follow it effect the saving of registers and the establishment of proper linkages. At the conclusion of your program, the RETURN macro instruction restores the saved registers and returns control to the system (the caller), which unlocks the keyboard.

It is good coding practice to place all variable data in your PSECT: refer to Appendix C for a more complete explanation of linkage conventions and TSS programming practices.

If the end of input data is reached before 10 cycles, the system transfers control to the location you specified in the end of data (EODAD) field in your DCB macro instruction. Your program then executes its normal return.

```
S,Y: DDEF INPDD,VS,INP6,DISP=OLD,RET=TLU
```

You define the data set from which your program reads input. Although the data is cataloged, you must define it for this task. The system locates it from the information in the catalog.

Some of the parameters you omitted in both the DCB macro instruction and the DDEF commands, such as the data set organization, are provided from the catalog entry. Others, such as RECFM, LRECL, and BLKSIZE, are obtained from the data set label. Appendix E explains these alternate sources. OLD indicates that the data set already exists. The retention code, TLU, specified by the RET parameter will cause the data set to become temporary with erasure at LOGOFF and with read-write access.

```
S,Y: DDEF OUT6A,VS,OUT6A
S,Y: DDEF OUT6B,VS,OUT6B
```

Now you define the data sets your program is to write. You decide to make the DDEF and data set names identical in each DDEF command. This makes it easy to relate the name of the DDEF to the output data set it defines. You default the disposition field by omitting it. NEW (the default) indicates that your two output sets do not already exist. Since these data sets are VAM organization, cataloging is automatic.

```
S,Y: CALL STRT6
```

Your object module is loaded and executed. Its output goes to the two output data sets. There will be no messages at your terminal until your program executes the GATWR macro instruction in your exit routine.

```
PGM: FINISHED WRITING TWO DATA SETS
```

This is the message from your GATWR macro instruction.

Your program contains the standard SAVE/RETURN linkage, so control is returned to you at the terminal; this is indicated by the underscore.

```
S,Y: RET OUT6A,R
```

OUT6A was automatically cataloged when opened, with access=U and you desire to protect it from further modification by issuing a RET command to change the access to R (read only). INP6 will be automatically erased at LOGOFF since you specified it as temporary in your DDEF command, with deletion at LOGOFF.

```
S,Y: LOGOFF
SYS:
```

The LOGOFF is accepted by the system.

Example 7: Multiple Assemblies and Program Linkage

In this example, you assemble three programs that refer to one another and then place them on two different libraries. Two programs are assembled in express mode from prestored source data sets, and the third program is assembled from the terminal. A control section is rejected during loading of the programs. You correct the error causing the rejection and run your programs.

After the LOGON procedure is completed, you begin processing.

```
S,Y: DEFAULT LPCXPRSS=Y
```

You request that your assembly be done in express mode which will allow you to assemble a number of source programs consecutively without a possibly time-consuming return to the Command System after each assembly.

```
S,Y: ASM MAIN7,Y
```

The Y specifies the existence of a prestored source data set named SOURCE.MAIN7. All modules that you assemble in the express mode will be governed by the parameters you specify with your ASM command for the first source module.

```
SYS:
```

Your first source program is assembled. When assembly is completed your keyboard is unlocked and the language processor control will read the next word entered at SYSIN as the name of the next module to be assembled.

```
S,Y: SUB7A
```

SUB7A is your next module. The language processor control will ignore any parameters you specify at this point and default to the parameters you specified on your entry into express mode. SUB7A is therefore assumed to be a prestored source data set.

SUB7A includes the following statements:

```
*SUBPROGRAM 7A
PST1    PSECT
        ENTRY   EP1
        DC      F'76'  SAVE AREA LENGTH
        DC      18F'0' REMAINDER OF SAVE AREA
         .
         .
         .
CST7A   CSECT
EP1     SAVE    (14,12) SAVE REGISTERS IN CALLER'S SV AREA
        L       14,72(0,13)   GET RCON FROM CALLER'S SAVE AREA
        ST      14,8(0,13)    STORE FORWARD POINTER
        ST      13,4(0,14)    STORE BACKWARD POINTER
        LR      13,14         SET PSECT AND SAVE AREA REGISTER
        USING   PST1,13       PSECT COVER REGISTER
        LR      12,15
        USING   EP1,12        CSECT COVER REGISTER
         .
         .
         .
        L       13,4(0,13)    RELOAD SAVE AREA BASE REG
        RETURN  (14,12)       RESTORE REGISTERS
        END
```

This and your other subroutine are reenterable programs. You use a standard form for such programs that require a separate PSECT for your variables. See Appendix C for more details.

SYS:

Your second source program is assembled. When assembly is completed, your keyboard is unlocked for you to enter the next module name.

YOU: SUB7B

A prestored source data set was expected as a result of the STORED=Y parameter when you entered express mode. No such prestored module can be found.

SYS:

The language processor control causes an exit from express mode, issues a diagnostic message and returns to the Command System with an underscore. To return to non-express mode without causing a diagnostic message you should have typed an underscore when the keyboard was unlocked, or you could have interrupted the language processor by pressing the attention button, issued the command DEFAULT LPCXPRSS=, and then continued processing with the GO command.

You decide to have your two listing data sets printed now as background tasks before proceeding with the assembly of your third module.

S,Y: PRINT LIST.MAIN7,,,EDIT,Y
SYS:

MAIN7 was successfully assembled, and the printing of its listing data set is assigned as a separate task (BSN=0624). The MAIN7 object module is stored in the library at the top of your program library list (in this case, your USERLIB).

The format of the PRINT command as given above is recommended for use whenever practical. The EDIT option allows line spacing to be regulated by the print control character in the first byte of each record. The ERASE option (Y) affords timely public storage release so that unwanted data is not occupying space in the system.

Note: Example 5 indicates the individual operands of the PRINT command.

S,Y: PRINT LIST.SUB7A,,,EDIT
SYS:

The print task is accepted and assigned a BSN (0625). The object module for SUB7A is also stored in your USERLIB.

```
S,Y: DDEF DDNAME=LIBADD,DSNAME=LIBA,OPTION=JOBLIB,DISP=OLD
```

Now you define a cataloged job library (LIBA), which the system places at the top of your program library list. Object modules from any successive assembly are placed in it instead of in your USERLIB. The OLD indicates that LIBA already exists. The omitted data set organization (VP) is supplied from the catalog entry.

You are now ready to proceed with the assembly of your third module. Since you are no longer in express mode you must enter the ASM command with all appropriate parameters.

```
S,Y: ASM SUB7B,N
```

The N specifies a data set to be entered dynamically (i.e., it does not yet exist). It will automatically be named SOURCE.SUB7B. You enter your third source program from the terminal, using tab stops at columns 17, 23 and 25. By default, line numbering will start at 100, with increments of 100.

```
S,Y: 0000100* SUBPROGRAM7B
     0000200PST2      PSECT
     0000300          ENTRY BGN7
     0000400          DC    F'76'          SAVE AREA LENGTH
     0000500SAVE      DC    18F'0'         REMAINDER OF SAVE AREA
                      .
                      .
                      .
     0000900EP1       CSECT
     0001000BGN7      SAVE  (14,12)        SAVE REGISTERS IN CALLER'S SAVE AREA
     0001100          L     14,72(0,13)    GET RCON FROM CALLER'S SAVE AREA
     0001200          ST    14,8(0,13)     STORE FORWARD POINTER
     0001300          ST    13,4(0,14)     STORE BACKWARD POINTER
     0001400          LR    13,14          SET PSECT AND SAVE AREA REGISTER
     0001500          USING PST2,13        PSECT COVER REGISTER
     0001600          LR    12,15
     0001700          USING BGN7,12        CSECT COVER REGISTER
                      .
                      .
                      .
     0002200          B     SYMB
     0002300CST7B     CSECT
     0002400SYMB      EQU   *
                      .
                      .
                      .
     0003100          GATWR AR,LNG
     0003200          L     13,4(0,13)     RELOAD SAVE AREA BASE REG
     0003300          RETURN (14,12)       RESTORE REGISTERS
     0003400AR        DC    C'SUBPROGRAM7B FINISHED'
     0003500LNG       DC    A(L'AR)
     0003600          END
SYS:
```

The assembler scans the source input and finds no errors. The assembly process is completed and you are invited to enter your next command.

```
S,Y: PRINT LIST.SUB7B,,,EDIT
SYS:
```

The print task is accepted for non-conversational processing. If you attempt to assemble a program with a module, control section, or external entry point name that already exists in the current library, your assembly will be completed but the module will not be stowed. You will then be allowed to enter a new JOBLIB name, after which the stow will be performed. Assume BSN=0626.

Because SUB7A and SUB7B object modules reside on different libraries, you don't notice that a control section name (EP1) in SUB7B duplicates an entry point name in SUB7A.

```
S,Y: MAIN7
```

Here you run MAIN7. It contains V-type address constants that name external entry points in both SUB7A and SUB7B. During the loading of MAIN7, SUB7A and SUB7B are loaded.

```
SYS: *** CSECT(EP1     ) IN MODULE (SUB7B) IS REJECTED BECAUSE PREVIOUSLY NAMED AS
     ENTRY POINT IN MODULE (SUB7A)
```

Your duplicated name is discovered at load time. This message tells you that, although SUB7B was loaded, its duplicated control section was not. Execution of MAIN7 is cancelled.

You decide to change the control section's name so that it will be loaded when you again run MAIN7.

```
S,Y: CANCEL BSN=0626
```

First you cancel the separate task that may still be printing the listings from the erroneous SUB7B assembly.

```
S,Y: UNLOAD NAME=MAIN7
```

Then you unload the erroneous module from your virtual storage. To do so, you unload MAIN7, which had caused the loading of SUB7B.

```
YOU: DDNAME? JOBLIB=Y
```

Before you can reassemble, you must either cause a different library to be placed at the top of your program library list, or erase the erroneous object module. You choose to move one of your libraries. You request a review of your library chain.

```
SYS:     DDNAME   DSNAME
         LIBADD   LIBA
         LIBCDD   LIBC
         SYSULIB  USERLIB
         LIBEDD   LIBE
YOU: JOBLIBS DDNAME=SYSULIB
```

You request that your USERLIB be placed at the top of your program library chain.

```
YOU: DDNAME? Y
```

You request that your JOBLIB chain be displayed.

```
SYS:     DDNAME   DSNAME
         SYSULIB  USERLIB
         LIBADD   LIBA
         LIBCDD   LIBC
         LIBEDD   LIBE
```

The system redefines your USERLIB to place it at the top of your JOBLIB chain and displays the chain. Your reassembled program will be placed in your USERLIB, eliminating the problem of duplicate entry point names.

```
S,Y: MODIFY SETNAME=SOURCE.SUB7B
```

Using the MODIFY command, you change the duplicate control section name in your source statement.

```
SYS:
```

Modifications are now solicited using the # sign.

```
S,Y: #
     900,CST7B CSECT
     #
     %E
```

You correct the source statement and then indicate the completion of modification with %E. Since review was not requested, the original form of the corrected line will not be presented to you.

```
S,Y: ASM SUB7B,Y
```

Now you reassemble the corrected source data set, which already exists.

Note that you never need to issue a DDEF command for a source data set that has been typed in earlier during the current session.

```
SYS:
```

The assembler scans the source input (as updated) and finds no errors. The assembly process is completed and you are invited to enter your next command.

```
S,Y: PRINT LIST.SUB7B,,,EDIT
SYS:
```

The PRINT task is assigned a batch sequence number and accepted for non-conversational processing. Assume BSN=0627.

```
S,Y: CALL MAIN7
```

You run your main program again.

```
PGM: SUBPROGRAM7B FINISHED
```

This message is printed by the GATWR macro instruction in SUB7B, indicating its completion (and successful loading of your two subprograms). The RETURN macro instruction causes control to be returned to the caller (MAIN7). A RETURN macro instruction in MAIN7 is eventually executed, causing control to be returned to the terminal. For a more complete discussion of CALL/SAVE/RETURN linkage conventions, see Appendix C.

```
S,Y: ERASE DSNAME=SOURCE.MAIN7
S,Y: ERASE DSNAME=SOURCE.SUB7A
```

You no longer need your two prestored source data sets, so you erase them from storage and delete their catalog entries. You decide to retain the source data set named SOURCE.SUB7B.

```
S,Y: EXHIBIT BWQ
```

Before logging off, you decide to check on the status of your PRINT requests. This command causes a display of all Batch Work Queue entries assigned to your userid.

```
SYS:
                              BATCH WORK QUEUE STATUS AT 14:55       9/15/70
     BSN  USERID   TID TYPE STAT DEV STAID       DSNAME
     0625 ADUSERID 18  LIST A    UR              SUB7A
     0627 ADUSERID 18  LIST P    UR              SUB7B
```

This display indicates that 0625 is active (A) and that 0627 is awaiting execution and pending (P). The absence of 0624 from the BWQ tells you that the job has been completed; 0626 does not appear since you cancelled it earlier.

```
S,Y: LOGOFF
SYS:
```

The LOGOFF is accepted by the system.

Example 8: Use of PCS Immediate Statements

In this example, you are executing a program for the first time. Since the Program Control System (PCS) provides complete debugging capability at execution time, you have not included any debugging facilities in your assembled program. Anticipating the use of PCS, you requested an ISD when the source program was assembled.

After the LOGON procedure is completed, you enter your first command.

```
S,Y: DDEF LIB8DD,,LIB8,OPTION=JOBLIB,DISP=OLD
```

You use this command to define the job library data set LIB8, which contains your assembled object modules. Although LIB8 is cataloged, you must define it with a DDEF command to make it available during this session.

```
S,Y: DEFAULT LIMEN=I
```

You desire all information messages to be presented at your terminal.

```
S,Y: PGM8
```

You cause the named object module to be loaded and executed.

When a module name is given as the command, execution begins at the module's standard entry point. It is the instruction named by the operand of the assembled END statement, or, if no operand is given, the first executable instruction in the first CSECT.

```
YOU: (press attention button)
```

When you do not receive the expected output after several minutes, you interrupt your program.

```
S,Y: !
     STOP
```

The system prints an exclamation mark to indicate its readiness to accept a command after an attention interrupt. You enter the PCS STOP command to determine the location in your program of the next instruction that was to be executed when you interrupted execution.

```
SYS: STOP AT PGM8EXT.(X'DA') PSW 2 0 0 004A3070
```

The STOP command causes the display of the symbolic location and the PSW at the point the interrupt occurred. In this case, the interrupt occurred just prior to the instruction X'ØA' (hexadecimal) bytes beyond the external symbol PGM8EXT. You can use this information to determine the corresponding source statement in the object listing.

The location of the interrupt is indicated as a displacement beyond the nearest internal symbol if you have issued a PCS command such as SET or AT. They make the ISD (and its internal symbols) available.

The rightmost field of the PSW gives the virtual storage address of the next instruction to be executed.

```
S,Y: LINE? DSNAME=SOURCE.PGM8,(2500,2600)
```

Suspecting that an undesirable loop has occurred in the convergence portion of your program, you request a printout of several of the source lines at the end of your convergence.

```
SYS: 0002500     CP     DIFF,EPSILON
     0002600     BH     RECYCLE
S,Y: DISPLAY PGM8.EPSILON,PGM8.DIFF
```

You request a printout of the appropriate variables, and you explicitly qualify the internal symbols (EPSILON and DIFF by the module name (PGM8).

```
SYS: PGM8.EPSILON=+.10000000E+04
     PGM8.DIFF=+.21301962E-02
S,Y: SET PGM8.EPSILON=1E-4
```

You decide to reestablish the value of the constant EPSILON to cause your program to converge more quickly.

```
SYS: PGM8.EPSILON=+.10000000E-03
```

After each SET is performed, a printout confirming the modification is available. The message filter code of I must be specified to obtain such information messages.

```
S,Y: GO
```

You issue the GO command to resume execution of the program. Since no operand is specified, execution resumes at the point of interruption.

```
PGM: JOB COMPLETED
```

Your program issues a message to the terminal to indicate successful completion of the program. Your program's RETURN macro instruction causes the typing of an underscore requesting the next command.

```
S,Y: LOGOFF
SYS:
```

The LOGOFF is accepted by the system.

Example 9: Use of PCS Dynamic Statements

In this example, you use some of the most powerful commands of the Program Control System to debug a complex program. PCS provides trace facilities, conditional program interruptions and modification of variables, and dumps.

After completing the LOGON procedure, you begin processing.

```
S,Y: DDEF DDCURR,,CURRENT,OPTION=JOBLIB,DISP=OLD
```

This DDEF command causes your job library CURRENT to be placed at the top of your program library list. CURRENT has been previously cataloged and contains assembled object modules.

```
S,Y: DDEF PCSOUT,VI,PCSOUT9
```

With the second DDEF command, you define the data set that will be filled by the PCS DUMP command, which you may print later. It requires the data definition name PCSOUT and virtual indexed (VI) organization. You name the data set PCSOUT9. It is automatically cataloged, since it will reside in public storage. The system defaults disposition to NEW, since the data set is being created in this task.

```
S,Y: LOAD MAIN9
```

The CALL command causes an object module to be loaded and then executed. Here you cause it to be loaded only. You do this so that you can insert AT statements in the module before executing it.

```
SYS: ***** UNDEFINED REF(FABLE) IN MODULE (MAIN9).  ADDRESS FFFFF000 ASSIGNED.
```

The system issues a message indicating that MAIN9 has a reference to an external symbol (FABLE) that does not exist in the libraries searched. An invalid address had been assigned for the reference in MAIN9 that will cause an interrupt if program execution reaches it.

When you realize that the symbol has been misspelled in the source program, you enter the necessary commands to correct the situation.

```
S,Y: LOAD TABLE
```

You request the object module defining the external symbol TABLE to be loaded into virtual storage. The module would have been implicitly loaded when MAIN9 was loaded if the spelling had been correct. Loading the module at this point, however, does not correct the problem entirely. MAIN9 still contains the invalid reference, a V-type adcon. Before entering a SET command to place the proper value into the adcon, you qualify your program's internal symbols.

```
S,Y: QUALIFY MNAME=MAIN9
```

After issuing this command, you can refer to internal symbols without the qualifying module name; they will be automatically qualified with the prefix "MAIN9."

```
S,Y: SET   ADDTAB=A'TABLE'
```

You request that the adcon defined in your source program by the name ADDTAB be set to the value of the address of eternal symbol TABLE. If you had not already explicitly loaded TABLE, you would have been prompted at this point to load the module containing it.

```
SYS:
```

Before it prints a symbol that you have qualified, the system reminds you of the qualification.

```
SYS: ADDTAB=0084C000
```

The contents of the modified adcon are displayed in hexadecimal. This is the virtual storage address of TABLE.

```
S,Y: AT LAST;STOP
```

This statement will cause your program to be interrupted when execution reaches the address corresponding to the statement named LAST.

```
SYS: 00001
```

The system assigns a number to each statement containing an AT (here 00001) that can be used for reference in removing the statements.

```
S,Y: CALL MAIN9
```

You initiate execution of the module. You must provide an operand in this call command, since a LOAD command naming another module has been entered after the loading of MAIN9.

```
SYS: E008 FIXED POINT DIVIDE INTERRUPT.  PSW = BFC000900280A1A
     INTERRUPT OCCURRED IN CSECT MAIN9C WITH DISPLACEMENT 000A1A FROM THE BEGINNING
     OF THE CSECT
```

Your program does not contain a routine for handling this type of interrupt.

```
S,Y: LINE? DSNAME=SOURCE.MAIN9,(3200,3500)
```

You request a printout of several of your source statements that correspond to the location of the interrupt. Tab stops are set at columns 18 and 24.

```
SYS: 0003200 DIVRTN  L    2,DVND
     0003300         SRDA 2,32
     0003400         D    2,DVSR
     0003500         ST   3,QUOT
```

Your program does not provide protection against division by zero, so you insert the necessary checking.

```
S,Y: AT DIVRTN; IF DVSR=0;SET 3R=0;BRANCH DIVRTN.(12)
```

With this statement, you request that the value of DVSR be compared to zero upon arrival at DIVRTN. If it is equal, general register 3 is set to zero, and control transfers to the instruction twelve bytes beyond DIVRTN.

```
SYS: 00002
```

This is the number the system assigns to your AT command.

```
S,Y: BRANCH DIVRTN
```

You cause your program to begin execution at DIVRTN so that you can immediately check the effectiveness of the PCS statements.

```
SYS: AT DIVRTN PSW 3 0 0 00280A12 2
```

The system issues a response to the statement indicating that the IF command has resulted in a "TRUE" comparison. The 2 to the right of the PSW printout is the number of the AT statement that caused the printout.

```
SYS: RUNNING FROM DIVRTN.(12)
```

This indicates that the branch has been taken.

```
SYS: STOP AT LAST PSW 2 0 0 00280F02 1
```

Execution has reached the location corresponding to LAST. The STOP you specified earlier is executed.

```
S,Y: DISPLAY QUOT, RESULT
```

You request a printout of two key variables in your program.

```
SYS: QUOT=0
     RESULT=1726
S,Y: REMOVE 1
```

You are convinced that the program is operating correctly, so you remove the dynamic STOP to prevent future interruption.

```
S,Y: GO
```

You cause your execution of your program to resume at the point of interruption.

```
PGM: JOB COMPLETED
```

Your program issues a message indicating its successful completion and then returns control to the keyboard.

```
S,Y: DUMP MN9PST
```

You request a hexadecimal dump of your PSECT by specifying its external name. It will be written in the PCSOUT9 data set you defined earlier. If you had used the internal name of the PSECT (MAIN.MN9PST), you would receive a formatted dump showing symbols and code in source format.

```
S,Y: RELEASE DDNAME=PCSOUT
```

If you wish to print the data set during this session, you must first issue a RELEASE command for its data definition. This causes the data set to be closed.

```
S,Y: PRINT PCSOUT9,,,,Y
```

Positional operand notation is used. The data set will be erased (Y) after it is printed. The commas specify options for which you have chosen the default.

```
SYS:
```

The data set will be printed as a separate task. A batch sequence number will be assigned for system control.

```
S,Y: LOGOFF
SYS:
```

The LOGOFF is accepted by the system. Remember that the alterations you made to your program with the PCS commands (SET,AT) exist only in virtual storage. If you want to make permanent changes to a program, you must reassemble from an altered source data set. This causes the changes to be incorporated into the object module, which you would then load.

Changes you make with the SET command remain in effect as long as the program is loaded. By contrast, changes you make with AT commands in *any* of your programs are completely removed if you issue an UNLOAD command, even if the program you unload does not contain AT statements and is not linked to other programs. Logging off causes all of your programs to be unloaded from virtual storage.

Example 10: Input and Output on Magnetic Tape

In the previous examples, all of your data sets resided on direct-access devices (disks). In this example, your data sets reside on tape.

In Part 1, you execute a program that reads a cataloged data set from tape and writes a data set on a new tape, which you then catalog.

In Part 2, the same program reads a cataloged data set created on tape by OS or OS/VS and then writes a data set on the same tape you cataloged in Part 1.

Part 1: Reading a Cataloged Labeled Data Set

You complete the LOGON procedure and begin your task. Your previously-assembled program includes the following source statements:

```
MN10PST  PSECT
         ENTRY   CST10
         ENTRY   EOD10
         ENTRY   SYN10
         DC      F'76'                          SAVE AREA LENGTH
         DC      18F'0'                         REMAINDER OF SAVE AREA
AREA     DS      80C
DCBIN1   DCB     DDNAME=IN10DD,EODAD=EOD10,SYNAD=SYN10
DCBOUT1  DCB     DDNAME=OUT10DD,RECFM=F,LRECL=80,SYNAD=SYN10
MN10CST  CSECT
CST10    SAVE    (14,12)                        SAVE REGISTERS IN CALLER'S SAVE AREA
         L       14,72(0,13)                    GET RCON FROM CALLER'S SAVE AREA
         ST      14,8(0,13)                     STORE FORWARD POINTER
         ST      13,4(0,14)                     STORE BACKWARD POINTER
         LR      13,14                          SET PSECT AND SAVE AREA REGISTER
         USING   MN10PST,13                     PSECT COVER REGISTER
         LR      12,15
         USING   CST10,12                       CSECT COVER REGISTER
           .
           .
           .
         OPEN    (DCBIN1,,DCBOUT1,(OUTPUT))     OPEN DCBS
AGAIN    READ    IN1DECB,SF,DCBIN1,AREA         READ INPUT RECORD
         CHECK   IN1DECB
           .
           .
           .
         (modify record)
           .
           .
           .
         WRITE   OUT1DECB,SF,DCBOUT1,AREA       WRITE OUTPUT RECORD
         CHECK   OUT1DECB
         B       AGAIN                          RECYCLE
EOD10    CLOSE   (DCBIN1,,DCBOUT1)              CLOSE DCBS
         LA      1,OKMSG
EXIT     EXIT    (1)                            RETURN TO TERMINAL
SYN10    LA      1,SYNMSG
         B       EXIT
OKMSG    DC      AL1(L'M1)                      MESSAGE LENGTH IN FIRST
M1       DC      C'PROGRAM FINISHED OK'
SYNMSG   DC      AL1(L'M2)                      BYTE OF MESSAGE
M2       DC      C'SYN ERROR'
         END
```

The program reads a record from one data set, modifies the record, and then writes it in a data set on another tape.

The CHECK is required to complete a READ or WRITE I/O request. It detects any errors or exceptional conditions that may occur and, when these arise, transfers control to the external symbol specified in the EODAD or SYNAD DCB field. When the program attempts to read past the last record, control is passed to EOD10 and the DCB's are closed.

The EXIT macro instruction causes an appropriate message to be printed, and then returns control to the terminal. Notice that each DCB contains the name of one of your DDEF commands, which in turn specifies the data set described by the DCB.

```
S,Y: DDEF IN10DD,,IN10,DISP=OLD
```

Since your input data set is already cataloged, you need entry only these parameters. The omitted information is provided by the catalog entry and by the tape label, which was created by the system when the data set was written. OLD indicates that the data set exists.

```
SYS:
```

You will be notified that your task is waiting for the system operator to mount your tape reel (a private volume). The system obtains its volume serial number from your catalog. When the tape volume has been mounted and activated, your task will proceed.

```
S,Y: DDEF DDNAME=OUT10DD,DSORG=PS,DSNAME=OUT10,UNIT=(TA,9),VOLUME=-
     (PRIVATE),LABEL=(,SL)
```

You found it necessary to continue your command operands on a second line. Here you define the output data set your program will write on tape. It is not cataloged and is not yet created, so you must supply all the necessary parameters in your DCB and DDEF. Note that the continuation hyphen may occur anywhere so long as it is the last non-blank character in the line.

The data set is to have physical sequential (PS) organization, as do all data sets residing on magnetic tape.

The UNIT field of this DDEF command indicates that your data set is to reside on a 9-track tape.

The DISPOSITION field is defaulted (by omission), indicating that the data set is NEW (i.e., does not now exist).

The VOLUME field specifies a private volume (all tapes are private). You have not specified the volume serial number of a tape reel in VOLUME field, so the system instructs the operator to choose a tape reel from the installation pool.

The LABEL field specifies that the system is to create standard labels (SL) on the tape when creating the data set.

```
SYS:
```

You must wait for the operator to select a tape reel, mount it, and inform the system of its volume serial number. You will then be informed of the selected volume, at your terminal, for future reference.

```
SYS: MAIN10
```

Now you run your program.

```
PGM: EXIT, RELEASE ALL UNNEEDED DEVICES
     PROGRAM FINISHED OK
```

At the conclusion of your program, EXIT prints the messages and returns control to the terminal; this is indicated by the underscore.

```
YOU: CATALOG OUT10
```

You catalog your data set. Only VAM data sets are automatically cataloged when created.

```
S,Y: LOGOFF
```

SYS:

LOGOFF was accepted by the system. In previous examples, your data sets were all automatically cataloged on public disks. Public disks remain mounted while the system is operational; they contain data sets belonging to the users with whom you share the system.

Your private volumes (all tapes and your own private disks) are dismounted at the end of your task and later retained. Thus, it is not necessary to catalog them in order to preserve them. However, cataloging your private volumes (disks and tapes) enables you to write DCB macro instructions and issue DDEF commands with the minimum required parameters. The system obtains the missing parameters from your catalog entry.

Part 2: Reading a Cataloged but Unlabeled OS or OS/VS Data Set

You complete the LOGON procedure and enter your first command.

S,Y: DDEF IN10DD,,*STUFF,DISP=OLD

You define your input data set. It differs from the input data set in Part 1 in several ways:

- It was created under OS or OS/VS. You indicate this by specifying the data set name with an asterisk preceding.
- It is unlabeled. This means that you cannot use the label to provide any of the DCB and DDEF parameters. Those not specified in your DCB macro instruction or DDEF command are taken from the catalog entry. Appendix E provides further details about these sources and the order in which they are searched.

SYS:

You will be notified that your task is waiting for the system operator to mount your tape reel (a private volume). The system obtains its volume serial number from your catalog. When the tape volume has been mounted and activated, your task will proceed.

S,Y: DDEF OUT10DD,PS,OUT10A,UNIT=(TA,9),VOLUME=(,101010),LABEL=(2,,)

You decide to write your output data set on the same tape reel you used in Part 1. You indicate that it will be the second data set on the tape with the 2 in the LABEL field. You previously omitted this parameter (default=1, or first). The data set does not now exist, so you choose the default for disposition (NEW) and omit the field.

S,Y: CALL MAIN10

You run the program described in Part 1 of this example. Note that the DDEF command enables you to supply various data set and volume information at the terminal.

PGM: EXIT, RELEASE ALL UNNEEDED DEVICES
PGM: PROGRAM FINISHED OK
S,Y: RELEASE IN10DD

This command deletes the DDEF command you issued earlier, thereby withdrawing definition of the data set. Accordingly, the system instructs the operator to dismount your tape reel and save it. The unit on which it was mounted is now free for other use.

S,Y: CATALOG OUT10A,N

Positional operand notation is used here. You catalog your new data set, indicating with the N that the catalog entry to be created for it is new (not currently cataloged). By default, access will be unlimited.

S,Y: LOGOFF
SYS:

The LOGOFF is accepted by the system. The label for a data set on tape is similar to the data set control block (DSCB) provided by the system when a data set on a disk is first created. Both contain information about the data set that may be required when the data set is processed. Appendix E provides further details.

Example 11: Conversational Initiation of Nonconversational Tasks

It is often more convenient to have your programs run after you have left the terminal—that is, to have them run in nonconversational mode. Two ways of doing this are shown in this example.

In Part 1, you begin your task conversationally and then use the BACK command to switch its execution to the nonconversational mode.

In Part 2, you construct a nonconversational task and then use the EXECUTE command to cause it to be executed at a later time.

Part 1: The BACK Command

You complete the LOGON procedure and begin processing.

S,Y: DATA DSNAME=BACKPROG

With this command, you build the SYSIN data set (named BACKPROG) that will provide input to your task after you have switched to nonconversational mode. You do not need to issue a DDEF command for the data set created by a DATA command.

S,Y: #DDEF OUT11,,OUTDS,DISP=OLD

The system prompts (with #) for the first command to be executed in your nonconversational task. This DDEF command defines the cataloged data set that MAIN11 has previously written and which will be written over when MAIN11 is run. It resides on a public disk.

S,Y: #CALL MAIN11

This program is already assembled and its object module resides on your USERLIB data set, which never requires a DDEF command. It includes the following statements:

```
MN11PST  PSECT
         ENTRY  STRT11
         DC     F'76'                      SAVE AREA LENGTH
         DC     18F'0'                     REMAINDER OF SAVE AREA
AREA     DS     80C                        DATA AREA
DCBOUT   DCB    DDNAME=OUT11
MN11CST  CSECT
STRT11   SAVE   (14,12)                    SAVE REGISTERS IN CALLER'S SV AREA
         L      14,72(0,13)                GET RCON FROM CALLER'S SAVE AREA
         ST     14,8(0,13)                 STORE FORWARD POINTER
         ST     13,4(0,14)                 STORE BACKWARD POINTER
         LR     13,14                      SET PSECT AND SAVE AREA REGISTER
         USING  MN11PST,13                 PSECT COVER REGISTER
         LR     12,15
         USING  STRT11,12                  CSECT COVER REGISTER
           .
           .
           .
         OPEN   (DCBOUT,(OUTPUT))          OPEN OUTPUT DCB
ALPHA    GATRD  AREA,LNG                   READ FROM SYSIN
         CLI    AREA+3,C'E'                REACHED END?
         BE     FINISH                     BRANCH IF YES
           .
           .
           .
         create a record
           .
           .
           .
```

```
         PUT     DCBOUT,AREA                             WRITE RECORD
         B       ALPHA                                   RECYCLE
FINISH   CLOSE   (DCBOUT)                                CLOSE DCB
         .
         .
         .
LNG      DC      F'80'                                   AREA LENGTH
         END
```

Notice how the DDEF command specifying the same data definition name as specified in your DCB statement contains the name of the desired output data set.

```
S,Y: #
     PART 0049628-49-11-MODEL 879

S,Y: #
     PART 0078928-49-11 MODEL 127

S,Y: #
     PART 0078927-49-10 MODEL 127
       .
       .
       .
```

This is part of the data to be read by GATRD.

```
S,Y: #
     E
```

You identify the end of your data with E. Your task will be abnormally terminated if GATRD attempts to read past the end of SYSIN.

Note that errors made while entering this data and the entire command sequence under the DATA command are not detected until the nonconversational task is executed.

```
S,Y: #
     LOGOFF

S,Y: #
     %E
```

You enter %E to indicate the end of your BACKPROG data set. The system then prompts you with the underscore to continue your conversational task.

```
S,Y: BACK DSNAME=BACKPROG
SYS: _
```

Beginning with the first command in your BACKPROG data set (DDEF OUT11 . . .), your task is continued nonconversationally. If the system is not able to accept your request, the BACK command is rejected. It can be re-issued later during this session.

Your nonconversational task will be abnormally terminated if it attempts to access a data set on a private volume for which you have not issued a DDEF command prior to issuing the BACK command. The operator must have mounted any required volumes before the nonconversational task is created.

The conversational part of the task is now finished. You can leave the terminal or log on again.

WARNING: The BACK command may not complete its operation if the attention key is depressed shortly after issuing the command. The result is a non-conversational task still connected to a terminal. Wait a few seconds before reinitiating LOGON procedures.

Part 2: The EXECUTE Command

You complete the LOGON procedure and begin your task.

```
S,Y: DATA DSNAME=EXECPROG
```

You now build the SYSIN for the separate nonconversational task that is to be executed later. The system prompts for commands and data with the number sign. When the data set EXECPROG is completed, it will be automatically cataloged. This cataloging is a function of the DDEF issued when the DATA command is used.

```
S,Y: #
        LOGON ADUSERID,,,ADACCT29
```

You provide information for the LOGON of the nonconversational task (LOGON starts in column 3). Except for the inclusion of this command, the tas kis exactly the same as the one you constructed in Part I of this example. No password is given for a nonconversational task.

If any private volumes were to be used during the nonconversational task, a SECURE command would be needed at this point. The command would notify the operator to secure a unit for your private volume(s) and mount them before the task is initiated.

```
S,Y: #
      DDEF OUT11,,OUTDS,DISP=OLD
```

This command defines the cataloged data set on which your program (MAIN11) will write its output.

```
S,Y: #
      MAIN11
```

This program is already assembled and stored on your USERLIB, which doesn't require a DDEF command. Its relevant statements are shown in Part 1.

```
S,Y: #
      PART 0049628-49-10  MODEL 879
S,Y: #
      PART 0078928-49-11  MODEL 127
S,Y: #
      PART 0078927-49-10  MODEL 127
        .
        .
        .
```

This is your input data.

```
S,Y: #
      E
S,Y: #
        LOGOFF
S,Y: #
      %E
S,Y: EXECUTE DSNAME=EXECPROG
SYS:
```

Your request for a nonconversational task has been accepted by the system. The task is initiated when system resources are available.

The data set EXECPROG will provide the SYSIN for your nonconversational task. Its SYSOUT will consist of system messages and any output to SYSOUT generated by your program. SYSOUT is printed later as a separate nonconversational task, and the listing is identified as yours. The system prompts you with an underscore (below). You are free to enter any command sequence to continue your conversational task.

```
S,Y: LOGOFF
```

This LOGOFF is for your conversational task.

```
SYS:
```

LOGOFF is accepted by the system.

Example 12: Preparing a Job for Nonconversational Processing

It is not always convenient or efficient to use remote terminals to create or initiate nonconversational tasks. In this example, your task is on punched cards. You submit to the machine room the card deck that contains the commands and data for your task. The operator then enters them into the system.

```
  LOGON ADUSERID,,ADACCT29
```

You initiate your nonconversational task with the LOGON command. All LOGON parameters must be included on one card. For nonconversational tasks, the password parameter must be omitted. This command must begin in the third column. The first two columns *must* be blank.

```
ASM AT2EX12,N,LISTDS=N
```

You construct the same program as in Example 4 specifying different external symbols to avoid duplication when your assembled object module is placed on USERLIB.

Remembering that you cataloged the source data set in Example 4, you use a different module name so that the source data set created during this assembly will have a different name. You indicate that you do not want a list data set. Your listings will therefore be printed automatically on SYSOUT and no record of them retained after printing.

```
PST12      PSECT
           ENTRY  BEGIN12
           DC     F'76'
           DC     18F'0'
AREA       DC     F'0'                RD/WR AREA
CST12      CSECT
BEGIN12    SAVE   (14,12)
           L      14,72(0,13)
           ST     14,8(0,13)
           ST     13,4(0,14)
           LR     13,14
           USING  PST12,13
           BASR   11,0
           USING  *,11                LOCAL BASE REG
HERE       EQU    *
           GATRD  AREA+3,LENGTH       READ FROM SYSIN
           CLI    AREA+3,C'E'         CHECK IF END
           BE     LEAVE               BRANCH IF YES
           MVZ    AREA+3(1),=X'00'    CONVERT TO BINARY
           L      5,AREA
           SLA    5,1                 MULT BY 2
* RESTRICTED TO INTEGERS FROM 0 TO 4
           ST     5,AREA
           MVZ    AREA+3(1),=X'FF'    CONVERT TO EBCDIC
           GATWR  AREA+3,LENGTH       WRITE ON SYSOUT
           B      HERE
LEAVE      EXIT   'PGM FINISHED'
LENGTH     DC     F'1'                LENGTH OF AREA
           END
```

You could have assembled from a prestored source data set, just as in conversational mode.

```
AT2EX12
2
1
3
```

When your program is executed, it will attempt to read data from SYSIN, which is this stream of commands and data. Here you supply the input for three cycles of GATRD.

```
E
```

This entry signals the end of data input and will cause your program to branch to EXIT, which will return control to SYSIN. The next command in the SYSIN stream below will then be executed.

At the completion of your task, SYSOUT will be printed as a separate task. It will contain commands, listings, and the program output generated by GATWR. SYSOUT will include neither program input (2,1,3) nor assembler parameters (AT2EX12,N). It will include your listings since you specified in your ASM command that no list data set was required.

```
  LOGOFF
```

Unlike the LOGON command, LOGOFF permits no parameters. All data set disposition must be completed before LOGOFF. The data set named SOURCE.AT2EX12 will be automatically cataloged by the system. No user action is necessary. A DDEF is issued at ASM time, accommodating the module name AT2EX12, by prefixing SOURCE to the object module name.

After completion of this task, you may execute AT2EX12 again, since the object module will be retained on USERLIB.

The LOGOFF command must begin in card column 3.

Example 13: Storing DDEF Commands for Later Use

In this example, you create a data set containing DDEF commands for frequently used data sets. Your DDEF commands create a library hierarchy that permits you to select various versions of identically named subroutines. You complete the LOGON procedure and enter your first command.

```
S,Y: DATA DDPACK,I,(100,100)
```

Positional parameter notation is used here. The DATA command can be used to store any data, source statements, or commands you wish to enter through the terminal. Here you store a set of frequently-used DDEF commands in a data set you name DDPACK. They are stored as character strings in a line data set, but are interpreted as commands when they are later retrieved with the CDD command. The system prompts you by typing a line number for each line, since indexing was specified. The data set named DDPACK will be automatically defined and cataloged by the system.

```
S,Y: 0000100DDEF JOB1DD,,JOB1,OPTION=JOBLIB
S,Y: 0000200DDEF JOB2DD,,JOB2,OPTION=JOBLIB
```

A mixture of positional and keyword parameter notation is used here. These two DDEF commands define the cataloged job libraries that contain the object modules of your subroutines. They already exist, so you specify their dispositions as OLD, by default.

```
S,Y: 0000300DDEF IN13DD,,IN13,DISP=OLD
```

Your program is to retrieve its input from the cataloged data set in IN13.

```
S,Y: 0000400DDEF OUT13DD,VS,OUT13,VOLUME=(,131313),UNIT=(DA1,2311)
```

Your output data set will reside on a private disk whose volume serial number is 131313.

```
S,Y: 0000500%E
```

You signal the end of the data set containing your DDEF commands. The system then prompts for the next conversational command with the underscore.

```
S,Y: CDD DSNAME=DDPACK,(JOB2DD,IN13DD,OUT13DD)
```

This command causes the three DDEF statements in DDPACK that you specify to be entered in SYSIN. If you omit the DDNAME field, all the DDEF commands are entered in SYSIN.

The data set containing the DDEF commands must be defined for the current session (which it was when created by the DATA command), or be cataloged.

```
SYS: DDEF JOB2DD,,JOB2,OPTION=JOBLIB
SYS: DDEF IN13DD,,IN13,DISP=OLD
```

Each DDEF command that is issued is printed at the terminal.

```
SYS: DDEF OUT13DD,VS,OUT13,VOLUME=(,131313),UNIT=(DA,2311)
SYS:
```

You must wait for the operator to mount your private disk. No wait was required for the two data sets above because they are on public volumes which remain mounted while the system is operational. A message will advise you of the wait state.

```
S,Y: CALL MAIN13
```

Your program, which is stored on your USERLIB, calls two subroutines (SUB13A and SUB13B).

You previously assembled a version of SUB13A and stored it on JOB1. Later you reassembled another version and stored it on JOB2. For this session, you want to use the version on JOB2, so you issue a DDEF command for JOB2, but not for JOB1 (see CDD parameters). Thus, your program library list begins with JOB2, then USERLIB, and then SYSLIB; this is the order in which the loader searches for object modules.

Versions of SUB13B are stored in JOB1 and USERLIB. Since JOB1 job library is not in your program library list, the version on USERLIB will be loaded.

PGM: MAIN13 FINISHED

Your program prints this message with a GATWR macro instruction and then returns control to the caller (the system) with a RETURN macro instruction. The system then prompts with the underscore.

S,Y: LOGOFF
SYS: _

The LOGOFF is accepted by the system.

All VAM data sets are automatically cataloged. Your private volume is dismounted by the operator and retained at the installation.

Example 14: Writing and Updating Virtual Index Sequential Data Sets

In the first part of this example, you run a program that reads a virtual sequential (VS) data set. After reading a record, the program adds a key to it and then writes the record into a virtual index sequential (VI) data set. The process is then repeated until all the records have been indexed.

In the second part, another program modifies the VISAM data set by deleting records.

Both of these data sets are on disks.

Part 1: Writing a VI Data Set

You complete the LOGON procedure and begin processing your task.

You are going to run a program which adds an index to each record of a VS data set, and then writes the modified records in a VI data set. The program includes the following statements:

```
MN14APST  PSECT
          ENTRY  MN14
          ENTRY  EOD14A
          ENTRY  SYN14A
          DC     F'76'                          SAVE AREA LENGTH
          DC     18F'0'                         REMAINDER OF SAVE AREA
AREA      DS     84C                            TEMP RECORD STORAGE
DCBIN14   DCB    DDNAME=IN14DD,EODAD=EOD14A
DCBOUT14  DCB    DDNAME=OUT14DD,SYNAD=SYN14A,LRECL=84,RECFM=F,DSORG=VI,KE-
                 YLEN=4,RKP=0
MN14ACST  CSECT
MN14      SAVE   (14,12)                        SAVE REGISTERS IN CALLER'S SV AREA
          L      14,72(0,13)                    GET RCON FROM CALLER'S SAVE AREA
          ST     14,8(0,13)                     STORE FORWARD POINTER
          ST     13,4(0,14)                     STORE BACKWARD POINTER
          LR     13,14                          SET PSECT AND SAVE AREA REGISTER
          USING  MN14APST,13                    PSECT COVER REGISTER
          LR     12,15
          USING  MN14,12                        CSECT COVER REGISTER
            .
            .
            .
          OPEN   (DCBIN14,,DCBOUT14,(OUTPUT))   OPEN DCB'S
          LA     10,0                           INITIALIZE KEY VALUE
ALPHA     GET    DCBIN14,AREA+4                 GET VSAM RECORD
          ST     10,AREA                        SET KEY VALUE
          LA     10,1(10)                       INCREMENT VALUE
          PUT    DCBOUT14,AREA                  PUT VISAM RECORD
          B      ALPHA                          RECYCLE
EOD14A    CLOSE  (DCBIN14,,DCBOUT14)            CLOSE DCB'S
          EXIT   OKMSG
SYN14A    CLOSE  (DCBIN14,,DCBOUT14)            CLOSE DCB'S
          EXIT   SYNMSG
OKMSG     DC     A(L'OK)
OK        DC     C'FINISHED OK'
SYNMSG    DC     A(L'SYN)
SYN       DC     C'SYN ERROR OCCURRED'
          END
```

S,Y: DDEF IN14DD,,MYVSDATA

You define the VS data set which provides input to your program. At execution time, the parameters that you omitted from the DCB macro instruction and from this command will be provided from the catalog and the data set's DSCB. These fields are: LRECL(80),RECFM(F),UNIT(DA,2311).

S,Y: DDEF OUT14DD,VI,MYVIDATA,UNIT=(DA,2311),VOLUME=(,141414)

The data set which your program creates is to reside on one of your own disks. Since there is no DSCB until the data set is created, and no catalog entry, you must specify all the required parameters in your DCB macro instruction and in this DDEF command. The DSORG parameter is not necessary, but supplying it prevents the system from providing a default option that you may not want.

S,Y: MAIN14A

Now you run your program. Entry is at the first byte of the first CSECT since the operand of the END statement was blank.

PGM: EXIT, RELEASE ALL UNNEEDED DEVICES
FINISHED OK

When it has finished, the EXIT macro prints a message and causes control to be returned to the terminal. It then prompts with the underscore.

S,Y: ERASE MYVSDATA

No longer needing your input data set, you erase it from public storage and delete its catalog entry.

S,Y: LOGOFF
SYS:

LOGOFF is accepted by the system.

Part 2: Updating a VI Data Set

You complete the LOGON procedure and begin your task.

A program named UPDATER reads records from the data set you saved in part 1. It deletes any that begin with "A". It includes the following statements:

```
UPDPST   PSECT
         ENTRY    START
         ENTRY    EODUPD
         ENTRY    SYNUPD
         DC       F'76'                          SAVE AREA LENGTH
         DC       18F'0'                         REMAINDER OF SAVE AREA
DCBDEL   DCB      DDNAME=OUT14DD,SYNAD=SYNUPD,EODAD=EODUPD
KEYLOC   DS       F
AREA     DS       84C
UPDCST   CSECT
START    SAVE     (14,12)                        SAVE REGISTERS IN CALLER'S SAVE AREA
         L        14,72(0,13)                    GET RCON FROM CALLER'S SAVE AREA
         ST       14,8(0,13)                     STORE FORWARD POINTER
         ST       13,4(0,14)                     STORE BACKWARD POINTER
         LR       13,14                          SET PSECT AND SAVE AREA REGISTER
         USING    UPDPST,13                      PSECT COVER REGISTER
         LR       12,15
```

```
         USING    START,12                             CSECT COVER REGISTER
           .
           .
           .
         OPEN     (DCBDEL,(UPDAT))                     OPEN DCB FOR INPUT & UPDATE
         LA       4,0                                  INITIALIZE KEY VALUE
ALPHA    ST       4,KEYLOC                             SET VALUE IN KEYLOC
         READ     DECBDEL,KY,DCBDEL,AREA,KEYLOC        READ RECORD AT KEY VALUE
         CLI      AREA+4,C'A'                          BEGINS WITH A?
         BNE      BUMP                                 BRANCH IF NO
         LA       0,KEYLOC                             SET KEYLOC ADDRESS
         DELREC   DCBDEL,K,(0)                         DELETE RECORD
BUMP     LA       4,1(4)                               INCREMENT KEY VALUE
         B        ALPHA                                RECYCLE
EODUPD   CLOSE    (DCBDEL)                             CLOSE DCB
         LA       1,OKMSG                              SET OK MESSAGE
LEAVE    EXIT     (1)                                  EXIT
SYNUPD   LA       1,SYNMSG                             SET ERROR MESSAGE
         B        LEAVE
OKMSG    DC       A(L'OK)
OK       DC       C'FINISHED DATA SET'
SYNMSG   DC       A(L'SYN)
SYN      DC       C'SYN ERROR OCCURRED'
         END
```

`S,Y: DDEF OUT140D,,MYVIDATA`

You define the data set which your program reads, and from which it deletes records. Since it is cataloged, you need provide only the minimum DDEF parameters.

`S,Y: UPDATER`

You have previously assembled this program. Its object module resides on your USERLIB.

```
PGM: EXIT, RELEASE ALL UNNEEDED DEVICES
     FINISHED DATA SET
```

After your program is completed, the EXIT macro instruction prints its messages and returns control to the terminal.

```
S,Y: LOGOFF
SYS:
```

LOGOFF is accepted by the system.

Example 15: Missing Subroutines

In this example, you attempt to execute a program that refers to a missing subroutine. In Part 1, you proceed without the subroutine; in Part 2, you alter your program library list to make the subroutine available.

Part 1: Proceeding Without a Missing Subroutine

You complete the LOGON procedure and enter your first command.

```
S,Y: CALL MAIN15
```

The CALL command causes the object module to be loaded from your USERLIB and then executed.

```
SYS: *****UNDEFINED REF(SUB15   ) IN MODULE (MAIN15     ) ADDRESS FFFFF000 ASSIGNED.
     (MAIN15) ERROR IN LOADING MODULE.
     STATEMENT REJECTED.
```

MAIN15 contains a reference to SUB15. During loading, SUB15 could not be found, so the loader assigned an invalid address that will cause an interrupt if the execution of MAIN15 ever reaches that reference.

The second message indicates that MAIN15 has been loaded but that the execution part of CALL has been rejected.

```
S,Y: CALL MAIN15
```

You know that this reference will not be encountered during execution, so you decide to run without SUB15.

```
PGM: MAIN15 COMPLETED
```

Successful completion of your program is indicated by this message.

```
S,Y: LOGOFF
SYS:
```

LOGOFF is accepted by the system.

Part 2: Supplying the Missing Subroutine

```
S,Y: MAIN15
```

After logging on, you again attempt to load and execute your program.

```
SYS: *****UNDEFINED REF(SUB15) IN MODULE (MAIN15   ) ADDRESS FFFFF000 ASSIGNED.
     (MAIN15) ERROR IN LOADING MODULE.
     STATEMENT REJECTED.
```

And again your subroutine cannot be found. But this time you decide to provide it.

```
S,Y: UNLOAD MAIN15
```

First you must unload MAIN15 from virtual storage, since it contains the invalid references to SUB15.

```
S,Y: DDEF LIB15DD,VP,LIB15,DISP=OLD,OPTION=JOBLIB
```

The object module for SUB15 is stored in your LIB15 job library. Issuing this DDEF command places the library at the top of your program library list. Now, when MAIN15 is loaded, the reference to SUB15 will be satisfied.

```
S,Y: MAIN15
```

This time when MAIN15 is loaded the reference is satisfied from your LIB15 job library.

```
PGM: MAIN15 COMPLETED
```

And your program runs to completion.

```
S,Y: LOGOFF
SYS:
```

LOGOFF is accepted by the system.

Example 16: Entering Data for Later Use

In this example, you use the terminal keyboard to enter statements of a source data set directly into the system, rather than keypunching them and then entering the card deck.

After completing the LOGON procedure, you enter your first command.

```
S,Y: EDIT SOURCE.READER
```

Positional operand notation is used.

You specify the fully-qualified name of the source data set that you are about to enter at the keyboard. The EDITOR will create a virtual indexed-sequential data set. This indexing is necessary if you should enter an erroneous line and want to correct your keyboard input. ASM also requires the line numbering (indexing). The data set is automatically defined, cataloged, and placed on a public disk.

The system prompts for your statements by issuing the next line number it has assigned.

Seven digit line numbers are shown, to indicate the maximum length of the numbers.

```
S,Y: 0000100PST16     PSECT
S,Y: 0000200          ENTRY  BEGIN16
S,Y: 0000300          DC     F'76'
S,Y: 0000400          DC     18F'0'
S,Y: 0000500CST16     CSECT
S,Y: 0000600BEGIN16   BASR   11,0
S,Y: 0000700          USING  *,11                LOCAL BASE REGISTER
S,Y: 0000800          L      13,72(0,13)
S,Y: 0000900          USING  PST16,13
S,Y: 0001000HERE      EQU    *
S,Y: 0001100          GATRD  AREA+3,LENGTH       READ FROM SYSIN
S,Y: 0001200          CLI    AREA+3,C'E'         CHECK IF END
S,Y: 0001300          BE     LEAVE               BRANCH IF YES
S,Y: 0001400          MVZ    AREA+3(1),=X'00'    CONVERT TO BINARY
S,Y: 0001500          L      5,AERA
                                REA
```

You notice your typing error before you press the carriage return key. You backspace three times, move the paper up a line to avoid over-typing, and enter the correct letters. Then you press the carriage return key.

```
S,Y: 0001600          SLA    5,1                 MULTIPLY BY 2
S,Y: 0001700*RESTRICTED TO INTEGER FROM 0 TO 4
S,Y: 0001800          ST     5,AREA
S,Y: 0001900          MVZ    AREA+3(1),=X'FF'    CONVERT TO EBCDIC
S,Y: 0002000          GATWR  AREA+3,LENGTH       WRITE ON SYSOUT
S,Y: 0002100          B      THERE
S,Y: 0002200LEAVE     EXIT   'PGM FINISHED'
S,Y: 0002300AREA      DC     F'0'                READ/WRITE AREA
S,Y: 0002400LENGTH    DC     F'1'                LENGTH AREA
S,Y: 0002500_REVISE 2100
```

After you are prompted for the number 2500, you notice an error in line 2100. You enter _REVISE 2100 and then the new line. To insert a new line, simply give it a line number that falls between two existing line numbers.

```
S,Y: 0002500          B      HERE
S,Y: EXCISE 1700
```

Here you delete line number 1700.

```
S,Y: INSERT 0002500
S,Y: 0002500          END
```

You are again prompted to enter line number 2500 and you do so.

```
S,Y: 0002600_END
```

The _END indicates completion of your data set. The system then prompts for another command with the underscore.

```
S,Y: EDIT SOURCE.READER
S,Y: LIST
```

To check on the accuracy of your corrections, you cause your source data set to be printed at the terminal. (You could also have printed it on the high-speed printer by issuing a PRINT command; the dsname parameter would be SOURCE.READER).

```
S,Y: 0000100 PST16   PSECT
S,Y: 0000200         ENTRY  BEGIN16
S,Y: 0000300         DC     F'76'
S,Y: 0000400         DC     18F'0'
S,Y: 0000500 CST16   CSECT
S,Y: 0000600 BEGIN16 BASR   11,0
S,Y: 0000700         USING  *,11                LOCAL BASE REGISTER
S,Y: 0000800         L      13,72(0,13)
S,Y: 0000900         USING  PST16,13
S,Y: 0001000 HERE    EQU    *
S,Y: 0001100         GATRD  AREA+3,LENGTH       READ FROM SYSIN
S,Y: 0001200         CLI    AREA+3,C'E'         CHECK IF END
S,Y: 0001300         BE     LEAVE               BRANCH IF YES
S,Y: 0001400         MVZ    AREA+3(1),=X'00'    CONVERT TO BINARY
S,Y: 0001500         L      5,AREA
S,Y: 0001600         SLA    5,1                 MULTIPLY BY 2
S,Y: 0001800         ST     5,AREA
S,Y: 0001900         MVZ    AREA+3(1),=X'FF'    CONVERT TO EBCDIC
S,Y: 0002000         GATWR  AREA+3,LENGTH       WRITE ON SYSOUT
S,Y: 0002100         B      HERE
S,Y: 0002200 LEAVE   EXIT   'PGM FINISHED'
S,Y: 0002300 AREA    DC     F'0'                READ/WRITE AREA
S,Y: 0002400 LENGTH  DC     F'1'                LENGTH AREA
S,Y: 0002500         END
S,Y: INSERT 450
```

You notice a modifiable work space named AREA in the CSECT. You decide to follow recommended programming practice and put it in the PSECT.

```
S,Y: 0000450 AREA    DC     F'0'                READ/WRITE AREA

S,Y: EXCISE 2300

S,Y: _END
```

You have successfully deleted line 2300. This was the AREA entry in the CSECT. Now you signal the end of modifications by typing _END.

```
S,Y: LOGOFF
SYS:
```

LOGOFF is accepted by the system.

The data set SOURCE.READER will be available for future use, since it was automatically cataloged. If you decide at a later time to add extensively to the data set named SOURCE.READER (line 2500 and beyond), you simply issue a DDEF command for the data set. After removing the present END statement (using the MODIFY command), a DATA command with a start line number of 2500 will allow you to continue the building of the line data set begun earlier. The DATA command will function with existing data sets as well as with ones that are being created in the current task.

Example 17: Data Set Considerations When Interrupting Program Execution

In this example you discover that a program you are running is reading the wrong data set. You interrupt it, supply the correct data set, and continue.

After completing the LOGON procedure, you begin processing your task.

```
S,Y: EVV 2311,171717
```

The data set from which your program will receive its information (IN17A) resides on a private volume which was created originally for another system and has therefore not been cataloged for your system. You present it to the present system with the Enter Vam Volumes (EVV) command, specifying device type (2311) and volume number (171717). All data sets on volume 171717 that have not been previously cataloged for this system will be automatically cataloged.

```
S,Y: DDEF IN17DD,,IN17A
```

The data set was automatically cataloged as a result of the EVV command. Only minimum parameters are now required to describe it in the DDEF command.

```
SYS:
```

A message will advise you the task is waiting for the operator to mount your private disk volume. Once the volume is mounted, you will be advised that your task is ready to proceed.

```
S,Y: MAIN17
PGM: AVERAGE I.Q. OF COMPUTER PROGRAMMERS IS: 0.30103
```

Your program contains a GATWR macro instruction to print the results of its computation on the terminal.

```
YOU: (press attention button)
```

Noticing slightly incorrect output, you interrupt your task during printout of output. See Appendix D for a discussion of attention interrupt handling.

```
SYS: !
YOU: UNLOAD MAIN17
```

The system prompts you with the exclamation point; you can now enter any command.

Suspecting incorrect input data, you decide to select another input data set that resides on a different disk. The DCB for data set IN17A in your program has been opened. It contains information that describes the incorrect data set. Unloading your program closes all of its open Data Control Blocks (DCB).

```
S,Y: RELEASE IN17DD
```

Now you release the DDEF for the incorrect data set. The previous DDEF is canceled and the operator is instructed to remove your private disk. The unit on which it was mounted is now available for other use.

```
S,Y: DDEF IN17DD,,OTHER17
```

You define another data set that is on a public disk. No waiting is necessary since public volumes remain mounted while the system is operational.

```
S,Y: MAIN17
```

You again load your program. This time the DCB for DDNAME=IN17DD is filled in for the data set named OTHER17.

```
PGM: AVERAGE I.Q. OF COMPUTER PROGRAMMERS IS:  198.6
```

Your program prints correct results, and returns control to the terminal.

```
S,Y: LOGOFF
SYS:
```

LOGOFF is accepted by the system.

Example 18: Sharing Data Sets

This example shows how data sets can be shared by several users of the system. Part 1 shows a session during which another user makes one of his data sets available to you. Part 2, shows how you copy the data set so that you can make changes to it.

```
Part 1: Permitting Access to a Data Set
USR: (presses attention button or dials up the system)
     LOGON ABPALID,PASSME,,ABACCT2
```

Another user's LOGON is shown here. His identification is ABPALID.

```
SYS:
```

The system will complete the LOGON procedure and invite the user to begin his task.

```
S,Y: PERMIT DATA,ADUSERID,RO
```

Positional operand notation is used here. He makes available to you all of his cataloged data sets whose left-most name qualifier is DATA. Thus you may share his data sets below this level of index in the catalog, such as those named DATA.Q1, DATA.Q2, DATA.Q1.AA, DATA.Q1.AB, etc. He could have made only a certain data set available by specifying its fully qualified name, e.g., DATA.Q5.

Read-only (RO) means that you may only read the specified data sets. He could have permitted read/write (RW), or unlimited (U) access. The latter would allow you to erase the data set. Notice that these access types differ from those of the CATALOG command.

```
S,Y: PERMIT INFO,ADUKU,R
```

With this command the user withdraws the access (R means restrict) to his INFO data set that he has previously permitted to the user whose identification is ADUKU.

```
S,Y: LOGOFF
SYS:
```

LOGOFF is accepted by the system.

The LOGOFF command has caused this user's (ABPALID) catalog entry for the specified group of data sets to be marked so that you can access them by issuing the proper SHARE command. An ABEND command could also have been used to update his catalog.

```
Part 2: Accessing a Shared Data Set
YOU: (press attention button or dial up the system)
     LOGON ADUSERID,MYPASS*,,ADACCT29
```

Now you log on to share ABPALID's data set.

```
SYS:
```

The system will complete the LOGON procedure and invite you to begin your task.

```
S,Y: SHARE MYDATA,ABPALID,DATA
```

Positional operand notation is used. For each data set the owner has cataloged with the left-most qualifier "DATA," an entry will be created in your catalog under the left-most qualifier "MYDATA." This command is rejected if the owner has not granted access to you with the PERMIT command.

Since you have been permitted read-only-access, you must make your own copy of the data set MYDATA.Q1.AA before you can modify it. Before issuing the CDS command, you may define the data set into which the contents of MYDATA.Q1.AA will be copied. However, if you do not define it, CDS will do it for you.

S,Y: CDS DSNAME1=MYDATA.Q1.AA,DSNAME2=MYOWND.Q1.AA

To refer to a specific data set to which you have been permitted access, you must append to the partially-qualified name you have assigned it, the same rightmost name that the owner has assigned it; in this case Q1.AA.

The name you assign to the new data set (MYOWND.Q1.AA) makes it easy to relate it to the original data set; you could have assigned any name.

You still have read-only access to the original data set (MYDATA.Q1.AA in your catalog).

S,Y: LOGOFF

The SHARE command caused entries to be created in your catalog for the group of data sets whose left-most qualifier is MYDATA. They point to the owner's data sets.

You must remember that if the owner erases or deletes one of his data sets which you share, its entry in *your* catalog is not removed. You would then use the DELETE command to update your catalog.

You should also note that the ERASE command for a shared data set is disregarded if there are any active users for that data set. Conversationally, a diagnostic will be issued, to alert you to system action.

SYS:

LOGOFF is accepted by the system.

Example 19: Switching Between Terminal and Card Reader for Input

If a card reader is available at your terminal, it can be used for entering commands and data.

In this example, you switch the SYSIN of your conversational task from keyboard to card reader. You have prepared a deck of cards that contains part of your command stream, your source program, and input data.

Your card deck is shown below. It includes a source program similar to the one you used in Examples 4 and 12.

```
ASM ATMS2,N                                                        LOGON and
                                                                   assembler
PST19    PSECT                                                     parameters
         ENTRY  BEGIN19
         DC     F'76'
         DC     18F'0'
AREA     DC     F'0'                    READ/WRITE AREA
CST19    CSECT
BEGIN19  BASR   11,0
         USING  *,11                    LOCAL BASE REG
         L      13,72(0,13)
         USING  PST19,13
HERE     EQU    *
         GATRD  AREA+3,LENGTH           READ FROM SYSIN
         CLI    AREA+3,C'E'             CHECK IF END
         BE     LEAVE                   BRANCH IF YES
         MVZ    AREA+3(1),=X'00'        CONVERT TO BINARY          Source
         L      5,AREA                                             Statements
         SLA    5,1                     MULT BY 2
*  RESTRICTED TO INTEGERS FROM 0 TO 4
         ST     5,AREA
         MVZ    AREA+3(1),=X'FF'        CONVERT TO EBCDIC
         GATWR  AREA+3,LENGTH           WRITE ON SYSOUT
         B      HERE
LEAVE    EXIT   'PGM FINISHED'
LENGTH   DC     F'1'                    LENGTH OF AREA
         END
  PRINT  LIST.ATMS2(0),,,EDIT,ERASE
  KB     Return to keyboard
  ATMS2
2        Input
1        data for
3        program ATMS2
E        End of data
KB
```

After completing the LOGON procedure, you place your card deck in the card reader and begin your task. The cards must be cut in the upper left corner.

YOU: CB

You switch SYSIN to the card reader. You can do this anytime the system is waiting for keyboard input if the desired cards are ready in the card reader.

Since the terminal is in send-receive mode, each card image is printed on the terminal as if it had been typed in at the keyboard.

S,C: ASM ATMS2,N

Positional operand notation is used. The system prompts for source statements by issuing a line number. It then reads the statement from the card reader.

```
S,C: 0000100PST19     PSECT
S,C: 0000200          ENTRY   BEGIN19
S,C: 0000300          DC      F'76'
S,C: 0000400          DC      18F'0'
S,C: 0000500AREA      DC      F'0'                 READ/WRITE AREA
S,C: 0000600CST19     CSECT
S,C: 0000700BEGIN19   BASR    11,0
S,C: 0000800          USING   *,11                 LOCAL BASE REGISTER
     (rest of source program)
                 .
                 .
                 .
S,C: 0002200LENGTH    DC      F'1'                 LENGTH OF AREA
S,C: 0002300          END
```

SYS:

The source input is scanned and you are notified that your assembly has been completed.

S,C: PRINT LIST.ATMS2,,,EDIT,Y
SYS:

Your program has been assembled without error. The printing of its listings has been assigned to a separate nonconversational task with a unique BSN. You decide to erase the data set as soon as the printing is completed.

S,C: KB

This card causes SYSIN to switch to the keyboard, and the system then prompts with the underscore. SYSIN is also transferred to the keyboard when you press the attention button.

S,Y: CANCEL 0137

You decide that you don't want your listing data set printed now, so you cancel the print request. BSN 0137 is assumed here.

S,Y: CB

You can enter CB to switch SYSIN to the card reader at any time. Your program reads data from the card reader and prints results (with GATWR) on SYSOUT.

S,C: ATMS2

Execution of your program has reached the GATRD macro instruction. The system prompts you for SYSIN data, which you have included in the card deck.

CIP: 2
PGM: 4
CIP: 1
PGM: 2
CIP: 3
PGM: 6

CIP: E

END of data for program ATMS2.

S,C: KB

This card causes SYSIN to be switched to the keyboard, and the system then prompts with an underscore.

S,Y: ERASE SOURCE.ATMS2

The data set named SOURCE.ATMS2 will have been automatically cataloged by the system. You have no further need of it, so you erase it.

S,Y: LOGOFF
SYS:

LOGOFF is accepted by the system.

Example 20: Anticipating an Interrupt in a Nonconversational Task

In this example, you create a nonconversational task on cards. To prevent your task from being abnormally ended if a fixed-point overflow occurs during the running of your program, you include in the program an interrupt handling routine. Your program uses the SPEC, SIR and DIR macro instructions. Refer to Appendix D for more details.

The SYSIN for your task consists of the cards below.

```
  LOGON ADUSERID,,,ADACCT29
```

A nonconversational task is initiated with a LOGON command that must include the parameters on the same card. The password is not used.

This command must begin in column 3; columns 1 and 2 must be blank.

```
SECURE (DA=1,2311)
```

All devices required for private volumes in a nonconversational task must be specified by a SECURE command. It must immediately follow the LOGON command.

You specify that your task requires one 2311 disk unit.

```
DDEF JOBLDD,,JOBL,UNIT=(DA,2311),VOLUME=(,202020),OPTION=JOBLIB
```

This command will define your job library for this session. It causes the operator to mount your disk (volume serial number 202020) on the device secured in the preceding command. If you do not issue it, your task will be terminated when it reaches this point.

```
EDIT SOURCE.MYPGM
```

You specify that a data set is to be constructed from the cards that follow, and is to be named MYPGM. You plan to assemble from it later in your task.

Your program computes $\Sigma (A_i + B_i)$ where i=10. If a fixed-point overflow occurs, Σ is set to zero.

Your program includes the following source statements.

```
PSTMYP  PSECT
        ENTRY   MYSTRT
        ENTRY   SIGMA
        DC      F'76'                         LENGTH SAVE AREA
        DC      18F'0'                        REMAINDER OF SAVE AREA
DELTA   SPEC    EP=SIGMA,COMAREA=RHO,INTTYP=IF
RHO     DS      4F                            INTERRUPT COMMUNICATION AREA
ALPHA   DS      10F                           ALPHA VALUES
BETA    DS      10F                           BETA VALUES
CSTMYP  CSECT
MYSTRT  SAVE    (14,12)                       SAVE REGISTERS IN CALLER'S SV AREA
        L       14,72(0,13)                   GET RCON FROM CALLER'S SAVE AREA
        ST      14,8(0,13)                    STORE FORWARD POINTER
        ST      13,4(0,14)                    STORE BACKWARD POINTER
        LR      13,14                         SET PSECT AND SAVE AREA REGISTER
        USING   PSTMYP,13                     PSECT COVER REGISTER
        LR      12,15
        USING   MYSTRT,12                     CSECT COVER REGISTER
          .
          .
          .
        SIR     DELTA,127                     SPECIFY INTERRUPT ROUTINE
        LA      8,4                           INCREMENT BY 4
        LA      9,40                          SET FOR 10 SUMMATIONS
        SR      6,6                           INITIALIZE SUMMATION REGISTER
        SR      5,5                           AND BEGINNING DISPLACEMENT
OMEGA   L       4,ALPHA(0,5)                  GET ALPH VALUE
        A       4,BETA(0,5)                   ADD BETA VALUE
        AR      6,4                           SUM PO R6
        BXLE    5,8,OMEGA                     RECYCLE UNTIL FINISHED
```

```
       DIR     DELTA                                      DELETE INTERRUPT ROUTINE
        .
        .
        .
       EXIT    'MYPGM FINISHED'
*INTERRUPT ROUTINE FOR FIXED-POINT OVERFLOW
SIGMA  SAVE (14,12)
       SR      4,4                                        SET REG 4 (SUM) TO ZERO
       LR      3,0                                        SAVE AREA TO REG 3
       ST      4,36(0,3)                                  ZERO IN SAVE AREA'S REG 4
       RETURN  (14,12)                                    RETURN
       END
```

Since you cannot process from the terminal any interrupts which may occur during the execution of your nonconversational task, you include in your program a routine to handle a type of interrupt which may occur. The SPEC macro instruction generates an interrupt control block (ICB), and specifies the entry point of that routine (SIGMA). The name specified on the SPEC macro instruction becomes the name of the ICB to which the SIR and DIR macro instruction will refer.

Control is transferred to the routine beginning at SIGMA when a fixed-point overflow (interrupt type IF) occurs. The routine sets the value of the register containing the overflowed sum to zero.

The SIR macro instruction establishes system references to the interrupt routine you specify in the SPEC macro instruction. When the specified interrupt occurs, your program's registers are stored in a save area to which register zero points. Your interrupt routine must therefore store in this save area any registers for the interrupted routine it wishes to alter. After executing the interrupt routine, your program's registers are restored from the save area, and execution of your program resumes at the next instruction past the interrupt.

The DIR macro instruction disables the specified SIR macro instruction. It is needed only when you want the same type of interrupt to be handled by a routine which specifies a lower priority, or by the system.

Execution of the EXIT macro instruction causes the specified message to be printed on SYSOUT and control to return to the system. The next record will then be read from SYSIN.

```
_END
```

This card will signal the end of your source data set. SOURCE.MYPGM will be automatically cataloged.

```
ASM MYPGM,Y
```

Positional notation is used for ASM operands. This command will activate the assembler and cause the program named in the parameter card to be assembled from the prestored source data set.

You are not asked for modification when running in nonconversational mode.

```
MYPGM
```

This command will cause your newly-assembled program to be run.

```
  LOGOFF
```

The LOGOFF command *must* begin in card column 3.

Example 21: Housekeeping

Periodically you should take inventory of your data sets and dispose of the ones you no longer need. In this example, you delete several old data sets residing on tapes, erase some unneeded data sets residing on public storage, convert a source data set to punched cards, and copy another data set from a public disk to tape.

After completing the LOGON procedure, the system invites you to begin your task.

```
S,Y: PC?
```

This command causes brief catalog entries for all of your data sets to be printed at your terminal. If you had specified a fully qualified data set name as operand, you would be presented with the status of that data set only; if you had specified a partially qualified data set name as operand, each data set possessing the same qualification would be presented.

```
SYS: DATA SETS IN CATALOG WITH QUALIFIER ADUSERID
     ADUSERID.ABLE.1, ACCESS=RW
     ADUSERID.ABLE.2, ACCESS=RO
     ADUSERID.BAKER, ACCESS=RO, OWNER=ROGERG**
       .
       .
       .
     ADUSERID.OUT10, ACCESS=RW
     ADUSERID.SOURCE.AT2, ACCESS=RW
     ADUSERID.SOURCE.AT2EX12, ACCESS=RW
YOU: (press attention button)
```

By pressing the attention button you have terminated the presentation of the catalog. You need more information about two of your data sets and request this with a DSS? command.

```
S,Y: DSS? OUT10

SYS: ADUSERID.OUT10
     VOLUME:101010 (9-TRACK TAPE)
     ORGANIZATION: PS

S,Y: DSS? SOURCE.AT2EX12

SYS: VOLUME: PB8171 (2314)
     ORGANIZATION: VI                    PAGES: 00001
     REFERENCE DATE: 195/70              CHANGE DATE: 195/70
     RECORD FORMAT: V                    RECORD LENGTH: 00132
     KEY LENGTH: 00007                   KEY POSITION: 00004

S,Y: DELETE OUT10
```

You decide to use tape 101010 as a scratch tape, and to ignore the data set named OUT10. So you issue this command to delete the entry in your catalog for the data set.

```
S,Y: PUNCH DSNAME=SOURCE.AT2EX12, STARTNO=9,ENDNO=88,ERASE=Y
```

You decide that your AT2EX12 program is used too infrequently to warrant retention on the system. With this command you cause its source program to be punched on cards which you will retain. By specifying the ninth character of your terminal line as the first character to be punched in column 1 (STARTNO=9) you will cause the suppression of the system-supplied 7-digit line number and of the input indicator in column 8. Your last character (ENDNO=88) will be punched in column 80 on the card. You specify that the data set is to be erased at the conclusion of the punch task.

SYS:

The system informs you that a nonconversational task has been established for your PUNCH request. A unique batch sequence number (BSN) will be given to you at the terminal.

S,Y: ERASE SOURCE.AT2

You decide to also erase another source program which is similar to the one you had punched on cards. It is erased from storage, and its catalog entry is deleted.

S,Y: POD? USERLIB,,Y

Now you request a list of each object module on your USERLIB. You specify that for each module, the listing of any aliases. An alias is an external name other than the module name such as CSECT name or external entry point name. Member oriented data (system and user) is omitted by default.

SYS:

The requested data for each member of USERLIB will be presented at your terminal.

S,Y: ERASE USERLIB(AT2EX12)

You erase your AT2EX12 object program module from your USERLIB and delete its entry from the USERLIB directory. All forms of your AT2EX12 program have been removed from the system. You retain only the source program on the card deck.

S,Y: WT DSNAME=SOURCE.ATIMES,DSNAME2=SVATIMES,VOLUME=999999,STARTNO=9,-
ENDNO=88,ERASE=Y

This command causes the creation of a separate nonconversational task to write your SOURCE.ATIMES data set on one of your tapes (volume number 999999). You specify start and end column numbers so that the tape can be later used to print or punch the data set.

SYS:

The system informs you that a nonconversational task, with a unique BSN, has been established for your WT (write tape) request.

After the data set is written, its public disk storage is erased and the catalog entry for SOURCE.ATIMES is deleted (because of the ERASE parameter). A new catalog entry is created for the data set SVATIMES.

S,Y: USAGE

You request a listing of all resources that have been used by you since you were joined to the system. This will be useful to you for accounting purposes and will show you whether you are currently using more storage than you reasonably need so that you may delete unnecessary data sets. The USAGE command will also list your user limits for each resource as well as the resources you have been using since the current LOGON, a facility you can use in planning a task and in making sure that you will not exceed the number of devices allocated to you.

SYS:

You will receive an accounting for the following resources: permanent storage (in pages used), temporary storage, direct-access devices, magnetic tapes, printers, card reader-punches, bulk input, bulk output, TSS tasks, total time that your terminal was connected to the system, and CPU time used.

S,Y: LOGOFF
SYS:

The system accepts the LOGOFF request.

Example 22: Use of Generation Data Groups

Successive, historically related data sets can be cataloged as a generation data group (GDG)—(for example, similar payroll records that are created every week). They can be stored and accessed by their relative generation number (mostly recently stored is 0, next previous is −1, etc.). Or, each data set (called a generation) in the group can be referred to by its absolute generation name (this is explained in Part 2 of this example).

This example shows the establishment of a GDG and both types of references. It also illustrates the use of private libraries and tapes.

Part 1: Establishing a Generation Data Group and Relative References

You complete the LOGON procedure and begin your task.

`S,Y: DDEF PAYLIBDD,,PAYLIB,OPTION=JOBLIB`

You define for this session your job library data set which is on your own private disk. In it resides your program for establishing a GDG for processing payroll records. Since it is the most recently defined library, the system places it at the top of your program library list.

`SYS:`

You will be notified that your task is waiting for the operator to mount the private volume defined above. The task will proceed when mounting is complete.

`S,Y: DDEF CRNTDD,,CURRENT`

You also define for this session the data set which contains this week's report for your payroll program.

`SYS:`

Your task will wait while the operator is mounting the private volume containing the data set named CURRENT.

`S,Y: CATALOG GDG=PAYROLL,GNO=5,ERASE=Y`

This command causes the establishment of a special entry in your catalog to describe the generation data group named PAYROLL. You specify that a maximum of five data sets (generations) are to be maintained in the GDG. You will be adding generations in chronological order, thus retaining only the most recent five. The oldest generation will be erased when there are more than five.

`S,Y: DDEF WK37DD,VS,PAYROLL(+1)`

You define the new data set (or generation) which your program will write. It will reside on public storage by default. The system will automatically catalog it. A positive relative number (in this case 1) designates a generation about to be created. (0) is used to refer to the most recently created generation, (−1) the generation just prior to the most recent, etc. When a new generation is cataloged, that generation assumes relative number (0) and all other relative generation numbers are decreased by one.

`S,Y: PAYUPDAT`

You cause your payroll updating program to be loaded from your private library, and run. It reads your data set named CURRENT, processes it, and then writes the data set (generation) you defined above. Then it prints a message at the terminal.

`PGM: 'WEEK37' PAYROLL FINISHED.`

A data set which was written during a previous session can be added to this GDG. You would simply use the CATALOG command to change its data set name to PAYROLL (+1).

```
S,Y: LOGOFF
SYS:
```

LOGOFF is then accepted by the system.

Part 2: Absolute Reference

You complete the LOGON procedure and begin your task.

```
S,Y: DDEF PAYLIBDD,,PAYLIB,OPTION=JOBLIB
```

You define the job library.

```
SYS:
```

You will be notified that your task is waiting for the operator to mount the private volume defined above. The task will proceed when mounting is complete.

```
S,Y: DDEF CRNTDD,,CURRENT
```

You define the cataloged data set which contains payroll information for this week.

```
S,Y: DDEF WK37DD,,PAYROLL(0)
```

You also define the data set you wrote and cataloged last week. Your program uses it for input too.

```
S,Y: DDEF WK38DD,VS,PAYROLL(+1)
```

This command defines the data set your program will write for this week, and automatically places the entry in your catalog, as PAYROLL (0).

The data set for last week can now be referred to as PAYROLL (−1).

You may also refer to a data set in a generation data group by its absolute generation number. The system provides a unique name for each generation (by which it is cataloged by appending to the group name the absolute generation number of the form GxxxxVyy, where xxxx is the generation number and yy indicates the version of a particular generation.

```
S,Y: PAYUPDAT
```

Again you run your payroll updating program.

```
PGM: 'WEEK38' PAYROLL FINISHED.
```

And your program prints its message.

The oldest generation in your catalog PAYROLL (−1) has been assigned the absolute generation name PAYROLL.G0001V00, and the current generation PAYROLL (0) has been assigned the name PAYROLL.G0002V00.

You can provide gaps in the sequence of absolute generation numbers by specifying a relative generation number greater than +1 when you define the latest generation. Thus, if you had specified +3 in the DDEF command above (instead of +1), the absolute generation names would be PAYROLL.G0001V00 and PAYROLL.G0004V00. However, the relative numbers would be PAYROLL(−1) and PAYROLL(0), respectively. If you now insert a generation with absolute name PAYROLL.G0003V00, the relative numbers would be adjusted.

Absolute generation names are useful if you want to update a generation with a new version. The system does not automatically create nonzero version numbers. You can do so with the CATALOG command, as shown below.

```
S,Y: DDEF CHNGDD,VS,CHNGWK37
```

You define a cataloged data set which one of your payroll programs will write.

```
S,Y: PAYCHNG
```

This program alters last week's payroll data set and writes a temporary data set.

```
S,Y: CATALOG DSNAME=CHNGWK37,STATE=U,ACC=U,NEWNAME=PAYROLL.G0001V01
```

You specify that your catalog entry is to be updated (U) so that it points to the altered data set, which is identified by a new version number. Since the data set which was replaced is on public storage, it is automatically erased.

```
S,Y: LOGOFF
SYS:
```

LOGOFF is accepted by the system.

Example 23: Creating and Using a User Macro-Library

A number of macro definitions are made available to you with TSS. Examples of these are CALL, SAVE, and RETURN. These macro are defined on a TSS library termed the "System Macro Library." You may, if you wish, create a macro library for your private use, termed a "user macro library." This example describes how you create and use such libraries. Appendix A gives more detailed information on user macro libraries.

Since user macro libraries contain many lines, they are usually created in nonconversational mode, as they are in this example.

```
LOGON ADUSERID,,ADACCT40
```

LOGON and LOGOFF must begin in the third column.

```
EDIT USERSYM,TOTAL,8
```

This command will cause the system to create a REGION data set named USERSYM from the statements that follow it. These statements will define your macros. The first region will be called TOTAL. The keys for each region will be 15 bytes long, position 1 to 8 for the name of the region, positions 9-15 for the line counter.

```
0000100 MACRO
0000200 TOTAL  &NUM,&REG,&AREA
0000300 L      &REG,&NUM(1)
0000400 A      &REG,&NUM(2)
0000500 A      &REG,&NUM(3)
0000600 ST     &REG,&AREA
0000700 MEND
```

The first macro you create is named TOTAL. The macro definition is given between the MACRO and MEND statements.

```
0000800_REGION PREFIX
```

This command describes the next region of your macro library.

```
0000100  MACRO
0000200  PREFIX &PSECT
0000300  L      14,72(0,13)  GET PSECT POINTER
0000400  ST     14,8(0,13)   FORWARD POINTER
0000500  ST     13,4(0,14)   BACKWARD POINTER
0000600  LR     13,14        PSECT BASE REG
0000700  USING  &PSECT,13
0000800  MEND
```

Your second macro is named PREFIX. This macro contains code that is generally executed following a SAVE macro instruction.

```
0009000_REGION TESTD
```

This command describes the next region of your macro library.

```
0000100 TESTD        DSECT
0000200 LENGTH       DS     F      RECORD LENGTH
0000300 LINE         DS     7C     RETR LINE NO
0000400 CODE         DS     X      INPUT CODE
0000500 INDATA       DS     80C    INPUT DATA
```

Your last entry to your macro library is not a macro but a DSECT copy parcel named TESTD. (Although TESTD is not a macro, it will sometimes be considered one in general discussions of your user macro library). The assembler will cause this DSECT to be brought into your program when it encounters your statement:

```
COPY   TESTD
```

```
0000600_END
```

The _END will signal entry of the last line of the USERSYM data set—your macro library.

```
DDEF SOURCE,VI,USERSYM
DDEF INDEX,VS,USERNDX
```

These two commands will define for the current task the macro definition and index data sets. SOURCE and INDEX are data definition names required by the SYSINDEX service routine. They will be automatically cataloged as new data sets, with access qualifier=U.

```
SYSINDEX
```

The SYSINDEX routine will scan the data set containing your macro definitions and copy parcel (USERSYM). It will then create a data set that you named USERNDX when it was defined in the previous command. This data set will contain the names of your two macro definitions and the copy parcel.

```
UNLOAD SYSINDEX
```

As the SYSINDEX routine is no longer needed, you may unload it.

```
ASM TESX,N,(SOURCE,INDEX),LISTDS=Y
```

You will then assemble a program using your newly-defined macro library. (Position operand notation is used for ASM in this example).

Your assembler parameters provide the module name TESX and specify that source lines are not prestored (N). SOURCE and INDEX are the ddnames for the definition and index data sets. In nonconversational mode you must ask for a list data set explicitly.

Note that DDEF commands for SOURCE and INDEX were issued preceding the execution of SYSINDEX, so you need not issue them again. The data set named SOURCE.TESX will be *automatically defined and cataloged* by the system.

```
             MACRO
```

The next 7 source statements show how you may define a macro (named MOVE) within your source statements. This macro would override a macro of the same name defined in your macro library.

```
&NAME        MOVE    &TO,&FROM
&NAME        ST      2,TEMPAREA
             L       2,&FROM
             ST      2,&TO
             L       2,TEMPAREA
             MEND
TESTP        PSECT
             DC      F'76' LENGTH SAVE AREA
TEMP         DC      18F'0' REMAINDER OF SAVE AREA
TEMPAREA     DC      F'0'
MESSAGE      DS      CL80    MSG LOCN
MSGLEN       DC      F'0'    MSG LEN
MSGIN        DC      F'92'
             DC      C'0000100'
             DC      X'00'
             DC      CL80'TEST MOVE EXECUTED'
             ENTRY   TESTE
             COPY    TESTD
```

The five lines in your macro library with the name TESTD will be copied without change into your source program at this point.

```
TEST         CSECT
TESTE        SAVE    (14,12)
             PREFIX TESTP
```

The statements that are normally used following a SAVE macro instruction will be copied here, including a USING TESTP,13 instruction.

```
            USING  TESTD,3
            USING  TESTE,12
            LR     12,15     LOAD COVER REG
            LA     3,LENGTH  COVER DSECT
            MOVE   MSGLEN,LENGTH  DATA LENGTH
```

The macro instruction defined in this assembly will be expanded here.

```
            MVC    MESSAGE,INDATA   MOVE DATA
            GATWR  MESSAGE,MSGLEN   WRITE DATA
            L      13,4(13)  RESTORE CALLING REG
            RETURN (14,12),T
            END    TESTE
PRINT LIST.TESX,,,EDIT,Y
```

Positional operand notation is used here. The system will establish a nonconversational task to print the current generation of LIST.TESX. The data set will be erased after printing is completed.

```
TESX
  LOGOFF
```

You run your program and then LOGOFF. SOURCE.TESX has been automatically cataloged at ASM time. The system issued the necessary DDEF command.

Example 24: Use of the Linkage Editor

In this example, the basic facilities of the linkage editor are shown. Object modules LEMOD1, LEMOD2, and LEMOD3 are assumed to already exist. You desire to first link these three modules that reside in a library on the program library list. You then wish to combine two control sections (CSECT1 and CSECT2) of a fourth module (EYLE1) and add it to the output. This example is divided into two parts, to portray the linkage editor use with both dynamic and prestored control statements.

Part 1: Conversational Linkage Editing: Control Statements Entered from the Terminal Keyboard

You complete the LOGON procedure and begin your task.

Define Libraries: You may then either enter DDEF commands that are required to identify the libraries to be used during this linkage editor run, or, as shown below, retrieve the DDEF commands from a previously cataloged line data set.

```
S,Y: CDD DSNAME=LNKED.DD
SYS:
```

Retrieves the DDEF commands cataloged under the data set name LNKED.DD, and prints each on SYSOUT. One DDEF is used to define the library LKLIB1 used in the LNK command.

```
S,Y: LNK NAME=TS2LNK,STORED=N,LIB=LKLIB1,ISD=N,PMD=Y,LINCR=(500,100)
SYS:
```

Processes the parameters specified. You have told the system that control statements are not prestored, the control statements will begin at line 500, a special library (LKLIB1) has been established for the object module produced, you do not want an ISD, you desire the PMD listing. When the linkage editor is ready for a control statement, you will receive the first line number at your terminal. You enter the content for the line and press the return key. The line is then made available to the linkage editor.

```
S,Y: 0000500INCLUDE (LEMOD1,LEMOD2,LEMOD3)
S,Y: 0000600COMBINE CSECT1,CSECT2
```

As each control statement is received by the linkage editor, it is analyzed for correctness and processed according to the particular functions it specifies. If errors are discovered by the linkage editor, a diagnostic message is typed at the terminal, prompting you to correct the statement in error. The modules LEMOD1, LEMOD2, and LEMOD3 exist in USERLIB or in the libraries named in the DDEF commands containing OPTION= JOBLIB. The CDD command produced these DDEF commands.

```
S,Y: 0000700INCLUDE, (EYLE1)
SYS: 0000700 E***ILLEGAL DELIMITER
S,Y: #
     700,INCLUDE (EYLE1)
```

You correct the statement in error.

```
SYS: #
YOU (press return key)
S,Y: 0000800END
```

You signify that all desired linkage editor control statements have been entered by specifying an END control statement. At this time the linkage editor attempts to resolve any unresolved external references by an automatic search of the libraries on the program library list. It then provides a list, at the terminal, of all finally outstanding unresolved external references, distinguishing those that can be resolved from SYSLIB from those that need resolution from USERLIB or job libraries at execution time.

```
SYS:
```

The linkage editor finds no errors and completes necessary processing. The output module is automatically stored in the library with ddname LKLIB1. The names of the original modules are retained as auxiliary entry points of the link edited module. Linkage editor processing is thus concluded. The system solicits your next command.

```
S,Y: LOGOFF
SYS:
```

The system accepts the LOGOFF request.

Part 2: Conversational Linkage Editing: Control Statements from a Prestored Data Set

This is identical to Part 1, except that the linkage editor control statements are obtained from a prestored data set. Thus, correction lines are treated in a slightly different manner.

.
.
.

Statements as shown in Part 1.

.
.
.

```
S,Y: LNK TSZLNK,Y,LKLIB1,,N,Y,(500,100)
SYS:
```

Processes the parameters specified. Here, however, you have indicated that control statements are prestored.

.
.
.

```
SYS: 0000700 E***ILLEGAL DELIMITER
```

The diagnostic message appears at the keyboard.

The keyboard is unlocked so that you may make a correction.

```
S,Y: #
     700,INCLUDE (EYLE1)
```

You enter the correction line.

```
SYS: #
YOU: (press return key)
```

Statements as shown in Part 1.

.
.
.

Example 25: Tape and Disk-Medium Transfers of Virtual Access Method Data Sets

In this example, three commands provided for manipulation of VAM data sets are presented. They are TV (TAPE to VAM), VT (VAM to TAPE), and VV (VAM to VAM). The data sets *to be copied* are assumed to exist, and are cataloged.

You complete the LOGON procedure and begin your task.

```
S,Y: DDEF DDVTOUT,PS,COPY1,UNIT=(TA,9),VOLUME=(PRIVATE)
S,Y: VT DSNAME1=ORIGIN1,DSNAME2=COPY1
```

Data set ORIGIN1 already exists as a VAM data set. COPY1 is the name assigned to the magnetic tape copy of this data set. The installation default for LABEL is assumed.

```
SYS:
```

When the data set is successfully copied, you will receive a message indicating the names of the input and output data sets, as well as the file sequence numbers and volume serial numbers used.

```
S,Y: RELEASE DDVTOUT
```

You wish to copy another data set (ORIGIN2) onto another tape. You therefore release DDVTOUT and issue command again.

```
S,Y: DDEF DDVTOUT,PS,COPY2,UNIT=(TA,9),VOLUME=(PRIVATE)
S,Y: VT ORIGIN2
```

Here the output data set name is not given. The output data set name will become ADUSERID.TA000001.ORIGIN2, where TA000001 is an arbitrary number to assure uniqueness for the fully-qualified data set name.

```
SYS:
```

The system will signify a successful copy. Any failure to copy successfully will result in a diagnostic message and cancellation of the command.

```
S,Y: DDEF DDCOPYB,VS,COPYBACK
S,Y: TV DSNAME1=ADUSERID.TA000001.ORIGIN2,DSNAME2=COPYBACK
SYS:
```

The data set just produced on a 9-track magnetic tape is copied on direct access storage (public) in VAM format. An appropriate system message will be issued to signify whether or not the copy attempt was successful. It is assumed that ORIGIN2 was a virtual sequential data set. COPYBACK is thus defined as having VS organization.

```
S,Y: DDEF DDCOPY3,VI,COPY3
S,Y: VV DSNAME1=ORIGIN3,DSNAME2=COPY3
```

The data set named ORIGIN3 is copied into public storage, assigning the name COPY3. It is assumed that ORIGIN3 has virtual index sequential organization. Therefore COPY3 is so defined.

```
SYS:
```

An appropriate system message will appear, signifying the success or failure of the copy operation.

```
S,Y: DDEF PRIVDD,VI,COPY4,UNIT=(DA,2311),VOLUME=(,333333)
S,Y: VV DSNAME1=ORIGIN4,DSNAME2=COPY4
```

You desire to copy the data set ORIGIN4 onto a private VAM volume #333333 and name the output data set COPY4.

```
SYS:
```

An appropriate message will appear, signifying the success or failure of the copy operation.

```
S,Y: LOGOFF
SYS:
```

Your LOGOFF request is accepted by the system. Your new data sets were automatically cataloged.

Example 26: The Text Editor Facility

In this example, the basic use of the Text Editor facility is illustrated. One of the most important applications of this facility is to create and edit data sets.

You complete the LOGON procedure and begin your task.

```
S,Y: EDIT DSNAME=EX26
```

You invoked the Text Editor with this command. A DDEF command is not required unless you are creating a new data set with a format differing from your default values. You will be prompted with line numbers to enter text.

```
S,Y: 0000100 DEM01
S,Y: 0000200 DEM02
S,Y: 0000300 DEM03
```

You enter data lines you wish to be part of the data set named EX26. Each time you press the return key, the Text Editor prompts with the next line number.

```
S,Y: 0000400 _UPDATE
```

You decide to make a change to the previous entries. By preceding UPDATE with an underscore, known as a break character (_), the Text Editor immediately executes the command.

```
SYS:
```

The system will issue a message prompting for line number and data.

```
YOU: 0000150 INSERT1
```

You add line number 150 to your data set.

```
YOU: _INSERT 0000400
```

You now want to continue entering data at the point where you left off earlier. INSERT is preceded by a break character, since the system expects data and not a command following UPDATE.

```
S,Y: 0000400 DEM04
S,Y: 0000500 DEM05
S,Y: 0000600 _END
```

You terminate Text Editor processing.

```
S,Y: EDIT DSNAME=EX26
```

You reinitiate editing on the same data set.

```
S,Y: EXCISE N1=0000200
```

Line number 200 of the data set will be deleted.

```
S,Y: INSERT 260,10
```

You wish to insert additional lines, starting with line 260 and proceeding in increments of 10.

```
S,Y: 0000260 INSERT2
S,Y: 0000270 INSERT3
S,Y: 0000280 _END
```

Text Editor processing is terminated.

```
S,Y: LOGOFF
SYS:
```

You decide to terminate your conversational task. The system accepts the LOGOFF request.

Example 27: The Text Editor Facility

In this example, the Text Editor is shown using most of the updating capabilities of the facility. It is probably much more complex than you might wish for a single terminal session, but attempts to portray the flexibility of the commands available.

After completing the LOGON procedure, you begin your task.

```
S,Y: EDIT DSNAME=EX27,RNAME=REGION2,REGSIZE=8
```

You invoke the Text Editor. A DDEF command is not required. Because you wish to produce a region data set, you define a region name for EX27 and assign a region name size.

```
S,Y: 0000100 LINEA
S,Y: 0000200 LINEB
S,Y: 0000300 LINEC
S,Y: 0000400 LINED
S,Y: 0000500 LINEE
S,Y: 0000600 LINEF
S,Y: 0000700 LINEG
S,Y: 0000800 LINEH
S,Y: 0000900 LINEI
S,Y: 0001000 LINEJ
S,Y: 0001100 _END
```

You enter the lines you wish to constitute REGION2 of data set EX27. You then terminate Text Editor processing.

```
S,Y: DEFAULT TRANTAB=Y
```

You wish to use the ENABLE, DISABLE, POST, or STET commands in editing your data set. Since no transaction table is normally kept (TRANTAB=N), you must reset the default to Y.

```
S,Y: EDIT DSNAME=EX26
```

You invoke the Text Editor for data set EX26 which you created in the previous example.

```
S,Y: NUMBER N1=300,N2=500,NBASE=300,INCR=50
S,Y: DISABLE
S,Y: EXCERPT DSNAME=EX27,RNAME=REGION2,N1=600,N2=1000
```

These lines will be inserted in the current data set, EX26.

```
S,Y: CONTEXT N1=300,N2=500,STRING1=DEMO,STRING2=XXXX
```

The data set is searched for the character string DEMO for occurrence in lines 300 to 500 only. Wherever it is found, XXXX will replace the occurrence.

NOTE: This facility is useful for symbol replacement in source language data sets.

```
S,Y: ENABLE
```

Up to this point, the revisions made since DISABLE was issued above were temporary. These revisions are now permanent, with the ENABLE command execution.

```
S,Y: CORRECT N1=100,SCOL=0
```

Standard correction characters are assumed, by default.

```
SYS: DEMO1
YOU: *   %
```

The result will be DEMO.

```
S,Y: POST
```

With this command you make permanent all editing commands issued for the current data set.

```
S,Y: END
```

You terminate Text Editor processing of EX26.

```
S,Y: EDIT EX27,RNAME=REGION2
```

You initiate Text Editor processing of EX27. The current region is now REGION2.

```
S,Y: LOCATE STRING=LINEF
```

The entire data set EX27 is searched for the character string LINEF.

```
SYS:
```

The line in which LINEF is *first* discovered is displayed at the terminal.

```
S,Y: LIST N1=100,N2=500,CHAR=H
```

The first five lines of the current region (REGION2) will be displayed in hexademical notation.

```
S,Y: END
```

Text Editor processing is terminated.

```
S,Y: LOGOFF
SYS:
```

The LOGOFF is accepted by the system.

Example 28: Use of Procedure Definition (PROCDEF)

In this example, you are shown how to create a procedure, tailored to your needs, to be called at a later time just as if it were a system-supplied command. You have also decided to change the system command prompt string for this terminal session by invoking the MCAST command.

You complete the LOGON procedure and enter your first command.

```
S,Y: MCAST CP=**:
```

The initial default for the system command prompt is an underscore and backspace. You decide to change this prompt to a pair of asterisks with no carriage return. Thus you issue the MCAST command with the CP (Command Prompt) parameter.

```
S,Y: **PROCDEF  NAME=ASMPGM
S,Y: 0000100  PARAM  MODULE
S,Y: 0000200  ASM  MODULE
S,Y: 0000300  PRINT  LIST.MODULE,,,EDIT,Y
S,Y: 0000300 _END
```

This procedure will now be available for calling using the name ASMPGM. It allows you to define a module name for assembly. By calling the established procedure, and giving it a unique module name to use, both the assembly and printing of the resulting listing data sets can be accomplished.

```
S,Y: **ASMPGM  MYMOD
```

The procedure established above (via PROCDEF) will now be activated. The actual module name (MYMOD) will replace the dummy module name (MODULE) wherever it occurs.

```
S,Y: **PROCDEF  NAME=SETUP
S,Y: 0000100 PARAM STORED=$1, ISD=$2, SYMLIST=$3, CRLIST=$4, MACROLIB=$5
S,Y: 0000200 DEFAULT STORED=$1, ISD=$2, SYMLIST=$3, CRLIST=$4, MACROLIB=$5
S,Y: 0000300 _END
```

This procedure will now be available to vary the default values for certain ASM parameters.

```
S,Y: **SETUP Y,N,Y,Y,(SRC,NDX)
```

Some ASM parameter default values have been adjusted to suit your requirements.

```
S,Y: **DDEF SRC,VI,MACSRC; DDEF NDX,VS,MACRNDX
```

MACSRC and MACRNDX are the DSNAMES for the symbolic and index portions of a macro library.

```
S,Y: **ASM MOD1
```

You now proceed with the assembly of MOD1 with the adjusted default values.

```
S,Y: **PROCDEF ZLOGON
```

Here the operand (ZLOGON) for the PROCDEF command is shown without the use of a keyword.

```
S,Y: 0000100 DDEF STOREIT,VP,MYLIB,OPTION=JOBLIB
S,Y: 0000200 _END
```

You decide that each time you LOGON you would like a certain job library defined for any object modules you may produce. By assigning ZLOGON as the procedure name, you insure its automatic call as soon as LOGON is accepted. MYLIB is assumed to be an existing, cataloged data set. Since PARAM was *not* used you *cannot change* any of the values in the DDEF command.

```
S,Y: **LOGOFF
SYS:
```

The LOGOFF command is accepted by the system.

Example 29: Use of the BUILTIN Procedure

In this example, you are shown the use of BUILTIN, as a user-defined procedure. This facility allows you to invoke an object program just as if it were a system-supplied command. You choose a program (already containing the BPKD macro in its PSECT) which causes a data set to be created. You will later invoke the KEYWORD command to obtain a listing of all your user-created commands existing in USERLIB.

You complete the LOGON procedure and begin your task. The program which you will later invoke, includes the following source statements.

```
PST29   PSECT
        ENTRY STRT29
          .
          .
          .
AREA    DS    80C                               DATA AREA
DCBNM   DCB   DDNAME=OUTDD,RECFM=FA
USER29  BPKD  STRT29
CST29   CSECT
STRT29  EQU *
          .
          .
          .
        LA    2,20                              SET FOR 20 CYCLES
        OPEN  (DCBNM,(OUTPUT))                  OPEN DCB
LABEL   EQU *
        (create record at AREA)
          .
          .
          .
PUT     DCBNM,AREA                              PUT RECORD IN DATA SET
BCT     2,LABEL                                 RECYCLE
CLOSE   (DCBNM)                                 CLOSE DCB
        EXIT
        END
```

Assuming that the above module was assembled without specifying a job library, your USERLIB will contain it. The sequence to follow indicates how you may invoke the module which creates a data set containing 80 character records, via BUILTIN.

S,Y: BUILTIN NAME=DOPROG,EXTNAME=USER29

The object program definition via your user-created command (DOPROG) is now established.

S,Y: DOPROG

You decide to invoke DOPROG. The program shown earlier will now be retrieved from your USERLIB, and executed beginning at entry point STRT29. Control will return to your terminal at EXIT time.

S,Y: KEYWORD

You request a listing of all the command names in your USERLIB.

SYS:

(The command names are printed at your terminal, one command string per line. Any associated parameters will be printed in the same format as they appear in the PROCDEF or BUILTIN commands.)

S,Y: LOGOFF

SYS:

The LOGOFF command is accepted by the system.

Example 30: The User Profile Facility

In this example, you are shown how to manipulate your copy of the prototype user profile, made available to you at JOIN time. This prototype profile is a member of your user library (USERLIB).

You complete the LOGON procedure and enter your first command.

```
S,Y: DEFAULT DSORG=VS
```

The data set organization field was originally defaulted by the system to VI (indexed sequential). You will now be using mostly VS organized data sets, so you set the default value (for the DDEF command) to virtual sequential (VS).

```
S,Y: DEFAULT DEPROMPT=YES
```

At some previous time, the value of DEPROMPT had been set to "no." For future use, you decide that all partially qualified names entered for either the ERASE or DELETE commands should be audited. The value of "yes" will cause individual data set names to be presented.

```
S,Y: SYNONYM DOPROG=PRINTDS
```

The BUILTIN procedure named in Example 29 can now be invoked with either name: DOPROG or PRINTDS.

```
S,Y: SYNONYM FINIS=DISPLAY 'TASK COMPLETED'
```

When FINIS is invoked, the message: TASK COMPLETED will appear on SYSOUT.

```
S,Y: PROFILE
```

You decide to make the changes applied to your session profile by the SYNONYM and DEFAULT commands, a permanent part of your user profile.

```
S,Y: FINIS
SYS: TASK COMPLETED
```

The command FINIS was established earlier in the session, using the SYNONYM command. Since the PROFILE command was later invoked, FINIS may be used in subsequent sessions to produce the same message.

```
S,Y: LOGOFF
SYS:
```

The LOGOFF is accepted by the system.

Example 31: Use of the OBEY Macro

The OBEY macro instruction allows the user to execute a command or command statement even though not in command mode. Upon execution of the OBEY macro instruction, the command or command statement specified via the macro instruction operands is executed; control is then returned to the user's program. OBEY may be used anywhere in the user's program.

```
CST5   CSECT
       ENTRY STRT5
STRT5  EQU   *
       OBEY 'DDEF OUTDD,,OUTDS'
         .
         .
```

```
         .
       LA     2,20                                  SET FOR 20 CYCLE
       OPEN   (DCBNM,(OUTPUT))                      OPEN DCB
LABEL  EQU    *
       (create record at AREA)
         .
         .
         .
       PUT    DCBNM,AREA                            PUT RECORD IN DATA SET
       BCT    2,LABEL                               RECYCLE
       CLOSE  (DCBNM)                               CLOSE DCB
       EXIT
AREA   DS     80C                                   DATA AREA
DCBNM  DCB    DDNAME=OUTDD,RECFM=FA,DSORG=VS
       END
```

Your program will write a data set with 80-character records from the storage area named AREA. Notice that your DCB macro instruction includes the DDNAME that is a parameter in the OBEY of the DDEF command, which in turn contains the name of the data set (OUTDS). The DDEF command relates the correct data set to your program because every data set name must be unique in your task.

Part IV. Appendixes

The Appendixes in this publication give detailed information on the use of TSS by assembler language programmers.

Appendix A, "Use of the TSS Assembler," describes the format of assembler statements, correction techniques, diagnostic actions, assembler parameters, assembler output, assembler restrictions, and user macro libraries.

Appendix B, "Problem Program Checkout and Modification," considers the use of the Program Control System (PCS). Prompting and diagnostic facilities, program listings, and use of the Linkage Editor are also discussed. Concerning PCS, only certain aspects are covered—in particular, diagnostic action. *Command System User's Guide* is the primary reference for the use of the Program Control System.

Appendix C presents assembler language programming considerations. The initial sections of this appendix describe programming techniques and sample programs that allow the programmer to write programs with a minimum of effort. Later sections of this appendix discuss more complex programming considerations.

Appendix D discusses interrupt considerations, including use of the terminal attention key and the various macro instructions provided with TSS/360 for control of interrupts. The publication *Assembler User's Macro Instructions* gives more detailed information on interrupt-handling macro instructions.

Appendix E is a guide to the use of DDEF command.

Appendix F describes various user-defined procedures available. Representative examples are given. Procedure Definition (PROCDEF), user's own code procedures (BUILTIN), and the User Profile Facility are portrayed.

The commands available in TSS are described in the examples given in Part III and are presented in detail in *Command System User's Guide*.

Appendix A. Use of the TSS Assembler

Problem-Program Preparation

In Time Sharing System, a problem program is the collection of instructions and data that the user specifies for the solution of some well-defined problem. The term problem program thus differentiates user-written application programs from system programs.

Problem programs may be in the system as source programs or object programs. A source program is in the symbolic form in which it was written by a user. It consists of a series of statements coded in one of the source languages available in TSS (FORTRAN IV, PL/I or assembler). An object program, in hexadecimal code or machine language form, is relocatable and can be loaded and executed by the system.

Language Processing

The operation that converts a source program to an object program, called language processing, is illustrated in Figure 14. Note the terms source program data set and object program module. A source program data set is the collection of all source statements submitted for processing during any single assembly or compilation. An object program module is the principal output of a single assembly or compilation. A source program data set and its corresponding object program module may represent all or part of the actual program required to solve a user's problem. A user can thus design his problem program in sections and, separately, assemble or compile each section. He can then, if he has supplied the proper symbolic linkages between sections, use the system to combine various sections prior to, or during, program execution.

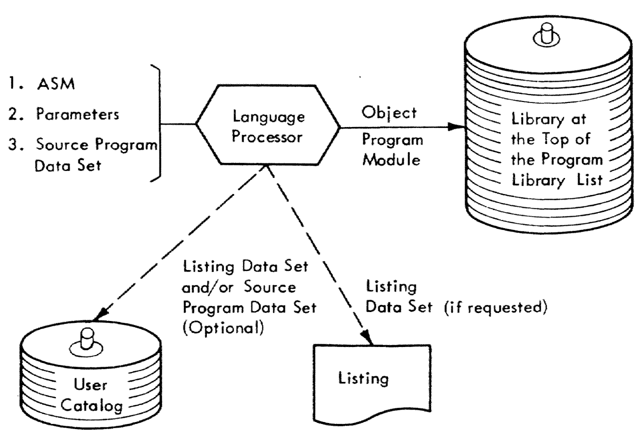


Figure 14. Language Processing

Language Processing in Conversational Mode

To initiate assembler language processing in conversational mode (see Figure 14), the user issues an ASM command with the desired parameters. He must enter them all at the same time as the system will not prompt for individual parameters. The parameters to be entered, listed below, are dependenet upon whether the source program is prestored and on options selected by the user.

- Module name of the object program module being created: The source data set for the object module will be named SOURCE.module; the associated listing data set will be named LIST. module.
- An indication of whether the source program data set is prestored or is to be made available via SYSIN: If the source program is made available via SYSIN, the user can also specify its starting line number and the value by which the line numbers are to be incremented (values of 100 and 100 are assumed, if the starting line number and increment value are not specified).
- Version identification of the module: This consists of one to eight user-supplied alphameric characters, the first of which need not be alphabetic. If a version identification is not supplied, the listing is time stamped.
- The ddnames of the symbolic and index portions of the user-written macro libraries to be used in addition to the system macro library: If this parameter is omitted, only the system macro library is assumed.
- An indication of whether these options are wanted:
  Internal symbol dictionary
  Source data set listing
  Object program module listing
  Cross reference listing
  Edited symbol table
  Internal symbol dictionary listing
  Program module dictionary listing

When these inputs are provided, language processing of the source program begins. The user can issue source program statements from his terminal, in response to system prompting; or he can make the source statements available from a prestored data set. When using a continuation character in statements extending beyond 80 characters, he must observe the continuation conventions of the source language. An END statement is always included in the source program data set to indicate the end of the input to the language processor.

Prompting and diagnostic facilities are available during language processing. These facilities vary with the way the source program data set is presented to the system (as part of SYSIN or as a prestored data set); they are described under "Problem Program Checkout and Modification," Appendix B.

At the conclusion of language processing, the system stores the object program module, by its module name, in the user's library or in his most recently defined job library, if there is one. This completes source language processing in the conversational mode.

Language Processing in Nonconversational Mode

The same commands are used to initiate nonconversational language processing as in conversational processing; the same outputs can be produced. The user must store the parameters required by the language processor in the SYSIN data set immediately after the ASM command. However, the user has the option of making the source program module available in the SYSIN data set or as a prestored data set.

Entry and Correction of Assembler Source Statements

This section discusses the format of assembler source statements entered at the terminal keyboard, the terminal card reader, and in card form for nonconversational mode processing. If conversational mode, correction of source statements is frequently performed by insertion or replacement. Since assembly speed can be influenced by the manner of making such corrections, guidelines for efficient correction techniques are given. A discussion of techniques for entering keyboard lines so that they can be punched and reentered in card form is also included.

Format of Source Lines

Input Sources

A source program is a sequence of source statements that have either been punched into cards and entered by card reader, typed at the keyboard of a remote terminal device, or both. Individual source statements may also contain card lines, keyboard lines, or both. Source statements formats differ between the two sources. The card format is identical to that used in other assembler languages. The keyboard format has been designed for ease of operation at typewriter-like terminal devices. A name field, comment line, or continued line must begin *immediately* following the line number (no space). If the line is not one of the above, *one or more blanks* must follow the line number.

Statement Boundaries—Card Format

Source statements are normally contained in columns 1-71 of initial cards and columns 16-71 of any continuation cards. Free form may be used, however. Therefore, columns 1, 71, and 16 are referred to as the "begin," "end," and "continue" columns, respectively. This convention can be altered by use of the input format control (ICTL) assembler instruction. The continuation character, if used, always immediately follows the "end" column.

Continuation Lines—Card Format

When necessary to continue a statement on another card, the following rules apply:

1. Enter a continuation character (not blank and not part of the statement coding) in column 72 of the initial card.
2. Continue the statement on the next card, starting in column 16. All columns to the left of column 16 are ignored.
3. When more than one card is needed, each card to be continued must have a character (not blank and not part of the statement coding) punched in column 72.
4. Not more than two continuation cards can be used for a statement, except in a macro instruction or macro prototype statement, where as many continuation cards may be used as are necessary.

NOTE: When the MODIFY command is used to alter existing source statements, the continuation column (72) is displaced to the right by as many positions as the line number and the required comma occupy. For example, if the following:

```
DC      F'70'     RESERVE A 19 WORD SAVE  -
                  AREA FOR CALLING PGM
```

represented two source statements, entered originally on cards, and the hyphen (-) in the first line appeared in column 72, alterations by the MODIFY command will cause adjustment. The user notices that '70' is not the correct value and must change the source entry. Assuming the line number was 400, the entry for MODIFY would be as follows: 400, DC F'76' RESERVE A 19 WORD SAVE -. The new source entry would now be offset four positions to the right, relative to the original entry.

Character Sets—Card Format

CA and CB can be used to specify the character set used during 1056 card reader input. With CA, the user indicates he wishes to convert card input from 1057 card punch code to EBCDIC. With CB, the user specifies conversion from 029 keypunch code to EBCDIC.

Statement Boundaries—Keyboard Format

When entered from a terminal, source statements occupy the statement area of each keyboard line. The statement area is that portion of the line between the column at which the system releases the keyboard to the user and the right-hand margin setting. This area may contain more or less than 80 characters, depending on the type of keyboard being used.

Many terminal keyboards available with TSS contain both upper and lower case forms of the letters A through Z. The upper case form must be used, except within character constants or comments. However, if the system is in KB mode (the default option), the lower case characters will be translated into their upper case equivalents, thus eliminating many of the previously required shifting operations. A good general practice is to set tab stops and make use of the tab key to separate the various fields of the source statement. This practice provides a simple method for formatting the input program on the terminal paper without excessive manual spacing. When entering keyboard lines, a single depression of the tab key is considered the equivalent of *one* blank. Thus, when reference is made to blanks, tab *or* blank is implied, unless specifically stated otherwise.

Continuation Lines—Keyboard Format

When it is necessary to continue a statement that is being entered from a keyboard, a hyphen (the keyboard continuation character) is typed at the point at which continuation is desired, followed immediately by a carrier return. The statement is continued at the first character of the statement area of the next line.

If a line with an asterisk (*) in column 1 follows a continued line, the * and following columns will be considered a continuation of the preceding line, not as a comment line. For example, if this sequence occurs:

```
LINE NO.     TEXT
0000500      L                  2,-(CR)
0000600*     COMMENT
```

the assembler will combine the two lines as follows:

```
L          2, *COMMENT
```

There is no restriction on the number of continuation lines which may be entered in keyboard format. The only restriction placed on the length of a statement is that imposed by available assembler working storage.

NOTE: ICTL statements in the source program apply only to lines in card format and have no effect on keyboard lines.

Character Sets—Keyboard Format

KA and KB can be used to specify the character set to be used during keyboard input. With KA, the user indicates he wishes to use the full EBCDIC character set during input. With KB, the user specifies that the lower case characters (a-z and ! " ¢) be translated into their upper case equivalents (A-Z and $ # @). When neither is specified, KB is assumed.

Mixed Card and Keyboard Input

Assembler language source statements entered at the terminal may be from the card reader (if one is available), from the keyboard, or from a mixture of the two, without restriction.

The procedure for changing input mode is as follows. The system will expect lines from the keyboard until the one of the commands C, CA, or CB is entered at the keyboard. Once these characters have been entered, input lines are expected from the card reader. If a card containing command K, KA, or KB is encountered, lines are once again expected to be entered from the keyboard. (See *Terminal User's Guide* for SYSIN Device Selection and Data Translation.)

Caution When Changing Card-Origin Statements

Source statements from punched cards may later be changed, using various commands of the TSS Text Editor (the Text Editor commands are described in *Command System User's Guide*).

On assembly, each source statement of punched card origin is treated as an 80-character record. Where the statement has been shortened to fewer than 80 characters by changing it with a Text Editor command after it has been stored, the assembler, before further processing, pads the statement to 80 characters with trailing blanks. Where the statement has been changed to contain more than 80 characters, the assembler truncates the statement to 80 characters.

Care must be taken in changing a card-origin source statement so that, after padding or truncation by the assembler, the statement will still conform to the coding conventions discussed in this section. (An example might be a statement containing a sequence number in the identification sequence field, columns 73-80. The statement is shortened one character during Text Editing. The assembler pads with one trailing blank in column 80, leaving columns 72-79 containing the sequence number. Since column 72 is normally the continuation column, an error results if the next source statement is not a continuation line.)

Efficient Correction Techniques

Conversational correction of assembler statements is normally made at one of two points in the assembly. The first, called *local* correction, is when the user's keyboard is unlocked for a new or correction line. This occurs following the assembler's scan of the statement just entered and the printing at the terminal of any diagnostic messages associated with that statement. The second point at which corrections are normally made is when the entire program has been entered (i.e., the END line has been entered) and a message soliciting modifications is typed at the terminal. This occurs only when errors have been detected by the system. Corrections made at this point are called *global* corrections. The distinction between local and global corrections and between different types of local corrections is important in that the user can minimize the amount of processing required for a given assembly by being aware of the effect of the correction upon the assembly process. The following paragraphs describe efficient correction techniques in detail.

After all global corrections are made, all the corrected lines are collected and applied to the source data set. The assembler then reinitiates the source scan of each individual statement, beginning with the first source line of the program. When local corrections are made, reinitiation of the source scan may or may not be required, depending upon the type of correction made. Since it is desirable to minimize the number of source scans, corrections that reinitiate the source scan generally should not be made until the time for global corrections is reached. This rule does not apply, of course, where failure to make a correction would result in many other diagnostics, such as an error in defining a symbol.

A simple correction rule that can be assumed in the majority of cases is: correction of a statement *immediately following* entry of that statement *does not* cause reinitiation of the source scan; all other corrections do cause reinitiation of the source scan. Example 1 below demonstrates immediate correction; example 2 demonstrates a correction causing reinitiation of the source scan.

Example 1:

LINE NO. TEXT	NOTES
0000600TABLE1 DC F	Entry of this statement produces a diagnostic.
0000600E *** DATA OMITTED FROM DC OPERAND	
#600,TABLE1 DS F	Immediate correction, entered after the system has typed #, causes the line to be replaced at once. User presses return key after #. User can then proceed without re-scan.
# (press return key)	
0000700 . . .	

Example 2:

LINE NO. TEXT	NOTES
0000600TABLE1 DC F	Entry of this statement produces a diagnostic.
0000600E *** DATA OMITTED FROM DC OPERAND	
# (press return key)	User presses return key after #.
0000700TABLE2 DS F	User ignores error by returning carriage after # has been printed, causing next line to be solicited.
0000800%600,TABLE1 DS F	Since line 700 was error free, line 800 is solicited. The user now decides to correct line 600, using % notation. This correction forces a rescan of the entire source module.
000800 . . .	When the module has been rescanned, the user is again solicited for the next line and can continue.

The position at which the keyboard is unlocked after the system solicits a new statement corresponds to the "begin" column in card format; i.e., a name field, comment line, or continued line must begin immediately following the line number. If the line is not a comment, named line, or continued line, one or more blanks must follow the line number.

If, after the keyboard has been unlocked for a new line, the user wishes to correct a statement, he types a percent sign (%) followed by the number of the statement he is correcting and then the corrected statement. In example 2 above, after line 800 is solicited, the user corrects line 600.

If a diagnostic has been issued for a statement, the system types a number sign (#) and then unlocks the keyboard. The user may then enter the number of the statement he is correcting (which is not necessarily the previous statement) and then the corrected statement. If the user decides at this point that previous errors in his assembly would result in excessive diagnostics and solicitations for corrections, he may respond by typing the letter I and press the return key. This will inhibit all further diagnostic messages from being printed at his keyboard. He may also print the letter C and press the return key; in that case he will continue to receive diagnostics but will not be solicited for corrections. If the error is a minor one, he may ignore the request for correction by pressing the return key. Example 1 above shows how the system continues to solicit corrections until the user ignores the request.

Note that, in a correction line, a comma must be used to separate the line number from the corrected statement.

Discussion of assembler response to more complex local corrections requires a definition of three terms:

1. *Partial Statement:* A partial statement is the statement currently being entered. Statements are partial until a line is entered that is not a continued line. An example of a partial statement is shown below. The first line is a continued line, but the continuation line has not yet been entered: (CR) notes carrier return.

```
ALPHA   DC   CL100'THIS IS AN EXAMPLE-(CR)
```

2. *Tentative Statement:* A tentative statement is the last statement completely processed by the assembler. Thus, while a new statement is being formed (i.e., is partial), the previous statement is defined as being tentative. When a statement has been completely entered and the next line has not yet begun, the statement just entered is termed tentative. An example of a tentative statement is shown below, where the second line is not a continued line.

```
HEXCON   DC   CL16'0123456789-(CR)
              ABCDEF'
```

The above lines are, of course, equivalent to the following single line.

```
HEXCON   DC   CL16'0123456789ABCDEF'(CR)
```

3. *Committed Statement:* The relation between a committed statement and a tentative statement is identical to the relation between a tentative statement and a partial statement; once a statement becomes tentative, the preceding statement becomes committed. In the following example, entrance of the second statement causes the second statement to become tentative and causes the first statement to become committed.

```
GAMMA   CSECT
        USING   *,15
```

The relation between the above types of statements and assembler response to corrections is as follows:

1. Tentative and partial statements can be corrected without causing a reinitiation of the source scan.

LINE NO.	TEXT	NOTES
0000100	MACRO	
0000200&NAME	MACNAME P1,—(CR)	Line 100 is tentative, Line 200 is partial, due to the continuation character.
0000300&P2, &P3,—(CR)		
0000400%200,&NAME	MACNAME &P1,—(CR)	Line 300 is partial due to the continuation character. Line 400 is solicited, but programmer notices an error. Since line 200 is still in partial status, this correction does not cause reinitiation of the source scan.
0000400&P4,&P5		Entry of this statement causes line 100 to become committed and the composite statement beginning at line 200 to become tentative.
0000500. . .		Before the statement beginning with line 500 is completed, corrections can still be made to lines 200 through 400 without causing rescan.

2. Correction of committed statements does cause reinitiation of the source scan.

LINE NO.	TEXT	NOTES
0000200RA	EQU 125	
0000300RB	EQU 13	
0000400	L RA,ALPHA	Causes a diagnostic due to an invalid value for symbol RA.
0000400E	*** R1 VALUE INVALID FOR FIELD	
#200,RA	EQU 12	Correction of a line prior to the committed statement (line 300) causes a reinitiation of the source scan.

3. Insertion of a new statement between a committed and a tentative statement does not cause reinitiation of the source scan of the entire program. For example, insertion of line 650 in the following example requires that line 700 be rescanned, but lines prior to 650 are not rescanned.

LINE NO.	TEXT	NOTES
0000600	SAVE (14,12)	At completion of this statement, line 600 is tentative.
0000700	EX 0,TAB(1)	Line 600 is committed at this point, line 700 is tentative.
0000800%650,	L 1,0(1)	User elects to insert a line at this point.
0000800. . . .		User continues.

The source and object listings (if requested) are not created until the entire program has been entered and all corrections have been made. Thus the conversational terminal may contain many diagnostic messages, but the listing contains diagnostics only for source errors remaining after all corrections have been made.

Entry of Keyboard Source Statements for Later Punching and Recompilation

It may be desirable to punch out source statements entered at the keyboard in order to enter these statements later by card reader; this is possible only if no source statement contains more than one line, because of the different conventions used in determining the initial significant character in continuation lines. Keyboard continuation lines *always* begin in type position 1; card continuation lines begin in column 16 or in a column specified in the ICTL instruction. (It is not permissible to specify that continuation lines begin in column 1 in the ICTL instruction.)

OPERATION	OPERAND
ASM	NAME = module name [,STORED = {Y|N}] [,MACROLIB = ({data definition name of symbolic portion, data definition name of index portion} [, , . .])] [,VERID = version identification] [,ISD = {Y|N}] [,SYMLIST = {Y|N}] [,ASMLIST = {Y|N}] [,CRLIST = {Y|N|E}] [,STEDIT = {Y|N}] [,ISDLIST = {Y|N}] [,PMDLIST={Y|N}] [,LISTDS={Y|N}] [,LINCR=(first line number, increment)]

Figure 15. Assembler Parameters

Terminal lines must contain no more than 80 significant characters if they are to be punched.

The means for punching terminal lines is the PUNCH command. This command contains two operand fields ("startno" and "endno") specifying the character positions relative to the data set record of the lines to be punched. Assembler source lines are stored in the data set as entered by the programmer, with the exception that the first character of each line entered becomes the ninth character of the data set record; a 7-byte line number and a 1-byte format character are provided by TSS during the formation of the data set. Thus, the startno operand of the PUNCH command should be 9, not 1. The endno operand is normally 88, but it can be less if all the keyboard lines contain some number of significant characters less than 80.

Assembler Options and Related Output

This section discusses three topics:

1. The parameters supplied when the ASM command is given.
2. The listings produced by the assembler when requested by user-supplied parameters.
3. The destination of all output from the assembly.

Assembler Parameters

When issuing an ASM command, the user must enter parameters providing such items as the module name for this assembly; the source of the input lines (prestored or to be entered at the terminal), etc. A list of assembler parameters is given in Figure 15. The notation used in Figure 15 is explained in Appendix G, Command Formats.

One of the assembler parameters listed in Figure 15 (module name) *must* be provided by the user; others may be left unspecified and system or user default values will be chosen.

Explicitly Defaulted

A comma is issued immediately following entrance of the preceding parameter, rather than entering a value for the new parameter followed by a comma. For example, module ALPHA, with prestored source lines, is to be assembled, explicitly defaulting supplementary macro libraries and the version identification, but supplying parameters for all other parameters. The proper parameter description is:

ALPHA, Y, , , Y, Y, Y, Y, Y, Y, Y

Implicitly Defaulted

In the example above, the user could have pressed the return key following entrance of the Y specifying that an ISD is to be produced. This action implicitly defaults all parameters following the ISD option.

The assembler parameters shown in Figure 15 are defined as follows:

NAME=

specifies the name of the object module to be created. Since the name used becomes a member of a virtual partitioned data set when the object module is created, partially-qualified names and generation data group names cannot be used. Virtual partitioned data set members must be identified with simple names. The source data set for that module is named SOURCE.module by the system; the listing data set for the module is named LIST.module-name.generation-number to cause actual printing to be accomplished. The module name must be unique to the library that is to include it; i.e., it must not be the same as any entry point, CSECT name, or externally defined symbol or module in that library. The name consists of one to eight alphameric characters, the module name may not be the same as any external definition supplied by the module.

STORED=

specifies whether or not the source data set is prestored; if so, it must have been named SOURCE.module. The allowable values are Y or N. The system assumes N.

MACROLIB=

The first ddname ($ddname_1$) specifies the symbolic portion of the supplementary macro library that is to be used. A DDEF for this ddname must be provided by the user. If a DDEF has not been entered by a conversational user, he is prompted for the required definition information; a nonconversational task is abnormally terminated.

Default: Only the system macro library is used. The second ($ddname_2$) specifies the ddname of the index portion of the supplementary macro library that is to be used. A DDEF for this ddname must be provided by the user. If a DDEF has not been entered by a conversational user, he is prompted for the required definition information; a nonconversational task is abnormally terminated.
Default: Only the system macro library index is used.
Note: $ddname_1$ and $ddname_2$ must both be given if either is given. Up to five more user supplementary macro libraries may be defined for use. The first additional one would use $ddname_3$ for the symbolic portion and $ddname_4$ for the index portion. The next one would utilize $ddname_5$ and $ddname_6$, and so on until the maximum allowable number of supplementary macro libraries is reached. As with $ddname_1$ and $ddname_2$, a pair of ddnames must always be specified. User macro libraries are searched in the reverse order of specification; the first one defined will be the last one searched.

VERID =
specifies the version identification to be assigned. The version identification consists of one to eight alphameric characters. The version identification appears in the PMD (and PMD listing if requested).

ISD =
specifies whether an Internal Symbol Dictionary (ISD) is to be produced. An ISD is used by the Program Control System (PCS) in order to refer to internal program symbols during checkout. The allowable values are Y or N. The system assumes Y.

SYMLIST =
specifies whether a symbolic source program listing is to be produced. The allowable values are Y or N. A source listing displays the card or keyboard images supplied as input to the assembler. The system assumes N.

ASMLIST =
specifies whether an object program listing is to be produced. The allowable values are Y or N. An object program listing shows the concatenated source lines together with the associated absolute value or location counter assignment. Where required by the statement, the hexadecimal representation of the binary text is also displayed. The system assumes Y.

CRLIST =
specifies whether a cross-reference listing is to be produced. The allowable values are Y or N. A cross-reference listing is a table of the defined symbols and the locations of all references to those symbols. The system assumes N.

STEDIT =
specifies whether the edited symbol table is to be listed. The allowable values are Y or N. The symbol table edit displays symbol names, their attributes, and the absolute or relocatable value assigned to each symbol. Either a cross-reference listing or a symbol table edit may be requested but not both. The system assumes N.

ISDLIST =
specifies whether an ISD listing is to be produced. The allowable values are Y or N. An ISD listing displays the internal symbol entries found in the ISD. The system assumes N.

PMDLIST =
specifies whether a program module dictionary (PMD) listing is to be produced. The allowable values are Y or N. A PMD listing displays the contents of the PMD by control section. The system assumes N.

LISTDS =
specifies whether the requested listings are to be placed in a list data set or placed directly on SYSOUT. When listings are placed on SYSOUT, no record of them is kept in the system after printout. The system assumes N for non-conversational tasks (no list data set), and Y for conversational.

LINCR = first line number
specifies the line number to be assigned to the first line of the data set. The line number can contain three to seven digits, the last two of which must be 00.
Default: The first line number is 100.

increment

specifies the increment to be applied to develop successive line numbers. The increment can contain three to seven digits, the last two of which must be 00.
Default: The increment is 100.

Although the line number values are explained in the context of the STORED parameter, such values have meaning only in conjunction with a negative (N) response to the STORED parameter. So that the syntax analysis for both prestored and non-prestored data sets may be identical, the LINCR parameter now resides at the end of the parameter list.

The source code in your object program listing can appear in either aligned or unaligned format. *Aligned* format means that regardless of how you entered your input, all name fields will appear in column 1, all instruction mnemonics will appear in column 10 (or one blank following the name field, whichever is further to the right), and all operands will begin in column 16 (or one blank following the mnemonic, again whichever is further to the right). *Unaligned* format means that the fields of the source code will appear exactly as you entered them.

If you do not specify otherwise, the source code will be aligned. To achieve an unaligned format, issue the DEFAULT command with an operand of ASMALIGN=N prior to issuing the ASM command. If you have issued DEFAULT ASMALIGN=N and wish to revert to an aligned format (the system default), issue DEFAULT ASMALIGN=Y.

Structure and Description of Assembler Listings

The assembler prepares a listing data set if one or more of the six listing options are requested. In nonconversational mode, if a list data set is not specifically requested, listings are placed on SYSOUT and no record of them is retained in the system after printout. In conversational mode, a list data set is automatically created for your listings. You may, however, choose to have them placed on SYSOUT (printed at your terminal). The six types of listings are: source program listing, object program listing, cross-reference listing, symbol table listing, internal symbol dictionary listing, and program module dictionary listing. Various combinations of these listing options are possible. Operation codes are now aligned on output listings from the Assembler to provide a more orderly presentation. This applies to both macro expansions and user-created code. Printing of the listing data sets prepared by the assembler is not automatic. Each time a unique module name is encountered, a generation data group is established, containing two generations. Each time the limit (two generations) is reached, the oldest generation is erased. The user may print only when he desires the output listings, using: PRINT LIST.module-name.generation (absolute) or LIST.module-name (generation) (relative). *Command System User's Guide*, GC28-2001, presents a complete explanation of the language processor listing data set maintenance process.

Since a pending BULK/IO task will be established when the PRINT command is issued for the language processor listing data set, the user must not attempt to erase the data set (or otherwise remove it from the system) unless the BSN is canceled first.

The formats of assembler-produced listings are illustrated below. The programs were designed so that diagnostics would be produced and certain assembler instructions and assembler functions (e.g., literal pooling and reordering of control sections) could be illustrated. All types of assembler output are shown; the circled numbers on the listings correspond to the numbers in bold face type in the text.

Source Program Listing

The source program listing presents, in the order received, the original source language line images submitted for assembly by the user. Each source line, **2**, is preceded by a decimal statement number, **1**. Terminal input greater than 120 characters is continued on the next line. Figure 16 is an illustration of a source program listing.

Diagnostic messages, **3**, are collected and presented at the end of the listing. Only those messages produced prior to the text generation phase are listed. Each message is preceded by the statement number, **4**, of the line to which it applies. Messages are listed in ascending order by line number.

Object Program Listing

The object program listing documents, in control section order, the hexadecimal representation of the binary text assembled for each source statement. Continued source statements are shown in concatenated form. No characters before the continue column in continuation lines appear in the object listing. Unless an ICTL instruction is used to change assembler treatment of card records, column 16 is the continue column on card records. The first non-blank, non-tab character is used as the continue column in keyboard continuation lines. The ASMALIGN default value may be used to align the source code in the object program listing.

Figure 17 is an illustration of an object program listing. The sample listing in Figure 17 contains three control sections, one of which has been written noncontiguously (for illustrative purposes only) and has

```
                                                                                     PAGE 0001

                                                  SOURCE LANGUAGE LISTING
   ①              ②
  LINE          SOURCE TEXT

0000100  *              INDIAN PROBLEM
0000200  INDIANP        PSECT
0000300                 ENTRY   INDIAN
0000400                 DC      F'76'                   SAVE APEA
0000500                 DC      18F'0'
0000600  YEAR           DS      D                       CONVERSION AREA
0000700  MESSAGE        DS      CL20                    MESSAGE LOCATION
0000800  LENGTH         DC      F'20'                   MESSAGE LENGTH
0000900  PRINCP         DC      PL7'+24.00'             PRINCIPAL AMT
0001000  INT            DC      PL2'+1.04'              INTEREST RATE
0001100  THEN           DC      F'1626'
0001200  NOW            DC      F'1965'
0001300  ROUND          DC      PL2'50'
0001400  TEMP           DS      PL66
0001500  EDITOR         DC      C' $ '
0001600                 DC      X'2120',C','
0001700                 DC      X'202020',C','
0001800                 DC      X'202020'
0001900                 DC      X'2020'
0002000  ** THE FOLLOWING INSTRUCTIONS ARE INCLUDED TO CAUSE MODIFIERS TO PRINT
0002100  ** ON THE PROGRAM MODULE DICTIONARY LISTING
0002200                 EXTRN   XYZ
0002300                 DC      V(AA)
0002400                 DC      A(XYZ)
0002500                 DC      AL3(TEMP)
0002600  ** END OF SPECIAL INSTRUCTIONS
0002700  INDIANC        CSECT   READONLY
0002800                 USING   INDIAN,15
0002900  INDIAN         SAVE    (14,12)
0003000                 L       14,72(0,13)             GET PSECT COVER REG
0003100                 ST      14,8(0,13)              STORE FORWARD LINK
0003200                 ST      13,4(0,14)              STORE BACKWARD LINK
0003300                 LR      13,14                   SET REG 13 TO ADDRESS OF PSECT
0003400                 USING   INDIANP,13
0003500                 LR      12,15                   LOAD COVER REG
0003600                 DROP    15
0003700                 USING   INDIAN,12
0003800                 L       5,NOW                   SET UP DATE COUNTER
0003900                 L       3,THEN
0004000                 L       4,1
0004100  INDIANCL       CSECT   READONLY
0004200                 EDIT    MESSAGE+6(14),PRINCP+1  EDIT ANSWER
0004300                 CVD     3,YEAR                  CONVERT YEAR FOR PRINTOUT
0004400                 UNPK    MESSAGE(4),YEAR+5(3)
0004500                 OI      MESSAGE+3,X'F0'         REMOVE SIGN FROM DATE
0004600                 GATWR   MESSAGE,LENGTH          WRITE ANSWER ON SYSOUT
0004700                 L       13,4(0,13)              RESTORE CALLING REG 13
0004800                 RETURN  (14,12)
0004900  ** THE FOLLOWING INSTRUCTIONS ARE INCLUDED TO ILLUSTRATE CERTAIN TYPES
0005000  ** OF ASSEMBLER INSTRUCTIONS AND THE POOLING OF LITERALS
0005100  THISYEAR       EQU     NOW
0005200                 GBLA    &A

                                                                                     PAGE 0002

                                                  SOURCE LANGUAGE LISTING

  LINE          SOURCE TEXT

0005300  &A             SETA    2
0005400                 ORG     *+100
0005500                 MVC     YEAR(2),=X'4040'
0005600                 L       1,=F'123'
0005700                 CCW     2,YEAR,X'48',80
0005800                 CNOP    2,8
0005900                 MACRO
0006000                 MM      &P1,&P2
0006100                 MNOTE   &P1,'THIS ILLUSTRATES AN MNOTE &P2'
0006200                 MEND
0006300                 MM      1,DIAGNOSTIC
0006400                 MM      *,COMMENT
0006500  ** END OF ILLUSTRATIONS
0006600  INDIANC        CSECT
0006700  LOOP           MP      PRINCP,INT              COMPUTE INTEREST
0006800                 AP      PRINCP,ROUND            ROUND OFF
0006900                 MVN     PRINCP+5(1),PRINCP+6    MOVE SIGN
0007000                 MVC     TEMP,PRINCP             EFFECTIVELY SHIFT OFF 2 DIGITS
0007100                 ZAP     PRINCP,TEMP
0007200  PRINT          BXLE    3,4,LOOP
0007300                 MVC     MESSAGE+4(16),EDITOR    SET UP EDIT MASK
0007400                 END

                                                                                     PAGE 0003

                                                  SOURCE LANGUAGE LISTING

   ④          ③                                     DIAGNOSTIC MESSAGES

0001400 E *** VALUE OF LENGTH MODIFIER INVALID FOR TYPE OF CONSTANT
0004200 E *** 'EDIT    ' UNDEFINED MNEMONIC OPERATION
```

Figure 16. Source Program Listing

```
   9          12           13       14       19        20                                                      PAGE 0004
LOCATION  INSTRUCTION    ADDR 1   ADDR 2   STATEMNT  SOURCE                                           07/22/7   08:39:19

                                           0000100   *         INDIAN PROBLEM
10  11  01 00000                           0000200   INDIANP  PSECT
                                           0000300            ENTRY INDIAN
01 00000 0000004C                          0000400      23    DC    F'76'                    SAVE AREA
01 00004 00000000                          0000500            DC    18F'0'
01 00050                                   0000600   YEAR 24  DS    D                        CONVERSION AREA
01 00058                                   0000700   MESSAGE  DS    CL20                     MESSAGE LOCATION
01 0006C 00000014                          0000800   LENGTH   DC    F'20'                    MESSAGE LENGTH
01 00070 0000000002400C                    0000900   PRINCP   DC    PL7'+24.00'              PRINCIPAL AMT
01 00077 104C                              0001000 8 INT      DC    PL2'+1.04'               INTEREST RATE
01 00079 000000                                  +
01 0007C 0000065A                          0001100   THEN     DC    F'1626'
01 00080 000007AD                          0001200   NOW      DC    F'1965'
01 00084 050C                              0001300   ROUND    DC    PL2'50'
01 00086                                   0001400 E TEMP        DS    PL66
01 00096 405B40                            0001500   EDITOR   DC    C' $ '
01 00099 2120                              0001600            DC    X'2120',C','
01 0009B 6B
01 0009C 202020                            0001700            DC    X'202020',C','
01 0009F 6B
01 000A0 202020                            0001800            DC    X'202020'
01 000A3 2020                              0001900            DC    X'2020'
                                           0002000 28** THE FOLLOWING INSTRUCTIONS ARE INCLUDED TO CAUSE MODIFIERS TO PRINT
                                           0002100   ** ON THE PROGRAM MODULE DICTIONARY LISTING
                                           0002200            EXTRN XYZ
01 000A5 000000                                  +
01 000A8 00000000                          0002300            DC    V(AA)
01 000AC 00000000                          0002400            DC    A(XYZ)
01 000B0 000086                            0002500            DC    AL3(TEMP)
                                           0002600   ** END OF SPECIAL INSTRUCTIONS
         01 000B3                                +   INDIANP  PSECT
01 000B4                                         +            DS    0F                    FULL WORD ALIGNMENT
01 000B4 0000                                    +            DC    H'0'                 SIC CODE
01 000B6 0002                                    +            DC    H'2'                  TYPE CODE
01 000B8 000000B4                                +   CHD0002  DC    A(*-4)
01 000BC 00000058                                +            DC    A(MESSAGE)
01 000C0 0000006C                                +            DC    A(LENGTH)
01 000C4 00000000                                +            DC    A(0)
01 000C8 00000000                                +            DC    A(0)
         02 00000                          0002700   INDIANC  CSECT READONLY
         02 00000                          0002800            USING INDIAN,15
                                           0002900   INDIAN   SAVE  (14,12)
02 00000 90EC D00C                               +   INDIAN   STM   14,12,12(13) SAVE SPECIFIED REG'S
02 00004 58E0 D048                         0003000            L     14,72(0,13)              GET PSECT COVER REG
02 00008 50E0 D008                         0003100            ST    14,8(0,13)               STORE FORWARD LINK
02 0000C 50D0 E004                         0003200      15    ST    13,4(0,14)               STORE BACKWARD LINK
02 00010 18DE                              0003300            LR    13,14                    SET REG 13 TO ADDRESS OF PSECT
         01 00000                          0003400            USING INDIANP,13
02 00012 18CF                              0003500            LR    12,15                    LOAD COVER REG
                                           0003600            DROP  15
         02 00000                          0003700      16    USING INDIAN,12
02 00014 5850 D080                01 00080 0003800            L     5,NOW                    SET UP DATE COUNTER
02 00018 5830 D07C                01 0007C 0003900            L     3,THEN
02 0001C 5840 0001                         0004000 W             L     4,1

                                                                                                              PAGE 0005

LOCATION  INSTRUCTION    ADDR 1   ADDR 2   STATEMNT  SOURCE                                          07/22/71  08:39:19

         02 00020                          0006600   INDIANC  CSECT
02 00020 FC61 D070 D077 01 00070 01 00077 0006700   LOOP     MP    PRINCP,INT               COMPUTE INTEREST
02 00026 FA61 D070 D084 01 00070 01 00084 0006800            AP    PRINCP,ROUND             ROUND OFF
02 0002C D100 D075 D076 01 00075 01 00076 0006900      18    MVN   PRINCP+5(1),PRINCP+6     MOVE SIGN
02 00032 D20F D086 D070 01 00086 01 00070 0007000            MVC   TEMP,PRINCP              EFFECTIVELY SHIFT OFF 2 DIGITS
02 00038 F86F D070 D086 01 00070 01 00086 0007100            ZAP   PRINCP,TEMP
02 0003E 8734 C020                02 00020 0007200   PRINT    BXLE  3,4,LOOP
02 00042 D20F D05C D096 01 0005C 01 00096 0007300          MVC   MESSAGE+4(16),EDITOR   SET UP EDIT MASK
                                           0007400            END
02 00048 0000007B                                       25          =F'123'
02 0004C 4040                                                       =X'4040'
         03 00000                          0004100   INDIANCL CSECT READONLY
                                           0004200 E    29       EDIT   MESSAGE+6(14),PRINCP+1 EDIT ANSWER
03 00000 4E30 D050                01 00050 0004300            CVD   3,YEAR                   CONVERT YEAR FOR PRINTOUT
03 00004 F332 D058 D055 01 00058 01 00055 0004400            UNPK  MESSAGE(4),YEAR+5(3)
03 0000A 96F0 D05B       01 0005B          0004500      17    OI    MESSAGE+3,X'F0'          REMOVE SIGN FROM DATE
                                           0004600            GATWR MESSAGE,LENGTH           WRITE ANSWER ON SYSOUT
                                                 +            CHDPSECT CHDX0002
         03 0000E                                +   INDIANCL CSECT
03 0000E 4110 D0B8                01 000B8       +   CHDX0002 LA    1,CHD0002
                                                 +            CHDINNRA ,,(CZATC1),X'9D'     GENERATE LINKAGE
03 00012 41F0 009D                               +            LA    15,X'9D'                     LOAD REG. 15 WITH ENTER CODE
                                                 +            ENTER TYPE II LINKAGE
03 00016 0A79                                    +            SVC   121                              SUPERVISOR CALL
03 00018 58D0 D004                01 00004 0004700            L     13,4(0,13)               RESTORE CALLING REG 13
                                           0004800            RETURN (14,12)
03 0001C                                         +            DS    0H
03 0001C 98EC D00C                01 0000C       +            LM    14,12,12(13)
03 00020 07FE                                    +            BR    14
                                           0004900   ** THE FOLLOWING INSTRUCTIONS ARE INCLUDED TO ILLUSTRATE CERTAIN TYPES
                                           0005000   ** OF ASSEMBLER INSTRUCTIONS AND THE POOLING OF LITERALS
         01 00080                          0005100   THISYEAR EQU   NOW
                                           0005200            GBLA  &A
         00000002                          0005300   &A       SETA  2
         03 00086                          0005400            ORG   *+100
03 00086 D201 D050 C04C 01 00050 02 0004C 0005500            MVC   YEAR(2),=X'4040'
03 0008C 5810 C048                02 00048 0005600      21    L     1,=F'123'
03 00090 02000050 48000050                 0005700            CCW   2,YEAR,X'48',80
03 00098 0700                              0005800            CNOP  2,8
                                           0005900      22    MACRO
                                           0006000            MM    &P1,&P2
                                           0006100            MNOTE &P1,'THIS ILLUSTRATES AN MNOTE &P2'
                                           0006200            MEND
                                           0006300            MM    1,DIAGNOSTIC
                                                +W      26       MNOTE  1,'THIS ILLUSTRATES AN MNOTE DIAGNOSTIC'
                                           0006400            MM    *,COMMENT
                                                 +   * THIS ILLUSTRATES AN MNOTE COMMENT 27
                                           0006500   ** END OF ILLUSTRATIONS

                                                                                                              PAGE 0006

                                          5 WARNING AND ERROR MESSAGES

0001400 E *** VALUE OF LENGTH MODIFIER INVALID FOR TYPE OF CONSTANT
0004000 W *** OPERAND REQUIRES FULL-WORD BOUNDARY
0004200 E *** 'EDIT     ' UNDEFINED MNEMONIC OPERATION
0006300 W *** THIS ILLUSTRATES AN MNOTE DIAGNOSTIC

NUMBER OF WARNING AND ERROR MESSAGES     004  6

HIGHEST SEVERITY CODE ENCOUNTERED        002  7
```

Figure 17. Object Program Listing

been put in order by the assembler. The user should avoid writing non-contiguous control sections, if possible, as they assemble much less efficiently.

Warning and error messages, **5**, are collected and presented at the end of the listing. All diagnostic messages (including MNOTE messages with a severity code) produced by the assembler will be listed. This listing differs from the listing presented at the end of the source language listing. Messages are listed in ascending order by line number. A count of the number of messages, **6**, and an indication of the highest severity code encountered, **7**, are also presented. The severity code is 1 if only warning messages were produced, or 2 if error messages were produced.

The listing contains the following types of lines in addition to the column heading line: machine instructions, assembler instructions with related values, assembler instructions without values, CCW instructions, CNOP instructions, constants, literal pools, diagnostic messages, MNOTE messages, and commentary lines (i.e., lines which were written as commentary by the user or which, due to diagnostic action, were made commentary by the assembler). In addition, a space for required boundary alignment or a statement generated by a macro instruction contains a plus sign (+), **8**, immediately following the statement field. A source statement is edited in the following manner: (a) the name field will begin in column 1; a sequence symbol in the name field is suppressed; (b) the operation code is shifted to begin in the location corresponding to card column 10 or the next available location thereafter; (c) the operand is shifted to begin in the location corresponding to card column 16 or the first available location thereafter; and (d) the comment field will follow the operand field by the number of blanks coded in the source program. *No editing is performed if the statement is in error.* Each type of line is described below.

1. *Machine Instructions:* Under location, **9**, the section number, **10**, and location counter displacement, **11**, are listed in hexadecimal. The instruction, **12**, addr 1, **13**, and addr 2, **14**, fields differ according to the type of instruction. If the instruction type is RR, **15**, the first part of the instruction field contains the hexadecimal text, and the addr 1 and addr 2 fields are blank. If the instruction type is RX or RS, **16**, the hexadecimal text for the R1 and R2 fields (R3, if RS) and the hexadecimal text for the B2 and D2 fields appear under instruction heading. The addr 1 field is blank. Under the addr 2 heading appear the section number and the location counter displacement of the symbolic S2 field, if applicable. If the instruction type is SI, **17**, the instruction field contains the hexadecimal text of the I field and the hexadecimal text for the B1 and D1 fields. The addr 1 field contains the section number and location counter displacement of the symbolic S1 field, if applicable. The addr 2 field is blank. If the instruction type is SS, **18**, the three subfields of the instruction field contain the hexadecimal text of the R1 and R2 fields, the hexadecimal text of the B1 and D1 fields, and the hexadecimal text of the B2 and D2 fields, respectively. The addr 1 field contains the section number and location counter displacement of the symbolic S1 field, if applicable. The addr 2 field contains the section number and location counter displacement of the symbolic S2 field, if applicable.

The seven-digit decimal statement number appears under the statement heading, **19**. The number is edited to contain leading zeros, e.g., line 4200 prints as 0004200. The 80-character source statement is listed under source, **20**, and is preceded by the letter W or E if a warning or error message has been issued. If a statement exceeds 80 characters in length, it is continued on as many lines as necessary, with continuation lines beginning in the location corresponding to card column 16.

2. *Assembler Instructions With Related Values:* This format description applies to the instructions CSECT, PSECT, DSECT, COM, START, END, EQU, LTORG, ORG, USING, SETA, SETB, and SETC. Under the instruction heading, the value of the instruction is listed. The types of values are described below. The location, addr 1, and addr 2 fields are blank. Other fields are as described under machine instructions.

Relocatable value fields are associated with CSECT, PSECT, DSECT, COM, START, EQU, END, LTORG, ORG, and USING. They contain the section number and location counter displacement, both in hexadecimal. Absolute value fields are associated with EQU, USING, SETA, and SETB; they contain the 32-bit value expressed as 8 hexadecimal digits. External or complex relocatable value fields are associated with EQU and USING, and are blank. Character-string value fields are associated with SETC; they contain an alphameric character string.

3. *Assembler Instructions Without Values:* The location, instruction, addr 1, and addr 2 fields are blank for the following instructions: COPY, DROP, ENTRY, EXTRN, AIF, AGO, GBLA, GBLB, GBLC, LCLA, LCLB, LCLC, PRINT, ICTL, ISEQ, PUNCH, REPRO, and macro instructions. The remaining fields are as described under machine instructions.

4. *CCW Instructions, **21**:* The location field contains the section number and location counter displacement, both in hexadecimal. The instruction field contains the text of the command code and data address fields, expressed as eight hexadecimal digits, followed by the text of the flag and count fields, also expressed as eight hexadecimal digits. The addr 1 and addr 2 fields are blank. The remaining fields are as described under machine instructions.

5. *CNOP Instructions,* **22:** The location field contains the section number and location counter displacement, both in hexadecimal. The instruction field contains the hexadecimal text of one, two, or three NOPR instructions, if required. The addr 1 and addr 2 fields are blank. The remaining fields are as described under machine instructions.

6. *Constants:* The location field contains the section number and location counter displacement, both in hexadecimal, for a DC, **23**, or DS, **24**, statement. If DC, up to eight bytes of the constant are listed on the first line under instruction. If DS, this field is blank. If the DATA print option is on, the remainder of the constant is listed eight bytes per line. If a duplication factor greater than one is present, each duplication is listed as if it were a new constant. The addr 1 and addr 2 fields are blank. The remaining fields are as described under machine instructions.

7. *Literal Pools:* The location, instruction, addr 1, and addr 2 fields are as described under constants. The source text of the literal, beginning with an equal sign (=), **25**, appears in lieu of the 80-character source statement.

8. *Diagnostic Messages:* Diagnostic messages are collected at the end of the listing. Each message is preceded by the statement number of the line to which it applies, and its severity code. Messages are listed in ascending order by line number.

9. *MNOTE Messages:* MNOTE messages that contain a severity code, **26**, are printed as diagnostic messages. MNOTE messages that contain an asterisk for the severity code, **27**, are printed as commentary lines.

10. *Commentary Lines:* Commentary lines are lines which were written as commentary by the user, **28**, or which, due to diagnostic action, were made commentary by the assembler, **29**. The location, instruction, addr 1, and addr 2 fields of these lines are blank. Other fields are as described under machine instructions.

Cross-Reference Listing

The cross-reference listing is a presentation in alphabetical order of all the symbols defined within the assembly. It includes a list of all hexadecimal program locations where a reference to the symbol is made in the source language. Figure 18 is an illustration of a cross-reference listing.

Each symbol, **30**, in an index line is followed immediately by the type attribute (defined in Table 6), **33**, and the length attribute, **34**, in bytes, of the symbol. Under location, **35**, are listed the section number, **31**, and displacement, **32**, both in hexadecimal, of the location where the symbol is defined (if relocatable), or an eight-digit hexadecimal number (if absolute). The references field, **36**, contains the section number, **37**, and displacement, **38**, in hexadecimal, of each lo-

```
 30       33    34      35          36                                  CROSS R
SYMBOL   TYPE  LNG   LOCATION   REFERENCES
                   31          32 37    38
CHDX0002 I    00004  03 0000E
CHD0002  A    00004  01 000B8   03 0000E
EDITOR   C    00003  01 00096   02 00042
INDIAN   M    00004  02 00000   02 00000, 02 00014
INDIANC  J    00001  02 00000
INDIANCL J    00001  03 00000
INDIANP  J    00001  01 00000   02 00012
INT      P    00002  01 00077   02 00020
LENGTH   F    00004  01 0006C   01 000C0
LOOP     I    00006  02 00020   02 0003E
MESSAGE  C    00014  01 00058   01 000BC, 02 00042, 03 00004, 03
NOW      F    00004  01 00080   02 00014, 03 00022
PRINCP   P    00007  01 00070   02 00020, 02 00026, 02 0002C, 02
PRINT    I    00004  02 0003E
ROUND    P    00002  01 00084   02 00026
TEMP     P    00010  01 00086   01 000B0, 0200032, 02 00038
THEN     F    00004  01 0007C   02 00018
THISYEAR U    00004  01 00080
XYZ      T    00001  00 00001   01 000AC
YEAR     D    00008  01 00050   03 00000, 03 00004, 03 00086, 03
```

Figure 18. Cross-Reference Listing

Table 6. Type Attributes

TYPE ATTRIBUTE	DESCRIPTION OF SYMBOL REPRESENTED BY ATTRIBUTE
A	A-type address constant, implied length, aligned.
B	Binary constant.
C	Character constant.
D	Long floating-point constant, implied length, aligned.
E	Short floating-point constant, implied length, aligned.
F	Full-word fixed-point constant, implied length, aligned.
G	Fixed-point constant, explicit length.
H	Half-word fixed-point constant, implied length, aligned.
I	Machine instruction.
J	Control section name.
K	Floating-point constant, explicit length.
M	Macro instruction.
N	Self-defining term (inner and outer macro instruction operands only).
O	Omitted operand (inner and outer macro instruction operands only).
P	Packed decimal constant.
Q	Q-type address constant, implied length, aligned.
R	A-, Q-, R-, S-, V-, or Y-type address constant, explicit length.
S	S-type address constant, implied length, aligned.
T	External symbol.
U	Undefined. Used for symbols whose attributes are not available, and for inner and outer macro instruction operands that cannot be assigned another attribute. This includes inner macro instruction operands that are symbols or literals. This letter is also assigned to symbols that name EQU statements.
V	V-type address constant, implied length, aligned.
W	CCW assembler instruction.
X	Hexadecimal constant.
Y	Y-type address constant, implied length, aligned.
Z	Zoned decimal constant.
#	R-type address constant, implied length, aligned.

PAGE 0002

TABLE OF DEFINED SYMBOLS

39 SYMBOL	40 TYPE	41 LENGTH	42 VALUE
CHDX0002	I	00004	03 0000F
CHD0002	A	00004	01 000B8
EDITOR	C	00003	01 00096
INDIAN	M	00004	02 00000
INDIANC	J	00001	02 00000
INDIANCL	J	00001	03 00000
INDIANP	J	00001	01 00000
INT	P	00002	01 00077
LENGTH	F	00004	01 0006C
LOOP	I	00006	02 00020
MESSAGE	C	00014	01 00058
NOW	F	00004	01 00080
PRINCP	P	00007	01 00070
PRINT	I	00004	02 0003E
ROUND	P	00002	01 00084
TEMP	P	00010	01 00086
THEN	F	00004	01 0007C
THISYEAR	U	00004	01 00080
XYZ	T	00001	00000000
YEAR	D	00008	01 00050

SYMBOL	TYPE	LENGTH	VALUE

Figure 19. Symbol Table Listing

cation where a reference is made to the symbol. Reference locations are listed in ascending order.

Symbol Table Listing

The symbol table listing is a presentation, in alphabetical order, of all the symbols, 39, defined within the assembly. It includes their type, **40**, length, **41**, and value, **42**, attributes. This listing is similar to the cross-reference listing but excludes references. This listing is produced only if the symbol table listing option has been selected and the cross-reference listing option has not also been specified. Figure 19 is an illustration of a symbol table listing.

Each symbol in the listing is followed by its type, length, and value attributes. The value attribute, in hexadecimal, is either a section number and location counter displacement, if relocatable, or an eight-digit hexadecimal number, if absolute.

Internal Symbol Dictionary Listing

The internal symbol dictionary listing is a presentation of the symbols and related information placed, on request, in the ISD portion of the program module to assist the program checkout system. Figure 20 is an illustration of an internal symbol dictionary listing.

Each column after the first presents symbols and information as described by the first column. The first line of a column contains the eight-character name of a symbol, **43**. The second line contains the type of symbol, **44**, that is represented by one of the following: INSTR, ADCON, BINARY, HEX, SECTION, REAL, INTEGER, CHAR, ZONED, PACKED, S-CON, or VALUE. The third line contains a duplication factor, **45**, in hexadecimal. The fourth line contains an eight-digit length, **46**, if any, in hexadecimal. The length is normally the length attribute of the symbol; if the symbol is the name of a control section, the length represents the length, in bytes, of the control section. The fifth line contains an eight-digit immediate value or an eight-digit section number and location counter displacement, **47**, in hexadecimal.

Program Module Dictionary Listing

The program module dictionary (PMD) listing presents the contents of the PMD. The PMD is created at assembly time and stored as part of the object module. Information in the PMD directs the loading of the object module. The PMD contains external symbol definitions, references, and relocation information.

The PMD listing is helpful in determining the structure of the user's object module and its relocation properties. Figure 21 is an illustration of a PMD listing.

The initial portion of the PMD listing contains a description of the program module. The module name is listed first, **48**, followed by the version (or time stamp), **49**, the length of the PMD, **50**, in hexadecimal, and the highest severity code encountered, **51**. The severity code is 0 if no diagnostic messages were produced, 1 if only warning messages were produced, or 2 if error messages were produced. The succeeding parts of the PMD listing contain descriptions of the control sections in the module. The name of the control section is listed first, **52**, followed by the type, **53**, which is CONTROL, COMMON, or PROTOTYPE. A time stamp is always assigned to the version, **54**. The attributes, **55**, may be one or more of the following: FIXED, VARIABLE, READONLY, PUBLIC, SYSTEM, or PRVLGD. Each attribute, except FIXED, prints if it was specified by the user. FIXED prints if VARIABLE was not specified. The length in bytes of the control section dictionary, **56**, is listed next (in hexademical), followed by the byte length, **57**, in hexadecimal, of the binary text for the control section.

Following each control section description is a description of the relocatable, absolute, and complex definitions, **58**, within the control section. These definitions, for which the name and value are listed, include only those symbols and CSECT names that have been declared entry points by the ENTRY instruction. The next part of the control section description contains the names of the references, **59**, within the con-

```
                                                   INTERNAL SYMBOL DICTIONARY                                     PAGE 0001

      NAME  43   INDIANP   YEAR      MESSAGE   LENGTH    PRINCP    INT       THEN      THISYEAR  NOW       ROUND     TEMP      EDITOR
  44  TYPE       SECTION   REAL      CHAR      INTEGER   PACKED    PACKED    INTEGER   INTEGER   INTEGER   PACKED    PACKED    CHAR
      DUPL  45   000000    000000    000000    000000    000000    000000    000000    000000    000000    000000    000000    000000
  46  LENGTH     000000CC  00000008  00000014  00000004  00000007  00000002  00000004  00000004  00000004  00000002  00000010
00000003
      LOC/VAL 47 01 00000  01 00050  01 00058  01 0006C  01 00070  01 00077  01 0007C  01 00080  01 00080  01 00084  01 00086  01
00096

      NAME       CHD0002   INDIANC   INDIAN    LOOP      PRINT     INDIANCL  CHDX0002
      TYPE       ADCON     SECTION   HEX       INSTR     INSTR     SECTION   INSTR
      DUPL       000000    000000    000000    000000    000000    000000    000000
      LENGTH     00000004  0000004E  00000004  00000006  00000004  0000009A  00000004
      LOC/VAL    01 000B8  02 00000  02 00000  02 00020  02 0003E  03 00000  03 0000E
```

Figure 20. ISD Listing

```
                                   PROGRAM MODULE DICTIONARY LISTING                                  PAGE 0008
MODULE          48
  NAME        49   INDIANX
  VERSION          07/22/7    08:39:19
  LENGTH       50  00000200
  DIAG SEVERITY    002 51


SECTION 01      52
  NAME             INDIANP
  TYPE          53 PROTOTYPE
  VERSION          07/22/7    08:39:19 54
  ATTRIBUTES 55    FIXED
  CSD LENGTH       000000EC 56
  SECT LENGTH      000000CC 57

  COMPLEX DEFINITIONS 58
    NAME    INDIAN
    VALUE   00000000

  REFERENCES 59
    REF #   0000       0001       0002       0003
    NAME    AA         XYZ        INDIANP    INDIANC

  MODIFIERS FOR COMPLEX DEFS 60
    PAGE 00    # MODIFIERS 0001
 61   LENGTH    4
  62  REF #     0003
 63   TYPE      +
  64  BYTE      09C

  MODIFIERS FOR TEXT (EXTERNAL REFS, Q-CONS, AND CXDS)
    TEXT PAGE 00    VIRTUAL PAGE 00   # MODIFIERS 0002
      LENGTH    4          4
      REF #     0001       0000
      TYPE      +          +
      BYTE      0AC        0A8

  MODIFIERS FOR TEXT (INTERNAL REFS)
    TEXT PAGE 00    VIRTUAL PAGE 00   # MODIFIERS 0004
      LENGTH    4          4          4          3
      REF #     0002       0002       0002       0002
      TYPE      +          +          +          +
      BYTE      0C0        0BC        0B8        0B0


                                                                              PAGE 0009
SECTION 02
  NAME             INDIANC
  TYPE             CONTROL
  VERSION          07/22/7    08:39:19
  ATTRIBUTES       FIXED,READONLY
  CSD LENGTH       00000054
  SECT LENGTH      0000004E


SECTION 03
  NAME             INDIANCL
  TYPE             CONTROL
  VERSION          07/22/7    08:39:19
  ATTRIBUTES       FIXED,READONLY
  CSD LENGTH       00000070
  SECT LENGTH      0000009A

  REFERENCES
    REF #   0000
    NAME    INDIANP

  MODIFIERS FOR TEXT (INTERNAL REFS)
    TEXT PAGE 00    VIRTUAL PAGE 00   # MODIFIERS 0001
      LENGTH    3
      REF #     0000
      TYPE      +
      BYTE      091

END OF MODULE
```

Figure 21. PMD Listing

trol section. The last part within a control section description contains a description of each modifier with the control section, **60**. Modifiers for definitions are listed first, followed by modifiers for text, with external references preceding internal references. For each modifier there is an entry for the length, **61**, a reference number (corresponding to the reference listed above), **62**, a type code (+, −, C, Q, or R), **63**, and byte displacement, **64**, within the text of the control section where the reference appears. See Table 7 for an explanation of the type code.

Table 7. Type Code Significance in PMD Listing

TYPE CODE	SIGNIFICANCE
+	The definition value of the reference at "reference number" is added to the adcon starting at the indicated byte of the page to which the modifier applies.
—	Same as +, except that the value is subtracted.
C	Store cumulative external dummy section length (CXD value) in storage indicated by modifier.
Q	Same as "+" but use Q-type constant value associated with external dummy section named in reference.
R	Same as "+" but use R-value rather than definition value.

Destination of Output

Assembly variations and the destination of output associated with each variation and shown in Table 8.

Object Program Module Format

Each of the language processors produces object program modules that always have a program module dictionary and text; an internal symbol dictionary is produced only if specified by the user (see Figure 22).

Program Module Dictionary

The program module dictionary consists of a header and a series of control section dictionaries.

- The header contains the name of the standard entry point to the module and other information common to the entire module.
- Each control section dictionary describes its associated control section so that the system can produce, from the text, a fully linked, executable object module.

Text

The text portion of the module contains the instructions and constants generated by the assembler or compiler; it is the executable portion of the module. The text is organized by control sections, the basic unit of all TSS programs.

A control section is a block of coding whose virtual storage location assignments may be adjusted inde-

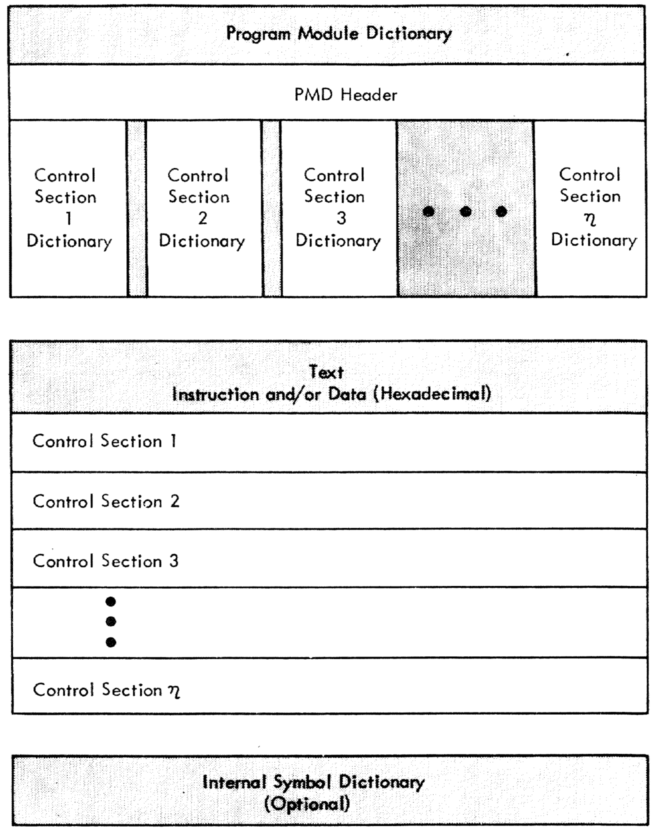


Figure 22. Format of an Object Program Module

pendently of other coding at linkage editing or load time, without altering or impairing the operating logic of the program. At least one page (4096 bytes) of virtual storage is assigned to each control section; a control section may require more than one page. However, at LOGON time the user may specify that control sections with like attributes be packed in virtual memory. This allows several related control sections to be collected into less memory space. Control section packing is encouraged so that the modules to be executed may be compressed into fewer pages, thus reducing the time required for paging operations by the system.

When virtual storage space is allocated to an object program module at load time, all its control sections are allocated. The contents of each control section occupy contiguous virtual storage addresses; however, the individual control sections may be scattered throughout virtual storage.

When object program modules are placed in main storage for execution, they are brought in page-by-page. The contents of each page occupy contiguous main storage locations; however, individual pages may be scattered throughout main storage. Only the pages required for execution are kept in main storage during a user's time slice.

Table 8. Destination of Output

ASSEMBLY VARIATION	OUTPUT: OBJECT MODULE	SOURCE	LISTINGS	ASSEMBLER DIAGNOSTICS
Conversational—Input from terminal keyboard or card reader	Latest JOBLIB defined in task or USERLIB	Data set named SOURCE. module name created by system.	Data set named LIST. module name unless printout to terminal is requested.	To terminal, and to list data set if listings requested.
Conversational—Prestored Data Set		Data set named SOURCE. module name will be updated to reflect modifications.	If a printout of the listing data is desired, it must be requested using the PRINT command.	
Nonconversational—Prestored Data Set			Same as conversational, if listing data set is specifically requested. To SYSOUT if no listing data set requested.	To SYSOUT data set if no listing data set requested; otherwise to list data set only.
Nonconversational—Input After ASM		Same as conversational, not prestored.		

The object program module code contains virtual storage addresses during execution. These are translated into actual main storage addresses, based on relationships established between each page's virtual storage base address and its main storage base address, at the time it is placed in main storage. If a page that is executing is swapped out and then relocated in main storage, it may well be assigned a new location in main storage. However, because a new relationship has been established between the page's virtual storage base address and its new main storage base address, the system can execute the page in its new main storage location.

Assembler users can control the organization of text into control sections.

Internal Symbol Dictionary

The internal symbol dictionary contains information, such as symbol definitions and data descriptions. It permits users to write program control system commands, using the same symbolic names for data and instructions that were used in their source coding. The internal symbol dictionary should be requested if the Program Control System is to be used.

Assembling in Express Mode

When a number of modules are to be assembled consecutively in one task, time may be saved by assembling in express mode. This will cause the language processor control to read the name of the next module from SYSIN whenever it would normally have returned to the Command System for a further command.

The express mode is turned on by issuing a command

DEFAULT LPCXPRSS=Y

anywhere in the task before the first assembly. The ASM command is issued only once, for the first assembly, and the assembly options (operands) are issued at the same time. The assembly options cannot be changed for subsequent assemblies.

The express mode can be turned off by entering an underscore as first character in a line, which will cause an exit from the language processing control system and a return to the command system. It is also possible to turn off the express mode by pressing the attention button any time during the assembly process and issuing a command

DEFAULT LPCXPRSS=,

Assembly can then be continued in non-express mode by issuing a GO command.

If an invalid module name is entered when the language processor control expects a new module name, the express mode will be turned off and a diagnostic message will be issued.

Assembler Restrictions

Limitations of virtual storage available to the assembler and of the object programs generated by it impose a number of restrictions on the size and contents of source programs capable of being assembled. These restrictions are categorized according to complexity. The first category, simple source program restrictions, can easily be applied to individual source statements or particular types of source statements. Simple program restrictions are listed in Table 9.

The second category, complex restrictions, is composed of restrictions that generally are too complex to anticipate in advance of assembly (e.g., the storage requirements of the various tables internal to the assembler are, in many cases difficult to compute accurately, as the table sizes are complex functions of the source program). Very few programs are of such a size or configuration that these complex limitations are exceeded. Therefore, the assembler user may not wish to concern himself with the complex restrictions until he receives a diagnostic message; then he can proceed to remedy the situation. Complex program restrictions are listed in Table 10.

The assembler working storage is separated into three parts. Within each working storage area are collections of file and table entries linked together by chain words. In the tables that follow, the table entries have been identified by the assembler phase that produces them.

Some of the more probable causes of working storage overflow are: specifying FULLGEN as an operand in a PRINT statement; infinite nesting of macro instructions; infinite looping within a macro expansion; and too many source statements. If a work area overflows, the assembler will attempt to dynamically acquire additional storage. If the storage is either unavailable or unaddressable, the assembly will terminate with an appropriate diagnostic message.

Table 9. Simple Source Program Restrictions

ITEM	MAXIMUM NUMBER OR SIZE	ASSOCIATED DIAGNOSTIC MESSAGE	USER CORRECTIVE ACTION	ASSEMBLER CONTINUATION ACTION
Unique control sections	255	MAXIMUM NUMBER OF CONTROL SECTIONS EXCEEDED	Split assembly in several parts and assemble each separately.	Control section statements for sections numbered 256 or greater are made commentary; the associated coding becomes part of the section in effect at the time of the error.
LTORG statements	253	None	Combine literal pools or split assembly into several modules.	The 254th and following LTORG statements are made commentary, literals associated with these statements are pooled by dafult at the end of the first CSECT and/or PSECT.
External References	2^{16}-1	None	Reduce external references by combining assembly modules or reducing size of assembly.	A number is assigned to each relocatable reference required by the assembly (EXTRN symbol, V-type address constant, and control section name). Reference 65535 is lost, and the loader resolves reference 65536 as if it were reference 0, etc., thus producing erroneous relocation of the module.
&SYSNDX	10,000	None	Reduct number of inner and outer level macro instructions.	The assembler continues modulo 10000 for &SYSNDX values. Macro expansions not referring to this system variable are correct; the first 10000 inner and outer level macro instructions generated are not effected. The 10001st use produces a &SYSNDX value of 0001 again; use of this value may produce duplication or conflicts with earlier macro-generated statements.
Unsublisted positional macro instruction operands	255	CHARACTER STRING ACCUMULATION IN EXCESS OF 255	Rewrite macro definition in order to concatenate operands longer than 255 characters or change macro instruction.	The length of the character string is reduced to 255 characters and the macro expansion continues.
Macro instruction suboperands within a sublist	255	None	Rewrite macro definition making suboperands operands.	Macro processing continues with the number attributes (N') of the positional operand computed modulo 255.

Table 9. Simple Source Program Restrictions *(Continued)*

ITEM	MAXIMUM NUMBER OR SIZE	ASSOCIATED DIAGNOSTIC MESSAGE	USER CORRECTIVE ACTION	ASSEMBLER CONTINUATION ACTION
Number of characters for card format input excluding macro instruction and prototype statements	240	TOO MANY CONTINUATION LINES	Compact statement. Operand fields can be contracted by using a variable character symbol in lieu of the desired operand. The variable symbol must, of course, be set to the desired character string for the operand. Comments can be continued on separate comment statements.	Processing continues with the fourth and following cards of the statement treated as commentary.
Location counter value	2^{24}-2	LOCATION COUNTER EXCEEDS MAXIMUM SEGMENT ADDRESS	Use multiple control sections. A symbol with location counter value 2^{24}-2 may not have a length attribute greater than 1.	The assembly is terminated. No object module is created.
Number of parenthesis levels per expression	64	EXPRESSION CONTAINS EXCESSIVE PARENTHESIS	Simplify expression possibly through the use of nested EQU or SET statements.	Evaluation of the expression noted in the diagnostic is terminated, causing incomplete assembly of the statement.
DS length modifier	65,535	VALUE OF LENGTH MODIFIER INVALID FOR TYPE OF CONSTANT	Write more than one consecutive DS statement.	Length of the DS statement is reduced to 65,535 and processing continues.
SYS symbol prefix reserved for system use	—	ENTRY POINT DECLARED IN CONTROL SECTION WITHOUT SYSTEM ATTRIBUTE	All external symbols starting with the characters SYS should be removed from the nonsystem program.	The assembly continues normally.

Table 10. Complex Restrictions

ITEM	OVERFLOW CAUSE	COMMENTS	ASSOCIATED DIAGNOSTIC MESSAGE
Assembler Work Area 1: (1⅔ pages = fixed usage 98⅓ pages = variable usage as outlined below)		Initial allocation only. Expansion is possible.	
Macro Level Dictionary 52 words + 4 to 16,384 words per entry (average 7 words per entry)	Assembler Phase II-A: Excessive nesting of macro instructions and/or usage of variable symbols within each macro level	A separate macro level dictionary is created for each macro instruction and lasts until the macro has been expanded as determined by the macro definition. Encountering a MEND or MEXIT statement will cause the area occupied by the current macro level dictionary to be returned to a scratch status. A macro level dictionary contains an entry for each positional macro instruction operand, an entry for each prototype keyword operand, and global and local variable symbol.	ASSEMBLER WORKING STORAGE EXHAUSTED —WORK1
Using-Register Tables (33 words per table) (Overlays area occupied by Page Usage Tables in Phase II-B)	Phase II-C: More than 1500 control section, USING, and/or DROP statements	A Using-Register Table is created for every control section break, USING, or DROP statement encountered in user level or macro generated source statements.	Same

Table 10. Complex Restrictions *(Continued)*

ITEM	OVERFLOW CAUSE	COMMENTS	ASSOCIATED DIAGNOSTIC MESSAGE
Cross Reference Item Sort Keys (2 words per entry) (Overlays area occupied by Using-Register Tables)	Phase III: Too many internal symbols and references to internal symbols. (Approximately 20000 cross-references)	A two word sort key is developed for each internal symbol and each reference to it if a cross reference listing has been requested.	Same
Assembler Work Area 2: (255 pages initially) Main Dictionary Items (5 to 1025 words per item or greater (average 7 words per item))	Assembler Phase I and II-A: Too many user local and global symbols in combination with other uses of the WORK2 area		ASSEMBLER WORKING STORAGE EXHAUSTED —WORK2
Logical Order File (Normally 5 words per source or macro generated statement except DCs, DSs, DXDs, or CXDs. Ten words per address constant, 8 words per DS, and 8 words plus the text length for one occurrence for each nonaddress constant DC are reserved)	Too many source statements or statements resulting from macro expansions in combination with other uses of the WORK2 area		Same
Global-Section-Macro Chain (3 words per entry)	Too many user level control section, macro instruction, GBLA, GBLB, GBLC (SETA, SETB, or SETC associated with user level global statements), USING, DROP, ENTRY, PRINT, and/or LTORG statements in combination with other uses of the WORK2 area		Same
Source statement continuation lines	Too many continuation lines		Same
Macro Name Dictionary Items (5 words per item)	Too many different macro definitions called by user program		Same
Variable Information for Diagnostics	Too many diagnostics		Same
Character string operands from TITLE and MNOTE instructions	Too many TITLE and MNOTE instructions		Same
Logical Order File (Alignment Entries) (2 words per entry)	Phase II-B: Excessive number of DS, DC, CNOP, CCW, CXD, LTORG, or machine instructions requiring alignment		Same
Main Dictionary (1) Relocatable EQU items (5 words per item)	Phase II-B: Excessive number of EQU statements	Simply relocatable items are those resolvable into one relocatable value plus an absolute offset.	Same
(2) Literal items plus associated literal trailer items (8 words per literal item and 5 words for each trailer)	Excessive number of nonduplicate literals within a literal pool	A literal item is created for the occurrence of each unique literal string. A trailer is created each time a literal with the same character string appears in a different literal pool.	Same

Table 10. Complex Restrictions *(Continued)*

ITEM	OVERFLOW CAUSE	COMMENTS	ASSOCIATED DIAGNOSTIC MESSAGE
Logical Order File (Diagnostic entries) (4 words per entry)	Phase III: Diagnostic messages		Same
Assembler Work Area 3: Original Source Statements (20 page blocks are acquired as needed)	Assembler Phase I: Source program too large or referring to a lengthy COPY element or containing backward AGO and/or AIF statements.	It is impossible to state exactly what number of statements produces this overflow condition since the usage of assembler working storage is a function of the type of statement. However, for an average program this number is usually larger than 10,000. In addition it should be noted that the FULLGEN operand of a PRINT statement will cause all conditional macro generated statements to be saved for printing on the output listing, thus requiring more WORK3 area than is otherwise the case.	VIRTUAL STORAGE EXHAUSTED. ASSEMBLER CANNOT CONTINUE.
Macro-Generated Statements	Phase II-A: Macro expansions	An infinite loop during macro expansion may result in virtual storage exhaustion PRINT statements with a FULLGEN operand may also result in total usage of virtual storage since each macro model statement is retained after string substitution is performed during the processing of each macro instruction.	Same
Additional Working Storage: Program Module Dictionary (2 pages + ⅛ page for each page of Assembled Program Text + total number of DEFs and REFs multiplied by 28)	Phase III: PMD too large. An excessive number of external definitions and/or external references will cause overflow		PMD FILE OVERFLOWED. ASSEMBLY TERMINATED.
External Name List (2 pages + number of DEFs multiplied by 28; two words per entry)	More than 512 external definitions (ENTRY operands, CSECT, and PSECT names) were specified.	Either splitting up the assembly or removal of external names should solve the problem.	EXT NAME FILE OVERFLOWED. ASSEMBLY TERMINATED.
List Data Set (VISAM)	Listing contributed to exhaustion of virtual storage.	Problem external to the assembler.	THE SIZE OF VIRTUAL MEMORY HAS BEEN EXCEEDED.
Assembled Program Text	Virtual storage exhausted.	Problem external to the assembler.	VIRTUAL STORAGE EXHAUSTED. ASSEMBLER CANNOT CONTINUE.
Internal Symbol Dictionary (255 pages)	Phase IV: ISD too large; excessive number of control sections, USING, DROP, control section breaks, and/or internal symbols	Four words are used for each control section; 31, for each USING, DROP, or control section break; and 5 to 6, for each internal symbol.	ISD FILE OVERFLOWED. ISD NOT PRODUCED.

Assembler Diagnostic Action

This section describes the format of diagnostic messages produced by the TSS assembler. It includes a description of error severity codes and error levels, and describes the effect of error severity upon requests to execute the assembled program. Refer to the publication *System Messages* for a description of each diagnostic and the source program errors that cause it.

All but a few of the diagnostic messages produced by the assemblers are issued in response to source program errors. In conversational mode, all diagnostics produced by the assembler appear on the terminal. In addition, messages from the conversational phases of the assembler for conditions which have been left uncorrected and all messages from the nonconversational phases of the assembler will appear in the output program listings, if any list option is selected. In nonconversational mode, all messages appear either in the output program listings or on SYSOUT if listing data set is not specified.

A few messages pertain to violations of assembler space and size restrictions and malfunctions in the assembler's operating environment. Some of the conditions which produce these messages also cause termination of the assembly and a return of control to the command-language level. The assembler does not produce object modules under conditions of abnormal termination; it will, however, place the terminating diagnostic message in the list data set, if one is available, and/or on SYSOUT.

The format of a diagnostic message is:

number   code   ***   text

When the assembler is used in conversational mode with prestored source data set, the actual line in which the error occurred is printed out immediately before the diagnostic message. Diagnostic messages produced after Phase I when running conversationally without a prestored source data set also cause the error line to be printed out at the keyboard.

The text for all messages produced by the assembler itself will be contained on one line. The text portion of messages produced by macro instructions through the MNOTE facility may extend to more than one line.

The "number" parameter is the source program line number of the first line of the statement to which the message applies. Messages concerning errors that the assembler does not associate with any specific statement carry the line number of the source program END statement.

The "code" parameter is a one-letter indicator of the severity of the error. The letters used, the severity of errors associated with each letter, and a brief description of assembler action taken are given in Table 11.

If a symbol is validly defined in a machine instruction or in a DC, DS, LTORG, or CCW statement, and the statement is then discarded for syntactic errors, the symbol will nevertheless be assigned the relocatable value it would have had, had the statement been correct.

Errors occurring in statements that are bypassed due to conditional assembly statements (AIF, AGO), do not produce diagnostic messages except in the case of an improperly formed sequence symbol in the name field.

When an assembled program is to be executed, the module named, and all modules called by this module, are inspected during the loading process to see whether any have been assembled with level-2 errors (severity code E). Any module containing an error level of 2 causes a diagnostic message naming the module and the error level to be printed on the user's SYSOUT.

Use and Structure of a User Macro Library

This section describes possible uses of a user macro library, the mechanism by which the TSS assembler operates on user macro libraries, and a detailed description of their creation and format. Up to seven macro libraries may be used in conjunction with the TSS Assembler. The libraries will be searched in the hierarchy specified by the DDEF sequence associated with ASM. Example 23 in Part III of this publication illustrates the procedure for building and using a user macro library.

Reasons for Using a User Macro Library

There are a number of ways in which user macro libraries may be of value. A few of these are listed below.

1. The same macro instruction is to be made available to more than one program or programmer. The macro instruction could be defined in each program, but a change in the macro definition would then require that each individual copy of the macro definition be changed rather than just one copy.

2. A modified form of a system macro instruction is to be used, employing the same macro instruction name as the system library macro instruction name. As the user macro library will be searched first, the modified form could be placed in the user macro library.

3. A program that must operate in more than one operating environment (TSS and OS or OS/VS, for example) is being written. The source program may

be identical in both systems if all code required to be different is contained in macro definitions.

4. Debugging output code is to be included in a program during checkout, and removed when the program is complete. The debugging code could, of course, simply be removed everywhere it appears in the source program. Another technique is to include all such code in user macro definitions, then change the macro definitions when the program is complete so they no longer produce the debugging code.

5. A program is being written that must interface with other programs, but the interface design is not yet firm. Programs on both sides of the interface may wish to place code in user macro definitions, so that when the interface needs changing all code associated with the change is centralized.

TSS Assembler Processing of Macro Definitions

When the TSS assembler encounters an operation code not defined as a machine instruction, the assembler does not produce a diagnostic labeling the operation code as invalid until it has been determined that the operation code is not a macro instruction. If the assembler is to know that a macro instruction is being used, a definition of the macro must be in one of three places. These places are listed below, in the order in which the assembler will look for them.

1. Macro instructions may be defined in the program using them.

2. Macro instructions may be defined in a user macro library.

3. Such macro instructions as CALL, SAVE, RETURN and all other macro instructions described in the publication *Assembler User's Macro Instructions* are defined in a macro library supplied with TSS and available to all users of this system. These macro instructions are referred to as "system" macro instructions. Macro definitions defined in sources 1 or 2 above will be used prior to any definitions in the system macro library.

Detailed Description of User Macro Library Creation and Format

The following paragraphs describe user macro library creation and format. This description applies equally to the system macro library.

A macro and COPY library is a collection of macro definitions and symbolic statements. It is from such a library that the TSS assembler retrieves and expands macro definitions when the corresponding macro instruction appears in a source program. The operand of a COPY instruction identifies the section of coding to be copied and included in the program currently being assembled.

Associated with each macro library is a macro library index. The entries in the index relate the name of each macro definition and group of COPY statements to its location in the macro library. Thus, source lines in the macro and COPY library can be located by matching the operation of the corresponding macro instruction or operand of the corresponding COPY statement to the appropriate entry in the index.

The first card of each macro or group of COPY statements in a library must contain a header character (normally a right parenthesis) as the first character followed by the macro or COPY name. This name has a maximum of eight characters; if less, it must be left justified. In source lines to be copied, the symbolic statements (for example a DSECT) begin at the second line. If the source lines are not naturally delimited (as a MEND statement delimits a macro definition), a delimiting statement must appear between items. For the TSS macro and COPY library the delimiter statement contains a right parenthesis in the first position of the text. It should be noted that the right parenthesis may also serve as the header character.

The second card of each macro definition must contain MACRO. This is followed by a macro instruction prototype statement, model statements (if any) and a macro definition trailer statement, i.e., MEND.

The macro and COPY library may be created and modified by the DATA and MODIFY commands. Alternatively, it may be created or modified by user-supplied routines using VISAM.

The organization and format of the symbolic component of the system macro and COPY library is shown in Figure 23. The format of each symbolic line, which is shown in Figure 24, follows that described for line data sets.

The lines of information within the symbolic component are ordered by line number. The number of the first line of each parcel (i.e., macro or group of COPY statements) is used to index the symbolic component.

Having established the macro and COPY library, the user must create the associated macro library index by executing either the SYSINDEX or SYSXBLD IBM-supplied service routines. Use of the SYSINDEX routine is illustrated in Example 23. The SYSINDEX routine requires that the macro definition and index data sets be explicitly defined for the task with respective ddnames

Table 11. Assembler Diagnostic Action

CODE	SEVERITY	DESCRIPTION	ACTION
blank	Informational message, Level-0 (no error)	The expansion of macro instructions may generate diagnostic messages if an MNOTE statement is encountered within the macro definition. The MNOTE statement allows a severity code to be associated with the message. When the value of the code is zero, the message is treated as a diagnostic for the purposes of printing at the terminal and inclusion in the diagnostic portion of the object listing; however, the message is considered to be informational only and does not contribute to the count of error messages or the error severity level of the assembled module.	Action is determined by the design of the individual macro definition.
W	Warning message, Level-1 error	A message with this code is produced under the following two conditions, which result in the associated actions:	
		1) The assembler detects a situation that either may not be as the programmer intended, or is incompatible with other assemblers.	1) The statement is assembled as written
		2) The assembler encounters an MNOTE statement with a severity code of 1 during macro expansion.	2) Action is determined by the design of the individual macro definition
E	Error message, Level-2 error	A message with this code is produced under the following circumstances, which result in the associated actions:	
		1) The operation code of any instruction cannot be identified or the syntax for the operands of machine instructions cannot be analyzed or the syntax for the operand field of assembler instructions cannot be correctly analyzed.	1) Statement is not assembled; however, symbols contained in name fields of such instructions are considered defined for the assembly.
		2) The syntax for the operands of machine instructions is correct but the values obtained for the various operands are incorrect.	2) The instruction is assembled but those subfields for which correct values could not be obtained are set to zero in the machine language text.
		3) The syntax for the operand field of assembler instructions is correct but the values or definitions for some of the operands are incorrect.	3) An attempt is made to process the correct operands (when there are more than one) and ignore the incorrect ones.
		4) An MNOTE statement with a severity code of 2 or greater is encountered during macro expansion.	4) Action is determined by the design of the individual macro definition.

Note: When an MNOTE instruction is encountered in the source program of a conversational assembly, the assembler will interrupt processing and prompt the user for corrections.

SOURCE and INDEX. The user must also specify the header character and the length of the macro name. The SYSINDEX routine receives the user's input parameters, prompts him for missing parameters and processes those parameters. It then calls the SYSXBLD routine which creates the index. Alternatively, the user can set up the parameters required and pass them directly by calling SYSXBLD in a problem program.

SYSXBLD makes a sequential pass through the entire macro and COPY library to determine, based on parameters supplied, which statements must have an index entry. The user-supplied header character will be compared with the first character of each symbolic statement to determine whether that statement contains an index entry. Or, the user may supply the name of a routine to be called after each symbolic statement is obtained. This routine must determine if a statement requires an index entry. If it does, the user routine returns to the appropriate library service routine with a name and associated line number that are to be placed in the index. If the user routine determines that a statement does not require an index entry, it must pass this fact to the service routine and request the next statement.

For detailed information on the use of SYSINDEX and SYSXBLD, see the publication *Assembler User's Macro Instructions*.

Figure 25 shows the format of the symbolic library index. The index component is a table that relates the name of each parcel to the number of its first line. It

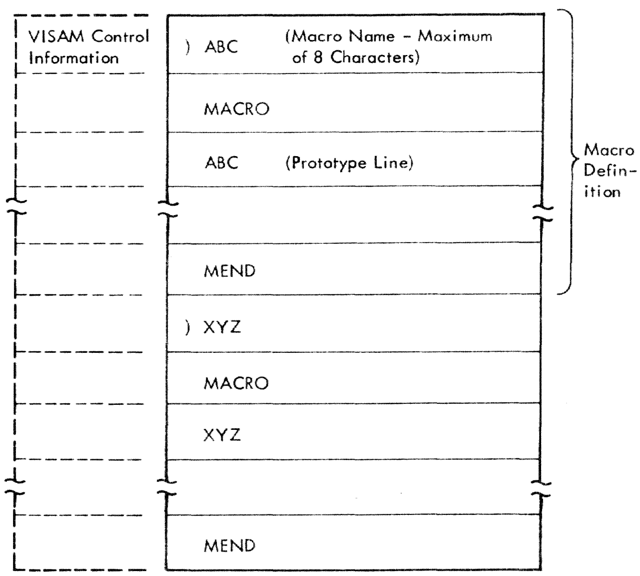


Figure 23. Format of a Macro Definition Symbolic Component

LL	LN	C	T
4 Bytes	7 Bytes	1 Byte	(LL-12) Bytes

LL is the length of the line including the LL field
C is a code whose values and their meanings are:

Code	*Meaning*
01	The line originated at a terminal keyboard
00	The line was obtained as a card image

Note: C is normally 00 for all lines of the system macro and COPY library
LN is the line number
T is the text of the symbolic line consisting of LL minus 12 characters

Figure 24. Format of a Line in a Symbolic Component

consists of a single record containing a header and as many entries as there are parcels in the associated library. The header contains information describing the index as a whole; each index entry contains a parcel name and retrieval information for the corresponding symbolic parcel in the associated library. Entries appear in ascending order according to the EBCDIC collating sequence of parcel names. Thus, any parcel in the system macro and COPY library can be located within the symbolic component by matching the operation of the corresponding macro instruction or operand of the corresponding COPY statement to the appropriate entry in the index.

Index Header

1. *Name Length:* a two-byte binary integer specifying the length, in bytes, of parcel names. In the IBM-supplied macro and COPY library, this value equals eight. The two high-order bytes of this word are reserved for future use and currently are set to zero.
2. *Index Length:* the location, relative to byte zero of the first index entry, of the first unused byte in the index. This value is used to indicate the length of the index.
3. *Search Starting Point:* the location, relative to byte zero, of the first index entry. This is the point at which the routine is to begin its binary search procedure.

Index Entry

1. *Parcel Name:* the parcel name, whose length in bytes is given in the header. It is left-justified and, if necessary, filled with trailing blanks.
2. *Retrieval Line Number:* the retrieval line number associated with the corresponding parcel in the symbolic library. The line number is given in EBCDIC and is right-justified with leading blanks.

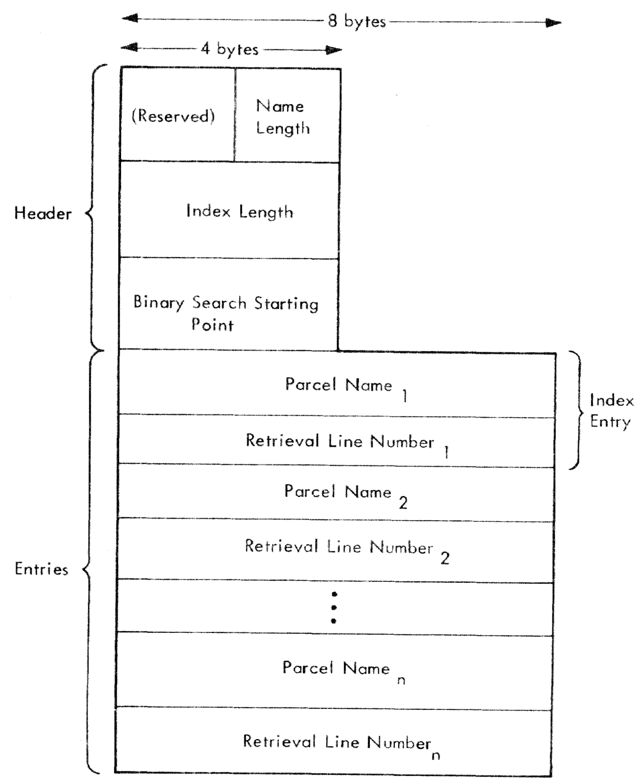


Figure 25. Format of Symbolic Library Index

Control Section Names and Attributes

Control sections are named to assist the assembler in assigning consecutive virtual storage locations to them during assembly. The consecutive assignment of virtual storage locations, once begun, is continued throughout assembly. Control section contents may be written in an intermixed manner. If the assembler detects several statements defining a particular type of control section, all containing the same name, it considers the first such statement as the beginning of the control section; the rest of the statements are continuations of that control section.

Only control sections with the PSECT attribute (described below) need be named. However, there may only be one unnamed control section in a source program module. As with named control sections, the unnamed control section is provided with a location counter; its contents are assigned consecutive virtual storage addresses throughout assembly.

In addition to controlling assembler address assignments, the identification of control sections

- Enables symbolic linkages, based on control section names, to be made between control sections.
- Allows the dynamic loader to allocate noncontiguous storage for different control sections of an object program module during loading.
- Allows dynamic control section rejection at load or link edit time.

Table 12 summarizes the ways in which control sections can be named and assigned attributes at source coding time.

The attributes of control sections describe the characteristics of the instructions and data they contain. Attributes are described at the control section level because the linkage editor and dynamic loader operate on control sections. These attributes can be specified by assembler users:

READONLY—The control section contains instructions and/or data that are not to be modified by a user. If this attribute is not specified, the control section is assumed to have a read/write attribute. READONLY control sections are allocated storage with a protection key that prevents the user from storing in the control section.

PUBLIC—The control section contains instructions and/or data that can be shared by other tasks if (1) the owner of the library containing the object program module that includes this control section issues an appropriate PERMIT command authorizing its sharing, (2) each sharer issues a SHARE command updating the system catalog so that the system can locate that library by each sharer's name, and (3) the owner and sharers define the library by a DDEF command (by specifying OPTION-JOBLIB in the command operand) in their respective tasks prior to attempting to use the object program module involved; the modules may not contain relocatable address constants.

If the public attribute is not specified, the control section is assumed to be private.

NOTE: If two users refer to the same public control section, both share the same physical copy. If two users refer to a private control section, each uses a separate copy.

PSECT—The control section contains modifiable storage (variable program data, save areas, or working storage areas). Control sections with the PSECT attribute are normally used for the modifiable storage associated with READONLY, PUBLIC control sections. Each such control section has its own private copy of the modifiable storage (PSECT) control section.

COM—The control section is used as a common storage area by independent assemblies that have been linked and/or loaded for execution as one overall program. The required storage area is allocated at assembly time.

PRVLGD—The control section is to be supplied with a storage protection key at load time, such that only privileged system service routines have access to it. If this attribute is not specified, the control section is assumed to be nonprivileged. This attribute is reserved for system routines resident in the SYSLIB.

VARIABLE—The length of the control section may vary during program execution. If this attribute is not specified, the fixed-length attribute is assumed. The number of pages allocated for variable-length control sections is determined by each installation and is specified at system generation time. Fixed-length control sections are allocated an integral number of pages (the minimum number that will contain the bounds of the control sections).

NOTE: The SYSTEM and PRVLGD attributes may also be specified by system programmer-users, if the control section is to be part of a system object program module. This attribute is never specified for problem programs. The user should keep in mind that control sections may be packed on double-word boundaries by the dynamic loader at execution time, if control section packing was designated at LOGON time. Only control sections with like attributes may be packed, however. This packing technique more efficiently utilizes virtual storage space, and is encouraged whenever practical.

Shared Object Program Modules

In TSS, shared object program modules normally contain one or more control sections with READONLY and PUBLIC attributes, and a prototype (PSECT) control section for the modifiable storage required by the READONLY portions.

A simplified format of a shared object program module is illustrated in Figure 26.

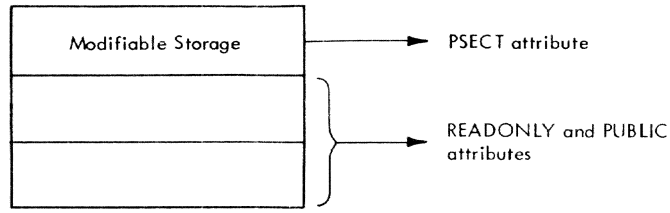


Figure 26. Shared Object Program Module

Table 12. Assembler Statements Used to Name Control Sections and Describe their Attributes

ASSEMBLER STATEMENT NAME FIELD	OPERATION FIELD	OPERAND FIELD	USE	REMARKS
Symbol or blank	START	Self-defining value or blank	May be used to identify first (or only) control section of object module; may be used if self-defining value is included in operand to specify initial virtual storage location counter value for first control section	Control section identified this way assumed to have fixed-length and read/write attributes but not these attributes: PRVLCD, PUBLIC, PSECT, SYSTEM, or COM
Symbol or blank	CSECT	READONLY PUBLIC PRVLGD VARIABLE SYSTEM blank (none of above)	Identifies control section without PSECT or COM attribute; is not a DSECT	Assembler assigns control section's attributes based on specification in operand field
Symbol	PSECT	READONLY PUBLIC PRVLGD VARIABLE SYSTEM blank (none of above)	Identifies control section containing address constants and save area, and/or working area	Assembler assigns PSECT attribute to control section; also other attributes specified in operand field
Symbol or blank	COM	READONLY PUBLIC PRVLGD VARIABLE SYSTEM blank (none of above)	Identifies control section serving as common storage area	Assembler assigns COM attribute to control section; also other attributes specified in operand field
Symbol	DSECT	blank	Identifies control section describing layout of storage area; does not actually reserve storage; storage area reserved by another statement	

Appendix B. Problem Program Checkout and Modification

The system elements that contain facilities for simplifying problem program checkout and modification are:

- Assembler
- Linkage editor
- Program control system

Assembler

The assembler includes conversational prompting and diagnostic facilities to assist the user in debugging his source program modules as he enters source statements at his terminal. It also includes optional facilities for:

- Storing and cataloging the source data set and object program module,
- Providing various listings,
- Including an internal symbol dictionary (ISD) in his object program module.

The ISD allows the user to employ the full capabilities of the program control system when the object program is subsequently checked dynamically during execution.

Prompting and Diagnostic Facilities

The diagnostic facilities available during source language processing vary with the manner in which the user has specified that source language processing is to proceed.

- As part of a conversational task in which the user enters his source statements from the terminal.
- As part of a conversational task in which the source statements are made available from a prestored data set specified by the user.
- As part of a nonconversational task in which the source statements are made available in the SYSIN data set.
- As part of a nonconversational task in which the source statements are made available from a prestored data set other than SYSIN, but which is specified by the user in the SYSIN data set.

NOTE: To be acceptable for language processing, a prestored source data set must have a line organization. If source statements are submitted conversationally, or if they form part of the prestored SYSIN of a task, a source data set will be constructed with line organization. Each physical line output to the system, whether as a single card or as a single line typed at the terminal, becomes a physical record of the line data set (input length is limited to 120 characters). Continuation conventions specified for commands do not apply to line data sets. Continuation conventions for combining two or more physical records into a single logical statement for a language processor are as specified by that processor.

Conversational Mode, Source Statements from Terminal

When the assembler is ready for a source statement, the system unlocks the user's keyboard and prints a line number at his terminal. The user then types in the contents for the line. The system stores both the line number and the line and then locks the user's keyboard.

If the syntax analysis indicates that the statement is correct, the system again unlocks the keyboard and prints the next line number at the terminal, so that the user can enter his next statement.

If the syntax analysis indicates that one or more parts of a statement are incorrect, a diagnostic message identifying each error is sent to the user's terminal. The system then types out the line in error and a # sign, and unlocks the user's keyboard so that the user can enter corrections. He can insert required lines between previously entered lines, replace erroneous lines, or delete lines.

When a user enters a correction line as an insert or a replacement, the first part of it must be a percent sign (%), followed by the appropriate line number. He then types a comma, and enters the correction. The % identifies the line as a correction rather than as the contents of the line for which the system entered a line number. Example: to replace line 500, the correction line might read

```
600        %500,        DC A(TRIAL)
```

If the user wishes to delete one or more lines, he must type a %D after the system-supplied line number, then a comma, and then the line number or range of line numbers to be deleted. *Example:* the correction line for a deletion entry might read

```
400   %D, 200
```
— for a single line

```
1800    %D, 900, 1100
```
— for lines 900-1100

The indicated lines are permanently removed from the source data set.

Each modification is stored by the system until all modifications are completed. A user restarting a long program may, thus, have a long wait before he can enter his next statement. To signal the system that he has entered all his modifications, the user enters a normal line (he does not enter % or %D), in response to the system's prompting. The corrections are then made by the assembler, after which the just-entered line is processed. If the assembler must restart, all uncorrected diagnostics will be reissued.

When the user enters an END statement, the assembler completes its first phase. If any diagnostic messages are issued at this point, the user is prompted for a decision: Does he want to terminate language processing, make modifications and restart, or continue language processing? The user may request that all further diagnostic messages, or solicitations for corrections, be inhibited, by typing 'I' or 'C' respectively when prompted and pressing the return key. Diagnostics will still be issued with the listing after completion of assembly, but the operation in conversational mode will not be needlessly impeded by messages and promptings should the user decide not to effect modifications at the keyboard. If he elects to modify and restart, the user repeats the above procedure after making modifications required by the diagnostic messages just received.

The second phase of the language processor is then executed. If any errors are detected during this phase, the assembler indicates the number of an erroneous line, but does not issue the line itself. If the user wants to see the actual contents of the line, he follows this procedure:

1. He presses the ATTENTION key to interrupt source language processing.

2. When the system prints an exclamation point (!), he types in the LINE? command and specifies the source data set name, together with the line number supplied in the diagnostic message.

3. After the line has been presented, he issues a GO command to resume source language processing from the point of interruption.

At the completion of this phase, the user is informed whether the assembler found no errors, minor errors, major errors, or errors that prevented it from producing an object module. The assembler will continue processing if it can; if it cannot, it will so inform the user.

NOTE: The user can terminate language processing at any time by pressing the ATTENTION button.

Conversational Mode, Source Statements from Prestored Data Set

In this form of language processing, successive lines of the source program module are fetched from the specified prestored data set. Communication, when source statement errors are detected, takes place between the user and the system via his terminal. The user's terminal is locked until diagnostic (or prompting) messages are produced.

If the system's syntax analysis indicates that one or more parts of the statement being processed are incorrect, the system prints out at the terminal the line in which the error occurred, followed by the diagnostic message and a number sign (#) at the beginning of the next line to prompt corrections. (The line printed out by the system may happen to be a continuation line. If the user wants to see the contents of some previous line he can press the attention key and then type in the LINE? command, specifying source data set name and desired line number(s).) The user may then proceed to correct his program, based on the diagnostic messages. He can add lines between existing lines, or replace or delete existing lines.

If he wants to enter a correction line as an insert or a replacement, he types in the line number of the line involved, a comma, and the actual correction. For example, to insert line 450 (between, say, lines 400 and 500), the insertion line might read:

```
#    450,    CATRD AREA+3, LENGTH
```

The correction line is permanently inserted by the system in the prestored source data set. The system then types out another # at the beginning of a new line, and unlocks the user's keyboard.

If the user wants to delete one or more lines, he types a D following the #, then a comma, and the line number or range of line numbers to be deleted.

```
#    D,    400          — to delete line 400
#    D,    900, 1100    — to delete lines 900-1100
```

The lines to be deleted are permanently removed from the prestored source data set.

The user may decide at some point that he has too many errors in his source program to try correcting them conversationally but wishes to allow the assembly to continue without further diagnostic messages coming to his terminal to slow down the process needlessly. In that case he can inhibit all further diagnostic messages by typing the letter 'I' in response to the number sign (#) and pressing the return key. If he wishes to continue receiving diagnostic messages but not be prompted for corrections until the completion of the SOURCE.data set scan, he can type in the letter 'C' at the terminal and press the return key. He may also elect to ignore only the current error message by pressing the return key. In all cases, all unsatisfied diagnostics are included in the LIST.dataset when the first phase of the assembly process is completed.

To signal the system that he has entered all the corrections required, in response to the diagnostic messages for the previous statement, the user responds to the # with a carriage return. The system processes the correction lines and then retrieves the next line of the prestored source data set. This may cause further diagnostic messages and a repetition of all previously issued messages.

After the END statement of the source data set has been processed, the system and user communicate in the same way as described about for "Conversational Mode, Source Statements from Terminal."

Nonconversational Mode, Source Statements from SYSIN

In this situation, there is no system communication with the user during language processing. The source data set is read, one statement at a time, from the SYSIN data set. As each statement is read, a line number is prefixed to it, to serve as the key by which the line can be identified later. The new data set created in this way can be modified, or otherwise used after language processing is completed. Any diagnostic messages are sent to the task's SYSOUT data set.

Nonconversational Mode, Source Statements from Prestored Data Set

This is essentially the same as the previous type of language processing, except that the source program module already exists as a line data set. The system picks up the source statements, line by line, and processes them. No corrections are made, and any diagnostic messages are written on the task's SYSOUT data set for later analysis by the user.

Program Listings and Related Aids

The user can specify that any, or a combination, of the following be made available as a result of source language processing:

1. Listings
   - Object program module listing
   - Source data set listing
   - Cross reference listing
   - Edited symbol table
   - Internal symbol dictionary listing
   - Program module dictionary listing
2. Internal symbol dictionary

NOTE: A cross reference listing and an edited symbol table cannot both be requested.

Linkage Editor

In addition to its basic function(static linking of object program modules), the linkage editor can

- Control the libraries from which input object program modules are to be obtained, and the order in which searches occur to satisfy unresolved references during linkage edit or input processing.
- Provide an automatic search (of the libraries in the program library list) at the completion of linkage editor input, to satisfy all unresolved external references (where resolution has not been explicitly excluded during linkage editor input).
- Replace, delete or rename control sections within modules. (Automatic rejection of control sections occurs when more than one section has the same name; the first control section received as input is retained in the object program module; all others subsequently detected with the same name are ignored.)
- Rename or delete entry points within an object program module.
- Change the attributes of control sections within an object program module.
- Combine two or more control sections of an object program module, thus reducing the number of virtual storage pages required.
- Collect automatically and include a reserved storage area within the output object program module, for common control sections received as input.

Prompting and Diagnostic Facilities

The linkage editor may be run under the same four conditions as the language processors. It also issues prompting and diagnostic messages in the same way, and the user can correct control statements in the same manner as for source statements.

Program Listings and Related Aids

The user may optionally specify that either or both of these be produced by the linkage editor:

1. Internal symbol dictionary for the output object module.
2. Program module dictionary listing.

The linkage editor automatically prepares a list of the symbols that cannot be resolved by automatic calls, and those symbols whose resolutions are deferred to the dynamic loader.

Object Program Module Linking

Time Sharing System Program Structure

In TSS, a problem program, at execution time, may be a single object program module, or a series of object program modules that are linked together.

Symbolic Linkage

Symbols may be referred to (used as an operand in a statement) in one control section, and be defined (used as the name of a statement) in other control sections to establish linkages between those control sections.

External References: A control section may contain external references (the symbols that are referred to in control sections of one object program module, but defined in control sections of separately assembled object program modules). Symbols that are external references are used in source program modules to

- Identify an entry point in another object program module or
- Identify the location of data (as a table) that is contained in another object program module.

External Symbols: Each object module has at least one external symbol (a symbol that can be used as an external reference in another module). These are valid external symbols in TSS object program modules:

- The module's name (standard entry point of the module).
- Name of any control section in the module, including common blocks; if blank COMMON is declared, the name is a name of blanks; if an unnamed control section is declared, its name is a name of hexadecimal 0s.
- Any symbol that is included in an ENTRY statement in the object program module and is used as a name in any statement, except those in dummy sections.

External Symbol Values: The values associated with each external symbol are V-value and R-value.

The V-value specifies the location at which execution of the object program module is to begin when control is transferred to that object program module. This is the conventional external symbol value.

These are the V-values for external symbols:

EXTERNAL SYMBOL	V-VALUE GIVEN CALLING PROGRAM
1. Module name	1. Virtual storage location of expression included in END statement in called program module; or, if END statement is blank, origin of first CSECT in called program module
2. CSECT name	2. Virtual storage location of origin of named CSECT in called program module
3. Symbol in operand of ENTRY statement	3. Actual virtual storage location of symbol in called program module

These R-values specify various locations, depending upon the type of external symbol specified:

EXTERNAL SYMBOL	R-VALUE GIVEN CALLING PROGRAM
1. Module name	1. Virtual storage location of origin of PSECT control section (if there is one) in called program module; if called program module does not contain PSECT control section, R-value gives virtual storage location of origin of first CSECT in called program module
2. CSECT name	2. Virtual storage location of origin of named CSECT (same as V-value)
3. Symbol in operand of ENTRY statement	3. Virtual storage location of origin of control section containing ENTRY statement in called program module; if called program module contains PSECT, ENTRY statement should be in PSECT even though symbol is defined in another control section

An illustration of the rules is given in Figure 27. Module M consists of two control sections: a CSECT (X) and a PSECT (Y). It has a standard entry point (W) and a deferred point (Z).

Note in the references to module M *by entry point* (W and Z), how the V-value indicates the location to which control is transferred, and the R-value gives the location of the PSECT. If module M were a shared program module, CSECT X would have READONLY and PUBLIC attributes, and a single copy of this control section could be used by any task permitted to share it. However, each sharing task would be given a separate copy of the PSECT, and for each copy there would be a separate R-value indicating where that task's private copy of the PSECT is located. Each calling program could then pass that address to module M, to be used as its private variable area, when module M is executed on its behalf. *Example:* Assume that module M is shared by task 1 and task 2. Also, assume that module A of task 1 and module B of task 2 are in main storage simultaneously and both are using module M, with task 2 making the first reference. Main storage might appear as in Figure 28.

Linkage Conventions

Standard linkage conventions have been defined to govern the communication between all TSS programs.

- Type I—Between two nonprivileged or between two privileged programs.

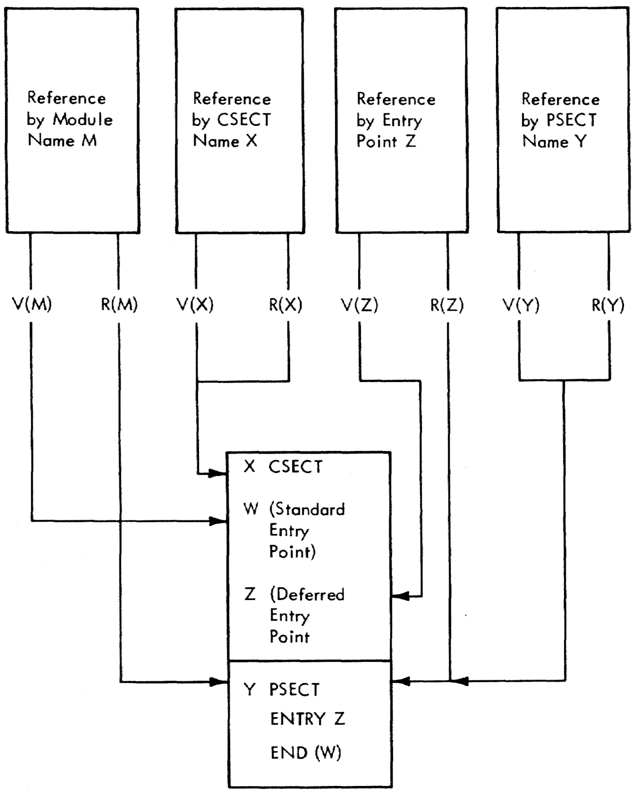


Figure 27. V- and R-Values of External Symbols

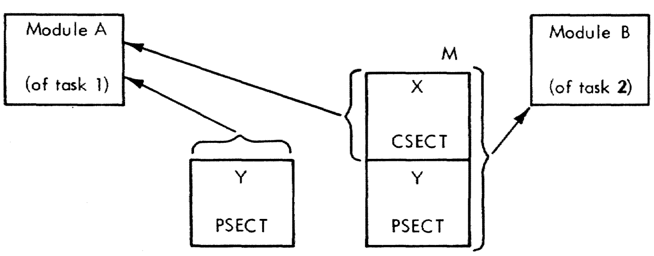


Figure 28. Sharing a Module

- Type II—From a nonprivileged to a privileged program.
- Type III—From a privileged to a nonprivileged program.

Only the type-I linkage between nonprivileged programs is covered here; type-I linkage between privileged programs is described in *Assembler User Macro Instructions*. Types-II and -III linkages are described in *System Programmer's Guide*.

Type-I linkage conventions include three basic standards to which the assembler user must adhere:

1. Using the proper registers in establishing a linkage;
2. Reserving a parameter area in the calling program, to which the called program may refer;
3. Reserving a save area (in the calling program) in which the called program may save the contents of the calling program's registers.

Proper Register Use: Four general registers are used for the type-I linkage between nonprivileged programs.

- *General register 1*—Set up by the calling program to give location of the parameter list to be passed to the called program.
- *General register 13*—Set up by calling program to give location of save area to called program.
- *General register 14*—Set up by calling program to give called program the location of standard return address in the calling program.
- *General register 15*—Set up by calling program to indicate location to which control is to be transferred in the called program (V-value); on return to the calling program, the called program may supply a return code to the calling program in this register.

Reserving a Parameter Area: Every calling program may reserve a storage area (parameter area) in which the parameter list used by the called program is located. The first entry in a variable-length parameter area contains the length (in bytes) of the entire parameter area. Each succeeding entry contains the address of an argument to be "passed" to the called program.

Reserving a Save Area: Every calling program must reserve a storage area (save area) in which certain registers (those used in the called program and those used in the linkage to the called program) are saved by the called program.

The minimum amount of storage needed for the save area of a program that is both calling and called, is 19 words. Table 13 shows the layout of the save area and the contents of each word.

NOTE: It is the responsibility of the called program to maintain the integrity of general registers 2 through 12, so their contents will be the same at exit as they were at entry to the called program. It is the calling program's responsibility to maintain the floating-point registers around a call. General registers 0, 1, 13, 14, and 15 must conform to the indicated conventions.

Linkage Macro Instructions

The CALL, SAVE, and RETURN macro instructions provide linkage between object program modules; the RETURN and EXIT macro instructions define the end of program execution.

CALL Macro Instruction: The forms of the CALL macro instruction are implicit CALL and explicit CALL. The principal difference between the two forms is the time at which the called program is brought into virtual

Table 13. Save Area Contents

WORD LOCATION	CONTENTS
1	Contains length of save area
2	Address of calling program's save area; field is set by called program in its own save area
3	Address of next save area; that is, save area of program to which this program refers
4	Contents of register 14 containing address to which return from this program is made; field is set by called program in calling program's save area
5	Contents of register 15 containing address to which entry into this program is made; this field is set by called program in calling program's save area
6	Contents of register 0
7	Contents of register 1
8 . . . 18	Contents of register 2 . . Saved by called program . Contents of register 12
19	Address of the called program's PSECT belong to calling program (R-value)
20 ff	User data

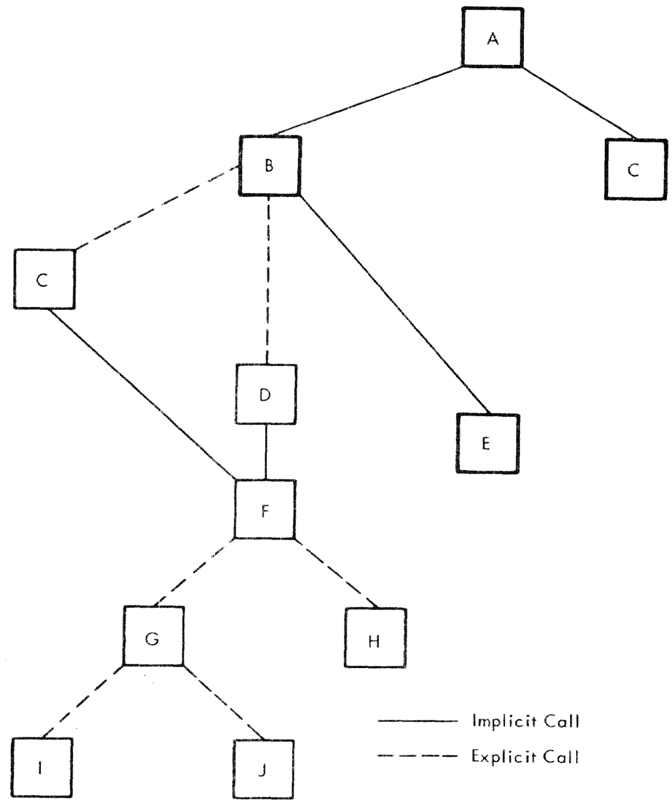


Figure 29. Program with Implicit and Explicit Linkages

storage. Consider the program illustrated in Figure 29. An implicit linkage between module A and module B means that allocating A implies allocating B. In other words, when A is allocated, so is B. Explicit linkage, between two modules (e.g., B and C in Figure 29) means that C is not to be loaded unless, in the execution of B, C is actually (or explicitly) required.

The usefulness of the differentiation of implicit and explicit linkage is illustrated by Figure 29.

- Linkages are provided so that any combination of the object modules needed for any conceivable run can be selected.
- If only A and C, or A, B, and E are needed in a given run, none of the others need be allocated; C, B, and E would be allocated with A, each time A is allocated.

SAVE Macro Instruction: The SAVE macro instruction may be used as a convenient means of automatically storing the contents of registers in the calling program's save area. It may be written at every entry point of the called program. An entry point identifier may be specified in the SAVE macro instruction to identify the entry point to which control is to be transferred.

RETURN Macro Instruction: The RETURN macro instruction is placed in called programs at the points where control is to be transferred back to the calling program. Used in this way, RETURN is also a convenient way of restoring the registers of the calling program that were saved by the SAVE macro instruction.

The RETURN macro instruction is used in the first object program module of a program to indicate the end of the program's execution.

EXIT Macro Instruction: The EXIT macro instruction terminates a program and switches the task to command mode.

ABEND Macro Instruction: System programs use the ABEND macro instruction to terminate a task when an uncorrectable error occurs. The user may use ABEND as an error exit in his program.

A more detailed description of these macros is contained in *Assembler User Macro Instructions.*

Object Module Combination

If a program consists of one or more object program modules (see Figure 30), the user can:

- Statically combine two or more (or all) object program modules prior to program execution by using the linkage facilities of the linkage editor program, or
- Dynamically link the object program modules during execution of the program by making use of the automatic facilities of the system's dynamic loader.

Static Linking

Static linking, although optional, may be used for object program module build-up during program development. It may also be used to combine a number of short object modules together and thereby save on paging time during program execution. A similar saving of page space may be achieved by specifying the control section packing option for the dynamic loader during LOGON, and the user will probably find dynamic linking through the dynamic loader more convenient than using the linkage editor. He may, however, find the editing facilities of the linkage editor to be of value in program development. (*Example:* He may want to update object program module information without recompiling or reassembling.)

In cases where the linkage editor is used for object program module combination, the object program modules to be combined may be in one of the libraries included in the program library list, or they may be in other libraries identified in the control statements that are input to the linkage editor program. On the other hand, all object modules to be dynamically linked must be contained in libraries currently identified on the program library list.

When the linkage editor is to be invoked a number of times consecutively in one task, time may be saved by running in express mode. This will cause the language processor control to read the name of the next output module from SYSIN whenever it would normally have returned to the Command System for another command. The express mode is turned on by issuing a command

DEFAULT LPCXPRSS=Y

anywhere before the LNK command is issued. The LNK command is issued only once, for the first linkage edited object program, and all parameters are issued at the same time. The link editing parameters cannot be changed for subsequent object modules.

The express mode can be turned off by entering an underscore as first character in a line; this will cause an exit from the language processor control and a return to the command system. It is also possible to turn off the express mode by pressing the attention button any time during the link editing process and issuing a command

DEFAULT LPCXPRSS=,

Link editing can then be continued in a non-express mode by issuing a GO command.

If an invalid module name is entered when the language processor control expects a new object module name, the express mode will be turned off and a diagnostic message will be issued.

Conversational Linkage Editing: To initiate conversational linkage editor processing, the user issues these commands (refer to Figure 30):

- DDEF or CDD command—Defines each library to be used during execution of the linkage editor program.
- LNK command—Loads and initiates linkage editor processing.

When LNK is entered, the necessary parameters must be included. Parameters not included will assume system default values, where applicable. The parameters available are:

- Name of the object module being created.
- Version identification of the module being created.
- An indication of whether the control statement data set is prestored or is to be made available via SYSIN. If that data set is not prestored, the user can also specify its starting line number, and the value by which the lines are to be incremented (values of 100 and 100 are assumed, if the starting line number and increment value are not specified).
- Name of the library in which the new object program module is to be stored. If this library is not specified, the object program is placed in the library that is currently at the top of the program library list.
- An indication of whether an internal symbol dictionary (ISD), or a program module dictionary (PMD) listing is wanted.

When these parameters have been provided, linkage editor processing of the control statement data set begins. The user can issue the control statements from his terminal in response to system prompting, or he can make the control statements available from a prestored data set. The user must observe the continuation conventions for linkage editor control statements; prompting and diagnostic facilities are available from the system to assist him in entering his control statements correctly.

To combine object program modules, the user specifies the appropriate INCLUDE control statements. Any external references in the input modules that were not satisfied (or excluded) by INCLUDE control statements, are satisfied at the end of the run by automatic call. Automatic call searches the program library list to satisfy the remaining external references. All required modules, except those in SYSLIB are placed in the output module. Linkage editor forms one PMD for the resulting object program module and, if the ISD option is specified, a chained ISD for all input object modules.

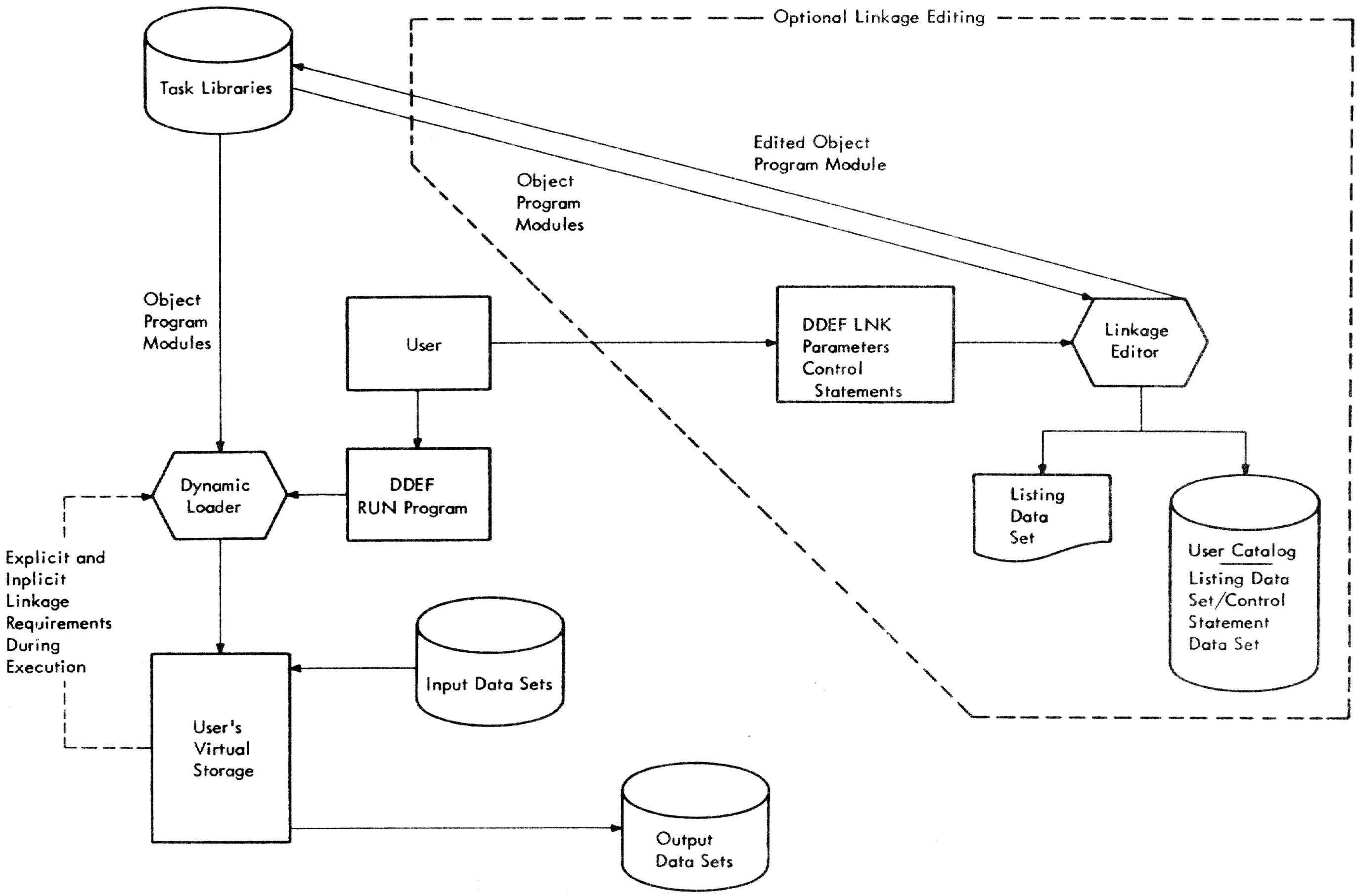


Figure 30. Object Program Module Combination

At the conclusion of linkage editing, the system stores the object program module, by its module name, in either a library specified in the parameters defined for the linkage editor run or, if no special library has been specified, the module is stored in the library currently at the top of the program library list. The original input module names are retained as auxiliary entry points.

Nonconversational Linkage Editing: The same commands are used to initiate nonconversational linkage editing. The user must be careful to store the DDEF commands, required to define the libraries, ahead of the LNK command (which must contain the desired parameters), in the SYSIN data set. However, the user has the option of making the control statements available either in the SYSIN data set, or as a prestored data set.

Dynamic Linking

Object program modules are linked dynamically during execution by the dynamic loader. To initiate program execution, the user issues these commands (see Figure 30):

- DDEF or CDD command—Defines each data set to be processed by the program.
- Either a LOAD command followed by CALL command (or an implicit CALL) or merely a CALL command—Loads and initiates execution of the specified object program module.

LOAD Command: When the user issues a LOAD command, the first object program module of the user's program is explicitly loaded in the user's virtual storage. The dynamic loader will allocate space in the user's virtual storage for the module named in the LOAD command. It then allocates space for all the object program modules implicitly called by that module, and all other modules implicitly called by them. Modules that are explicitly called are not known to the loader at this time.

NOTE: The LOAD command need not be issued prior to CALL or implicit CALL. The LOAD command, however, is useful for placing object modules in virtual storage to examine them prior to execution.

CALL Command: When the CALL command is issued, the first object program module and all implicit modules are allocated space in the user's virtual storage, if they have not already been loaded. When execution begins, pages are loaded into main storage as needed.

If, during execution of the first object module, a reference is made to another module and the linkage is implicit, these events occur:

1. The linkage is completed so that control can be transferred.
2. The required pages of the new module are brought into main storage (from its library), relocated (if necessary), and then are given control.

Explicit linkage to another module is detected when an explicit CALL macro instruction is executed. Then, the dynamic loader action is similar to that of the LOAD command (it allocates space for the explicitly called object module, for all other modules it implicitly calls, and for all modules they implicitly call). However, when the allocation is completed, control is transferred to the explicitly called object program module.

LOAD Macro Instruction: The LOAD macro instruction allocates object modules during program execution when a module is loaded; it is not given control. This procedure is useful for inserting PCS commands and statements into modules before execution.

UNLOAD Command: The UNLOAD command deletes object program modules and all linked modules that will not be needed by other tasks.

DELETE Macro Instruction: The DELETE macro instruction is used during program execution to perform the same function as the UNLOAD command.

Program Control System

The user can employ program control commands and statements to perform one, or any combination, of these operations:

1. Request output of data fields and instruction locations within his program, specifying them by their symbolic names in the source language program or by their virtual storage addresses; he can also request output of machine register contents, specifying the registers by type and number.
2. Modifying variables within his program, specifying them by their symbolic names or by their virtual storage addresses, and specifying the new value for each variable in standard representation as an integer, floating-point number, or character string.
3. Specify, either symbolically or by virtual storage address, instruction locations within his program at which execution is to be stopped or started. When program execution has been stopped, the user may intervene, as described in items 1 and 2, before he directs resumption of program execution.
4. Specify, either symbolically or by virtual storage addresses, instruction locations within his program at which the actions described in items 1 and 2 are to be performed automatically.
5. Obtain the values of his program's variables at a specified point in its execution, with the variables automatically formatted according to their types.
6. Establish logical (true or false) conditions which allow or inhibit the actions described in items 3, 4 and 5.

The use of program control system facilities does not involve any restrictions on the user relative to

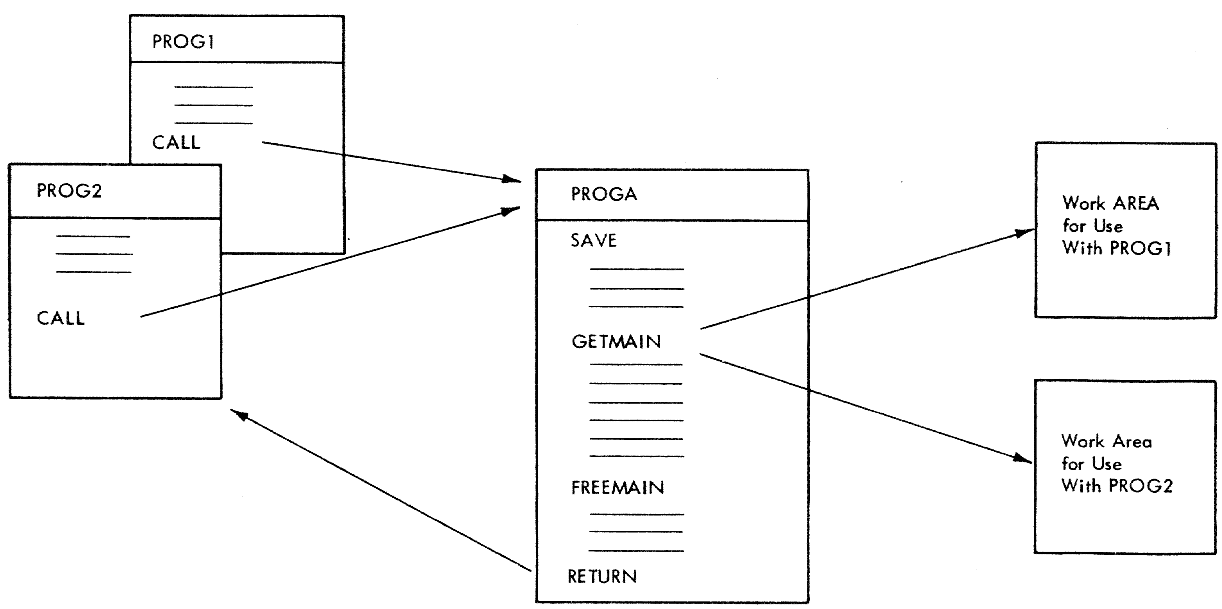


Figure 31. A Reenterable Routine that Requests its own Temporary Storage

source coding. In general, the use of program control facilities will greatly simplify the preparation of source programs, because many functions previously source-coded may conveniently be made available after compilation. Typical of these: the routines used for debugging programs, and the conventional I/O statements usually included in source programs.

Program Control Commands

The program control commands (DISPLAY, DUMP, and SET; CALL, GO, BRANCH, and STOP; AT; and QUALIFY and REMOVE) may be issued individually, or (except for QUALIFY and REMOVE) may be combined into program control statements.

The DISPLAY and DUMP commands allow the user to obtain the values of variables, the contents of machine registers, and the contents of specified virtual storage locations. These two commands differ in that the DISPLAY command delivers the designated information to SYSOUT (the terminal in conversational mode); the DUMP command delivers the designated information to the PCSOUT data set which must be subsequently transferred to an output medium (by the PRINT command). The SET command allows the user to change the contents of machine registers, or the values of variables within his program.

The STOP command interrupts execution of the user's program and outputs the instruction location at which the interruption was honored. Thus, except for this output information, the STOP command, used alone, is equivalent to the operation of the ATTENTION button at the user's terminal. The STOP command is more useful when included in a statement that designates the instruction locations at which execution is to be interrupted.

The CALL command, used after a LOAD command, initiates execution of the loaded program, either at its entry point or any other specified point. The CALL, GO, and BRANCH commands may also be issued after a STOP command to perform any of the following:

- Resume program execution from the point of interruption (GO).
- Resume program execution at a point other than the point of interruption in the current program (BRANCH).
- Load and initiate execution of another program (CALL).

The AT command, issued individually, does not interrupt program execution; it simply informs the user when execution of his program has reached the instruction locations designated in the command.

When the AT command is included in a program control statement, it designates the instruction locations at which the actions specified in the statement are to be performed. Program control statements that include one or more AT commands are dynamic statements.

When an AT command is accepted by the system, it is assigned a number to identify it uniquely, and that number is printed at the terminal. This is done whether the AT command is individually issued, or is the first of a string in a statement. When an AT command subsequently becomes effective, the standard output presented to the user includes the AT command's identification number.

The QUALIFY command allows the user to designate, before he refers to a group of internal or external symbols, the program in which these symbols are defined; he may, thereafter, refer to these symbols without explicitly qualifying them by program name. If the user does not issue the QUALIFY command, he must prefix each symbol with the program name. If the user is uncertain of his qualification, he may issue the command: "DISPLAY .(0, 1)". PCS will supply the symbol last named in a QUALIFY command and display one byte. If no QUALIFY command has been issued during the task, L'0' is the assumed qualification.

The REMOVE command enables the user to delete previously issued dynamic statements (those which include one or more AT commands), thereby terminating their subsequent execution.

Program Control Statements

The program control commands just described (except QUALIFY and REMOVE) can be combined into program control statements, either to request immediate execution of several PCS commands with one entry, or to request deferred execution of one or more PCS commands. Deferred execution of the actions specified in any statement is achieved by including an AT command in the statement. The AT command makes it a dynamic statement, in the sense that its execution is dependent upon arrival at the instruction locations designated in the AT command.

Execution of a program control statement, either immediate or dynamic, can also be made conditional by inclusion of an IF command. The IF command defines a logical expression (two-valued, or true-false) that must be true to allow execution of the statement's actions. When a dynamic statement includes an IF command, the IF command is evaluated only when control arrives at the instruction locations designated in the AT command.

This is the general format of a program control statement

(AT) . . .;(IF);(DISPLAY) . . .;(DUMP) . . .;(SET) . . .;(BRANCH)
(STOP)

The rules for the inclusion of program control commands in program control statements are:

AT: None, one, or more than one AT command may be included in a statement. If included, the AT commands must be the first entered in the statement.

IF: The IF command must be entered after any AT commands in the same statement. Multiple IF commands may be entered in a single statement.

DISPLAY: None, one, or more than one DISPLAY command may be included in a statement. If included, they must follow any AT commands and/or the IF command entered in the same statement. The DISPLAY command is performed (together with any DUMP command in the same statement) before any other action commands specified in that statement.

DUMP: None, one, or more than one DUMP command may be included in a statement. If included, they must follow any AT commands and/or the IF command, if present. The DUMP commands may be entered before or after other action commands (except BRANCH or STOP) in the statement. SET commands, if present in a statement, are performed after any DISPLAY and DUMP commands in the statement, but before BRANCH or STOP.

BRANCH: Only one BRANCH command may be entered in a statement; it must be the last command or entry in the statement. If the BRANCH command is included in a statement, the STOP command may not be included. If other action commands are included in the same statement, they are performed before the BRANCH command is executed.

SET: The SET command enables the user to change the contents of a specified data location to a new value. When the command becomes effective, the new value of the data location is produced in the same format that would be produced if the name of the data location had appeared in a DISPLAY command. The output is produced from the changed field itself and reflects the results of all conversions and expression evaluation.

STOP: Only one STOP command may be entered in a statement. If included, it must be entered after any AT commands and/or the IF command in the same statement. If the STOP command is included in a statement, the BRANCH command may not be included. If other action commands are included in the same statement, they are performed before the STOP command is executed.

PCS and the Internal Symbol Dictionary

When the user selects the ISD option in the ASM parameters, the assembler includes the internal symbol dictionary as part of the object module. The ISD contains the length and type attributes and the location of each symbol that appears in the "name" field of a statement in the source program (note exceptions below).

PCS uses the information in the ISD to determine the location of an instruction or data area in virtual storage, to select the proper conversion and format when a variable is displayed or dumped, and to determine the method of arithmetic to be used in evaluating a SET or IF expression.

PCS recognizes the following type attributes:

- Immediate values (absolute EQU statements)
- Instructions
- Character
- Integer (halfword and fullword)
- Floating point (single- and double-precision)
- Address constants
- Hexadecimal
- Relocatable EQU statements
- Names on LTORG statements

All other type attributes are treated as hexadecimal data. In addition, if the user designates, in his source program, a length attribute for a type having an implied length (i.e., HL2), the type attribute for the symbol is recorded in the ISD as hexadecimal. The reason for this is that the assembler does not force boundary alignment when a length attribute is specified. Since the data area, therefore, may be unaligned, it cannot be considered as having the characteristics that the type attribute implies.

When an offset is specified with an internal symbol, the type is considered hexadecimal, and the length, unless designated with the offset, is treated as one byte.

The assembler statements listed below are not included in the ISD; therefore they may not be referred to in PCS statements.

- Complex EQU statements
- Local and global variable symbols
- Sequence symbols
- Names on macro instructions that are not carried forward in the expansion.

Symbols in DSECTS, although recorded in the ISD, should not be referred to in PCS statements.

If PCS statements are to refer to internal symbols, the original object module must have included an ISD.

When object modules are link-edited and the ISD option is selected, the linkage editor automatically retains each module's ISD. In addition, a new ISD is formed, allowing PCS to trace back from the new composite object module to each original control section.

Using PCS Without an ISD

When an ISD is not selected at assembly time, the user is restricted to the use of external symbols in his program control statements. PCS commands can refer to lines in the source program by using the control section name with a hexadecimal offset equal to the location shown on the object listing.

If PCS statements are to refer to internal symbols, the original object module must have included an ISD.

No conversion or formatting is done for variables that are referred to by external symbols. The type is considered to be hexadecimal, and, unless an explicit length is specified, one byte is assumed as the length.

Evaluating Expressions

When two operands are joined by an operator to form an expression, the length and type attributes of both are used to determine the method of arithmetic to be used in performing the operation. Integer, floating-point, or logical arithmetic is selected. However, logical (i.e., address) arithmetic is performed only when requested by the user in response to the system's prompting.

When it is not possible to determine the arithmetic to be used from the type and length of the operands, the conversational user is prompted to supply the method. In a nonconversational task, since the expression cannot be evaluated, the PCS statement is rejected.

References to variables that are not aligned on the proper boundary result in a warning diagnostic. However, the operation will be successfully performed without a specification interrupt. PCS automatically provides intermediate moves to proper word boundaries.

Table 14 illustrates the possible combinations of operands for arithmetic and relational operations. Logical operators are not included since they are always performed in a general register and must be one, two, or four bytes in length.

Program interrupts can occur any time an expression in a PCS statement must be evaluated. These interrupts are recognized as being caused by a PCS statement and not by the user's object program. Five such interrupts can occur:

1. Fixed-point overflow exception
2. Fixed-point divide exception
3. Exponent overflow exception
4. Exponent underflow exception
5. Floating-point divide exception

When any of these interrupts occurs, a diagnostic is issued and the action requested in the PCS statement is not performed. These interrupts are not recognized by the user's program interrupt routine if one has been specified.

Floating-Point Constant Conversion

The assembler and PCS use different methods to convert floating-point constants to their internal binary values. As a result, the two processors may develop two slightly different internal values for the same floating-point number. Thus, although a single-precision floating-point number gives up to seven decimal places of precision, the user should assume only six decimal places of precision between the two processors. With double-precision numbers, a maximum of sixteen decimal places should be assumed instead of seventeen.

This difference in conversion techniques should also be kept in mind when using PCS IF and SET commands to debug an assembler language program. For example, if an IF command were used to test the value of an object-module variable which was initialized with an assembled floating-point constant, equality might not occur because different internal values were obtained by the two processors for the floating-point number.

To avoid the possibility of such a problem, the user should take into consideration the allowed variation between the two numbers being tested. One method to use when testing two numbers for equality is to take the absolute value of the difference against the allowed variation. (A similar problem might develop when a floating-point variable that is to be used for comparison by the object program is initialized using a floating-point number in a SET command.)

If the programmer has access to an assembly that indicates the internal value to which a floating-point number has been converted by the assembler, the problems resulting from different conversion techniques may be avoided by using the hexadecimal equivalent of the internal value in the IF or SET command.

PCS Diagnostics

PCS examines each statement for validity and issues diagnostics alerting the user to errors.

Diagnostics usually are issued immediately upon receiving the command. In conversational mode, the user can reenter the statement with the necessary correction made. Nonconversationally, the user has no chance to correct errors; a diagnostic is printed on SYSOUT and the PCS statement is simply ignored.

Certain errors are not detected until execution has begun. These errors are the result of an action the user has requested in a dynamic PCS statement (i.e., one containing an AT command). In a conversational task, after the diagnostic is issued, the terminal is placed in command mode. The user can then REMOVE the erroneous statement, reenter it correctly if he desires, and continue execution with a GO. If he wishes to perform the corrected statement immediately, he must use the operand of the AT statement as the operand of the BRANCH.

In a nonconversational task, the diagnostic is written on the SYSOUT data set and the next command is read from SYSIN. This may result in prematurely terminating program execution.

Miscellaneous Considerations

CALL, GO, and BRANCH Commands

When the CALL command specifies an external symbol as an operand, the object module defining that symbol is automatically loaded if it has not been previously loaded. If, however, there is a serious error in loading the module, the CALL command is rejected. Serious errors are caused by level 2 errors when the module was assembled, as described in Appendix A, under "Assembler Diagnostic Action"; or dynamic loader errors as described in Appendix C, under "Recovering from Errors when Dynamically Loading."

If the error condition does not preclude execution of the object module that was loaded, the CALL command may be reissued to initiate execution. The module name must be repeated on the second CALL.

If the user anticipates that such errors will occur, he should first issue a LOAD command naming the module, followed by a CALL with operand. This method ensures that program execution will always be initiated. This is most important to the nonconversational user, since it is not always possible to anticipate loading errors. The conversational user may use the LOAD and CALL procedure, or a CALL followed by a duplicate CALL command if the first CALL is rejected.

Table 14. Possible Combinations of Operands for Arithmetic and Relational Operations

OPERAND 2 \ OPERAND 1	H	F	E OR SINGLE-PRECISION REGISTER	D OR DOUBLE-PRECISION REGISTER	OTHER TYPES LENGTH = 1	OTHER TYPES LENGTH = 2	OTHER TYPES LENGTH = 4 OR GENERAL REGISTER	OTHER TYPES LENGTH = 8	OTHER TYPES LENGTH OTHER THAN 1, 2, 4, OR 8 BYTES
H	I	I	F703	F048	I	I	I	F048	F048
F	I	I	F705	F049	I	I	I	F049	F048
E or Single-Precision Register	F703	F705	E	D	F703	F703	E	D	F048
D or Double-Precision Register	F048	F049	D	D	F048	F048	D	D	F048
Other Types Length = 1	I	I	F703	F048	F703	F703	F703	F048	F048
Other Types Length = 2	I	I	F703	F048	F703	F703	F703	F048	F048
Other Types Length = 4 or General Register	I	I	E	D	F703	F703	F705	F049	F048
Other Types Length = 8	F048	F049	D	D	F048	F048	F049	a	F048
Other Types Length Other Than 1, 2, 4 or 8	F048	F048	F048	F048	F048	F048	F048	F048	b

H = halfword integer
F = fullword integer
I = integer arithmetic
E = single-precision, floating-point arithmetic
D = double-precision, floating-point arithmetic
F048 = operation cannot be performed because operands are incompatible
F049 = only floating-point arithmetic is possible
F703 = user is prompted to select integer or logical arithmetic
F705 = user is prompted to select integer, logical, or floating-point arithmetic
a = if operation is relational, the user is prompted to select logical or floating-point arithmetic; if the operation is not relational, diagnostic F048 is issued
b = if operation is relational and the two operands have the same length, a logical compare is made; if not relational or if the lengths are unequal, diagnostic F048 is issued

AT Command

The AT command should only refer to instructions in the user's own program. AT should not be used in a label-processing or end-of-volume routine.

For each operand in an AT command, the instruction at that location is replaced by an SVC causing PCS to be activated when the SVC is executed. For this reason, the user should not designate as an operand in an AT command any instruction location that is the subject of an Execute (EX) instruction, or any instruction residing in a public control section.

The instruction replaced by the SVC is moved to an area of virtual storage remote from the user's program and is executed after all PCS actions have been performed. It will, in fact, never be executed if a CALL is issued that specifies a different restart point.

If a program interrupt should occur when this instruction is executed and a user's interrupt-handling routine has not been specified, a diagnostic message is issued. Since the interrupt is recognized as being caused by an instruction in the user's object program, the diagnostic is issued by the system's program interrupt routine. The control section name and displacement in the message are not shown in this case, since the instruction does not reside in a user's CSECT. If the user suspects that this situation has occurred, he can remove the PCS statement containing the appropriate AT operand and enter a CALL to re-execute the instruction(s). An interrupt at this time isolates the invalid instruction.

Operational Considerations

The user cannot make full use of the program control facilities until he has loaded his program; e.g., with a LOAD command. After loading his program, the user can initiate investigatory actions, e.g., he may DISPLAY the contents of machine registers or of locations in his virtual storage, or he may issue AT or SET commands.

Even after his program is loaded, the user's utilization of program control facilities will be restricted, if he failed to request an internal symbol dictionary (ISD) when his program was assembled or compiled. Lacking the ISD for a program, the user may refer only to external symbols in his commands; with the ISD, he is free to refer, also, to internal symbols within the program.

Once the user has loaded, but not initiated execution of his program, he may input program control commands and statements that refer to his program, and then initiate execution. If he is operating in conversational mode, he can interrupt execution of his program by pressing the ATTENTION button at his terminal; then he may input additional commands and statements or cancel previous statements.

The user can then resume program execution by entering the GO command. Alternatively, he can enter dynamic statements prior to program initiation, and specify control points at which execution is to be stopped. At these points he may enter data, change program sequence, etc.

When execution of the program is completed, the user may want to enter more commands. *Example:* restart execution of the program from a specified entry point by use of the GO command.

Program control operations may be continued on an object program until the program is unloaded by an UNLOAD command (or, in the case of an assembler-written program, by a DELETE macro instruction). When a program referred to in a dynamic PCS statement is unloaded, all dynamic statements are deleted. The user can reenter any dynamic PCS statements that refer to programs that are still loaded, if he wants to reinstate these statements.

Conversational Mode

In the conversational mode, PCS commands received from the user are checked for valid syntax; the symbols he refers to are checked against the object program's external and internal symbol dictionaries. Syntax errors and references to undefined symbols are reported to the user at his terminal together with appropriate messages to direct his corrective actions. The user is thus assured of entering only a valid set of program control commands and statements.

All output is produced at the user's terminal, except for the output developed by a DUMP command. *Example:* A dynamic statement that calls for the interruption of program execution causes output at the user's terminal when the statement is executed, to inform the user of the action taken and its location within his program.

Nonconversational Mode

Program control facilities may also be used in nonconversational mode, but with these differences:

1. Program control commands containing errors produce diagnostic messages that are sent to the task's SYSOUT data set; the commands are ignored.
2. No prompting is performed; incompletely entered commands, which would cause the user to be prompted in conversational mode, are ignored.
3. Program control output is sent to the nonconversational tasks' SYSOUT data set, and may be interspersed with other data that appears there.
4. After object program execution is interrupted by a STOP command (alone, or in a statement), the next command is taken from the nonconversational task's SYSIN data set.

Appendix C. Programming Considerations

This appendix describes procedures that the TSS assembler language programmer should follow in the preparation and execution of his programs. The initial sections in this appendix describe concepts basic to the writing of any assembler language program; included are discussions of writing programs in TSS assembler language, creation of unnamed control sections, pooling of literals, system macro instruction usage, floating-point computations, references to module names of link-edited modules, use of the EXIT and PAUSE macro instructions, assembler language linkage conventions, shared code considerations, efficient use of virtual storage, control section rejection and linking control sections, and recovering from errors when dynamically loading.

Next, a discussion is given of library management, including a description of the libraries available to the programmer and the use of the program library list.

The final section of this appendix describes those TSS naming conventions of which the user should be aware in order to avoid creating names that conflict with names reserved for system use.

Users writing privileged system programs should refer to the *System Programmer's Guide* for proper programming procedures.

Writing Programs in TSS

While not all programs need be written in accordance with the guidelines given here, users of TSS will find it easier to use the assembler if these guidelines are followed. More detailed information concerning assembler language programming is given later in this Appendix, and in Appendix D, "Interrupt Considerations."

The programming procedures described in this section use two programs for demonstration purposes. The first will be referred to as PGM. The name PGM is assigned as the module name when the ASM parameters are entered. Consistent with TSS terminology, the output of an assembly will be referred to as a "module," rather than the more general term "program."

The assembler instructions for PGM will be shown completely. It will be seen that PGM initializes two variables, then calls a second module, SUB. The instructions for SUB will be shown only in enough detail to make clear the details of the linkage between PGM and SUB.

The first assembler statement normally written will be a PSECT statement, as, for example:

```
PGMP     PSECT
```

where PGMP is the label of the PGM PSECT. The name of the PSECT may not be identical to the module name. (Here, the P is affixed to the module name PGM as a convenient notation technique.)

At this point it is pertinent to briefly discuss TSS naming conventions. A more detailed discussion is given later in this appendix. The user should not use names in his program that start with the following three characters: SYS, CHC, or CHD.

Following the PSECT statement, an ENTRY statement is given identifying the point in PGM at which execution begins. In PGM, the name of this entry point is PGME. The entry point name may not duplicate the module name or PSECT name. (The manner in which PGME is used will be shown later.) Execution at PGME is initiated by use of the CALL command or by an implicit CALL, as:

```
   CALL    PGME
```

(The CALL command is described in detail in the examples contained in this publication, as well as various appendixes, and *Command System User's Guide.*)

Following the ENTRY statement are two statements reserving a 19 word *save area:*

```
PGMP     PSECT
         ENTRY     PGME
         DC        F'76'
         DC        18F'0'
```

This save area is required by many of the system macro instructions (such as GET, PUT, CALL, SAVE, and RETURN). The DC F'76' statement gives the number of bytes in the save area (4 x 19); the DC 18F'0' statement sets the last 18 words of the 19 word save area to 0. When the contents of any word in the save area is altered, the new contents will, in general, not be 0. Thus, presetting each word to 0 allows one to determine if a word has been altered or not. This would not be possible if the statement DS 18F had been used, for example, as the contents of a storage area defined with a DS is not predictable.

The next statements to be placed in the PSECT are those reserving storage for items whose values may be changed by the program. This practice is required for reenterable programming, and is a convenient practice for all types of programs, as described in the section of this appendix discussing performance considerations. Assume that module PGM will be computing a new value for two words, to be referred to as ALPHA and BETA:

```
PGMP      PSECT
          ENTRY     PGME
          DC        F'76'
          DC        18F'0'

ALPHA     DS        F

BETA      DS        F
```

As stated earlier, PGM will make use of a subroutine SUB. The entry point to SUB is SUBE. The CALL, SAVE, RETURN group of system macro instructions will be used to accomplish the linkage to SUBE, and from SUB back to PGM. A necessary part of a CALL on SUBE is a specification in PGM of the entry name of module to be called. The normal way of specifying that a program will be called is by use of the ADCON macro instruction. The implicit form of the ADCON macro instruction will be used here:

```
PGMP      PSECT
          ENTRY     PGME
          DC        F'76'
          DC        18F'0'

ALPHA     DS        F

BETA      DS        F

SUBEVR    ADCON     IMPLICIT,EP=SUBE

SUBEVR    EQU       *-12
                    This and the following two
                    statements are generated by the
                    ADCON macro instruction and
                    will appear in the object listing
                    with a plus sign to the left of
                    the generated statement.
          DC        V(SUBE)
          DC        R(SUBE)
```

The ADCON macro instruction defines V- and R-type address constants for a later CALL to entry point SUBE, just as a program wishing to CALL module PGM at entry point PGME would write, for definition of the V- and R-type address constants:

```
PGMEVR   ADCON   IMPLICIT,EP=PGME
```

In addition to items varying at object time and V- and R-type address constants, all other types of address constants should be placed in the PSECT. In module PGM, assume the addresses of ALPHA and BETA are required:

```
PGMP      PSECT
          ENTRY     PGME
          DC        F'76'
          DC        18F'0'

ALPHA     DS        F

BETA      DS        F

SUBEVR    ADCON     IMPLICIT,EP=SUBE
          .
          .
          .
AALPHA    DC        A(ALPHA)
          DC        A(BETA)
```

That part of PGM that actually performs computations—the executable portion of the module—will now be written. The first statement in this portion of the module will be a CSECT statement. This CSECT will be assigned the attribute READONLY, as a CSECT should not contain information that will be altered at object time.

```
PGMP      PSECT
          ENTRY     PGME
          .
          .
          .
PGMC      CSECT     READONLY
```

The label on the CSECT statement was chosen to be the module name PGM with a C attached, a convenient notation technique. The CSECT name (PGMC), the PSECT name (PGMP), the entry name (PGME), and the module name (PGM) must all be different, as all are external symbols and external symbols may not be duplicated in a module.

When PGM is to be executed, the CALL command is used to transfer control to the entry point of PGM. The entry point is PGME, as noted in the ENTRY statement in the PSECT. PGME will be defined at the beginning of the CSECT. The question of base register usage should also be considered. When PGME is entered, general register 15 contains the address of PGME. This information should be furnished the assembler by a USING statement:

```
PGMP      PSECT
          ENTRY     PGME
          .
          .
          .
PGMC      CSECT     READONLY
          USING     PGME, 15

PGME      SAVE      (14, 12)
          STM       14, 12, 12(13)
                    This statement is generated by
                    the SAVE macro instruction,
                    and will store general registers
                    14, 15 and 0 through 12 in the
                    calling program's save area.
```

Note the use of the SAVE macro instruction at the entry point PGME above. It is a recommended convention to use the SAVE as shown to save the contents of the registers when entered from the CALL command or as the result of a CALL from another program. The SAVE allows PGM to restore registers to their value when PGME was entered with the RETURN macro, as shown later. (Use of RETURN is also a recommended convention.)

Following the SAVE of general registers, a sequence of five instructions will then be given:

```
PGMP     PSECT
            .
            .
            .
PGMC     CSECT      READONLY
         USING      PGME, 15
PGME     SAVE       (14, 12)
         L          14, 72(0, 13)
```

Loads register 14 with the R-value of PGME, which in this case is the address of the PGM PSECT, PGMP. The calling program placed the R-value of PGME in the 19th word of its save area prior to calling PGME.

```
         ST         14, 8(0, 13)
```

The address of the PGM save area is placed in the third word of the PGM save area. This save establishes the "forward pointer" in the calling program's PSECT.

```
         ST         13, 4(0, 14)
```

The address of the calling program's save area is saved in the second word of the PGM save area, PGMP+4. This save establishes the "backward pointer."

```
         LR         13, 14
```

Register 13 now contains the address of the PGM save area (by convention, its PSECT) as it will throughout execution of PGM.

```
         USING      PGMP, 13
```

Inform the assembler that any items referred to in PGMP should use register 13 as the base register.

The remainder of the PGM CSECT will now be written. In this example, PGM will initialize the two variables ALPHA and BETA to 12, then execute a CALL on subroutine SUB, supplying SUB with the address of a parameter list containing address constants for the two parameters required by SUB: ALPHA and BETA.

```
PGMP     PSECT
            .
            .
            .
PGME     SAVE       (14, 12)
            .
            .
            .
         USING      PGMP, 13
         L          0,F12
```

Set ALPHA to 12.

```
         ST         0,ALPHA
```

Register 0 is a convenient register to use for such temporary usage, as it is unsuitable for retaining the same value over large parts of a program. Many macro instructions destroy register 0 contents. Note that F12 is defined below in the CSECT. This is not in conflict with earlier suggestions for items to be placed in a PSECT, as F12 will not be altered by PGM, and is, of course, not an address constant.

```
         L          0,=F'12'
```

This demonstrates another means of setting a variable to 12. The assembler will generate a literal constant of 12, and place it in the CSECT, as all non-adcon literals are placed in the first defined CSECT unless a LTORG is declared.

```
         ST         0,BETA
         LA         15,SUBEVR
         CALL       (15),MF=(E,AALPHA)
```

The following five instructions are generated by the CALL macro instruction:

```
         LA         1,AALPHA
```

Load register 1 with the address of the parameter list.

```
         L          14, 16(0, 15)
```

Load register 14 with the R-value of the program to be called.

```
         ST         14, 72(0, 13)
```

Store the R-value in the 19th word of the save area to be supplied to the called program.

```
         L          15, 12(0, 15)
```

Load register 15 with the V-value (the entry point) of the program to be called.

```
         BASR      14, 15
                   Branch to the location in register 15, setting register 14 to the
                   address of the first byte following the BASR. The called module
                   will return to the address in register 14.
```

Note that the LA into register 15, required for this form of the CALL macro instruction, will destroy any previous contents of register 15. When PGME was entered, the USING PGME, 15 statement established a base register for the PGM CSECT. If a CALL or other use of 15 is made, as in this example, another register should be used. General register 12 is a good choice, as no system macro instructions (such as CALL) require that register 12 be altered. The code following PGME might then be:

```
         USING     PGME, 15
PGME     SAVE      (14, 12)
         L         14, 72(0, 13)
         ST        14, 8(0, 13)
         ST        13, 4(0, 14)
         LR        13, 14
         USING     PGMP, 13
         LR        12, 15
                   To load register 12 with the address of PGME.
         DROP      15
                   This instruction is required prior to the following USING, or
                   the assembler would continue to use register 15 as the PGME
                   base register.
         USING     PGME, 12
                   To inform the assembler that register 12 is now the base
                   register for PGME.
         .
         .
         .
         LA        15,SUBEVR
         CALL      (15),MF=(E,AALPHA)
         .
         .
         .
```

Many forms of the CALL macro might have been used to call SUBE. The form shown above requires that the V- and R-type address constants be prepared by the programmer; another form of the CALL would cause the assembler to generate these two address constants and place them in the PGM PSECT.

PGM is now completed, and will use the usual sequence of statements to return control to the calling program:

```
         L         13, 4(0, 13)
                   Load register 13 with the address of the calling program's
                   save area.
         RETURN    (14, 12)
                   Restore the remaining registers to the values they contained
                   when PGM was called.
         LM        14, 12, 12(13)
         BR        14
                   These instructions are generated by the RETURN macro
                   instruction, and loads registers 14, 15, and 0 through 12 from
                   the save area of the calling program, and then returns control
                   to the calling program.
F12      DC        F'12'
                   Required above.
         END       Notes the end of this assembly.
```

The assembler instructions for the entire PGM module are repeated below.

```
PGMP     PSECT     Declare the program PSECT.
         ENTRY     PGME
                   Identify the program entry point.
         DC        F'76'
                   Reserve a 19-word save area.
         DC        18F'0'
ALPHA    DS        F
                   Reserve space for items to be altered in program execution.
BETA     DS        F
SUBEVR   ADCON     IMPLICIT,EP=SUBE
                   Define V- and R-type adcons for the CALL of SUBE.
AALPHA   DC        A(ALPHA)
         DC        A(BETA)
                   Address constants of parameters used by SUBE.
PGMC     CSECT     READONLY
                   Declare the program CSECT.
         USING     PGME,15
                   The calling program established register 15.
PGME     SAVE      (14, 12)
                   Save the contents of registers.
         L         14, 72(0, 13)
                   Establish the backward and forward chains.
         ST        14, 8(0, 13)
         ST        13, 4(0, 14)
         LR        13, 14
                   Load PSECT base register and inform the assembler.
```

```
         USING     PGMP, 13
         LR        12, 15
                   Establish base register for
                   PGME and inform the assem-
                   bler.
         DROP      15
         USING     PGME, 12
         L         0,=F'12'
                   Initialize ALPHA and BETA to
                   12.
         ST        0,ALPHA
         L         0,=F'12'
         ST        0,BETA
         LA        15,SUBEVR
                   Call SUBE, passing the address
                   of the parameter list.
         CALL      (15),MF=(E,AALPHA)
         L         13, 4(0, 13)
                   Restore the address of the call-
                   ing program's save area.
         RETURN    (14, 12)
                   Return to the calling program.
F12      DC        F'12'
         END
```

Module SUB, called by PGM at the SUBE entry point, will now be shown. First, the SUB PSECT.

```
SUBP     PSECT
         ENTRY     SUBE
         DC        F'76'
         DC        18F'0'
           .
           .
           .
```

Next the initial part of the SUB CSECT, ENTRY and SAVE statements and the code establishing the PSECT and CSECT base registers:

```
SUBP     PSECT
         ENTRY     SUBE
           .
           .
           .
SUBC     CSECT     READONLY
         USING     SUBE, 15
SUBE     SAVE      (14, 12)
         L         14, 72(0, 13)
         ST        14, 8(0, 13)
         ST        13, 4(0, 14)
         LR        13, 14
         USING     SUBP, 13
         LR        12, 15
         DROP      15
         USING     SUBE, 12
```

Module SUB will now move the values of ALPHA and BETA from the calling program into SUB. Note that: (1)SUB can obtain the values of ALPHA and BETA, as the CALL of SUBE by PGM placed the location of address constants pointing to ALPHA and BETA in general register 1; and (2) SUB should store the values obtained in the SUB PSECT, SUBP, in accordance with the practice of storing items that vary during execution in a PSECT. The assembler instructions might be:

```
SUBP     PSECT
         ENTRY     SUBE
           .
           .
           .
VAR1     DS        F
VAR2     DS        F
                   ALPHA and BETA from PGM
                   will be stored here.
           .
           .
           .
SUBC     CSECT     READONLY
         USING     SUBE, 15
SUBE     SAVE      (14, 12)
           .
           .
           .
         DROP      15
         USING     SUBE, 12
         LM        6, 7, 0(1)
                   Load the address of ALPHA
                   into register 6, and the address
                   of BETA into register 7.
         L         0, 0(0, 6)
                   Store ALPHA in VAR1.
         ST        0, VAR1
         L         0, 0(0, 7)
                   Store BETA in VAR2.
         ST        0, VAR2
           .
           .
           .
```

Following whatever other instructions might be executed, SUB returns to PGM. When PGM is reentered following the CALL to SUB, all general registers will contain the same values as when SUBE was called.

```
L           13, 4(0, 13)
RETURN      (14, 12)
  .
  .
  .
END
```

Creation of Unnamed Control Sections

Certain statements within the TSS assembler language require that a control section be defined when they are encountered because they assume that a location counter value is available. If the programmer has not declared a control section, the assembler defines an unnamed CSECT which will be the first control section in the object module produced by the assembler. This assembler action does not normally need to concern the programmer, except when

(1) statements generated in this control section will require a base register which has not been provided, or when

(2) the programmer expects to call the module by its name.

The following statements will cause the assembler to produce an unnamed CSECT if necessary:

- all machine operation instructions
- all system or user macro instructions
- CCW
- CNOP
- CXD
- DC, DS, and ORG
- EQU
- ENTRY
- USING and DROP
- LTORG
- END

If a module will be called by name and no symbol is specified on the END statement, then the entry point for execution of the module will be the first byte of the first CSECT (if one exists). If the assembler has defined an unnamed CSECT for the reasons above, the unnamed CSECT is the assumed execution entry point (standard entry point) for calls of the module by name, even though the programmer may have defined a named CSECT later in the program. This condition may prevent proper execution of the program. Further, if the unnamed CSECT has no text, that is, its length is zero, the address associated with the CSECT is location 0.

The programmer should define a control section before coding any of the statements mentioned to avoid any problems caused by an assembler-produced unnamed control section. A suggested practice is to define all control sections to be used with CSECT, PSECT, COM, or DSECT statements before coding anything else except comments. For example, the following sequence of statements guarantees that the first two control sections in the program will be a PSECT named PS and a CSECT named CS. The assembler will build the control sections from subsequent statements.

```
PS      PSECT
        DC          F'76',18A(0)      define a save area
CS      CSECT
          .
          .
          .
PS      PSECT                         present the program
          .                           in this and following
          .                           statements
          .
        END
```

Pooling of Literals

When LTORGS are included in a program, non-adcon literals are pooled at the first LTORG following their occurrence. Any non-adcon literals not pooled by the end of the program are pooled at the end of the first CSECT. When no LTORG is present, all non-adcon literals are pooled at the end of the first CSECT. When a PSECT is present in the program, regardless of whether LTORGS are present, all adcon literals are pooled at the end of the first PSECT. If there is no PSECT in the program, adcon literals follow the pooling rules used for non-adcon literals. Finally, if a CSECT is not present and a PSECT is present, all literals are pooled as LTORGS are encountered or at the end of the first PSECT.

System Macro Instruction Usage

There are certain conditions to be met and certain conventions to be observed when using system macro instructions. The contents of registers 0, 1, 14, and 15 may be destroyed when system macro instructions are used. Some system macro instructions generate literals for which base registers must be provided by the user. With two exceptions, whenever a macro instruction generates a control section statement, that control section is a continuation of one previously declared by the source program, and the control section in effect at macro call time is continued before the macro generation is completed. The two exceptions are DCBD and ADCOND, which cause a unique DSECT to be generated, and this control section is in effect when macro generation is completed. A few macro instructions require register 13 to be preset to the address of a save area; CALL, SAVE, and RETURN are examples of this type of macro instruction. The publications *Assembler User Macro Instructions* and *System Programmer's Guide* contain more detailed discussions of these topics.

Floating-Point Computations

It must be kept in mind that, unlike integer arithmetic, floating-point computations are not in general exact, due to roundoff. This may cause the low-order bits of a result to be different from the expected value. This is true of PCS and FORTRAN programs as well as assembler language programs. Thus, the user should be particularly careful when comparing the results of an assembler language floating-point computation with that from a PCS computation, etc.

The order in which programs perform floating-point computations may be important. For PCS, this order is described in the publication *Command System User's Guide*. For FORTRAN programs, the object listings must be inspected to determine the order of computation.

References to Module Names of Link-Edited Modules

When modules are link-edited, the resultant module is assigned the name specified in the LNK parameters. The module names for those modules included in the link edit are retained as auxiliary entry points in the list of external symbols associated with the link-edited module.

The module names of link-edited modules are retained in the ISD of the resultant module, if an ISD was requested for the resultant module and the module(s) being included had been assembled with an ISD. The user can, in his PCS commands, refer to internal symbols, including the original module name, in the resultant module. In order to do so, the internal symbol must be qualified by both the resultant module name and the original module name, in that order.

EXIT and PAUSE Macro Instructions

Table 15 summarizes the use of the EXIT and PAUSE macro instructions in both conversational and nonconversational mode.

Assembler Language Linkage Conventions

This section discusses the coding practices to be observed when preparing modules to be used as subroutines that are called by other object modules and when preparing linkages to other object modules. The section concludes with an example describing the contents of the PSECTs of three modules at various points in the transfer of control from one to another. For information regarding the linking of assembler language programs with FORTRAN and PL/I subprograms, refer to the *FORTRAN Programmer's Guide* and *PL/I Programmer's Guide*, respectively.

Linkage Conventions

Standard linkage conventions have been defined to govern the communication between all TSS programs. Five types of standard linkage have been defined: I, II, IM/II, III, IV. Only type I linkage will be described here. The other linkage types are described in detail in the publication *System Programmer's Guide*.

Table 15. EXIT and PAUSE Macro Instructions

MACRO INSTRUCTION	EFFECT IN CONVERSATIONAL MODE	EFFECT IN NONCONVERSATIONAL MODE
PAUSE n or PAUSE 'message'	1. Prints the message "PAUSE n" or "PAUSE message" at the user's terminal.	PAUSE n or 'message' prints on SYSOUT data set, execution continues with the statement following the PAUSE.
	2. Prints an underscore at terminal requesting a command.	
	3. Program may be continued at the statement following the PAUSE by entering the RUN command.	
EXIT n or EXIT 'message'	1. Prints "EXIT, RELEASE ALL UNNEEDED DEVICES," followed by n or 'message' at the terminal.	1. Prints "EXIT, RELEASE ALL UNNEEDED DEVICES," followed by n or 'message' in the SYSOUT data set.
	2. Prints an underscore at terminal requesting a command.	2. Reads the next command from the SYSIN data set.

Associated with type I linkage conventions are three areas of concern; these are:

1. Register usage.
2. Parameter lists.
3. Save areas.

Proper Register Usage

TSS has assigned roles to certain registers used in generating a linkage. The function of each linkage register is illustrated in Table 16. Note that registers 2 through 12 are not assigned and, thus, are always available to the programmer for other purposes.

It is the responsibility of the *called* module to maintain the integrity of general registers 2 through 12 so that their contents are the same at exit as they were at entry to the called module. It is the *calling* module's responsibility to maintain the floating-point registers and program mask around a call. General registers 0, 1, and 13 through 15 must conform to the indicated conventions; 0 and 1 may be destroyed by the called module.

Table 16. Linkage Registers

GENERAL REGISTER	USAGE
1	Parameter List Register—contains the address of a list of pointers to input parameters.
13	Save Area Register—contains the address of the calling module's save area.
14	Return Register—contains address in calling module at which execution resumes upon return.
15	Entry Point Register—contains address of the entry point in the called module; also Return Code Register—contains return code set by called module.

Reserving a Parameter Area

If a called module requires input parameters, the calling module must supply the called module with the location of a parameter list in general register 1. Each entry in the parameter list must be on a full-word boundary and represents the address of a parameter being passed to the called module. If the parameter list is variable in length, the length is specified as a count of the number of addresses that compose the list. This count is located one word before the first word in the parameter list. Regardless of whether the parameter list is of fixed or variable length, the parameter list register points to the first word of the parameter list. The CALL macro instruction can be used to generate the parameter list, as well as to link to the called module.

Reserving a Save Area

It is the responsibility of the calling module to supply a 19-word area to be used by the called module. Figure 32 shows the layout of the save area and briefly describes the information saved in the area by the calling and called module. Of particular interest in this save area (for trace purposes) are the following two words:

Word 2 The "backward pointer." This word always points to the save area of the module that called the module whose save area is being inspected.

Word 3 The "forward pointer." This word contains the address of the save area of the module last called by the program whose save area is being inspected. The low order bit of this word is set to zero as the called program is entered and set to 1 upon exit if the T option in the RETURN macro is used. This bit is useful in determining the flow of control during program execution.

CALL, SAVE, and RETURN Macro Instruction Usage

The CALL, SAVE, and RETURN macro instructions are used to provide linkage between object program modules. Refer to the publication *Assembler User's Macro Instructions* for a detailed description of the notation and options of each macro instruction. In most cases, additional user-written instructions are necessary to complete the requirements of the linkage conventions. The following sections illustrate the points that should be considered when using the program linkage macro instructions.

CALL Macro Instruction

The CALL macro instruction generates all the necessary instructions to set the entry point and return registers, constructs a parameter list if parameters are specified and sets the parameter register, and stores the R-value for the called module in the 19th word of the save area indicated by register 13. It is the user's responsibility to ensure that register 13 is properly set to the address of the save area.

If the calling module requires the contents of registers 0, 1, 13, 14, and 15, the calling module must save and restore these registers around the CALL. (For example, register 15 may be used as a base register for code.) The calling module must also save and restore the program mask and any floating point registers used around the CALL.

NOTE: An implicit CALL refers to a V- and an R-type address constant pair. In order for the loader to properly resolve the value of the R-type address constant, the label appearing in the operand of the R-type address constant must be an external symbol.

Although the CALL macro instruction provides for specifying parameters that automatically cause a parameter list to be constructed, there are various other methods for communicating between object modules.

The user can, of course, construct his own parameter list, setting the parameter list register (1) prior to the CALL. The location of data (such as a table) can be communicated to the called program by an ENTRY statement in the calling program. The called program must then contain an EXTRN statement and an address constant naming the table.

SAVE Macro Instruction

The SAVE macro instruction generates the instructions for saving the contents of general registers as specified by the user. Register 13 is never saved and must not be specified. The user may wish to save all the registers but 13 (i.e., SAVE (14,12)). This provides full protection against inadvertently changing a register.

In addition to saving registers, SAVE can be used to develop a means of checking the program flow. If the T option is specified, registers 14 and 15 are stored in the fourth and fifth words of the save area. Also, if the first register specification in the macro instruction is 14, 15, 0, 1, or 2, all registers from 14 through the second register specification are saved. If an entry-point identifier operand is specified, the entry-point identifier character string is included in the macro expansion beginning on a half-word boundary preceding the entry point.

The entry-point identifier is placed so that either one or two bytes separate its end from the beginning of the entry-point. If an extra byte is needed to achieve half-word alignment, a character blank is added to the end of the string. A count byte will then follow. The count byte will always precede the entry-point and contain a value equal to the number of characters in the string, plus the blank (if used). The count byte itself is not included in this tally.

If the entry-point identifier operand is written as an asterisk, the entry-point identifier is the same as the symbol in the name field of this macro instruction. If the name field is blank, the name of the control section containing the SAVE macro instruction is used as the identifier character string.

The additional instructions to be supplied by the user following the SAVE are dependent on the type of module being prepared.

If the called module does not perform any further linkages, the only additional instruction necessary is one that loads into a base register the PSECT address from the nineteenth word of the save area as pointed to by register 13.

If register 15 is not to be used in further calls, it can be used as a base register for the code. If register 14 is used by the called module, it should be specified in the SAVE and RETURN.

A program that does not perform any calls may be coded as:

```
SUB1EP   SAVE     (14,12),,*
                  Save registers 14-15, 0-12 and
                  place SUB1EP preceding the
                  entry point
         L        12,72(0,13)
                  Pick up address of PSECT
         USING    SUB1P,12
                  PSECT base register
         USING    SUB1EP,15
                  Code base register
           .
           .
           .
         RETURN   (14,12),T
                  Restore and return
```

If the called module does perform further linkages, additional instructions must be supplied to perform the following functions:

- Establish a save area to be used by the modules being called.
- Save the contents of registers 13 and 14. Register 14 can be specified in the SAVE macro; the instructions to save register 13 must be supplied by the user. The backward pointer in the called module's save area is intended for this purpose.
- Establish the forward and backward pointers in the calling and called module's save areas. This facilitates checking of the flow of control from one object module to another.

The following is an example of a module that has been written so that its save area occupies the first 19 words of its PSECT. This is convenient in that the save area register (13) can also be used as a base register for the PSECT.

```
SUB1P    PSECT
         DC       F'76'
         DS       18F
           .
           .
           .
         ENTRY    SUB1EP
SUB1C    CSECT    READONLY
SUB1EP   SAVE     (14,12)
                  Save registers 14-15, 0-12
         L        14,72(0,13)
                  Get R-value from calling mod-
                  ule's save area
         ST       14,8(0,13)
                  Store forward pointer in calling
                  module's save area
         ST       13,4(0,14)
                  Store backward pointer in SUB1
                  save area; address of calling
                  module's save area will be re-
                  stored to 13 before return
         LR       13,14
                  Set base register for PSECT and
                  save area
         USING    SUB1P,13
         LR       12,15
                  Set code register
         USING    SUB1EP,12
```

Note that register 14 in the example is used to hold the PSECT address temporarily until 13 is saved. This is safe, since 14 has been specified in the SAVE and is to be restored on RETURN. Also, since this module is to CALL another module, register 12 is used as a base register for code, rather than saving and restoring 15 around each call. Finally, the occurrence of the ENTRY statement in the SUB1 PSECT identifies the origin of the PSECT as the R-value for the entry point SUB1EP.

Location	Content
SAREA (word 1)	Contains the length of the save area in bytes, a minimum of 76.
SAREA + 4 (word 2)	The address of the calling module's save area. This field is set by the called module in its own save area.
SAREA + 8 (word 3)	The address of the next save area; that is, the save area of the called module. This field is set by the called module.
SAREA + 12 (word 4)	The contents of register 14 containing the address to which return from the called module is made. This field is set by the called module in the calling module's save area.
SAREA + 16 (word 5)	The contents of register 15, containing the address to which entry into the called module is made. This field is set by the called module in the calling module's save area.
SAREA + 20 (word 6)	The contents of register 0. Value in register 0 set by calling module and saved by called module.
SAREA + 24 (word 7)	The contents of register 1.
SAREA + 28 (word 8)	The contents of register 2.
SAREA + 32 (word 9)	The contents of register 3.
	Eight words containing the contents of registers 4-11.
SAREA + 68 (word 18)	The contents of register 12.
SAREA + 72 (word 19)	The address of the PSECT of the called module. This field must be set by the calling module, by storing the R-value of the called entry point in it.

Figure 32. Save Area Format and Word Content

Although it is convenient to have a save area as the initial portion of the PSECT, it is not a requirement. Some alternatives are:

- The save area resides in the PSECT but not at the beginning.
- The save area resides in another control section but is located by an address constant in the PSECT.
- The module contains no PSECT; the save area resides in another control section type and is pointed to by an address constant in the module.
- The save area is dynamically allocated via GETMAIN on each entry to the routine and released prior to RETURN.

While any of these methods is possible, each has disadvantages. All require more instructions in locating the save area. Also, an extra register must be used as the PSECT base register, thus decreasing the number of registers available for the program.

RETURN Macro Instruction

The RETURN macro instruction expands into the instructions for restoring the general registers and returning control to the calling program. Additionally, if the T option is specified in the macro, the low-order bit in the third word of the caller's save area is set on to facilitate tracing save areas.

Register 13 must be set to the calling module's save area prior to the RETURN. If the contents have been stored as the backward pointer (word 2) in the called module's save area, the register contents can be restored easily by:

```
L         13,4(0,13)
RETURN    (14,12)
```

If neither register 14 nor the T option has been specified in the SAVE macro instruction, 14 must also be restored prior to the RETURN.

A return code can be specified in the macro or can be preset in register 15 by the program. Return codes can be any value from 0 to 4092. Codes must be assigned in multiples of four, so that the calling program can use them as an index to a branch table.

If the RETURN macro instruction is written with the T option, when the registers are restored, the low-order bit of the forward pointer in word 3 of the calling module's save area is set to 1. This is another means of tracing the flow of control between modules.

Object Modules Initiated by a CALL Command

Logically, the first module to receive control has the same linkage as lower-level routines that it, in turn, calls. The system, via the CALL command, allocates a save area to be used by the modules. On entry, registers 13, 14, and 15 are set by convention to the save area, the return point, and the entry point, respectively. The 19th word of the save area contains the R-value for the entry point. The return point, in this case, is to a system routine that conversationally places the terminal back in command mode or nonconversationally reads the next command from SYSIN. The contents of registers 0 through 12 are unpredictable.

If the initial module completes its execution with an EXIT macro instruction, the registers contain the values as last changed by the program. If, however, the program ends execution with a RETURN macro instruction, the contents of the registers depend on which registers were restored in the RETURN. Registers 13 and 14 must, of course, always contain the save area address and return point. Registers 0 through 12 and 15, if not specified in the RETURN, contain the values as set by the program. Register 15, if restored, contains the entry point. In this case, any investigation of the contents of registers 0 through 12 must be done prior to the execution of the RETURN.

Example of Module Interaction

The interaction of three modules is shown below (Figures 33, 34, and 35). Module A calls module B which in turn calls module C. Register 0 is loaded with "1," "2," and "3," in modules A, B, and C, respectively. The load is performed prior to the link to the next higher level. Table 17 shows the contents of the save areas for all the modules at the points specified. The backward and forward pointers between save area levels are contained in words 2 and 3. The next higher level entry point is in word 5, and the return point to the level being inspected is contained in word 4. The save area for the highest level will never contain useful information in words 4 through 18, the general register save area. This is because it is the responsibility of the called module to save the environment of the calling module in the calling module's save area.

It must also be emphasized that the setting of bit 31, word 3 in the calling module's save area results from the called module using the T option on the RETURN. As an example, see the save area for module B after control is passed to module A. The setting of this bit took place at line 2000 in module C.

Thus, by inspecting the save area belonging to a module, it may be determined whether it was the highest level executed, where it was called, the location of the calling module's save area and the descending chain of pointers, the entry point to the next higher level module, if any, and the return point to the level being inspected if a call was made to a higher level.

Interroutine Communication

External dummy sections provide a direct means of communication among several different routines. They are defined by DXD instructions and are located at execution time by a Q-type address constant naming the desired external dummy section. The information present in an external dummy section is available to all routines which have defined the dummy section within their code.

The CXD instruction must be specified in at least one of the routines. For all modules loaded simultaneously, the loader will calculate the cumulative byte length of all external dummy sections and place the sum in the fullword storage area allocated by the CXD instruction. Any routine which issued the CXD instruction can then use the cumulative sum to determine the amount of virtual storage which must be acquired to contain all the external dummy sections. The storage will be obtained dynamically, at execution time, by issuing a GETMAIN macro instruction. Alternately, storage may have been reserved earlier with DS, DC, or ORG assembler instructions.

The example which follows illustrates a use of external dummy sections as communications areas for a main routine and its subroutine. MAINPGM is the main routine which has been assembled in a program module. It will call the subroutine SUBPGM which has been assembled in a separate program module. The main routine will place the address of the message 'Sample Message' in the external dummy section named P1. The message byte length will be saved in the external dummy section P2. When SUBPGM gains control, it can access the two external dummy sections and write the message to SYSOUT.

The assembler instructions for the module MAINPGM are shown below. The standard entry and linkage conventions discussed earlier must be followed. This example, however, will only show the code necessary for establishing and manipulating external dummy sections.

```
MAINP    PSECT
            .
            .
            .
P1       DXD      A
                  External dummy section which
                  will be set to the address of
                  MSG.
P2       DXD      F
                  External dummy section which
                  will be set to the length of
                  MSG.
QP1      DC       Q(P1)
QP2      DC       Q(P2)
                  Q-type address constants used
                  to locate P1 and P2 in the ac-
                  quired virtual storage.
```

```
CUMLEN   CXD
                   After loading, this fullword in
                   storage will contain the cumula-
                   tive byte length of all external
                   dummy sections defined by the
                   two routines.
XDPGS    DS        F
                   This fullword will save the num-
                   ber of pages of virtual storage
                   which must be acquired.
         .
         .
         .
MSG      DC        C'SAMPLE MESSAGE'
LMSG     DC        A(L'MSG)
                   Set to the length of MSG.
         .
         .
         .
```

The executable portion of the main routine is now shown. MAINPGM will calculate the virtual storage request from the sum stored in CUMLEN, and issue the GETMAIN macro instruction.

```
MAINC    CSECT
         .
         .
         .
         L         12,CUMLEN
         LA        12,4095(0,12)
         SRA       12,12
                   The preceding code is necessary
                   to convert the number of bytes
                   needed into pages.
         ST        12,XDPGS
                   The page request is stored in
                   XDPGS for reference when the
                   storage is released.
         LR        0,12
                   Register 0 now contains the
                   number of pages which must be
                   acquired.
         GETMAIN   PAGE,LV=(0)
                   The GETMAIN macro instruc-
                   tion is issued to obtain the vir-
                   tual storage.
         LR        12,1
                   Register 12 is established as the
                   base register for addressing the
                   external dummy sections.
         .
         .
         .
```

The main routine will now place the proper values in the external dummy sections and call SUBPGM.

```
         L         5,QP1
                   The displacement of the external
                   dummy section P1 is loaded into
                   register 5.
         LA        6,MSG
                   The address of MSG is loaded
                   into register 6.
         ST        6,0(5,12)
                   The address of MSG is stored in
                   P1.
         L         5,QP2
         L         6,LMSG
                   The displacement of P2 is
                   loaded into register 5; the byte
                   length of MSG is loaded into
                   register 6.
         ST        6,0(5,12)
                   The length of MSG is stored in
                   P2.
         CALL      SUBPGM
                   A call is made to the subroutine
                   to print out MSG.
         .
         .
         .
```

After SUBPGM has completed execution, it will return control to the main routine. Since the storage needed for the external dummy sections is no longer required, a FREEMAIN macro instruction can be issued to release virtual storage.

```
         L         0,XDPGS
                   The page request is loaded.
         LR        1,12
                   The address of the external
                   dummy section block is loaded
                   into register 1.
         FREEMAIN  PAGE,LV=(0),A=(1)
                   Virtual storage is released.
         .
         .
         .
         END
```

The external dummy sections P1 and P2 must also be defined in the subroutine in order for interaction between the two routines to occur. Following is the necessary code for SUBPGM:

```
SUBP     PSECT
         .
         .
         .
P2       DXD       F
                   Length of MSG.
P1       DXD       A
                   Address of MSG.
QP2      DC        Q(P2)
QP1      DC        Q(P1)
                   Displacements of external
                   dummy sections within block of
                   storage.
LNGAREA  DS        F
MSGAREA  DS        CL256
                   Areas into which P2 and P1
                   will be moved.
XX       DXD       CL256
QXX      DC        Q(XX)
                   The external dummy section XX
                   is an area which will be used
                   only by SUBPGM. It has not
                   been defined in MAINPGM and
                   is thus not available to the main
                   routine even though it is repre-
                   sented in the area secured by
                   MAINPGM.
```

```
          .
          .
          .
SUBC      CSECT
          .
          .
          .
          L         7,QP2
                    The displacement of P2 within
                    the block of storage is loaded
                    into register 7.
          L         8,0(7,12)
                    SUBPGM assumes that register
                    12 points to the address of the
                    storage block containing the ex-
                    ternal dummy sections. By using
                    register 12 as the base and reg-
                    ister 7 as the index, the message
                    length can be obtained and
                    loaded into register 8. Note that
                    if SUBC is not defined with a
                    PUBLIC attribute, the preced-
                    ing two lines will be replaced
                    with the following:
                        L      7,0(12)
                        ORG    *-2
                        DC     QL2(P2)
                    These lines will generate one
                    LOAD instruction, thus saving
                    on the number of source lines.
                    SUBC may not have the PUB-
                    LIC attribute because the Qcon
                    for P2 must be resolved by the
                    dynamic loader.
          ST        8,LNGAREA
                    The message length is stored.
          BCTR      8,0
                    Decrement register contents by
                    one to prepare for EXECUTE
                    instruction.
          L         7,QP1
                    Load displacement of P1.
          L         7,0(7,12)
                    Obtain address of MSG.
          EX        8,MOVE
                    Move message to MSGAREA in
                    order to write to SYSOUT.
          GATWR     MSGAREA,LNGAREA
                    Issue message.
          .
          .
          .
MOVE      MVC       MSGAREA(0),0(7)
                    Move message.
          .
          .
          .
          END
```

```
VIRTUAL
MEMORY
LOCATION STATEMNT  SOURCE

         0000100   * MODULE 'A'
  300000 0000200   SUBRA     PSECT
         0000300             ENTRY SUBRA21
  300000 0000400             DC    F'76'               SAVE AREA
  300004 0000500             DC    18F'0'
         0000600   ADPR1     ADCON IMPLICIT,EP=SUBRB21
  30004C        +            CNOP  0,4
                +  ADPR1     EQU   *-12
  30004C        +            DC    V(SUBRB21)          V-CON
  300050        +            DC    R(SUBRB21)          R-CON
  300054 0000700   PLIST     DC    A(PARAM)
  300058 0000800   PARAM     DS    F
  301000 0000900   SUBRA1    CSECT READONLY
         0001000             USING SUBRA21,15
         0001100   SUBRA21   SAVE  (14,12)             SAVE REGS IN CALLER'S SAVE AREA
  301000        +  SUBRA21   DS    0H
  301000        +            STM   14,12,12(13)
  301004 0001200             L     14,72(0,13)         GET LOC THIS PROGRAM'S SAVE AREA
  301008 0001300             ST    14,8(0,13)          SAVE IN CALLER'S SAVE AREA
  30100C 0001400             ST    13,4(0,14)          SAVE LOC OF CALLER'S SAVE AREA
  301010 0001500             LR    13,14               LOC THIS SUBROUTINE'S SAVE AREA
         0001600             USING SUBRA,13
  301012 0001700             LA    0,1
  301016 0001800             LR    12,15               LOAD COVER REG
         0001900             DROP  15
         0002000             USING SUBRA21,12
  301018 0002100             LA    15,ADPR1
         0002200             CALL  (15),MF=(E,PLIST)
  30101C        +            DS    0H
                +            CHDINNRA PLIST      LOAD POINTER TO PARAM LIST INTO REG 1
  30101C        +            DS    0H                        HALF WORD ALIGNMENT
  30101C        +            LA    1,PLIST                      LOAD PARAM. REG. 1
  301020        +            L     14,16(0,15)
  301024        +            L     15,12(0,15)
  301028        +            ST    14,72(0,13)             STORE IN SAVE AREA
  30102C        +            BASR  14,15                   LINK
         0002300   CALLB     EQU   *                   RETURN POINT AFTER CALL
         0002400   *
         0002500   * ADDITIONAL PROCESSING BY SUBRA1
         0002600   *
  30102E 0002700             L     13,4(0,13)          RESTORE REG 13 TO CALLER'S SAVE AREA
         0002800             RETURN (14,12),T          RESTORE REGS AND RETURN
  301032        +            DS    0H
  301032        +            LM    14,12,12(13)
  301036        +            OI    11(13),X'01'
  30103A        +            BR    14
         0002900             END
```

Figure 33. Module A Source Listing

```
VIRTUAL
MEMORY
LOCATION STATEMNT   SOURCE

       0000100    * MODULE 'B'
350000 0000200    SUBRB1   PSECT
       0000300             ENTRY SUBRB21
350000 0000400             DC    F'76'                SAVE AREA
350004 0000500             DC    18F'0'
       0000600    ADPR2    ADCON IMPLICIT,EP=SUBRC21
35004C         +           CNOP  0,4
               +  ADPR2    EQU   *-12
35004C         +           DC    V(SUBRC21)           V-CON
350050         +           DC    R(SUBRC21)           R-CON
350054 0000700    R1SAVE   DS    F
351000 0000800    SUBRB2   CSECT READONLY
       0000900             USING SUBRB21,15
       0001000    SUBRB21  SAVE  (14,12)              SAVE REGS IN CALLER'S SAVE AREA
351000         +  SUBRB21  DS    0H
351000         +           STM   14,12,12(13)
351004 0001100             L     14,72(0,13)          GET LOC THIS PROGRAM'S SAVE AREA
351008 0001200             ST    14,8(0,13)           SAVE IN CALLER'S SAVE AREA
35100C 0001300             ST    13,4(0,14)           SAVE LOC OF CALLER'S SAVE AREA
351010 0001400             LR    13,14                LOC THIS SUBROUTINE'S SAVE AREA
       0001500             USING SUBRB1,13
351012 0001600             ST    1,R1SAVE             SAVE ADDRESS OF PARAMETER LIST
351016 0001700             LA    0,2
35101A 0001800             LR    12,15                LOAD COVER REG
       0001900             DROP  15
       0002000             USING SUBRB21,12
35101C 0002100             LA    15,ADPR2
       0002200             CALL  (15),MF=(E,(1))
351020         +           DS    0H
               +           CHDINNRA (1)     LOAD POINTER TO PARAM LIST INTO REG 1
351020         +           DS    0H                        HALF WORD ALIGNMENT
351020         +           L     14,16(0,15)
351024         +           L     15,12(0,15)
351028         +           ST    14,72(0,13)               STORE IN SAVE AREA
35102C         +           BASR  14,15                     LINK
       0002300    CALLC    EQU   *                    RETURN POINT AFTER CALL
       0002400    *
       0002500    * ADDITIONAL PROCESSING BY SUBRB2
       0002600    *
35102E 0002700             L     13,4(0,13)           RESTORE REG 13 TO CALLER'S SAVE AREA
       0002800             RETURN (14,12),T           RESTORE REGS AND RETURN
351032         +           DS    0H
351032         +           LM    14,12,12(13)
351036         +           OI    11(13),X'01'
35103A         +           BR    14
       0002900             END
```

Figure 34. Module B Source Listing

```
VIRTUAL
MEMORY
LOCATION STATEMNT   SOURCE

       0000100    * MODULE 'C'
400000 0000200    SUBRC1   PSECT
       0000300             ENTRY SUBRC21
400000 0000400             DC    F'76'                SAVE AREA
400004 0000500             DC    18F'0'
401000 0000600    SUBRC2   CSECT READONLY
       0000700             USING SUBRC21,15
       0000800    SUBRC21  SAVE  (14,12)              SAVE REGS IN CALLER'S SAVE AREA
401000         +  SUBRC21  DS    0H
401000         +           STM   14,12,12(13)
401004 0000900             L     14,72(0,13)          GET LOC THIS PROGRAM'S SAVE AREA
401008 0001000             ST    14,8(0,13)           SAVE IN CALLER'S PSECT
40100C 0001100             ST    13,4(0,14)           SAVE LOC OF CALLER'S SAVE AREA
401010 0001200             LR    13,14                LOC THIS SUBROUTINE'S SAVE AREA
       0001300             USING SUBRC1,13
       0001400    *
       0001500    * ADDITIONAL PROCESSING BY SUBRC2, USING 15 AS CODE COVER
       0001600    * AS NO SYSTEM MACROS USED.
       0001700    *
401012 0001800             LA    0,3
401016 0001900             L     13,4(0,13)           RESTORE REG 13 TO CALLER'S SAVE AREA
       0002000             RETURN (14,12),T           RESTORE REGS AND RETURN
40101A         +           DS    0H
40101A         +           LM    14,12,12(13)
40101E         +           OI    11(13),X'01'
401022         +           BR    14
       0002100             END
```

Figure 35. Module C Source Listing

Table 17. Save Area Linkage

SAVE AREA	WORD	AFTER A CALLS B	STATE-MENT WHERE SET	AFTER B CALLS C	STATE-MENT WHERE SET	AFTER C RETURNS TO B	STATE-MENT WHERE SET	AFTER B RETURNS TO A	STATE-MENT WHERE SET
A	1	F'76'		F'76'		F'76'		F'76'	
	2	xxxx*	A-1400	xxxx*		xxxx*		xxx1	
	3	R(SUBRB1) [350000]	B-1200	R(SUBRB1) [350000]		R(SUBRB1) [350000]		R(SUBRB1+1) [350000]	B-2800
	4	V(CALLB) [30102E]	B-1000	V(CALLB) [30102E]		V(CALLB) [30102E]		V(CALLB) [30102E]	
	5	V(SUBRB21) [351000]	B-1000	V(SUBRB21) [351000]		V(SUBRB21) [351000]		V(SUBRB21) [351000]	
	6	F'01'	B-1000	F'01'		F'01'		F'01'	
	7	A(PLIST) [300054]	B-1000	A(PLIST) [300054]		A(PLIST) [300054]		A(PLIST) [300054]	
	. . .								
	19	R(SUBRB1) [350000]	A-2200	R(SUBRB1) [350000]		R(SUBRB1) [350000]		R(SUBRB1) [350000]	
B	1	F'76'		F'76'		F'76'		F'76'	
	2	R(SUBRA) [300000]	B-1300	R(SUBRA) [300000]		R(SUBRA) [300000]		R(SUBRA) [300000]	
	3	F'0'		R(SUBRC1) [400000]	C-1000	R(SUBRC1+1) [401001]	C-2000	R(SUBRC1+1) [401001]	
	4			V(CALLC) [35102E]	C-0800	V(CALLC) [35102E]		V(CALLC) [35102E]	
	5	. .		V(SUBRC21) [401000]	C-0800	V(SUBRC21) [401000]		V(SUBRC21) [401000]	
	6	.		F'02'	C-0800	F'02'		F'02'	
	7			A(PLIST) [300054]	C-0800	A(PLIST) [300054]		A(PLIST) [300054]	
	. . .								
	19	F'0'		R(SUBRC1) [400000]	B-2200	R(SUBRC1) [400000]		R(SUBRC1) [400000]	
C	1	F'76'		F'76'		F'76'		F'76'	
	2	F'0'		R(SUBRB1) [350000]	C-1100	R(SUBRB1) [350000]		R(SUBRB1) [350000]	
	3			F'0'		F'0'		F'0'	
	4	.		.		.		.	
	5	.		.		.		.	
	6	.		.		.		.	
	7								
	. . .								
	19	F'0'		F'0'		F'0'		F'0'	

*Subroutine 'A' inserts the address of its caller's save area in this word.

Shared Code (PUBLIC) Considerations

The system recognizes a control section as being private or sharable. The latter type is identified by the specification of the PUBLIC attribute associated with the control section *and* the residence of the control section in a shared data set. Each task is allocated its own copy of a private control section; however, allocation of public control sections occurs in such a way as to make the same physical copy of the control section available to all tasks that have allocated the control section to their respective virtual storages.

Sharing object code enhances the efficiency of the system. Paging is reduced since only one copy need be in main storage or on the paging device; in addition, shared routines can be executed simultaneously by more than one CPU.

A reenterable program is one that can be interrupted at any point during execution, entered by another user, and subsequently, reentered at the point of interruption by the first user, and produces the desired results for all users.

The latter type is identified by the specification of the PUBLIC attribute associated with the control section *and* the residence of the control section in a shared data set. Each task is allocated its own copy of a private control section; however, allocation of public control sections occurs in such a way as to make the same physical copy of the control section available to all tasks that have allocated the control section to their respective virtual storages.

In TSS, a standard reenterable program normally consists of one or more named, read-only PUBLIC CSECTS containing instructions and invariant data (relocatable address constants can never be contained in these CSECTS), and a PRIVATE PSECT, consisting of save areas, working storage and variable program data. With this method, each task using the reenterable program is supplied a private copy of the PSECT, the location of which is passed to the reenterable program as a linkage parameter by the calling program.

Other variations of reenterable programs are possible; for example, temporary working storage can also be obtained dynamically by the reenterable program itself, using the GETMAIN macro instruction. In this case, storage is obtained for each task entering the reenterable program, and is private for that task.

To make the reenterable program sharable, the user specifies the PUBLIC attribute in the control section declaration. Specifying the READONLY attribute ensures that the shared code will not in any way modify itself during execution. If the READONLY attribute is not specified, it is the responsibility of each user to ensure the integrity of the routine at any stage of execution, preventing mutual interference.

Prior to assembling the module, a DDEF must be issued defining the job library where the object module is to be stored. Once the module is assembled, the user must grant access to the job library by issuing a PERMIT. This, of course, is not necessary if the object module is stored on a job library previously being shared.

Each user who has been permitted access must then issue a SHARE command, to make the appropriate entry in his catalog for the library. Again, this is not necessary if the user is already sharing the data set. Each time the sharer wishes to use the shared program, he must issue a DDEF for the JOBLIB prior to loading the object module. The object code actually is shared only when each user loads the public control section from the same shared job library. A sharer who link-edits a public control section onto another library receives a private copy each time the object module is loaded from that library.

A program requiring more than 256 shared pages of storage cannot be loaded in public storage. The program will be loaded on private pages, and each user sharing it will receive a private copy.

Efficient Use of Virtual Storage

This section discusses how to use virtual storage efficiently. To understand the guidelines that are presented here, the user should be aware of certain aspects of how the system allocates and manipulates the virtual storage associated with his task.

1. *Control Sections:* All PSECTS and CSECTS are allocated virtual storage starting on a page boundary. Thus, each specification of a new control section incurs a requirement for a new page.

2. *Auxiliary Storage:* As a result of task execution, pages will be brought into real storage as necessary. If the content of a page is altered while it is in real storage, the system will write the changed page on auxiliary storage (i.e., the paging drums and disks) when the real storage space occupied by this page is released for other use (primarily at the end of the time slice). Furthermore, the system will attempt to keep frequently-used pages on the paging drum and seldom-used pages on the disk.

   Pages that are read-only (i.e., not changed during execution) will not be placed on auxiliary storage since they can be reloaded from the initial source.

   Once the initial state of a virtual storage page changes, a copy of this page will be on auxiliary storage until the task explicitly deletes it (e.g., via FREEMAIN) or logs off.

3. *Sharing:* There are two levels of inter-user sharing available in the system: data set sharing via the PERMIT and SHARE commands, and control section sharing via the PUBLIC attributes of the control section declaration. In the latter case, two or more tasks will share a single real core page to reflect the status of a page of their respective virtual storages. Note that the PUBLIC attribute of the control section will not be effective except in the case where the library originally containing the control section is also shared.

   A shared page, once brought into real storage, tends to remain resident there for an extended period of time (as compared to private pages); the intent is to make it immediately available to other tasks besides the task that caused the initial load. Thus, seldom-used control sections should not be shared internally unless it is a requirement.

Guidelines For Efficient Use

Internal Organization of Program Modules

1. In general, the following conventions should be adhered to in setting up the contents of PSECTS and CSECTS:

   A CSECT should contain:
   Executable read-only code
   Data constants
   Non-relocatable literals
   Any other non-modifiable address-free information

   A PSECT should contain:
   Save areas
   Local temporary storage
   Parameter lists
   Address constants and relocatable literals
   Any other modifiable location-dependent information

   There are, however, cases where this results in a less efficient program. For instance, if a module consists of a large PSECT in comparison to the CSECT, and the sum of both is less than 4096 bytes, the CSECT can be incorporated into the PSECT, thereby reducing the page references when the module is executed.

2. Segregate code so that seldom-used code is allocated to a CSECT which is not in the main flow of the program logic.

External Organization of Program Modules

1. Concentrate changeable data (i.e., PSECTS and tables) into as few pages as possible—a changed page requires additional auxiliary storage space.
2. If program module A uses program module B, then attempt to package the read-only CSECTS of A and B together in the same page and the read-write PSECTS of A and B in another page.
3. In general, do not combine CSECTS or PSECTS such that they cross a page boundary. Optimally, a control section or combined control sections should be 4096 bytes in extent.

These combinations of control sections can be effected by using the combine feature of the linkage editor.

Programming Techniques

1. It is most useful to plan the use of virtual storage as if it were a one-page overlay environment—that is, as if only one page of virtual storage could be used without incurring the overhead of a page overlay.
2. Do not zero out virtual storage areas obtained by a GETMAIN; they are set to zero automatically. (Note, however, that the contents of an area reserved by a DS statement are unpredictable, i.e., they should not be assumed to be zero.)
3. Use open (in-line) as opposed to closed (out-of-line) subroutines. If increasing the size of the total program will reduce the page references, do so.
4. Perform your own page suballocation on return from GETMAIN; i.e., use the entire page provided by the GETMAIN before issuing another GETMAIN.
5. Utilize program common for parameters that are frequently referred to, small work areas, etc.
6. For large tables (i.e., greater than four pages), use GETMAIN to allocate the necessary space. When the space is no longer needed, the decision as to whether to reuse it or to release it via a FREEMAIN should depend on the total table size. If the table is less than 10 pages in size, it is more efficient to reuse the space, since the system overhead on a FREEMAIN-GETMAIN sequence is greater than the overhead attached to the paging operations necessary to reuse the space. If the table is greater than 10 pages in size, a FREEMAIN should be used to release the old space and a GETMAIN should be issued for the new table.
   Using GETMAIN-FREEMAIN for space allocation is of particular importance when the program is not to be unloaded. For example, initial virtual storage (IVM) is never unloaded and, if a large table appears in a PSECT, once the table is changed, the changed pages will remain in auxiliary storage until the owning task logs off.
7. Avoid repetitive nonsequential use of subroutines if data can be blocked into or out of the subroutine in a single call; for example, OPEN all data sets at the same time.
8. Avoid multi-page chained tables if possible—a linear search in one page is more efficient than a random search in two pages even if the CPU execution time of the former is greater (true virtual storage execution time is the sum of CPU time and paging time). If the table is larger than a page, use an index table to get to the proper table page directly.
9. Avoid the use of "push down" stacks where the depth of the stack at any time is larger than one or two pages.
10. Avoid sequential programs which build large program or data virtual storage images and then do not refer to them for an extended period of time; use data management (external storage) instead. This eliminates excessive build-up of inactive pages in auxiliary storage.
11. Use job libraries carefully to avoid excessive library searching for program modules.
12. In general, implicit loading of a module (via V-type constants or A-type constants with an EXTRN)

should be used in preference to explicit loading (via the LOAD command, the LOAD macro instruction, or an explicit CALL). An exception to this would be when implicit loading causes loading of many more modules than the program might actually use.

13. In general, unloading will not noticeably increase system performance. An exception would be when a very large number of pages of a program have been referred to, but will not be referred to again.

    Only modules that were explicitly loaded can be explicitly unloaded. If it is desired to unload a module that was implicitly loaded, it is necessary to unload the explicitly-loaded module that caused the implicit loading.

14. At LOGON time, specify control section packing whenever possible. This allows control sections with like attributes to be collected into less memory space. Modules to be executed may thus be compressed into fewer pages, reducing the time required for system paging operations.

Control Section Rejection and Linking Control Sections

During the dynamic linking of object modules, each control section name is checked against control section and entry point names that are already loaded in the task. If a duplication is found, the control section is rejected. Figure 36 summarizes the loader's rejection action.

CONTROL SECTION	LOADER REJECTION ACTION
Named CSECTs, PSECTs, or COMMON	Subject to automatic control section rejection if name duplicates a control section name or any other entry point name already present in the task.
Unnamed CSECTs	Given a unique internal numeric identification when processed by the loader; it is not subject to automatic rejection.
Unnamed (blank) COMMON	Subject to automatic rejection: after one unnamed COMMON control section is processed, any subsequently loaded will be assigned the same name and therefore rejected.

Figure 36. Dynamic Loader Automatic Control Section Rejection

Control sections may also be rejected because of the violation of a naming restriction.

Control section rejection may result in other errors, since none of the entry points defined by the control section are recorded by the system. References to these entry points will be unresolved unless they are satisfied by another control section.

Accidental control section rejection can be avoided by unloading following each execution. However, in some cases, it is desirable to allow a control section to be linked from one execution to the next.

If an UNLOAD command is issued after a module has completed execution all record of control section and entry names in that module are removed from the task's allocated storage. Any subsequent module that is loaded containing a CSECT with the same name would have storage allocated as if it were the first usage.

When the user wishes to pass the contents of the same named control section from one program to the next, the UNLOAD command should not be entered. In this case, the second program's references to the control section would be resolved to the control section that was allocated storage with the first program, if both have the same name.

Recovering from Errors When Dynamically Loading

If a program consists of more than one object module, the modules are dynamically linked by the system's dynamic loader at execution time. The dynamic loader takes all of the implicit external references in the module that is explicitly loaded or run and resolves them by searching the program library list. It is possible that while the loader is linking the object module(s) into the user's virtual storage, several error conditions may arise that affect the eventual execution of that program.

- *Name to be loaded or run not found in library:* Either the user has specified the wrong name in the LOAD or CALL command or the job library containing the object module has not been defined in the task and, therefore, is not in the program library list. If the latter is the cause of error, the user in conversational mode can merely enter the DDEF defining the job library and reissue the LOAD or CALL command.
- *Unresolved references:* If an object module has an external reference that cannot be located in any of the libraries in the program library list, a diagnostic is issued specifying the name in the reference. Further linking of other object modules is not suspended, however, so that the explicitly-named object module and, possibly, other object modules that were referred to implicitly have been placed in the user's virtual storage. If the error occurs in a CALL command, execution of the program is not initiated.

  If the user wishes to execute his program regardless of the error, he may reissue the CALL command. He must, however, repeat the name of the module named in the original CALL command. This is necessary to define the point at which execution is to be initiated.

  If the user anticipates that an object module will have unresolved references, he should first issue a

LOAD command naming the module, followed by a CALL with an operand. This procedure is recommended for a nonconversational task, since the user can be assured that execution will be initiated regardless of unresolved references.

If the user does not wish to run the version of the program that has been loaded into his storage, he must issue an UNLOAD command. If he does wish to run this version of the program, he can then enter a DDEF defining a job library that was missing in the first load attempt. A LOAD or CALL issued at this point causes the entire linking procedure to be redone.

- *Duplicate entry points:* This condition may occur when dynamically linking an object module from one library with a module from another library. In this case, since the second entry point definition is disregarded, all further references to the ENTRY name may be erroneously resolved.

  The user should take some corrective measures before attempting to LOAD or CALL again. (A possible correction might be to change the ENTRY name by link-editing the object modules onto another JOBLIB.) To avoid the possibility of such duplications when working with a new library, the POD? command can be used to list the directory of the library. The user can then circumvent the problem by setting up an appropriate program library list before he attempts to load his program.

Library Management

Program Library List Control

A program in TSS can consist of one or more object modules. All programs in TSS are stored in object module form in program libraries that are partitioned data sets. A program consisting of only one object module is stored entirely within one library; however, if a program consists of several object modules, these modules may reside in different libraries, depending on how the user has stored them. During linkage editing and during execution, the system can automatically retrieve all object modules required, if the user has defined the libraries in which those object modules are contained. The manner in which the user does this is described in the following paragraphs.

There are four categories of program libraries:

- System library (SYSLIB)
- User library (USERLIB)
- User-defined job libraries
- Other user-defined libraries used in linkage-editing

TSS does not allow a library to contain more than one declaration of any external symbol. In this sense, named and blank COMMON are not considered external symbols since they are not listed in the directory of the library.

The system library contains service routines provided by the installation. For example, it includes service programs, and the installation's standard subroutines and functions.

The user library is the private library assigned to each user when he is joined to the system. This library is automatically defined for him and an entry made in his catalog by the system. His user library is thus available each time he logs on. If the user does not employ job libraries in a task, all the object modules resulting from his use of the language processors are placed in his user library.

The user may wish to restrict his user library to checked-out, standard object modules that he executes frequently or that he uses frequently in the buildup of other object modules.

The program library list is a defined hierarchy of program libraries. It is set up at log-on time, and initially consists of the user library and SYSLIB.

The library at the top of the list automatically receives all object modules resulting from language processing. As noted above, if no job libraries are defined, the library at the top of the list is always the user library. However, the user can specify that a job library be added to the program library list to receive the output of the language processors. He does this by issuing a DDEF command defining that job library and containing the operand OPTION=JOBLIB. When this command is executed, the name of that job library is added to the top of the program library list. That library then receives all subsequent module output of the language processors until another job library is defined (and is thus at the top of the list) or until a RELEASE command is issued for that job library. A job library must always be a partitioned data set and may be defined on public or private volumes.

In addition to using the program library list to store object modules, the system uses this list to control its order of search when looking for object modules that must be loaded at execution time. The library at the top of the list is searched first, then the next-to-the-top library, etc.; finally, the user library and SYSLIB are searched.

In summary, the user has the following basic library setups for handling the object modules produced by the language processors.

- User Library—As this is always available and is always searched, the user may wish to reserve this for frequently used checked-out programs. All user's USERLIBS are kept on public volumes and, hence, are always mounted on system devices.

- Session JOBLIB—By issuing a DDEF command for a new library at the beginning of a session, a user can create a library to contain all modules assembled during the session. By not cataloging this new library during the session (if private), he can discard modules not to be used again or not yet debugged.

- Cataloged Private Volume JOBLIB—A user can direct output to and retrieve from a library of infrequently-used modules by issuing a DDEF command for a cataloged job library that resides on a private removable disk pack. In a nonconversational task when using private job libraries, the user must request (via SECURE) a device for that job library. Modules may be entered in such a library:

  Automatically if the library is the latest defined one in the session.

  By link-editing it from his USERLIB, session job library or public-device job library and specifying to the linkage editor the desired private device job library as the output destination. Cataloged libraries on private volumes may also be shared by several users.

- Cataloged Public Volume JOBLIB—This type of library may be useful to the user in setting up (and using) a library of frequently-used programs whose names and external symbols conflict with other programs in USERLIB. As an example, versions of frequently used programs may be set up with one in USERLIB and another in a job library. Cataloged libraries on public volumes may be shared among users.

The program library list can also be used, during link-editing, to define the following for the system:

- The library that is to receive link-edited object module.

- The sequence in which libraries are to be searched if the system must find other object modules to define references in the link-edited object module.

The fourth category of libraries may be defined by a DDEF command with the operand keyword JOBLIB omitted. Such libraries may be referred to by a specific link editor INCLUDE statement, but they are not listed in the program library list, and hence are not included in the automatic library search, nor are they available to the dynamic loader.

Refer to Appendix B and to the publication *Linkage Editor* for an explanation of link editor program libraries.

Program Versions

Since one library cannot contain more than one control section entry point or module with the same name, different versions of the same program must be kept in different libraries. For example, a user may have a checked-out program in his USERLIB and wish to reassemble the program with modifications, but retain his original version until the new version has been checked out. A DDEF with a JOBLIB option causes the new module to be stored on the job library rather than USERLIB. The user may continue after assembly with his checkout of the new version, since any subsequent LOAD or CALL command in the task naming the module retrieves the new version from the job library. If, when the new version has been successfully tested, he wishes to replace the old version with the new version, he may link-edit the new version onto his USERLIB. He may also use the TV, VV, or CDS commands to copy a program module from one library to another. If he does not wish to retain the new version, he must either ERASE the module on the job library or RELEASE the job library. Releasing the library removes it from the program library list, automatically causing subsequent retrievals of that module to revert to USERLIB. Erasing the module does not remove the job library from the program library list, but any subsequent references to that module are resolved from USERLIB after the job library has been searched unsuccessfully.

To facilitate orderly maintenance of programs within various job libraries (and USERLIB), the POD? command is available. POD? enables the user to obtain on SYSOUT a list of the member names (and optionally the alias names and other member-oriented data) of individual members of cataloged VPAM data sets.

Sharing Libraries

A user may allow another user to share (i.e., access) one or more of his cataloged job libraries. When the owner permits access to his job library, all of the object modules on that data set are usable by the sharer. This facility does not imply that if the owner and/or one or more sharers use the same program at the same time they are sharing (co-using) the same copy in real storage. This aspect is controlled by the PUBLIC attribute assigned to a control section at assembly time.

The data set owner issues a PERMIT command to designate the other users who may share his job library and indicate the level of access those users may have:

- Read-only access: The sharer may use the object modules on the library, but may not add, replace, or erase a module.

- Read-and-write access: The sharer may use any object module on the library and may add or replace modules. He may not use the ERASE command to delete a module from the library.

- Unlimited access: The sharer, in effect, can treat the library as his own; thus he may even erase modules.

(Note that the implications of "read-only," "read-write," and "unlimited" are slightly different when specified by the user for his use of his own data sets and when specified in a PERMIT command. The owner of a data set may permit any level of access he wishes regardless of the access designator in the owner catalog. For example, if the owner catalog is marked "read only", the owner may not write into his own data set, *but* he may permit a higher level of access (read/write or unlimited) to a sharing user. This flexibility must make the data set owner very cautious with critical data sets he has entered into the system.)

To gain access to a data set for which he has been previously authorized, the sharer must issue a SHARE command. The SHARE command places an entry for the owner's data set name in the sharer's catalog. The sharer may then enter a DDEF command for the data set (with the JOBLIB option) in each task where he wishes to include the library in his program library list.

Groups of job libraries with names having common higher-order components can be specified by using partially-qualified names when the PERMIT is issued. For example, an owner of two job libraries named TRACK.SUB1 and TRACK.SUB2 can allow sharing of both libraries by using the partially-qualified name TRACK in the PERMIT command. In this case, the sharer must also use the partially-qualified name (as the dsname_2 parameter) in the SHARE command, even though he only wishes to use one of the job libraries.

Table 18 lists the commands applicable to shared data sets and the effect of the command on the user's catalog.

System Naming Rules

User-Assigned Names

The following names resulting in external symbols are supplied by the user in his assembler language source program or during assembly.

- Module name
- Control section names
- ENTRY names

All external symbol definitions in a module, including the module name, must be unique. In addition, since the system does not allow any one library to contain more than one definition of a particular external symbol, each name (except names of COMMON control sections and unnamed CSECTS) must be distinct from any other symbol contained on the library that is to receive the object module. It is valid to have the same names on different libraries. Since a named or blank COMMON control section is not listed in the directory of the library as an external name associated with the module name, it does not have the preceding restriction. Also, since it is not listed in the directory, it cannot be explicitly referred to by name (i.e., it cannot be loaded by its COMMON name).

The POD? command can be used to list the external symbols in a library to avoid duplication.

Reserved Names

External Symbols

The user must never assign a name beginning with the characters SYS. These letters are reserved for certain system programs. Any module stored on the user library or a job library starting with these symbols can never be retrieved by that name for execution, since resolution of SYS symbols for loading and running is always attempted from the system library. In addition, a diagnostic is issued if a module, loaded by another name, contains an external symbol definition beginning with SYS.

The user should also be careful to avoid accidentally duplicating the names of IBM-supplied FORTRAN subprograms. Generally, he should avoid the use of all external symbols starting with the characters CHC, or any FORTRAN-supplied subprogram name (i.e., SIN, COS, etc.), unless he specifically wishes to use this FORTRAN subprogram.

Internal Symbols

The user should avoid assigning an internal symbol beginning with the characters CHD, since system macro instructions use these characters and might cause a duplication of internal symbols.

Reserved Names Associated with Data Sets

The following list contains the reserved names which are assigned to system functions:

RESERVED DDNAMES	RESERVED DSNAMES
SYSLIB	USERLIB
SYSULIB	SYSLIB
SYSIN	
SYSOUT	
PCSOUT	

The following names are assigned to the assembler output data sets:

SOURCE.module—is the data set name assigned to the line data set of source statements constructed during the assembly. If the input to the assembler is from a prestored data set, then the user must assign the name SOURCE.module to the data set prior to the ASM command.

LIST.module—is the data set name assigned to the data set created for all the listings optionally selected by the user. The system automatically catalogs each new generation. Printed output is optional and must be requested via the PRINT command. The listing data set is a generation data group, established the first time the module name was encountered during language processing.

Table 18. Shared Data Set Commands

COMMAND	BY OWNER	BY SHARER
PERMIT	Must be issued prior to the SHARE command by the sharer(s).	Not allowed. A user cannot permit access to a data set that he does not own.
SHARE	Not allowed.	Must be issued prior to any other references to the data sets. Once issued, the sharer may access the data set until he issues an ERASE or DELETE. The SHARE command places an entry in the sharer's catalog, so that a further CATALOG command is not necessary.
ERASE	The owner may always erase a member (object module) from his job library or erase the entire library. If he erases the job library, the entry in the sharer's catalog is not removed. The sharer(s) must issue a DELETE command to remove the entry from their own catalog.	A sharer may only erase if he has been granted unlimited access. If he then erases an object module, neither the sharer's nor the owner's catalog is affected. If he erases the entire job library, both his catalog entry and the owner's are removed.
DELETE	The owner may delete a library or group of libraries from his catalog. An object module alone cannot be deleted. When the owner deletes a shared job library, the sharer's catalog entry is not removed.	A sharer may delete his catalog entry for a job library without affecting the owner's catalog. The sharer must reissue a SHARE command if he again wants to refer to the data set that was deleted.
CATALOG	The owner may catalog a fully-qualified data set name. If that name is a component of a partially-qualified name that the owner has permitted to be shared, all sharers have immediate access to the newly cataloged data set. If an owner changes the name of a single data set to which he permitted access using a fully-qualified name, each sharer must delete his catalog entry and reissue the SHARE command with the owner's new name.	A sharer that has been granted unlimited access may change or add entries to the owner's catalog. If he is permitted to share a group of data sets, he may catalog a new data set into the group, but he must include as part of the name the partially-qualified name that he used in the SHARE command. If he changes the name of one of the data sets in the group, the new name must still contain the partially-qualified name. A sharer who has been granted unlimited access to an individual data set may never change the data set name.

Appendix D. Interrupt Considerations

This appendix discusses the more common interrupt considerations when programming in TSS. The sections of this appendix discuss:

1. TSS operation when a program interrupt occurs in a program where the programmer does not use the SIR, DIR, and SPEC macros to control interrupts.
2. The effect of an interrupt caused by pushing the attention button at the terminal, and resuming execution following the interrupt.
3. TSS facilities for user-written interrupt handling routines, and considerations for the processing of interrupts.

Program Interrupts

If a program interrupt occurs during program execution and the user has not specified his own interrupt-handling routine for the interrupt type, the interrupt is processed by a system service routine.

A diagnostic message appropriate to the cause of the interrupt and the virtual program status word is issued to the terminal (or in nonconversational mode, is written to SYSOUT) and the task is returned to command mode. In addition to a diagnostic specifying the type of interrupt that has occurred, the user is supplied the following information to locate the source of the interrupt:

PSW = xxxxxxxxxxxxxxxx. THE INTERRUPT OCCURRED IN CSECT xxxxxxxx WITH A DISPLACEMENT OF xxxxxxxx FROM THE BEGINNING OF THE CSECT.

If the interrupt occurs in a conversational task, an underscore is typed at the terminal requesting that the user enter a command. The user can then use PCS statements for problem investigation.

In a nonconversational task, if a data set has been supplied with a ddname of TSKABEND, this data set becomes the task's new SYSIN. This data set can contain a sequence of PCS commands and statements to obtain a selective dump of the program before terminating the task with a LOGOFF command. If a TSKABEND data set has not been supplied, the task is terminated.

The following table (Table 19) shows each type of program interrupt, and the possible causes for the interrupt.

Table 19. Types of Program Interrupts

TYPE OF INTERRUPT	CAUSE OF INTERRUPT
Operation Code	The operation code is not valid.
Privileged Operation	A privileged instruction has been encountered in a program executing in a nonprivileged state.
Execution	The subject instruction of an Execute (EX) is another Execute.
Protection	The storage key used in an instruction fetch or a data reference does not match the protection key in the PSW.
Addressing	An address specifies a location that has not been allocated to the task's virtual storage.
Specification	1. Reference has been made to virtual storage with an address that does not specify an integral boundary for the unit of information. 2. The R_1 field of an instruction specifies an odd register address for a pair of general registers that contain a 64-bit operand. 3. A floating-point register other than 0, 2, 4, or 6 has been specified. 4. The multiplier or divisor in decimal arithmetic exceeds 15 digits and sign. 5. The first operand field in a decimal multiply or divide is smaller than or equal to the second operand field.
Data	1. The sign or digit codes of operands in decimal arithmetic, editing operations, or Convert to Binary (CVB) are incorrect. 2. Fields in decimal arithmetic overlap incorrectly. 3. The decimal multiplicand has too many high-order significant digits.
Fixed-Point Overflow	A high-order carry has occurred or high-order significant bits have been lost in a fixed-point addition, subtraction, shifting, or sign control operation.
Fixed-Point Divide	1. Division by zero has been attempted. 2. The quotient has exceeded the register size in fixed point division. 3. The result of Convert to Binary (CVB) has exceeded 31 bits.
Decimal Overflow	The destination field is too small to contain the result in a decimal operation.
Decimal Divide	The quotient has exceeded the specified data field.
Exponent Overflow	The result characteristic has exceeded 127 in floating-point addition, subtraction, multiplication, or division.
Exponent Underflow	The result characteristic is less than zero in floating-point addition, subtraction, multiplication, or division.
Significance	The result of a floating-point addition or subtraction has an all-zero fraction.
Floating-Point Divide	Division by a floating-point number with a zero fraction has been attempted.

NOTE: Fixed-point overflow, decimal overflow, exponent underflow and significance interrupts can be masked off by setting the appropriate bits in the program mask. Such interrupts are ignored and are not recognized by the system. These interrupts can also be processed by the user with the SIR, DIR, and SPEC macro instructions.

Attention Considerations

Interrupting Execution

When an ATTENTION interruption has been encountered in a nonprivileged program, the system will respond with one of the following three characters: !, *, or ___. The response character will indicate to the user the situation at the time the interruption occurred. (Note that the user program may have been executing a privileged command string when it was interrupted.) The user may then request a particular system action by issuing the appropriate command or pressing the ATTENTION or RETURN key. The possible user actions and corresponding system responses are shown in Table 20. If the attention button is pushed while a PCS DISPLAY response is being typed on the terminal, it is assumed the user wishes to cancel any further lines from the DISPLAY request.

If a user program has invoked a privileged system program (such as the OPEN routine) and it is in operation when the attention button is pushed, the interruption is not honored until that program has completed its processing and has returned tc the user program. If a user's program is executing, the interruption is honored immediately.

Combining the PCS STOP command with an AT also produces an interruption. The symbolic location at which execution was stopped is printed at the terminal. To find the location when a program is interrupted by an attention, the user may enter a STOP command.

Once the underscore has been typed, the user may enter any commands or PCS statements he wishes. He may even REMOVE the PCS statement that caused the interruption, so no subsequent execution of the instruction will be interrupted.

Levels of Interruption

Whenever a nonprivileged program is interrupted, and another user program is invoked, the status registers and PSW of the interrupted program are saved in a system table called the stack table. The interruption may have been an ATTENTION, a PCS AT, a CLIC, CLIP, COMMAND, PAUSE, or OBEY macro instruction, or a program interruption. The interrupted program remains active, with its status saved, until such a time when it again receives control. The RTRN, PUSH, and EXIT commands, issued after an ATTENTION interruption, can be used to manipulate the stack table. The function of these commands and the system response are shown in Table 20.

The current level of interruption is an indicator of how much of the stack table is in use. One level is taken whenever a program's status is saved; the level is freed when the interrupted program regains control. A maximum of ten levels are available. The user is at level zero when no program's status is currently being saved. The STACK command, issued after an ATTENTION interruption, will display the names of all active programs in the stack table in descending order from the currently active down. See *Command System User's Guide* for a more detailed description of these commands.

Resuming Execution

The GO command is used to resume execution after an attention interrupt or a PCS STOP command. To resume at the point of interruption, a GO command is sufficient; however, the user can specify an alternate point at which to resume or restart, using a BRANCH command. He might, for example, in a two-phase program, want to skip directly to the second phase to continue processing. The interrupted program should not be unloaded and reloaded as it may not resume execution at all.

Intervention Prevention Switch (IPS)

The Intervention Prevention Switch (IPS) may be set by the user to protect a segment of code, preventing the interruption of his program during the execution of that segment. Setting and unsetting the switch is accomplished by means of the following being coded into his program, surrounding the code he wishes protected:

```
L       R5,=V(SYSAAA)
USING   CHAAAA,R5
AP      AAAIPC,=P'+1'   Sets the IPS
.       (code to be
.       protected)
.
SP      AAAIPC,=P'+1'   Resets the IPS
EX      AAASW           Passes control
                        to the user
.
.
.
COPY     CHAAAA         Creates DSECT
                        for symbols
```

Table 20. Responding to Attention Interruptions

When the ATTENTION key is pressed, the system responds with one of three condition symbols:

By these actions, the user calls for the system reaction listed in the block under the corresponding condition symbol:	! (to denote the interruption of nonprivileged programs or commands)	* (to denote the interruption of an unfinished, privileged command string)	_ , or user's command prompt (denotes completion of program or command string)
By issuing the GO command . . .	the current user program is resumed	the most recently interrupted user program is resumed and intervening command strings are cancelled	
By issuing any command . . .	the command is executed	the command is executed and the current command string is cancelled	the command is executed
By pressing the RETURN key . . .	the current user program is resumed	the current command string is resumed	the system prompts the user to enter a command
By pressing the ATTENTION key . . .	the system returns an exclamation point (nothing is changed)	the system returns an asterisk (nothing is changed)	the system prompts the user to enter a command
By entering STRING to list remaining commands in an interrupted string . . .	the system displays the unexecuted command string, if it exists	the system displays unexecuted commands in the current command string	the system returns a diagnostic message
By entering STACK to list names of active nonprivileged programs (in the order in which they may be retrieved). . .	the system displays the names of active user programs at every ATTENTION level		
By entering EXIT to end the currently active program . . .	ends the currently active program, resumes command string if it exists	ends the most recently interrupted program and resumes its associated command string, cancelling subsequent command strings	
By entering RTRN to cancel command strings and user programs at every attention level . . .	command strings and user programs are cancelled by the system at every ATTENTION level		
By entering PUSH to save the status of the currently active program	the system saves the status of the currently active program in ISA Long Save 1 (ISALS1)		

Where the IPS has been set and the user hits the attention key, an exclamation point will be printed out at his terminal. However, control will be passed to the user only after the EX AAASW instruction has been processed. If no interrupt has occurred preceding this instruction, it will be treated as a no-op and execution will proceed normally.

If the user wishes to ignore the setting of the IPS, he may do so by causing a second attention interrupt. The second attention overrides the IPS setting and causes the terminal to be opened.

Writing Interrupt-Handling Programs

Time Sharing System provides facilities to enable the user to write his own interrupt-handling programs. Interrupt-handling consists of responding to task interrupts. Significant features of the interrupt-handling facility are priority interrupt control and interrupt delay, both of which are discussed below.

Refer to the publication *Assembler User's Macro Instructions* for a detailed description of each interrupt macro instruction.

There are six types of task interrupts: program, supervisor call, external, asynchronous, timer, and input/output.

The problem programmer may decide how to respond to any of the six types of interrupts or he may elect to ignore certain interrupts.

This section discusses:

- How to establish an interrupt handling routine.
- Processing the interrupt.
- An illustration of the sequence of events as might occur when a user specifies interrupt-handling routines utilizing all of the facilities of TSS interrupt management.

Establishing Interrupt Routines

When a user wishes to provide an interrupt-handling routine to service any of the possible task interrupts, he must specify to the system the conditions for servicing an interrupt and the points at which he wishes to activate and deactivate his interrupt-handling routine.

The Specify Entry Condition macro instructions define Interrupt Control Blocks (ICBS) which specify what task interrupts are to be processed, under what conditions the user's interrupt routine is to be entered, and the entry point address of the user's interrupt routine. The following Specify Entry Condition macro instructions are available:

SPEC—Program interrupts
SSEC—Supervisor calls (SVCS)
SEEC—External interrupts
SAEC—Asynchronous interrupts (attention key)
STEC—Timer interrupts
SIEC—Synchronous I/O interrupts

When the normal L-type form of the macro instruction is written, no linkage is performed. The interrupt control block is built and the name included on the Specify Entry Condition macro instruction becomes the name of the ICB. This name is referred to in the SIR and DIR (Specify and Delete Interrupt Routines) macro instructions. The E-type form of the macro instruction may be used in-line to modify the contents of an already existing ICB.

Along with the macro instruction specifying the entry condition, the user must reserve 16 bytes of storage for a communication area (COMAREA). This area is used to communicate with the interrupt-handling routine and, upon interrupt, will describe the conditions causing the interrupt.

The SIR macro instruction makes the interrupt routine specified by the ICB address available and sets its priority. Any user interrupt routine can be made unavailable by the DIR macro instruction. Another SIR macro instruction will make the interrupt routine available again.

It should be noted that if the SAEC macro instruction is used to get control for interrupts from the attention key, the USATT macro instruction must be issued after the SIR macro instruction. This will cause control for attention interrupts to be taken away from the system and given to the user-designated routine. Likewise, the CLATT macro instruction should be issued before the DIR macro instruction when making the attention interrupt handling routine unavailable. This returns control of attention interrupts to the system. The DCB parameter required for the SAEC macro instruction to be used for attention is the system symbol SYSINDCB. An EXTRN statement must also be included for SYSINDCB.

Processing an Interrupt

When an interrupt occurs, an exit is taken from the interrupted routine and control is passed to the entry point of the user-specified interrupt routine. Information identifying the type of interrupt that occurs is made available in a communication area (COMAREA). Using this information, the interrupt routine written by the user can perform any calculations necessary, including issuing input/output macro instructions if the user should wish to do so, and whatever else is necessary to respond to the interrupt. The INTINQ macro instruction may be issued in the interrupt routine. Issuing a RETURN macro instruction causes control to be returned to the interrupted routine at the instruction following that which caused the interrupt, or to another enqueued interrupt routine.

If interrupts are not disabled by an SAI macro instruction, higher-priority interrupts will interrupt a routine of lower priority. If more than one interrupt-handling routine is defined with the same priority and for the same type of interrupt, the interrupt-handling routine associated with the first issued SIR macro instruction will take precedence.

Upon entry to an interrupt handling routine, register contents are as described below. See Figure 37 for a description of the items referred to below.

An interrupt-handling routine should be coded with normal type I linkage conventions. Figure 38 illustrates the interrupt control block format. On entry to the interrupt-handling routine, although it is not necessary to save the contents of the general registers, if a SAVE is issued, the registers will be saved in an area provided by the system and pointed to by register 13. Register 14, however, should always be saved if it is going to be changed in the interrupt routine, and restored prior to the RETURN.

REGISTER	CONTENTS
0	Address of a save area. This save area contains information on the condition of the interrupted program, including the contents of all registers at the time of the interrupt. The interrupt routine can use this area to change the return point to the interrupted program (old VPSW), or the contents of any of the other registers.
1	Address of the appropriate ICB whose first word contains the address of the COMAREA, and whose second word contains the address, if applicable, of the associated DCB.
2-12	Same as at the time of interrupt.
13	Address of the register save area to be used by the interrupt handling routine. The 19th word of this save area contains the location of the interrupt routine's PSECT.
14	Address of the location in the control program to which control will be returned after execution of the interrupt routine. If this is changed by the interrupt handling routine, it must be reset before the RETURN.
15	Address of the entry point of the interrupt handling routine. (This register can be used to provide addressability.)

It is important to remember that, on return to the interrupted routine, *all* general and floating-point registers will be restored to the contents as stored in the area pointed to by register zero. To illustrate, if an interrupt-handling routine is to set a particular value in general register 7, the interrupt routine must store that value into the appropriate word (in this case, 48 bytes) beyond the address supplied in register zero. Note that since register zero cannot be used as a base register, that address must first be transferred to a valid base register.

On a normal return from an interrupt-handling routine, once any queued interrupts have been serviced, execution resumes in the interrupted routine at the instruction following the point of interruption. If the user wishes to modify this return point, he must modify the old VPSW located in the area pointed to by register zero. The address of the desired return point should be stored in the second word of the double-word VPSW. The condition code and program mask in the first word

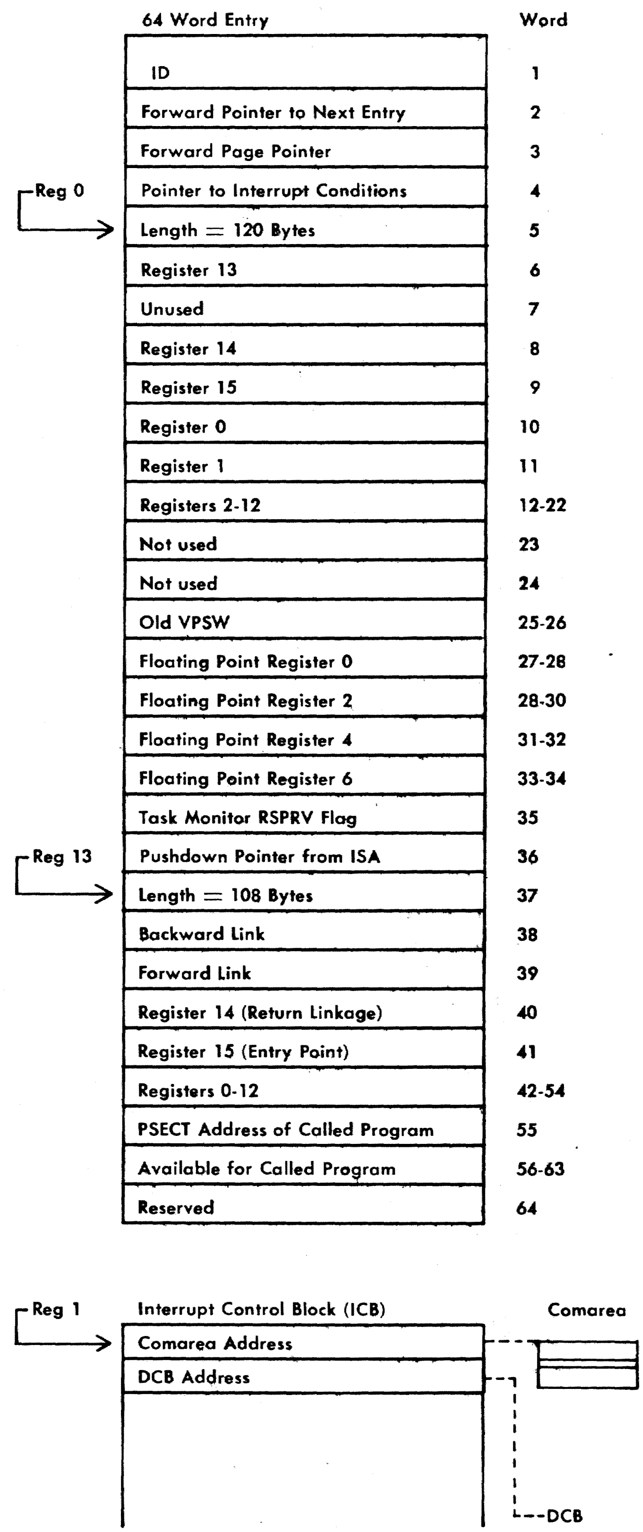


Figure 37. Information Available Upon Entry to an Interrupt Routine

of the VPSW may also be modified if desired. The format of the VPSW is given in Figure 39 for reference. Note that the preceding procedures apply to nonprivileged interruption-handling routines only. For a description of privileged interruption-handling procedures, see the *System Programmer's Guide.*

Interrupt delay is accomplished through the SAI and RAE macro instructions, which allow the user to inhibit or permit further interrupts while his interrupt handling routine is in control. The SAI macro instruction delays interrupts to the task. These interrupts are not lost, they are queued up in the supervisor. The RAE macro instruction permits interrupts to the task (including those that were queued while the SAI was in effect).

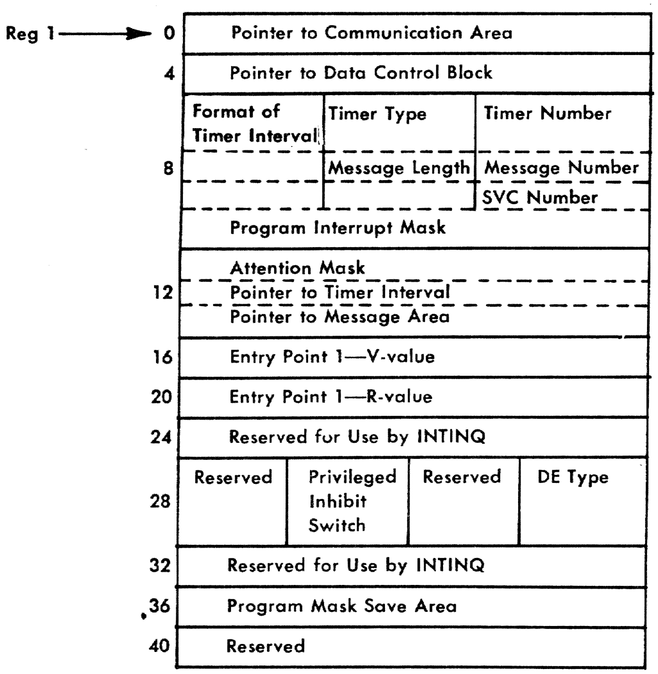


Figure 38. Interrupt Control Block (ICB) Format

The INTINQ macro instruction allows the user's interrupt-handling routine to optionally:

1. Relinquish control until a specified interrupt occurs (MODE=R).
2. Enter a wait-state until a specified interrupt occurs (MODE=W).
3. Branch conditionally, depending on whether a specified interrupt has occurred (MODE=C); if the interrupt has occurred, it is dequeued.
4. Delete pending interrupts (MODE=CLEAR).

Figure 40 illustrates the use of all the interrupt-handling facilities. The circled numbers in Figure 40 are explained below. The example shows the logical sequence of events as might occur in a situation where three interrupt routines are specified, two with equal priority.

In the PSECT of this program, the following instructions are written:

```
ICB1     SxEC     EP=ROUTINE1,COMAREA=CA1,
                                    INTTYP=. . . .
ICB2     SxEC     EP=ROUTINE2,COMAREA=CA2,
                                    INTTYP=. . . .
ICB3     SxEC     EP=ROUTINE3,COMAREA=CA3,
                                    INTTYP=. . . .
CA1      DS       4F    COMMUNICATION AREA 1
CA2      DS       4F    COMMUNICATION AREA 2
CA3      DS       4F    COMMUNICATION AREA 3
```

Note that in this illustration the second digit of the Specify Entry Condition macro instruction (shown as *x*), having no significance in the example, is not filled in. Also, the interrupt type is not specified since this parameter varies with the particular macro instruction.

1. In the main routine the user has activated interrupt-handling routines under the conditions described in the Specify Entry Condition macro instructions. Assuming that the third SIR specifies the same group of interrupts as the second SIR, the third has an implied priority over the second.
2. An interrupt of the type first specified occurs in the main routine. The system, recognizing that a user's interrupt-handling routine has been specified:
   a. Saves the general and floating point registers in the supervisor area (setting register zero appropriately).
   b. Saves the interrupt information in the indicated communication area (setting register one to the ICB, the first word of which contains the address of the communication area).
   c. Sets register 13 to a standard TSS save area in which the address of the PSECT of the interrupt-handling routine has been stored in the 19th word.
   d. Sets register 15 to the entry point of the interrupt-handling routine, and register 14 to the address (in the system) to which the interrupt routine should return.
3. The user issues a SAI macro instruction. With this instruction, the user requests that further interrupts of any type be delayed (enqueued) until this interrupt-handling routine re-enables them.
4. When the interrupt specified by the second SIR macro instruction occurs, it is not honored immediately because of the previously issued SAI. Instead, it is enqueued by the system and will be honored when the user's interrupt routine issues a RAE or returns. The interrupt may, in fact, never be honored if an INTINQ dequeues it.

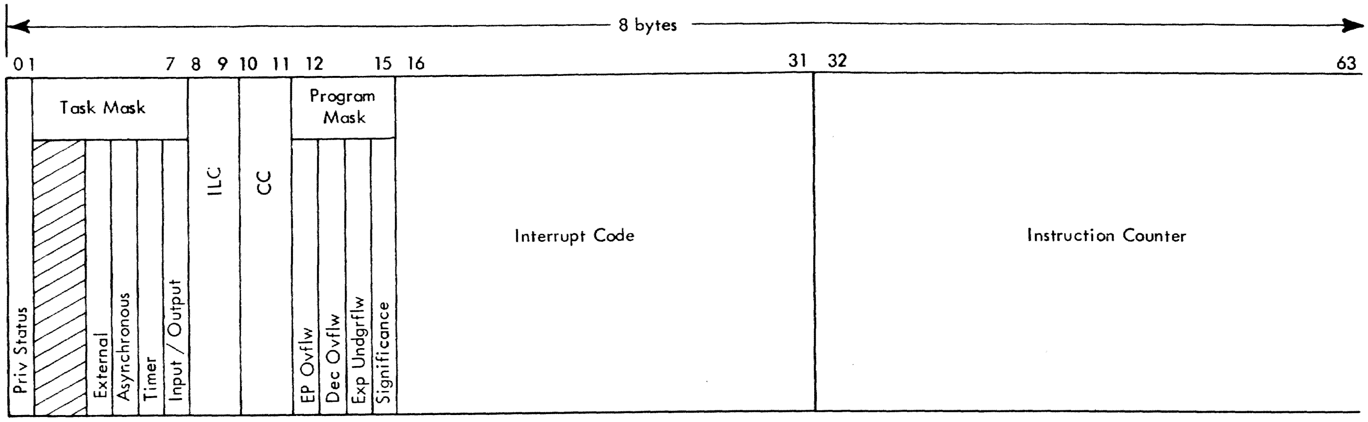


Figure 39. Virtual Program Status Word (VPSW)

5. This interrupt is treated the same as the interrupt at 4, except that its priority being higher will cause it to be honored before 4, unless it is dequeued.

6. This macro requests that the system examine the ICB for the interrupt types specified. If an interrupt of the type is queued, the branch will be performed. If it is not, execution will continue with the next sequential instruction. When the interrupt is found to be queued, and the branch is taken, the interrupt is then removed from the queue.

7. When the RAE macro instruction is issued, all interrupts that have been enqueued since the SAI are honored. In this case, since interrupt 2 was dequeued by the INTINQ macro instruction, the only enqueued interrupt in the list is interrupt 3. When the interrupt 3 routine returns control (to the next sequential instruction beyond the RAE in the main routine), the main routine then returns. The system then resets the general and floating-point registers to the contents of the saved area (as pointed to by register zero on entry) and resumes execution at the location specified in the old VPSW.

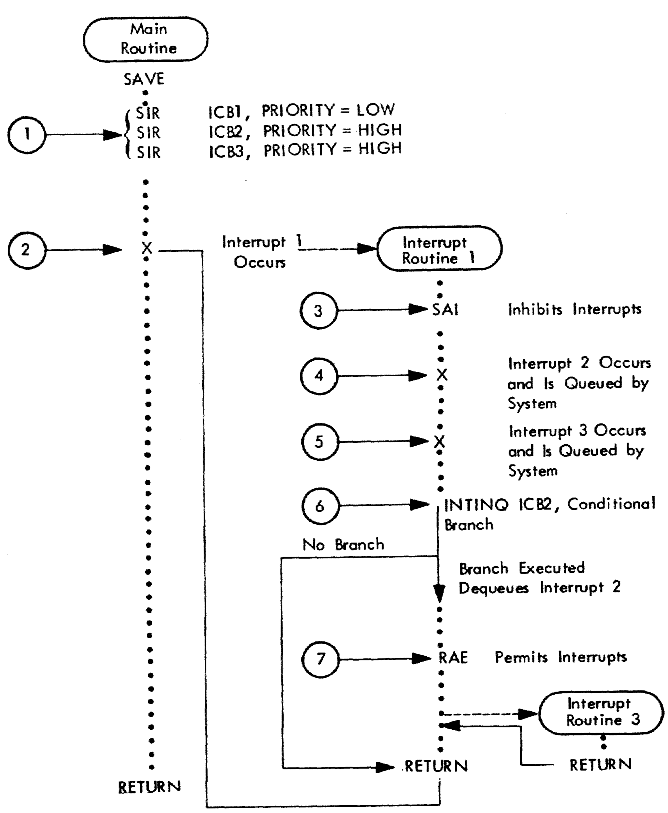


Figure 40. Illustration of Interrupts Being Serviced

This appendix discusses use of the DDEF command (Figure 41) and includes the following topics:

- Preparing a DDEF for a new data set (Table 21)
- Preparing a DDEF for an existing data set (Table 22)
- Specifying DCB parameters in a DDEF command (Table 23)
- DDEF and data set organization requirements for language processing
- DDEF and data set organization requirements for commands (Table 24)
- SECURE command requirements for non-conversational tasks
- DDEF considerations for multiple executions in the same session

The purpose of the DDEF command is to allow the user to specify those data sets that are to be created or processed during the execution of his program or the commands he has issued. The DDEF command:

- Associates a DCB with a named data set—the ddname appears in both a DCB and a DDEF.
- Names the data set to be created or processed—DSNAME parameter
- Requests, when necessary, device and device space allocation and volume mounting—UNIT, SPACE, and VOLUME parameters.
- Defines labeling conventions to be used and certain label information if desired—LABEL parameter
- Declares the disposition of the data set (i.e., whether a new data set is to be created or an existing one is to be processed)—DISP parameter
- Allows declaration of data set attributes and characteristics—DSORG and DCB parameters

The DDEF command (or macro instruction) causes the parameters included therein to be recorded in an entry in a table known as the Task Data Definition Table (TDT), which is maintained by the system. This table is not of concern to the user; it is described here only as an aid to understanding the functions of the DDEF command. Each entry is the result of one DDEF command or macro instruction and is identified by the ddname which appeared in that command or macro instruction.

The complete DDEF command is shown in Figure 41. For clarity, the DCB parameter, with all possible suboperands is shown separately.

The system locates data set characteristics and attributes by searching the TDT for an entry with the same ddname as appears in the data control block (DCB) specified in the user's program (or system routine).

The TDT is discarded each time a session is completed (i.e., log-off time). Hence, a DDEF command must be given for each data set to be processed in every session. For some data sets, the system itself effectively issues the necessary DDEF (see Table 24).

All VAM data sets are *automatically cataloged* when they are created. SAM data sets must be cataloged by the user, using the CATALOG command (or macro). When the user issues a CATALOG command naming a data set for which a DDEF has been given in the current session, label, organization, unit, and volume information from the TDT entry are recorded in the system catalog. A subsequent DDEF command which names that cataloged data set is significantly simpler, since the system will automatically extract that information from the catalog and place it in the TDT entry being built; i.e., for cataloged data sets, the user does not respecify in the DDEF the information that is already in the catalog.

When the data set is written, some of its attributes (e.g., record length (LRECL), record type (RECFM), etc.) are recorded in the data set control block (DSCB) on direct access devices or in the tape label. The system will also extract this information from the DSCB or label when an existing data set is being processed; hence, the user does not respecify that information in a DDEF command or within a DCB within his program (except for ASCII tapes).

There are three points to consider in providing DDEF commands:

1. For a new data set (and for *all* ASCII tapes), the user must supply all characteristics and attributes, or specify only those required and rely upon system-assigned defaults for the remainder.
2. For an existing data set, the user should specify only that information not recorded in the DSCB or label.
3. Further, if the existing data set is cataloged, the user should not specify information already in the catalog.

The following general guidelines and Tables 21 and 22 provide information for the proper use of the DDEF command (or macro instruction).

1. If a DCB is opened (OPEN macro instruction) and no DDEF with the same ddname has previously been issued and:

```
                                            ( PS )
DDEF      DDNAME= { name    } [,DSORG= { VI
                  { PCSOUT  }           { VS  } ]
                                        { VP
                                        ( RX )

          ,DSNAME= { name
                   { *name
                   { 'bname'

          [,DCB=(([data definition name] [,DSORG=code]      [,RECFM=code]
                                          [,LRECL=integer]    [,KEYLEN=integer]
                                          [,RKP=integer]     [,PAD=integer]
                                          [,MACRF=code]       [,DEVD=code]
                                          [,DEN=integer]      [,TRTCH=code]
                                          [,OPTCD={W|A}]   [,BLKSIZE=integer]
                                          [,IMSK=code]      [,NCP=integer]
                                          [,BUFNO=integer]     [,BUFL=integer]
                                          [,BFTEK=code]     [,PRTSP=code]
                                          [,STACK=code]      [,MODE=code]
                                          [,EROPT=code]      [,BUFOFF=integer])]

                    ( DA [,datype-integer]                    )
          [,UNIT= ( { TA [,tapetype- { 7                 ] } )]
                    {                 { 7DC }
                    {                 { 9
                    ( device                                  )

          [,SPACE= ( { TRK                 } ,primary-integer [,secondary-integer] [,HOLD])]
                     { CYL
                     { rcdlength-integer   }

          [,VOLUME= ( [ { PUBLIC   } ] [, { PRIVATE        } ] ) ]
                        { PRIVATE  }      { volserno,...   }
                        { volseqno }

                                                    ( NL  )
                                                    { SL
          [,LABEL= ([fileseqno-integer] [,labeltype- { AL  } ] [,RETPD=integer])]
                                                    { SUL
                                                    ( AUL )

          [,DISP= { OLD } ]
                  { NEW }
                  { MOD }

          [,OPTION= { CONC   } ]
                    { JOBLIB }
          [,RET=retention code]
          [,PROTECT={Y}]
                    {N}
```

Figure 41. The DDEF Command

a. the session is conversational, the user will be prompted for *all* parameters specifiable in the DDEF command.

b. the session is nonconversational, the session will be terminated.

2. More than one DDEF cannot be issued for the same ddname *unless* it specifies concatenation (OPTION=CONC) of existing physical sequential (PS) data sets.

3. A data set cannot be referred to by more than one ddname at a time. If a second DDEF specifies a different ddname for a data set, the ddname given in the earlier DDEF will simply be replaced. This is true even if the DSNAME parameter specifies different member names of the same partitioned data set.

4. With the exception of IOREQ, a data set created with one access method cannot be processed with a different access method. VS data sets cannot be read with VI techniques, VI cannot be read with VS, etc.

5. The SPACE parameters should be specified for all new data sets on direct access devices unless the installation-assigned defaults are satisfactory. The HOLD sub-parameter should not be specified unless the data set is to be extended or modified. The DSCB maintains secondary space allocation specification from the DDEF issued when the data set was created, thereby obviating the need to specify this parameter when extending an existing direct access data set.

6. When creating private volume data sets, the system will assign the volume to be used if the volume specification is VOLUME=(PRIVATE).
7. If no labels (LABEL=(,NL)) are specified for a tape data set, the data set must *not* contain a header label recognizable by TSS.
8. For a cataloged data set, if SPACE, UNIT, LABEL, or VOLUME operands are entered, diagnostics will be displayed as appropriate. However, the associated fields will be taken correctly from the existing catalog entry.
9. When a module has been loaded from the wrong library, the incorrect version of the module must be unloaded, a DDEF or RELEASE issued for the job library affected, and a LOAD or CALL issued for the correct version.
10. When defining an ASCII tape, the DDEF command (or macro instruction) must *always* be used to supply all necessary DCB information; there is no other source for this information. ASCII tapes cannot be cataloged.

Table 21. Form of DDEF for New Data Sets

OPERANDS	WHEN APPLICABLE						COMMENTS
	VAM				BSAM OR QASM		
	Public		Private		Private		
	USERLIB	JOBLIB	USERLIB	JOBLIB	DISK	TAPE	
DDNAME=data definition name	X	X	X	X	X	X	1-8 characters (1st alphabetic); cannot begin with SYS; must be same as DDNAME in macro instruction.
[,DSORG=data set organization]	VS VI VP	VP	VS VI VP	VP	PS	PS	
,DSNAME=symbol	X	X	X	X	X	X	Maximum of 35 characters (18 qualifiers).
[,DCB=(,...)]	•		•		•	•	
[,UNIT=((DA[{,3330 ,333B ,2311 ,2314}]			X	X	X	X	Must specify for new VAM data set on private volume and uncataloged physical sequential data set. If nonconversational, obtain devices with SECURE command.
			X	X	X	X	
TA[,tape type]					X	X	
device))]					X		
[,SPACE=(([{TRK					X		
CYL					X		
record length}]					X		
,primary	X	X	X	X	X		
[,secondary]	X	X	X	X	X		
[,HOLD]))]	X	X	X	X	X		
[,VOLUME=(([{PUBLIC	X	X					In nonconversational, use SECURE command to obtain private volumes.
PRIVATE			X	X	X	X	
volseqno}]							
[{PRIVATE			X	X	X	X	
volserno,...}]))]			X	X	••	••	
[,LABEL=([fileseqno-integer]					X	X	
[label-type {NL					X	X	Must be specified for ASCII tapes.
SL					X	X	
AL						X	
SUL					X	X	
AUL}]						X	
[,RETPD=integer])]	X	X	X	X	X	X	
[,DISP=NEW]	X	X	X	X	X	X	Data sets are defaulted to existence: if cataloged, OLD is assumed; if uncataloged, NEW is assumed.
[,OPTION=JOBLIB]		X		X			
[,RET=codes]	X	X	X	X			
[,PROTECT={Y						X	Defaulted to N
N}]						X	

*See Table 23
**Must be specified here

Table 22. Forms of DDEF for Existing Data Sets

OPERANDS	WHEN APPLICABLE: CATALOGED** VAM USERLIB	CATALOGED** VAM JOBLIB	BSAM OR QSAM CATALOGED DISK	BSAM OR QSAM CATALOGED TAPE	BSAM OR QSAM UNCATALOGED DISK	BSAM OR QSAM UNCATALOGED TAPE	COMMENTS
DDNAME=data definition name	X	X	X	X	X	X	1-8 characters (1st alphabetic); cannot begin with SYS; must be same as DDNAME in macro instruction.
[,DSORG=data set organization]					PS	PS	
,DSNAME= { symbol	X	X	X	X	X	X	Maximum of 35 characters (18 qualifiers).
*symbol }					X	X	For data sets created under OS or OS/VS; maximum of 44 characters.
[,DCB=(,...)]	*	*	*	*	*	*	
[,UNIT=({ DA [{,3330 ,333B ,2311 ,2314}]					X		Must specify for uncataloged physical sequential data set. If nonconversational, obtain devices with SECURE command.
					X		
TA[,tape type]						X	
device })]						X	
[,SPACE=([{TRK CYL record length}] ,primary [,secondary] [,HOLD])]	X	X	X		X		Never specified for DISP=OLD data sets.
[,VOLUME=([{PUBLIC PRIVATE volseqno}])[			X	X			If nonconversational, use SECURE command to obtain private volumes.
[{PRIVATE volserno,...}]]					X	X	
[,LABEL=([fileseqno-integer]						X	Normally defaulted; file sequence number is in catalog. However, must be specified for ASCII tapes.
[,labeltype {NL						X	
SL					X	X	
AL						X	
SUL					X	X	
AUL}]						X	
[,RETPD=integer])]							
[,DISP={ OLD	X	X	X	X	X	X	
MOD }]			X	X	X	X	MOD applies only to BSAM; positions after last record of data set.
[,OPTION= { JOBLIB		X					
CONC }]			X	X	X	X	Only OLD BSAM data sets can be concatenated.
[,RET=codes]	X	X					
[,PROTECT={ Y				X		X	Default when DISP= MOD is N
N }]				X		X	

*See Table 23

**Existing uncataloged VAM volumes must be cataloged with an EVV (Enter VAM Volumes) command before they can be accessed.

Forms of the DDEF Command

Tables 21 and 22 specify all allowable forms of the DDEF command (excluding DCB parameters) for new and existing data sets, respectively, where the data set organization is VS, VI, VP, or PS. Standard TSS metalanguage notation (as described in the Command System User's Guide) is used. The first column specifies the general structure of each parameter, including whether parentheses are to be entered or not and the exact entry for providing default (or specification omission) entries. IBM-assigned defaults are underlined and may be selected by omission of specification in the DDEF command.

DCB Parameter Specification

It is not required that DCB information be specified in the DDEF command (except for ASCII tapes, where it is *always* required). It is, however, sometimes desirable to code a general-purpose program for which the attributes of the data set(s) to be processed are not known or not specified at assembly time in DCB macro instructions. In this case, these parameters may be supplied dynamically (i.e., without program reassembly) from the DCB information maintained in DSCBs, data set labels, and DCB subparameters specified in the DDEF command.

If a field has been specified in the DCB at assembly time or by the user's program prior to OPEN, it will not be modified by the system if a later specification for the same field is given; e.g., if there were a LRECL (logical record length) specification in the DDEF command as a DCB subparameter and the DCB also contained an LRECL specification at assembly time, the DDEF specification would be ignored. Any field supplied dynamically by the system is reset when the DCB is closed. This permits successive dynamic DCB parameter specification between successive OPEN-CLOSE executions.

Information is used or filled into a user's DCB at OPEN time in the following order:

Table 23. Use of DCB Parameters in the DDEF Command

DCB OPERAND	SPECIFIES	APPLICABLE DSORG					IF DISP=OLD WILL BE FILLED FROM	
		VS	VI	VP	PS	RX	CATALOG	DSCB OR LABEL
DSORG	data set organization	x	x	x	x	x	x	VIP/VSP
RECFM	record format	x	x	x	x			x
LRECL	logical record length	x	x	x	x			x
KEYLEN	key length		x	VIP	x			x
RKP	key position		x	VIP				
PAD	space to be left on each page of VI(VIP) data set for subsequent insertions		x	VIP				x
MACRF	type of macro instructions used				x		x	
DEVD	device type				x		x	
DEN	tape density				x		x	
TRTCH	data conv, parity, translation				x		x	
OPTCD	write check or ASCII tape				x		x	x
BLKSIZE	maximum block length				x		x	x
IMSK	error recovery procedures				x	x	x	
NCP	number of consecutive READ, WRITE or IOREQ macro instructions issued before CHECK				x	x	x	
BUFNO	no. of buffers				x		x	
BUFL	buffer length				x		x	
BFTEK	buffer technique						x	
PRTSP	print spacing						x	
STACK	stacker selection						x	
MODE	mode of operation						x	
EROPT	error option						x	
BUFOFF	buffer offset				x			

1. DCB macro instruction assembly time specification
2. User's program prior to OPEN (can modify above)
3. TDT entry containing DDEF and catalog information
4. DSCB or data set label (existing data sets only)

The DCB parameters applicable to each data set organization and which are recorded in the system catalog or in a DSCB or data set label are shown in Table 23. Refer to the publication *Assembler User Macro Instructions* for the actual parameter specifications.

Data Set Definition Rules for Language Processing

No DDEF command is required to define the source and listing data sets, or the object modules, used in language processing. The DDEF command is required when the job library that is to receive the object module is not the library at the top of the program library list; that job library must be defined. Each library referred to by INCLUDE statements (except USERLIB), and each job library used by automatic call, must also be defined by a DDEF command.

Data Set Definition Rules for TSS Commands

Table 24 provides information relating to the structure of and DDEF requirements for data sets processed by TSS commands.

Secure Requirements for Nonconversational Tasks

Nonconversational tasks are enqueued until the system is able to fill the requirements for private devices. The list of requirements is made available to the system by means of a SECURE command, which the user must include in the task's command procedure as the first command after LOGON. Then, as each DDEF command is read and processed, the required devices are allocated from those that have been reserved for the nonconversational task. Any attempt to allocate more than have been secured causes the task to be terminated.

In determining the number of devices needed in a task, the following points should be considered:

- The number of devices should be at least equivalent to the number of data sets on different private volumes that are opened at any one time. Two or more data sets residing on the same private volume may require only one device (the exception is described below).
- If two different data sets are referred to in sequence (i.e., the first is closed before the second is opened) the system can be directed to allocate the same device to both by including the UNIT=AFF option in the second DDEF along with the ddname of the first DDEF. When the UNIT=AFF option is selected, the device types of both data sets must be compatible and neither should be a new data set residing on a direct access device. If several data sets are to be serially processed with unit affinity specified, each data set may have unit affinity with only the most recently processed data set. Note that unit affinity may only be specified for physical sequential data sets.
- If two different data sets on private volumes are referred to by the same ddname, the UNIT=AFF option cannot be selected. Since the first DDEF must be released prior to the second DDEF, two devices

Table 24. Data Set Definition Requirements for Commands

COMMAND	RELATED DATA SETS	DSORG	DATA SET DEFINITION
BACK	New SYSIN data set that is to control completion of this task in nonconversational mode.	VS, VI	New SYSIN data set must be cataloged, or defined by previous DDEF command in conversational portion of this task.
BUILTIN	USERLIB or a virtual partitioned data set specified by DSNAME will contain a member named SYSPRO.	VP	Virtual partitioned data set does not have to be defined or cataloged when BUILTIN command is issued.
	Virtual partitioned data set must contain an object module with entry point named by EXTNAME operand.	VP	Virtual partitioned data set must be in user's program library hierarchy when command defined by BUILTIN is issued. (DDEF with OPTION=JOBLIB)
CATALOG	Data set to be cataloged.	PS	Data set to be cataloged must be defined by previous DDEF command in this task, unless UPDATE option specified.
CDD	Data set containing only DDEF commands.	VI	Data set must be cataloged, or defined in current task.

Table 24. Data Set Definition Requirements for Commands *(Continued)*

COMMAND	RELATED DATA SETS	DSORG	DATA SET DEFINITION
CDS	Data set to be copied.	Any except RX (for user-controlled physical I/O with private devices)	Data set to be copied must be cataloged or defined by previous DDEF command in this task.
	Copy data set.	Any except RX (for user-controlled physical I/O with private devices)	VAM—does not have to be defined or cataloged. PS—should be cataloged or defined to insure proper volume and unit.
CLOSE	Data set to be closed.	any	Data set to be closed must be defined by previous DDEF command in this task.
DATA[1]	Data set to be entered.	VS, VI	No DDEF command is required if the data set is to reside on public storage; data follows this command in input stream. If the data set is to reside on private storage a DDEF must be issued before the command.
DEFAULT	User profile data set in USERLIB.	VP	Data set must be defined in current task.
DELETE	Data set whose name is to be removed from catalog.	any	No DDEF command required for this command.
DSS?	Data sets whose status is desired.	any	Each data set whose status is to be presented must be cataloged; no DDEF command required for this command.
DUMP	Data set to be printed as a result of program control command DUMP.	VI	DDEF command whose ddname is PCSOUT must be defined prior to execution of DUMP command.
EDIT[2]	Data set to be processed by the Text Editor.	VI	Data set must be cataloged, or defined in current task. This is done automatically.
END[2]	Data set being processed by the Text Editor, or indicates PROCDEF command completion.	VI	No DDEF command required for this command.
ERASE	Data set to be erased.	VS, VI, VP	Data set to be erased must be cataloged.
EVV	Private data sets whose names are to be entered in catalog.	VS, VI, VP	No DDEF command required for this command.
EXECUTE	SYSIN data set for nonconversational task set up by this command.	VS, VI	Data set must be cataloged; no DDEF command required by this command.
KEYWORD	SYSPRO data set in USERLIB, SYSPRO data set in SYSLIB.	VIP	Data set must be defined in current task.
LINE?	Line data set containing lines to be presented.	VI	Line data set must be cataloged or defined by previous DDEF command in this task.

[1] If the DATA command was used to create the data set within the current task, then the data set is defined as if a DDEF command had been issued by the user directly. If the data set is also VAM organized and resides in public storage, it is automatically cataloged.

[2] These are the basic directive commands of the Text Editor. See *Command System User's Guide* for details concerning the data manipulation commands of this facility.

Table 24. Data Set Definition Requirements for Commands *(Continued)*

COMMAND	RELATED DATA SETS	DSORG	DATA SET DEFINITION
LOAD FTN ASM LNK PLI User-written problem program	Object module to be loaded.	VP	Object module to be loaded is identified by external name specified in this command; it must be in a library in the current program library list.
MCAST	User profile data set in USERLIB, session profile in task virtual memory.	VP, VSP	Data set must be defined in current task.
MCASTAB	User profile data set in USERLIB, session profile in task virtual memory.	VP, VSP	Data set must be defined in current task.
MODIFY	Data set to be changed.	VI	Data set must be cataloged or defined by previous DDEF command in this task.
PC?	Data set whose status is required.	any	Each data set whose status is to be presented must be cataloged; no DDEF command required for this command.
PERMIT	Data sets for which sharing is permitted.	any	Data sets for which sharing is permitted must be cataloged; no DDEF command required for this command.
POD?	Virtual partitioned data set for which information about its members is given.	VP	Virtual partitioned data set must be cataloged, or defined by previous DDEF command in this task.
PRINT	Data set to be printed.	PS, VS, VI	Data set must be cataloged or defined by previous DDEF command in this task. A previous DDEF required for unlabeled tapes.
PROCDEF	USERLIB or virtual partitioned data set named by DSNAME will have a SYSPRO member created to contain procedure definition.	VP	Virtual partitioned data set does not have to be defined or cataloged.
PROFILE	User profile data set in USERLIB, session profile in task virtual memory.	VP	Data sets must be defined in current task.
PUNCH	Data set to be punched on cards.	VS, VI	Data set must be cataloged or be defined by previous DDEF command in this task.
REGION[1]	Data set to be processed by the Text Editor.	VI	Data set must be cataloged, or defined in current task.
RELEASE	Data set to be released.	any	Data set to be released must be defined in previous DDEF command in this task.
RET	VAM data set whose data set descriptor is to be changed.	VS, VI, VP	Data set must be cataloged.
SHARE	Data sets for which sharing is requested.	any	Data sets for which sharing is requested must be cataloged; no DDEF command required by this command.
SYNONYM	User profile data set in USERLIB, session profile in task virtual storage.	VP	Data sets must be defined in current task.
TV	Physical sequential data set (from a VT operation) to be written on a VAM volume.	PS	Data set (input) must be cataloged or defined in current task.
VT	VAM data set to be copied to magnetic tape as a physical sequential data set.	VS, VI, VP	Data set (input) must be cataloged or defined in current task.
VV	VAM data set to be copied into direct access storage.	VS, VI, VP	Data set (input) must be cataloged or defined in current task.
WT	Data set to be recorded on magnetic tape in print format.	VS, VI	Data set must be cataloged or defined by previous DDEF command in this task.

[1] These are the basic directive commands of the Text Editor. See *Command System User's Guide* for details concerning the data manipulation commands of this facility.

must be secured for the data sets even though both data sets are not open at the same time. Since the ddname must be unique in each DDEF, the first data set must be released prior to the second DDEF. Therefore, two devices are necessary since the RELEASE command removes the device from the task's allocation prior to the second DDEF command.

Data Definition Considerations for Multiple Executions in the Same Session

A DDEF command provides the linkage between the ddname used in the source program DCB and the actual physical data set. Once a DDEF has been entered, it remains in effect until log-off time, unless the definition is released or redefined.

If two programs are executed in succession, the following conditions could arise:

1. Both programs refer to the same data set with the same ddname. One DDEF command issued prior to the execution of the first program is sufficient for both executions if the data set is read in both programs or written in the first and read in the second. If, however, the data set is written in both programs, the data is not automatically concatenated. Data written in the first execution would be written over in the second execution. If the user does not wish this to occur, he must take the steps outlined in 3.
2. Both programs refer to the same data set with different ddnames. Each execution must be preceded by a DDEF command giving the ddname as appropriate for the execution. Since the second DDEF will contain the same DSNAME as the first, effectively redefining it, the first definition need not be released.
3. Each program refers to a different data set with the same ddname. Each execution must be preceded by a DDEF command giving the DSNAME for the ddname. In addition, since the second DDEF has the same ddname, the first definition must be released prior to the second DDEF. When a data set on a private volume is released, the input/output device is also released unless another defined data set resides on that same volume. In a nonconversational task, if a device is freed by a RELEASE command, the user must account for this when specifying the SECURE command. For example, if two programs read different data sets on separate private volumes and both are referred to by the same ddname, the following procedure is necessary:

a.	SECURE	Two devices—even though only one device is needed at any one time
b.	DDEF	For first data set
c.	CALL	First execution
d.	RELEASE	First data set
e.	DDEF	For second data set
f.	CALL	Second execution

Appendix F. User Defined Procedures

This appendix will depict representative uses of the Procedure Definition (PROCDEF), BUILTIN, and the User Profile. These facilities all enable the user to tailor his task to special situations, while still retaining the generalized scope provided by the system-supplied commands. *Command System User's Guide* is the primary source for explanation of these user-created procedures.

Procedure Definition (PROCDEF)

The PROCDEF command defines a command procedure which consists of other commands. When issuing PROCDEF, the user must specify the name to be assigned to the user-written command procedure. The system then prompts the user to enter his first line by issuing the line number 100. If the user wants to build his command procedure so that he can substitute values for the operands in the created procedure, the PARAM line should be incorporated. Without the PARAM line, the procedure remains fixed, as defined, with no adjustment of operand values possible at execution time. These dummy operands that comprise the PARAM line may be both keyword and positionally specified (see Example 1 in this appendix).

Entering Procedure Text

After PROCDEF is issued (optionally using the PARAM line) all subsequent lines issued without a single preceding break character (_) will be included in the procedure text. The system prompts for each line with a line number, and there is no apparent limit on the number of lines the user may enter.

The user may enter system-supplied commands (including PROCDEF and/or BUILTIN commands) or other user-created commands. The commands entered need not include all of the operands normally associated with them, but only those necessary for the successful performance of the functions requested. These operands may be indicated as variable (dummy names within a PARAM line) or may be fixed with explicit values. Fixed operand values are not included in the PARAM line, and therefore will be executed *exactly* as given in the text when the procedure is called.

A direct call to a language processor-produced object module may be produced by entering the name of the module in the procedure text.

Commands preceded by a break character (e.g., _END) are executed immediately and do not become part of the procedure text.

NOTE: To insert a command requiring a break character for execution (e.g., LIST in the text editor context) during PROCDEF generation, use two break characters to insure that one will appear with the command in the PROCDEF.

Terminating Procedure Definition

The user terminates PROCDEF processing by entering a break character followed by any *one* of these items: an END command, an EDIT command, another PROCDEF command. When the user enters another PROCDEF command, the same options for terminating its processing are available. Eventually, the last PROCDEF desired will have to be terminated with either an END or an EDIT command.

Nested Procedure Definitions

The text of a procedure, defined by PROCDEF, may contain other PROCDEF commands, entered just as any other system-supplied command, without a preceding break character. These additional PROCDEF commands are said to be nested in relation to the complete procedure.

Example 1:

```
PROCDEF NAME = MYJOB
100 PARAM DDNAME=ALPHA,DSNAME=
    DATASET,VOLUME=ANY,$N,STATE=$1,
    ACC=$2,NEWNAME=BETA
200 DDEF DDNAME=ALPHA,DSORG=VI,
    DSNAME=DATASET,VOLUME=ANY,$N
300 CATALOG DSNAME=DATASET,STATE=
    $1,ACC=$2,NEWNAME=BETA
400 _END
```

In the PARAM line in Example 1 above DDNAME, DSNAME, VOLUME, STATE, ACC, and NEWNAME are external strings (keywords) that associate the calling parameters with the internal strings (in the PARAM line) ALPHA, DATASET, ANY, $1, $2, and BETA respectively.

DDEF, on line 200, is a system-supplied command with the variable operands DDNAME, DSORG, DSNAME, VOLUME, and DISP. The keyword DISP is omitted and the dummy operand $N is supplied positionally. DSORG=VI is a fixed operand value and will be so treated when the procedure named MYJOB is called. Values for the other variable operands will be supplied when the procedure is called.

CATALOG, on line 300, is also a system-supplied command. Its operands are all variable and will be substituted when the procedure is called.

The _END, line 400, terminates the definition of this procedure. It can now be executed by the user.

Example 2: MYJOB DDNAME=SETUP,DSNAME=ONE, VOLUME=131313, OLD, STATE=U, ACC=U, NEWNAME=TWO

This parameter string associated with MYJOB will cause the dataset named ONE to be defined and retrieved as an existing (DISP=OLD) data set on private volume #131313. The catalog entry for ONE will then be updated, renaming the data set as TWO. The access qualifier of U (unlimited access) is retained.

Example 3: MYJOB SETX,FIRST,PRIVATE,NEW,N,R, SECOND

This parameter string associated with MYJOB will cause the data set named FIRST to be defined for a private volume. The data set does not yet exist (DISP=NEW). The data set will be cataloged with read-only access, under the data set name SECOND.

Object Program Definition (BUILTIN)

The BUILTIN command defines an object program that the user can invoke as if it were a command. It is useful for accomplishing actions not achieved by any current system-supplied commands. If the user wishes to define operands for his BUILTIN command, he must supply the code within his module to handle the parameter values supplied when the module is called. The BPKD macro instruction (BUILTIN Procedure Key Identifier) must be supplied in the object code as part of the PSECT and have the expected parameters defined. The BPKD macro instruction must also supply the names needed to provide linkage between the module and the BUILTIN command defining that module. The following source program could be assembled, with the object module being retained for future use. The program is only a randomly selected example to indicate the sequence of events necessary for BUILTIN, and the control features (BPKD) necessary for incorporation. Any other sequence of executable code would suffice equally as well.

```
PST16    PSECT
         ENTRY  BEGIN16
         DC     F'76'
         DC     18F'0'
USEREX   BPKD   BEGIN16
CST16    CSECT
BEGIN16  BASR   11,0
         USING  *,11                   LOCAL BASE
                                       REGISTER
         L      13,72(0,13)
         USING  PST16,13
HERE     EQU    *
         GATRD  AREA+3,LENGTH          READ FROM
                                       SYSIN
         CLI    AREA+3,C'E'            CHECK
                                       IF END
         BE     LEAVE                  BRANCH
                                       IF YES
         MVZ    AREA+3(1),=X'00'       CONVERT
                                       TO BINARY
         L      5,AREA
         SLA    5,1                    MULTIPLY
                                       BY 2
         ST     5,AREA
         MVZ    AREA+3(1),=X'FF'       CONVERT
                                       TO EBCDIC
         GATWR  AREA+3,LENGTH          WRITE ON
                                       SYSOUT
         B      HERE
LEAVE    EXIT   'PGM FINISHED'
AREA     DC     F'0'                   READ/WRITE
                                       AREA
LENGTH   DC     F'1'                   LENGTH
                                       AREA
         END
```

Assuming that the above module was assembled without specifying a job library, the task USERLIB will contain the object code available via entry point BEGIN16. Example 4 to follow shows how this code sequence may be retrieved.

Example 4: BUILTIN NAME=GETPROG,EXTNAME= USEREX

The object program definition via a user-created command (GETPROG) is now established.

GETPROG

The execution of this user-defined command will now result in the calling and running of the program shown earlier. Control will be transferred to the entry point named BEGIN16, with linkage established via parameters in the BUILTIN command and the BPKD macro specification.

The User Profile

The user profile is a specialized data set containing information pertinent to each user. Stored within this data set is information regarding the values the user generates for defaults and synonyms, and optionally, his command symbols. The user profile is a member of the partitioned data set named USERLIB.

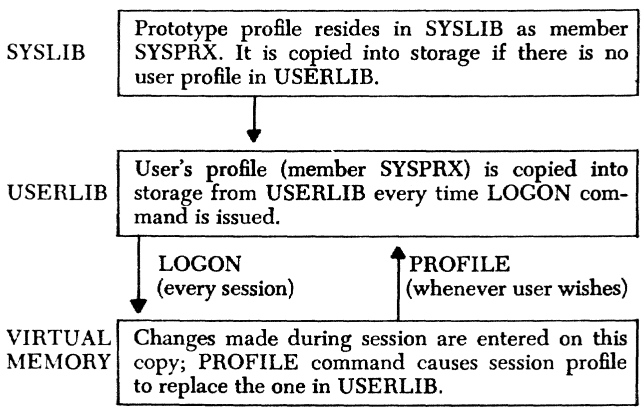
Initially, the system provides the user with a prototype user profile, resident in SYSLIB, which contains the default values for system-supplied commands and any initial synonym values.

When a user is joined to the system, a copy of the prototype profile in SYSLIB is made a member of his user library. He can make changes or add to the prototype copy in memory during a terminal session by issuing a SYNONYM or DEFAULT command, or by using the SET command to establish command symbols. Such changes affect only the session profile, unless followed by the PROFILE command, which permanently changes the user profile.

When the prototype profile is not permanently changed during a session, the memory copy is erased when LOGOFF is issued. When, during the course of a session, the user issues a PROFILE command, the entire profile copy, as it exists in memory, is written into USERLIB, and given the member name of SYSPRX.

When the user initiates his task, the system generates a search through USERLIB to locate the user's profile. If it is not found (i.e., the user has erased his user profile), the system copies the prototype profile from SYSLIB into memory, where it may be accessed and used. Unless changed via PROFILE, this memory copy of the prototype profile is erased at LOGOFF.

The user profile can exist concurrently on three levels: the prototype profile in SYSLIB, the user profile in USERLIB, and the session profile in storage.



At the user's first LOGON, the system provides initial default values for most operands. When the user does not explicitly define operand values when entering a command requiring these values, the system will default to the initial value that it has provided. If the initial value is null, the user must specify a value. The default table is a list of default values supplied by the system (see *Command System User's Guide*).

A user can specify his own default values to be used in place of or in addition to these system-supplied default values by using the DEFAULT command. Any changes become a part of his user profile for the session involved and may, of course, be saved for later sessions by issuing a PROFILE command.

Each user has a separate user library and therefore a separate user profile. At times the user may find it desirable to share the copy of the profile in his user library. Since his copy is addressable as a normal member, it can be shared by making USERLIB shareable. Normal sharing precautions and procedures should be used.

The user may erase his copy of the user profile, exercising the normal erasing procedure. He may also log on without it for a particular session by using the pristine operand of the LOGON command.

Command System User's Guide should be referenced for complete details concerning User Profile Management.

Index

GC28-2032-6

IBM Time Sharing System Assembler Programmer's Guide Printed in U.S.A. GC28-2032-6

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
360 Hamilton Avenue, White Plains, New York 10601
(International)