SPERRY UNIVAC
COMPUTER SYSTEMS

PUBLICATIONS REVISION

Operating System/3 (OS/3)

System 80

Concepts & Facilities

UP-8870 Rev. 1

This Library Memo announces the release and availability of "SPERRY UNIVAC® Operating System/3 (OS/3) System 80 Concepts and Facilities", UP-8870 Rev. 1.

This revision presents material some of which is new and some of which expands upon and clarifies topics covered previously. The new material includes:

- Menu processing
- Editor enhancements
  - COBOL editor
  - Screen mode processing
  - Error file processor
- New workstation features
  - Auxiliary printer
  - Remote workstation support
  - Workstation programming aids
- ICAM enhancements
- DDP enhancements
- Printerless System 80s
- New application programs
- IBM System/32 and System 34 conversion aids
- A new table of peripheral device characteristics

Topics covered in previous releases and clarified in this revision include the concepts and facilities behind consolidated data management, job control; and the differences among screen formats, dialogs, and menus (new to this release).

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ and MZ	Mailing Lists B00, B01, 28U, and 29U (Cover and 191 pages)	Library Memo for UP-8870 Rev. 1
		RELEASE DATE: September, 1982

Destruction Notice: If you are going to OS/3 release 8.0, use this revision and destroy all previous copies. If you are not going to OS/3 release 8.0, retain the copy you are now using and store this revision for future use.

Copies of UP-8870, UP-8870-A and UP-8870-B will be available for 6 months after the release of 8.0. Should you need additional copies of this edition, you should order them within 90 days of the release of 8.0. When ordering the previous edition of a manual, be sure to identify the exact revision and update packages desired and indicate that they are needed to support an earlier release.

Additional copies may be ordered by your local Sperry Univac representative.

System 80

OS/3

Concepts and Facilities

SPERRY UNIVAC

H

UP-8870 Rev. 1

This document was prepared by Systems Publications using the SPERRY UNIVAC UTS 400 Text Editor. It was printed and distributed by the Customer Information Distribution Center (CIDC), 555 Henderson Rd., King of Prussia, Pa., 19406.

PAGE STATUS SUMMARY

ISSUE: UP-8870 Rev. 1
RELEASE LEVEL: 8.0 Forward

Part/Section	Page Number	Update Level
Cover/Disclaimer		
PSS	1	
Preface	1, 2	
Contents	1 thru 8	
PART 1	Title Page	
1	1 thru 7	
PART 2	Title Page	
2	1 thru 5	
3	1 thru 5	
4	1 thru 37	
PART 3	Title Page	
5	1 thru 23	
6	1 thru 7	
7	1 thru 14	
PART 4	Title Page	
8	1, 2	
9	1 thru 5	
10	1 thru 8	
11	1 thru 5	
PART 5	Title Page	
12	1 thru 8	
PART 6	Title Page	
13	1 thru 15	
14	1 thru 3	
PART 7	Title Page	
15	1 thru 6	
PART 8	Title Page	
Appendix A	1 thru 3	
Appendix B	1 thru 6	
Index	1 thru 11	
User Comment Sheet		

*New pages

All the technical changes are denoted by an arrow (→) in the margin. A downward pointing arrow (↓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (→) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

Preface

This manual is one of a series designed to instruct and guide the programmer in the use of the SPERRY UNIVAC Operating System/3 (OS/3). It offers a presentation on the overall operation and use of the System 80 data processing system. Included are descriptions of all available software productions and explanations of such procedures as program preparation and file cataloging.

This manual is intended for the experienced programmer who is unfamiliar with System 80 or Operating System/3 (OS/3).

This concepts and facilities manual is divided into the following parts:

- PART 1. INTRODUCTION

  Presents an overall view of the system describing the types of users and the processing environment.

- PART 2. SYSTEM OPERATION FACILITIES

  Details the use and operation of the System 80 Workstation, describes the purpose and function of the supervisor, and explains the operation of the data management routines and discusses programmer considerations for using various types of data files.

- PART 3. PROGRAMMING FACILITIES

  Describes the various programming facilities available including the programming languages: BASIC, COBOL, RPG II, FORTRAN IV, ESCORT, and the basic assembly language; the job control language, including the job control dialog; and the interactive program development. It also describes how the user can develop his own interactive software.

- PART 4. SYSTEM UTILITIES

  Discusses the utilities used to install the system and support the day-to-day operation. Explanations of file cataloging, sorting, spooling, and diagnostic procedures are included.

- PART 5. APPLICATIONS PROGRAMS

  Describes the various user applications available to meet specific user data processing requirements.

- PART 6. COMMUNICATIONS AND DATA BASE FACILITIES

  Introduces the integrated communications access method, the information management system, and the data base management system and highlights their salient features.

- PART 7. CONVERSION

  Describes the conversion aids available to users migrating from other data processing systems. Systems for which conversion aids are provided are: SPERRY UNIVAC 9200/9300 and 9400/9480 (OS/4); IBM System 3 and Systems 32 and 34; Honeywell Series 60, 100, and 200/2000.

Each of the aforementioned parts consists of one or more sections that cover the different aspects of the subject matter contained in each part. Because of the nature of this manual, the discussions are necessarily brief; however, detailed descriptions are available in other Sperry Univac publications. The appropriate document is referenced in each section. In addition, you should be familiar with the OS/3 system index and publications guide, UP-8874.

Contents

FIGURES

TABLES

PART 1. INTRODUCTION

1. System Overview

1.1. GENERAL

The SPERRY UNIVAC System 80 is a multipurpose business computer system that provides a sophisticated, yet easy to use, data processing environment. Executing under the control of the Operating System/3 (OS/3) software and offering advanced programming facilities, it permits completely *interactive* access to the system and supports the development and implementation of interactive applications programs. Interactivity is a convenient method of accessing the system resources through a *workstation* to quickly perform work that might otherwise take hours or even days to complete. When using an interactive system such as System 80, you are, in effect, in direct communication with the system software.

The system provides a powerful set of interactive workstation commands enabling you to control its operation. Facilities are provided to assist programmers in the interactive development and execution of programs. In addition, facilities can be included that permit nonprogramming personnel to use system resources to perform a variety of operations. To extend the advantages of interactive operation to all phases of commercial data processing, the system also provides facilities for the development of applications programs that interface with workstations for online data entry and retrieval.

The interactive environment supports the configuration of a large number of workstations that can be placed where needed within the organization. These workstations can be used by programmers or by data entry and retrieval personnel. Individual programmers can use the workstation to perform all of their own processing. Workstations used as data entry and retrieval devices can be placed at the various departments or work locations within your organization, providing convenient access to the system.

Noninteractive applications programs can also be developed and processed concurrently with interactive operations. The system responds immediately to interactive requests and on an as-time-permits basis for the noninteractive applications where time is less critical. Even in a predominately interactive environment, a number of noninteractive programs must be executed to handle day-to-day operations. In fact, the very development of an applications program, interactive or noninteractive, necessitates the execution of a number of noninteractive programs during the normal course of program development. Even if no interactive applications programs are contemplated, the interactive program development and operational facilities can significantly improve programmer productivity and enhance system efficiency and usability.

The following subsections discuss the basic operating concepts of System 80 and describe how the system schedules and processes the various activities that can demand the use of system resources at any given time.

1.2. PROCESSING ENVIRONMENT

The productive capacity of any system is largely determined by how efficiently it uses its resources and, specifically, its central processing unit (CPU). Because all work performed by a system ultimately requires the CPU to execute one or more program instructions (a program being defined as an executable segment of code), productive capacity can be improved through more efficient CPU scheduling. System 80 achieves optimum CPU scheduling efficiency through *multiprogramming*.

If a system could only execute one program at a time, much valuable CPU time would be lost while the system performed operations not involving the CPU (such as an input/output operation). This lost CPU time could be recovered if the CPU could execute another program while the first program performed the I/O. This technique of switching control of the CPU between programs is the heart of multiprogramming. Because the performance of such activities as I/O operations is substantially slower than CPU operations, the multiprogramming technique can be applied to large numbers of programs. Figure 1-1 illustrates the switching of CPU control between two concurrently executing programs.

For System 80 to process multiple programs concurrently, it requires more information about the needs of a program than is supplied in the program itself. This includes the resources a program is to use, such as data files, peripheral devices, and main storage. For those programs that can be executed interactively - the general editor and so on - the system supplies the necessary control information. However, for executing your own programs or the noninteractive system programs, you must supply this information by using the OS/3 *job control language*.

Using the job control language, you prepare a series of job control statements to produce a *job control stream*. Together, the job control stream and the programs executed by it are called a *job*. When generating a job control stream, you must include information relevant to each program executed by the job and also information used to control the processing of the job itself.

The job control language statements are interpreted and acted upon by the various job control routines. These routines control the scheduling of your job and ensure the proper loading and execution of the programs included in the job. In addition, job control can be used to perform a variety of specific functions, such as file allocation and deallocation. Jobs can be run that perform only these job control functions and execute no other programs.

Each function performed by a job, a program execution, file allocation, etc, represents a single job step. Every job consists of one or more job steps, and the job steps are executed serially within the job. In other words, a job step is completely processed before the next job step is started. Transfer of CPU control between programs being concurrently executed occurs as a result of a request to perform an external event and also at job step completion or termination.

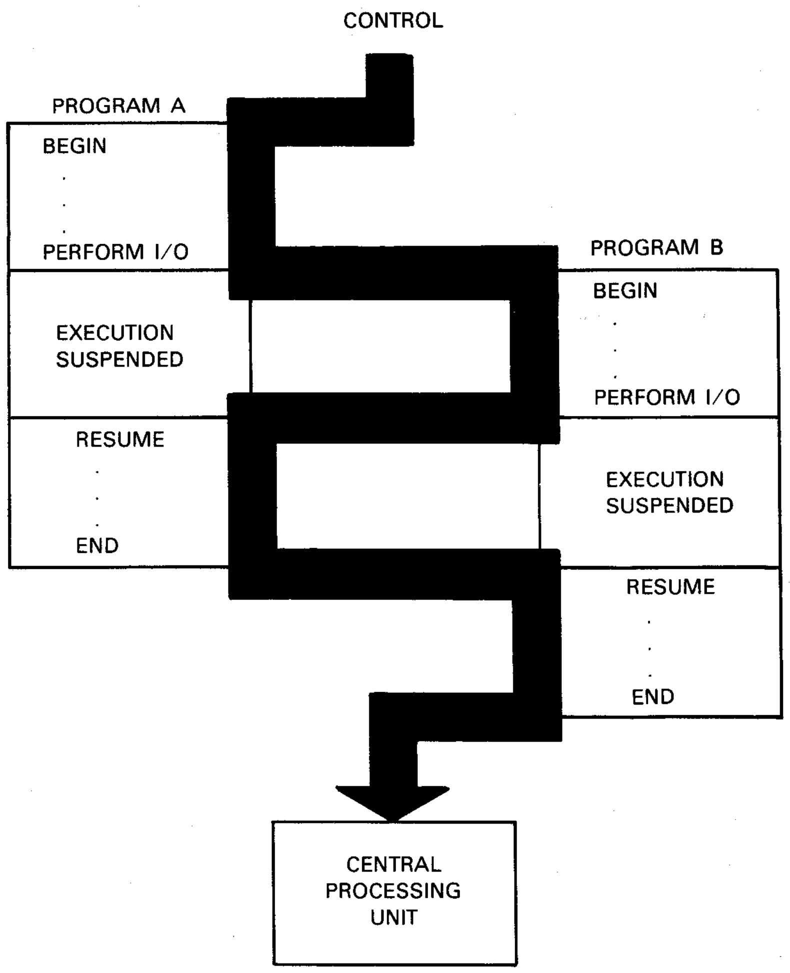


Figure 1-1. Switching CPU Control between Two Programs in a Multiprogramming Environment

Job control streams can be prepared interactively through an easy-to-use system program called job control dialog. This program displays a series of questions concerning the needs of your job on the workstation screen. Your responses to the questions are used to generate the required job control statements. The output produced by job control dialog is automatically placed in the system *job control stream library, $Y$JCS*. Once placed in this library, a job control stream can be run by issuing a workstation RUN command specifying the name assigned to the job.

Because of its multiprogramming capability, the system is able to process up to 14 jobs concurrently. The concurrent processing of jobs is referred to as *multijobbing*. The system maintains a control area for each job currently being processed. These areas are called *job slots*.

Figure 1-2 shows the path of a job from job entry through program execution. When a request to run a job is issued, the *run processor* interprets the request and locates the specified job control in $Y$JCS. The run processor places the name of the job on the *job queue*, generates a temporary run library ($Y$RUN) for the job, and places the job control stream into this run library.

Once a job is entered on the job queue, it is considered to be a scheduled job. A component called the *job scheduler* determines which job of those currently scheduled is to be the next job initiated. If a job slot is available, the job is usually processed immediately. The exception is when resources required by the job are currently unavailable. If the system is currently processing 14 jobs or if the resources for a job are not available, the job remains queued until a current job terminates or the resources become available.

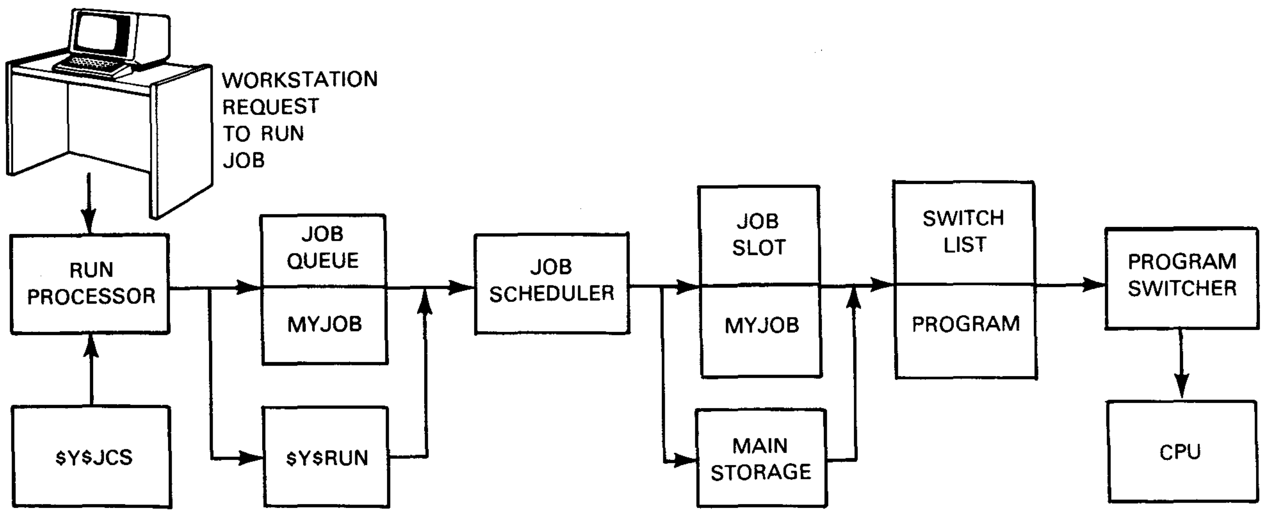


Figure 1—2. Job Path

If more than one job is currently on the job queue, the job scheduler selects the job that has the highest *scheduling priority.* You can specify one of three scheduling priorities in the job control stream or on the workstation command that entered the job: preemptive, high, and normal.

Preemptive jobs are initiated first; in fact, the system attempts to make room for preemptive jobs by temporarily suspending the execution of lower priority jobs and moving them from main storage to disk. If no preemptive jobs are currently on the job queue, the system initiates all high priority jobs and then any normal priority jobs.

If you do not specify a scheduling priority, the system automatically assigns normal priority to your job. Within each priority level, the first job entered is usually the first job initiated unless the resources for the job are not available. In such cases, the system selects the next job entered for which the resources are available.

When a job is selected for execution, the job scheduler removes the job name from the job queue and displays it on the system console. The job scheduler also allocates the main storage required to process the job. The amount of main storage required is equal to the size of the largest program executed by the job.

Under certain conditions, the system can determine how much main storage is required. In many cases, however, the main storage requirements must be specified in the job control stream. During the processing of a job consisting of more than one job step (that is, a job that executes more than one program), the first program is loaded in the allocated area and executed. The next program to be executed overlays the first and so on until all the job steps are processed.

The mechanism the system uses to coordinate the execution of programs is the *system switch list*. The switch list has an entry for every program currently being processed by the system. When a job is initiated, job control makes an entry on the switch list for that job. As the system processes a job by executing the programs, it deletes the entry made for a completed program and generates a new entry for the next program. Most jobs have only one entry on the system switch list at any point in time.

Control of the CPU is passed among the programs scheduled for execution based on the system switch list entries. The *program switcher* determines which program is to be executed based on each program's *execution priority*. The system can support up to 60 levels of program execution priority. The number of priority levels available is determined during system generation. Even though the system supports 60 levels of priority, 10 to 15 levels should be sufficient for most systems.

The execution priority of a program should not be confused with the scheduling priority of the job. The scheduling priority is used only to determine when a job should be processed in relation to the other jobs entered on the job queue. The program execution priority is used to determine when programs should be executed in relation to the other programs in the system after a job has been initiated.

When searching the switch list to determine which program to execute next, the program switcher selects the program with the highest execution priority. You can specify the execution priority of a program in the job control stream. When selecting a program for execution, the system does not have to check the availability of required resources because the job, and thus the program, would not have been selected for processing if the resources were unavailable.

If your job executes more than one program, you can assign a different priority level to each program. If no priority is specified for a particular program, the system automatically assigns the lowest program execution level configured into the system.

Interactive system programs are initiated directly through the workstation either by using the RUN/RV command or by entering the specific command that executes the program. Those executed through the RUN/RV command are processed as any job would be processed. They occupy job slots and are executed through the switch list.

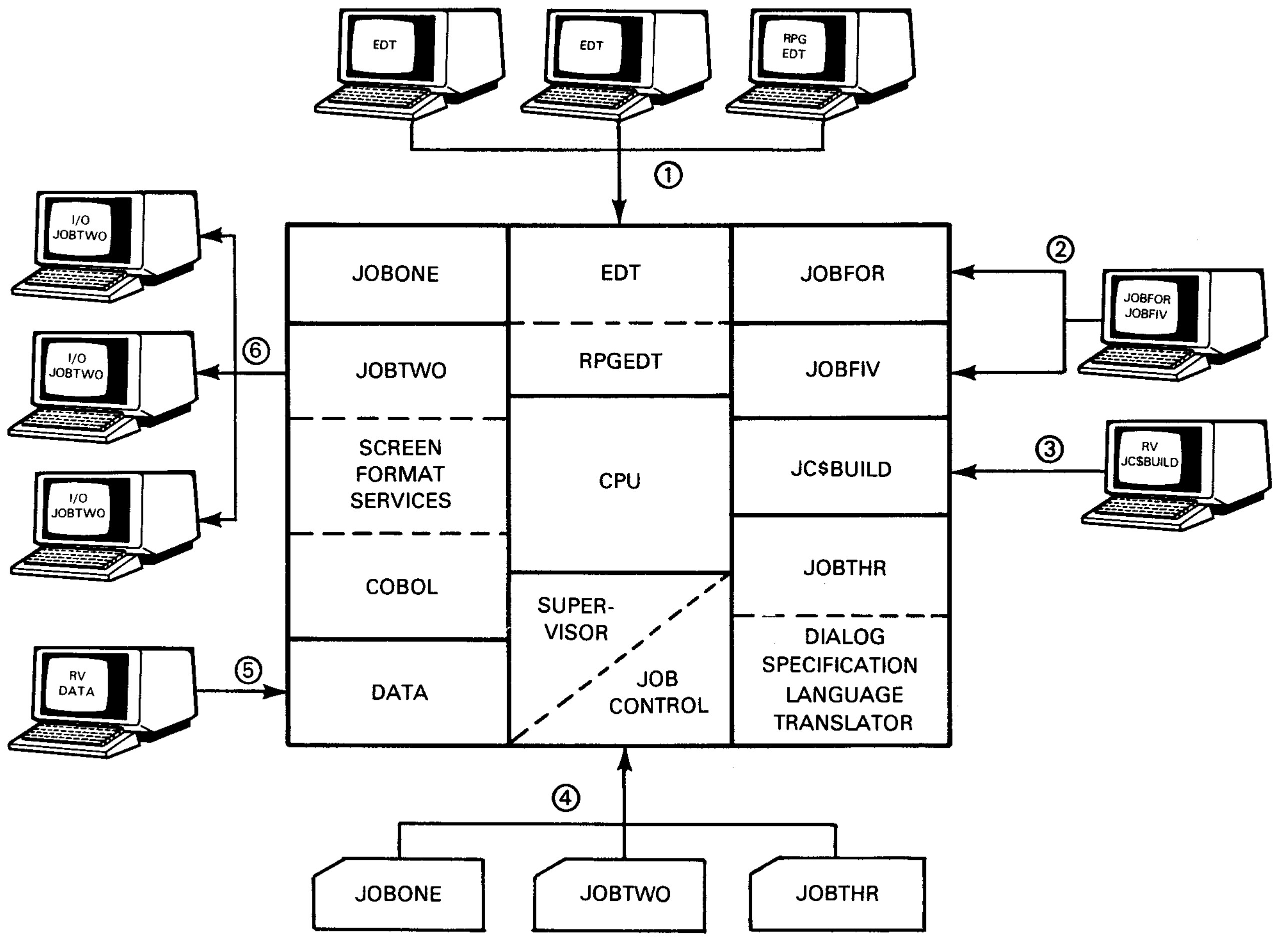
The programs that are executed by a direct workstation command, including the commands themselves, are executed as *supervisor symbionts*. A symbiont is a high priority program that is executed directly by the supervisor. Symbionts vie with the jobs running in the system for resources such as main storage and devices.

Symbionts have, however, a higher scheduling and execution priority and are usually the next function initiated when a symbiont request is issued. The symbionts are placed on the system switch list but at a high priority. When a request for the execution of a symbiont is made, the system interprets the request, locates the proper symbiont, loads it into main storage, and makes the appropriate entry for that symbiont on the system switch list.

Those programs executed through job control are scheduled in the same way described for your jobs. The job control information is stored in $Y$JCS. The RUN/RV command you enter to initiate the interactive program is interpreted by the run processor, the appropriate control stream is located, and the job is scheduled.

In addition to being able to interactively generate job control streams, programmers can also control, to a certain extent, the way a control stream is executed. It is possible to have portions of the control stream skipped or to have new values inserted into the control stream. Through available workstation commands, programmers can also interactively cancel a job and receive a display showing the jobs presently in the scheduling queue.

As you can see, the System 80 processing environment supports a variety of program and job types. It extends the resources of the system to more users through its interactive capabilities and offers a fast and efficient method of performing data processing activities. Figure 1-3 illustrates a typical processing environment detailing the different types of activities that can be going on at any given moment.



NOTES:

① Three workstation operators using the general editor. One is using the RPG II editor, which is a superset of the general editor used specifically for developing RPG II programs. The general editor provides interactive program and text editing capabilities as well as file generation and program and data storage operations.

② One workstation operator initiating two independent jobs. Results from the jobs are routed to the initiating workstation. Results can be routed to the system console if the system is directed to do so or if the originating workstation is unavailable (logged off).

③ One workstation operator initiating job control dialog (JC$BUILD). This program is used to generate and store job control streams that can be executed immediately or stored for subsequent execution.

④ Three jobs entered through an optionally configured card reader. The system executive, consisting of job control routines and the supervisor, ensures that the needs of all executing programs are satisfied.

⑤ Workstation operator executing the data utilities dialog to perform a data utilities operation, such as a file copy or compare.

⑥ JOBTWO is in interactive communications with three workstation operators who are providing input to and accepting output from an executing COBOL program through the screen format services.

Figure 1-3. Typical System 80 Processing Environment

PART 2. SYSTEM OPERATION FACILITIES

2. The Workstation

2.1. GENERAL

The workstation is the primary means of communicating with the system. All processing that can occur on the system can be prepared, entered, and controlled through the workstation. Through the services provided by the workstation and the various system processors that it interfaces with, you can:

- prepare source and edit source programs and job control streams;
- prepare and edit data;
- initiate a number of interactive programs, such as the data utilities, screen format services, and the editors;
- control the scheduling, running, processing of a job; and
- interactively create data files and program libraries.

The workstation can be used as a freestanding data entry or control device, a program and job preparation tool, or as a device dedicated to a job. Each workstation configured into the system can operate simultaneously with no significant impact on the performance of each device, even if all devices are using the same function.

The following subsections offer brief discussions on the operation and usage of the workstation. For complete information on the workstation, refer to the current version of the interactive services commands and facilities user guide/programmer reference, UP-8845.

2.2. WORKSTATION OPERATION

The workstation consists of a keyboard used to make entries and a video screen that displays all entries and the system responses or queries. The workstation must be turned on, and you must issue a LOGON command to connect the device with the system. The LOGON command must be entered with the appropriate user identifier. Once accepted by the system, you can begin to perform any function desired.

The keyboard is essentially the same as that of a typewriter with a number of additional keys, such as the transmit and cursor control keys. Your workstation may also include cursor control and arithmetic operation pads. The keyboard includes a variety of function keys. These keys, when pressed, cause a predetermined action to occur. Some of these keys are dedicated to specific functions, others perform a variety of functions depending upon the component with which your workstation is in communication. In addition, if you are using the basic assembler language, you can set certain function keys to cause a specific event to occur within your program.

The workstation display screen is a cathode ray tube providing an image area of 12 or 24 lines (user option), 80 characters to a line. The images are white characters against a dark background. You can highlight fields within the screen by reversing the display for dark characters against a white area. You can control the brightness of the screen image by using the intensity control knob located on the front bottom panel of the video display unit. The power on/off switch is located here also.

Your workstation operates in two modes: system mode and workstation mode. System mode is used to enter any of the available workstation commands or to initiate the various interactive features. Workstation mode is in effect when you have connected your workstation to a system program such as the editor. Each time you want to enter a workstation command, you must press the function and system mode keys simultaneously. After doing so, you can enter one command. For example, to execute the editor, you would follow this procedure:

1. Press and hold the function key
2. Press the system mode key
3. Enter EDT at the cursor
4. Press the transmit key

This initiates the editor and you can proceed to use it. Your workstation is now in workstation mode. If you wanted to enter another command to the system but did not want to terminate the editor, you can enter system mode by using the function key and system mode keys. The top two lines of your current workstation display are placed in a buffer; you can enter the system command and the cursor will move to the first line. You can then enter your system command. To return to the editor, press the function key and the workstation key. The first two lines are returned and the cursor moves to the appropriate last location. This procedure is the same if using any of the interactive features.

2.3. WORKSTATION USAGE

As stated, to use the workstation you must turn it on and connect it to the system by issuing a LOGON message. After doing so, you are free to perform any function you desire. Those things that you can initiate directly from the workstation and use are:

- the general editor, COBOL editor, and RPG editor;

- the data utilities;
- the job control stream preparation dialog; and
- the screen format services.

In addition, you have a set of workstation commands that provide a full range of capabilities to perform the following functions:

- Initiate and terminate workstation sessions
- Copy prefiled jobs and jprocs from a diskette to a library file
- Initiate the running of a job
- Alter the execution of a scheduled job
- Connect a workstation to a job
- Control the job processing environment
- Control the processing of a job
- Control the output spooling environment
- Allocate and manipulate data files and program libraries
- Run a prefiled stream of workstation commands as a batch job

These workstation commands and the commands that initiate the system components can only be issued from a workstation in system mode.

A workstation can also be used as an input/output device for an executing program. In such cases, it functions essentially as a substitute for input and output files. The workstation operator would enter input information and possibly receive output from the program.

You can have several workstations dedicated to a single program. The system offers a great deal of flexibility for workstations dedicated to a program. You can develop programs and assign workstations to meet your own requirements. However, it is essential that those who are to use dedicated workstations be aware of how their workstations are interfaced because it can affect the manner in which they use the workstation. You can identify specific workstations to be assigned to a program or allow the workstation operators to issue connects. You can also assign workstations to a program by user identifiers. This allows you to specify that any workstations logged on with specified user identifiers are to be connected to the program.

When dedicating multiple workstations to a program, you can specify that all workstations must be available (that is, logged on) before the program can be executed, or that only a specified number of those workstations need be available. Workstations can be connected to a program as they are logged on or operators can disconnect their workstations from a program.

Depending upon the particular program or your requirements, you can have workstations interfaced directly with the program or through system programs that provide screen format or dialog processing services. Screen format services generate screen displays that the workstation operator fills in to enter the required data. Dialog processing involves the presentation of a series of questions to which the workstation responds. Use of screen formats and dialogs permits you to enforce uniform data entry and can eliminate errors or omissions. Screen formats and dialogs are discussed in Section 7.

2.4. ADDITIONAL WORKSTATION FEATURES

In addition to the workstation capabilities we've described so far, there are other features that extend the usefulness of your workstation. These include:

- remote workstation capability
- auxiliary printer
- screen bypass
- menus
- security maintenance utility

These features are described in 2.4.1. through 2.4.5.

2.4.1. Remote Workstation Capability

One feature that allows you to extend the range of your workstation is remote workstation capability. Using communications software, you can operate workstations at a distance from your system ranging from a few feet to thousands of miles. Even at these distances, you can give remote workstations the same capability as local workstations (those attached directly to your system) to initiate and control jobs, perform other system-related functions, and act as input/output devices to user programs. (See 13.1 for more information.)

2.4.2. Auxiliary Printer

You can also extend the usefulness of your System 80 workstation by attaching an auxiliary printer directly to it. This feature gives you two advantages:

1. Using job control statements, you can direct program output to your auxiliary printer when the system printer is busy.

2. Auxiliary printers can be attached to local or remote workstations; thus you can get printed output on the spot even when your workstation is physically distant from the system.

More information on auxiliary printers is included in 10.3.

2.4.3. Screen Bypass

Closely related to the auxiliary printer feature, the screen bypass feature is available with some local System 80 workstations. Normally, data destined for output to the auxiliary printer has to use the same workstation buffer that the screen display uses. Thus, printer data first appears on the workstation screen, overwriting anything else there. With the screen bypass feature, the workstation has two separate buffers, one holding the data that's displayed on the workstation screen and the other holding the data that's being printed on the auxiliary printer. Neither buffer overwrites the other so screen and printer data remain separate.

2.4.4. Menus

Your OS/3 workstation provides you with menus. Like other types of menus, these are lists of items or options from which to choose.In OS/3, these lists are numbered; you simply read the menu, choose an item (usually a program or function to execute) and enter the item's number. After execution has finished,the menu usually returns for you to make another entry. To narrow your range of choices, a menu may call other menus. Or if you are not sure what to choose, you can ask the system to display "help" screens that explain the menu options in greater detail.

Sperry Univac supplies menus and help screens as aids in understanding and using the OS/3 interactive services. In addition, you can create menus for your own use either with interactive services or with your own programs. More information on menu creation and use is included in Section 7.

2.4.5. Security Maintenance Utility

Another workstation feature is the security maintenance utility. It performs the following functions

- helps the system administrator enforce system security;
- controls access to the system's interactive facilities;
- enables the administrator to account for computer time; and
- provides an automatic method of executing predefined sets of interactive commands when logging on

The security maintenance utility is discussed in Section 10.5.

3. The Supervisor

3.1. GENERAL

The SPERRY UNIVAC Operating System/3 (OS/3) Supervisor is a package of routines that form the heart of OS/3. It is the supervisor that allows other parts of OS/3 to work together and makes possible such useful OS/3 features as multijobbing and spooling.

As far as your user programs are concerned, the supervisor has two main functions:

- It interacts with user programs and symbionts to provide the services and control they need.
- It acts when necessary to handle randomly occurring external events, such as errors. It ensures that an error occurring in one job causes only that job to be terminated, leaving all other jobs unaffected.

The supervisor is built around executable modules, or routines, each of which has a specialized function. Those routines commonly used by the supervisor always reside in main storage. Other less often used routines, called transients, are stored on the SYSRES volume and are loaded in main storage only when the supervisor needs them. This arrangement promotes supervisor efficiency: it minimizes the amount of main storage the supervisor uses by overlaying unneeded transients with newly loaded transients, and it eliminates the input/output time needed to load the most commonly used routines by keeping them resident.

The following is a brief description of the operation of the supervisor and the services it provides. If you want to learn more about the supervisor, see the supervisor concepts and facilities user guide, UP-8831 (current version).

As stated, the supervisor provides the central control for all system activities. It manages the processing of multiple batch jobs by allocating resources, loading into main storage, and executing each batch job based upon information provided by the job control routines. For the interactive user, the supervisor initiates activities requested by the workstation operator. For all users, the supervisor, through the physical input/output control system, performs the actual movement of information within the system.

The supervisor also manages the main storage region available to users. It makes sure that sufficient main storage is available for each new job before execution can begin. If sufficient main storage exists but in scattered pieces, the supervisor can consolidate these into a single block large enough to hold the job.

The supervisor also provides the facilities to support the diagnostic and debugging aids. Included are the dump routines, the error logging facility, automatic recovery procedures for hardware and software failures, and the online maintenance procedures.

In a system configured with the spooling option, the supervisor manages the various spooling modules and ensures the proper operation of the spooling routines. Spooling is configured into the system during the generation of the supervisor, and it is at that time that the various optional features of spooling can be included.

A timer and day clock can be included in the supervisor to provide not only an interval timing facility but also date and time stamps for all applications. As part of its management of system resources, it can be configured to include a facility to suspend the execution of low priority jobs and place them onto a disk for temporary storage and load another job having preemptive priority. This is called the rollin/rollout facility. The preempted jobs are rolled out of main storage and are rolled back in once sufficient memory resources are available.

The operation of the system console is supported directly by the supervisor. This support includes the management of messages to and from the console and other workstations or executing jobs. The operator also has commands to directly control or alter the operation of the supervisor.

Your supervisor is configured during the system generation procedure (SYSGEN). During SYSGEN, you enter supervisor-related parameters to indicate those supervisor features you want to include which are resident (always in main storage), and which are transient (called into main storage when needed). Additional features increase the main storage requirements of your supervisor that may be a consideration for smaller systems.

A number of supervisor generation parameters are used to provide default values that the supervisor uses to handle certain conditions. You can generate a number of separate supervisors during system generation, but only one supervisor at a time can be operating. For example, if your system is used primarily for one type of operation during certain periods of the day, you may want to generate a separate supervisor to handle that activity more efficiently than your regular supervisor.

3.2. SUPERVISOR OPERATION

Every job that enters the system is divided into tasks. Every task is a unit of specific work to be done and is the smallest entity that can compete for central processing unit (CPU) time. Each job is made up of at least one task.

Each task is assigned a task switching priority. The supervisor uses these priorities to determine task-processing order. Because the supervisor passes control from task-to-task, several jobs can be processed concurrently. While one job is reading data from a disk file, for example, another job might be printing a report. In OS/3, up to 14 jobs can be active in the system at a given time.

The supervisor coordinates executing tasks through interrupts. An interrupt is a break in program flow that passes processing control to the supervisor, which in turn diverts processing to another program component or task. When the new program or task is completed or another interrupt occurs, the supervisor returns control to the task that was executing when the first interrupt occurred or to another task if the first task is not ready to resume processing.

The general types of interrupts are:

- Supervisor call – occurs in response to the SUPERVISOR CALL (SVC) machine instruction. Your programs routinely use the supervisor call to request supervisor services.

- Exigent machine check – indicates a malfunction in or around the processor from which the supervisor cannot recover.

- Repressible machine check – indicates a malfunction in or around the processor from which recovery is possible.

- External interrupt – generated either by the processor interval timer or the system console interrupt key.

- Program check – occurs when the processor attempts to execute a nonexistent instruction or to execute an existing instruction in an illegal manner.

- Program event recording (PER) – provides dynamic monitoring of executing programs by storing information about the current instruction whenever a specified event occurs.

- Input/output – occurs in response to signals from I/O channels.

- Restart – occurs when the restart key on the system console is pressed and can be used to take a dump (see 11.1.1) and reload the system.

Some interrupts, like the supervisor call or input/output, are routinely encountered; others, like program or machine checks, represent serious errors that the supervisor must handle with minimal system interruption. In addition to coordinating the activities of executing jobs, the supervisor is responsible for monitoring the activities of symbionts.

Symbionts are system programs or routines that execute directly under the control of the supervisor and perform functions that are usually critical in nature or are extremely time dependent. Symbionts are loaded by the supervisor in response to a request made by another system component, one of your executing programs, or an interactive user. For example, a number of the workstation command functions are symbionts that are executed by the supervisor in response to the command entry. Symbionts do not occupy job slots during execution.

3.3. SUPERVISOR SERVICES

While the supervisor performs its functions automatically in nearly all cases without user intervention, there are occasions that it becomes necessary to closely control its operations. These needs arise from special programming requirements and the like. Thus, the assembler program is provided with the capability to control the operation of the supervisor directly through a series of declarative and imperative macroinstructions that can be included as part of a basic assembly language (BAL) program. In some cases, the user is merely issuing the macroinstructions that other system components would have issued in response to a user request made, for example, through a higher level language. However, there are some features of which only the assembler programmer can take advantage. Those items that the assembler programmer can control directly through macroinstructions are:

- Disk and diskette space management
- System access technique
- Multitasking
- Program execution management
- Diagnostic and debugging features
- Spooling control

The disk and diskette space management macroinstructions allow you to obtain detailed information on files and volumes in your system.

The system access technique (SAT) declarative and imperative macroinstructions allow you to generate and manipulate block level disk and tape files. SAT files are partitioned sequential files. Most program libraries are partitioned SAT files.

The supervisor permits the BAL programmer to create multiple tasks within a job step. This capability is called *multitasking,* or the concurrent execution of multiple tasks. It enables you to overlap processing with external occurrences (input/output functions or operator intervention) within a job step to obtain maximum throughput. When a task is interrupted to perform external processing, the central processor is freed, and the system searches for another task to be performed (that is, not waiting for an external event to be completed). This task could be in the same job step, or be in any other job step currently being processed.

If you program in BAL, you can, through the inclusion of the program execution management macroinstruction, closely control the loading and execution of your program. By using these macroinstructions, you can directly control the loading of program phases and the initiation and termination of your job, and link your program to island code subroutines. In addition, you can obtain the current date and time to be used by your program and set an interval timer to control processing. Included also are macroinstructions to access the prologue of your job for information retrieval.

Diagnostic services are provided that allow you to obtain the following types of dumps:

- Terminate job and dump job region
- Dump and continue processing
- Complete system dump

These dump routines are discussed in Section 11.

In addition, supervisor macroinstructions are available for performing dynamic dumps of selected portions of main storage during your program's execution. The dynamic dumps do not affect the program's execution and are referred to as snapshot dumps.

Included are diagnostic macroinstructions to generate checkpoints and a checkpoint file to be used to recover program-generated data files in the event of a program termination to the last established checkpoint before the termination. The terminated program can be restarted and it will begin generating the file from the checkpoint. This is especially useful when dealing with large data files.

A number of message transfer facilities are available and they allow you to:

- write a message to the system log file, either with or without displaying it on the workstation or system console; and
- display a message on a workstation or console without writing it in the system log file and with or without requiring an operator response.

These message transfer facilities are explained in the consolidated data management macroinstructions user guide/programmer reference (current version).

The supervisor provides a breakpoint facility for those systems configured with spooling. Breakpointing allows you to have incomplete print and punch files processed by the output writer by including the breakpoint macroinstruction in your assembler program. The print or punch spool file is closed at the point where the breakpoint is issued, the incomplete file is printed or punched, then the file is reopened and your program can continue to place records in the spool file for subsequent printing or punching.

4. Consolidated Data Management

4.1. GENERAL

As you know, all computer programs process data in one form or another; however, the data and the program are in two different places. The program is executed in the main storage section of the central processing unit (CPU) and the data is contained on devices external to the CPU.

To process the data and produce the desired results, data must be moved in from and out to these peripheral devices. Because the physical and electronic characteristics of the various devices differ, this can lead to problems if you have to take the characteristics of a device into consideration each time you want to perform an input/output operation.

Obviously, there is a need for some way to specify an input/output operation in a program on a logical level. The answer to this need is consolidated data management.

4.2. WHAT IS CONSOLIDATED DATA MANAGEMENT?

Consolidated data management is a collection of program modules that are written for each of the input/output devices supported by your system. These modules handle the actual movement of data. They take care of all the device characteristic requirements; consequently, you need not worry about this when you want to perform input/output operations. All you need to do is make a formal request in your program to data management and it moves the data in from or out to the particular input/output device. Figure 4-1 illustrates the relationship between data management and your program. As you can see, data management acts as the data transfer mechanism between your program and the input/output devices.

4.3. HOW CONSOLIDATED DATA MANAGEMENT WORKS

When you write your program, you establish a unique file name for each input/output file you intend to use. You then describe the characteristics for each file. Once this is done, you use these file names in conjunction with input/output commands at each point that you want to move data into or out of your program. The input/output commands act as formal requests to data management.

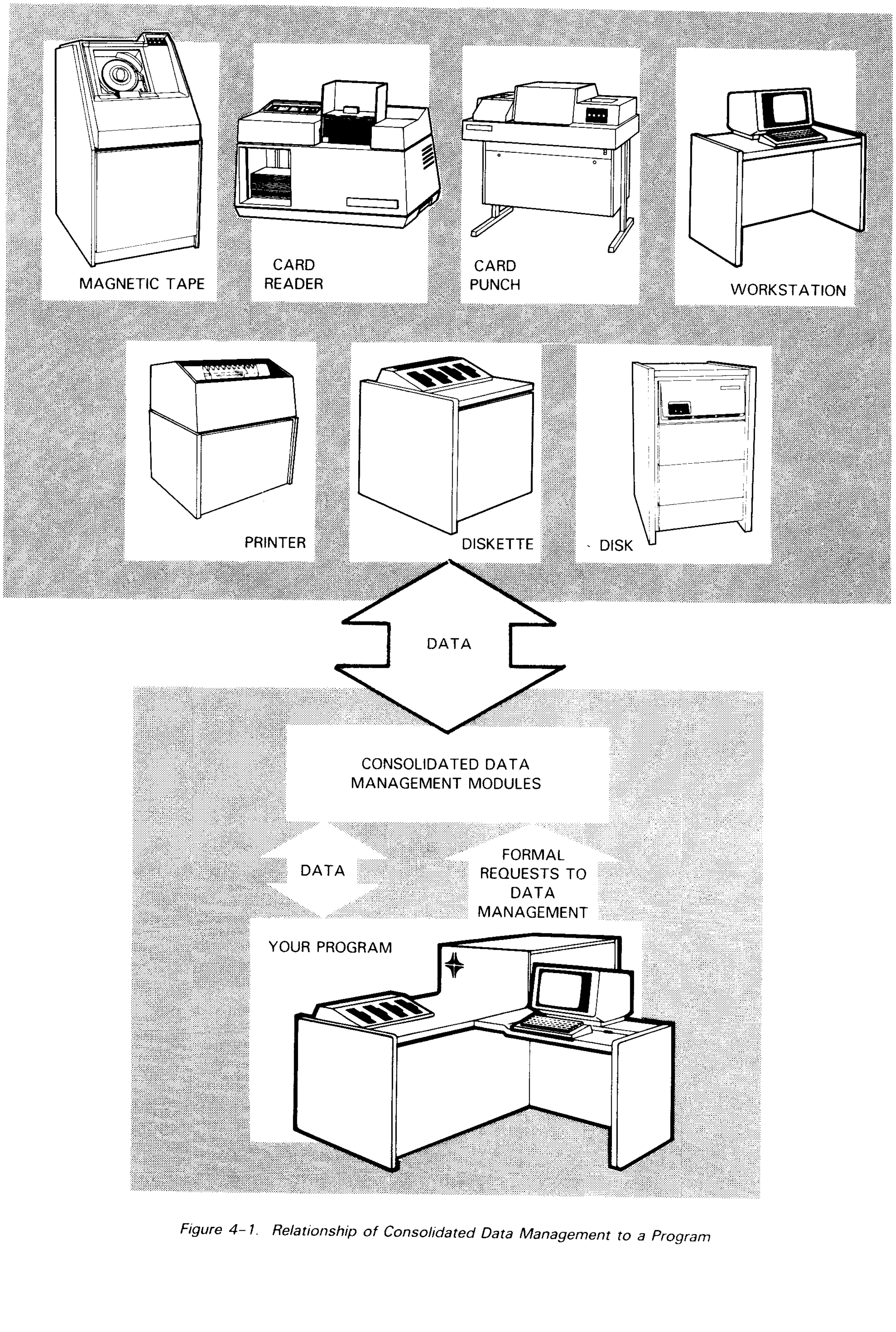
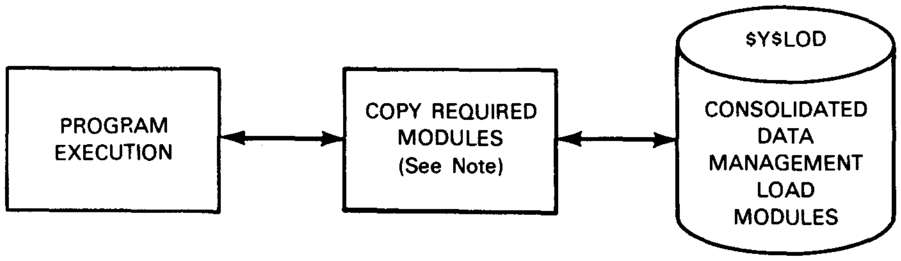


Figure 4-1. Relationship of Consolidated Data Management to a Program

When the time comes to execute your program, the appropriate data management modules are placed in main storage at this time as shown in Figure 4-2. Thereafter, each time an input or output command is encountered during processing, the applicable data management module gets data from or sends data to the appropriate device.


PROGRAM EXECUTION
COPY REQUIRED MODULES (See Note)
$Y$LOD
CONSOLIDATED DATA MANAGEMENT LOAD MODULES

NOTE:

Based on device assignment sets for files in the job control stream, the required modules are copied at execution time.

Figure 4-2. Consolidated Data Management and Program Execution

4.4. DATA STRUCTURE

Consolidated data management recognizes the following as structural entities:

- Volume

  The largest physical unit for data storage such as tape reel or disk pack.

- File

  A delimited storage space having an identifying file name and consisting of a collection of related data.

- Record

  A collection of contiguous characters within a file that you have designated to be handled as a unit.

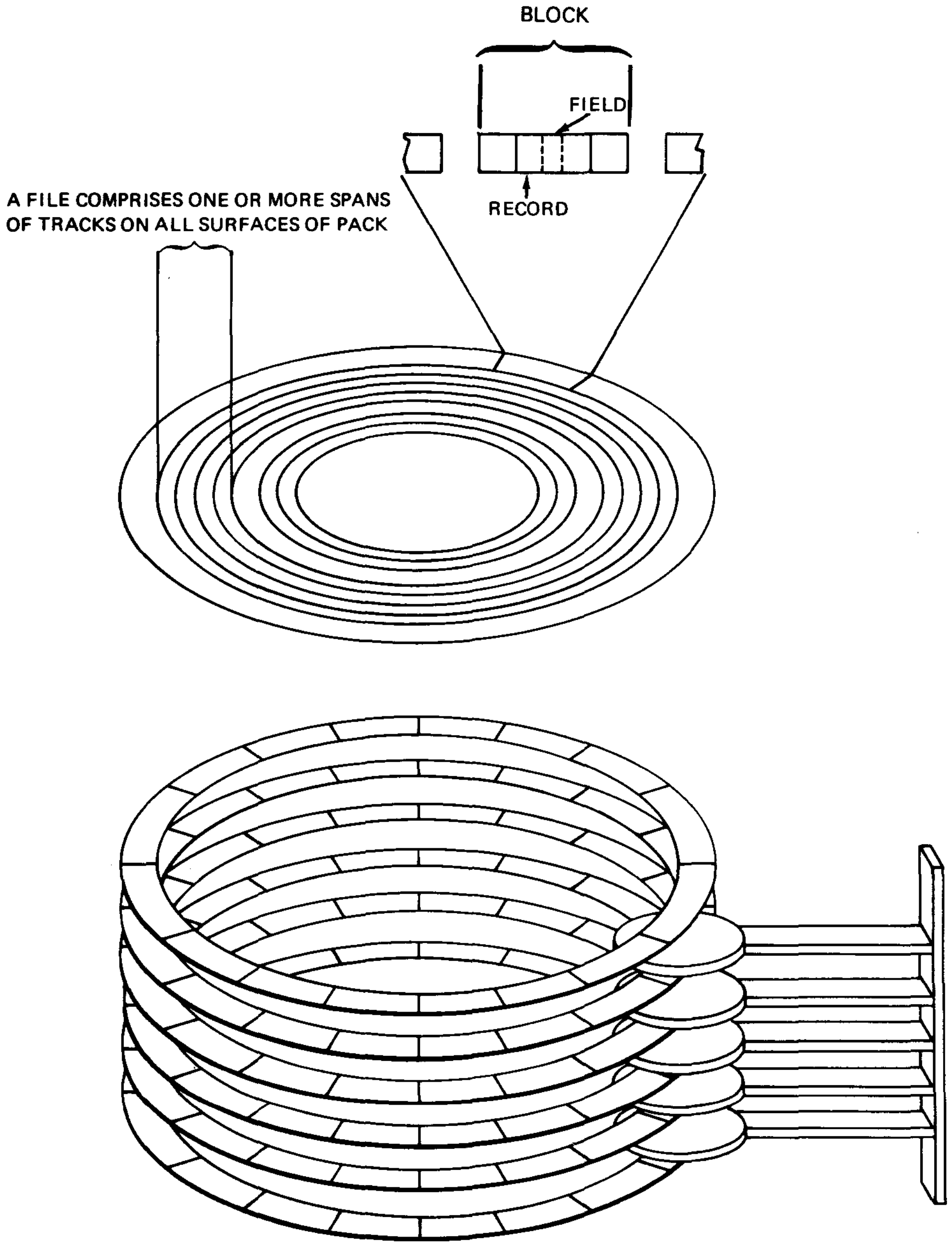
- Block

  That portion of a file that is transferred into or out of main storage by a single access. A block may contain a single record or, for some devices, it may contain several records.

- Field

  One or more contiguous characters within a record that represents a single piece of information.

The volume concept is not truly applicable to printers, workstation, or card devices. On disk, diskettes, and magnetic tape, a file may be larger then a volume; that is, a file may require more than one physical unit to hold it. In this case, you have what is called a multivolume file. Figure 4-3 shows the organization of data on the peripheral devices supported by consolidated data management.

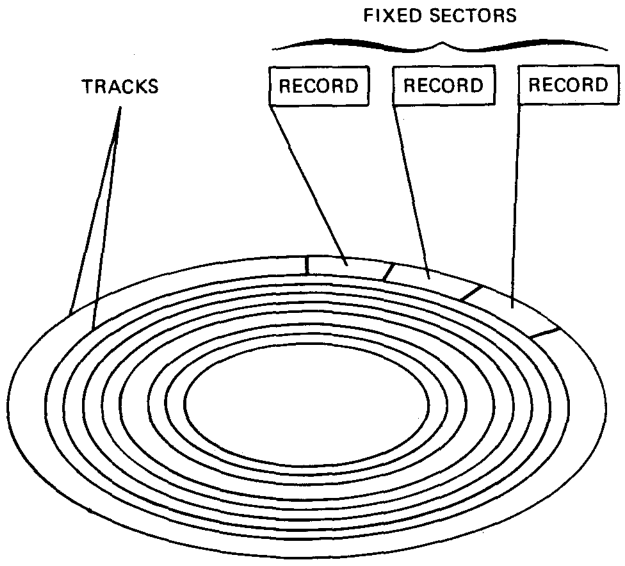

BLOCK
FIELD
RECORD
A FILE COMPRISES ONE OR MORE SPANS OF TRACKS ON ALL SURFACES OF PACK

a. Disk pack

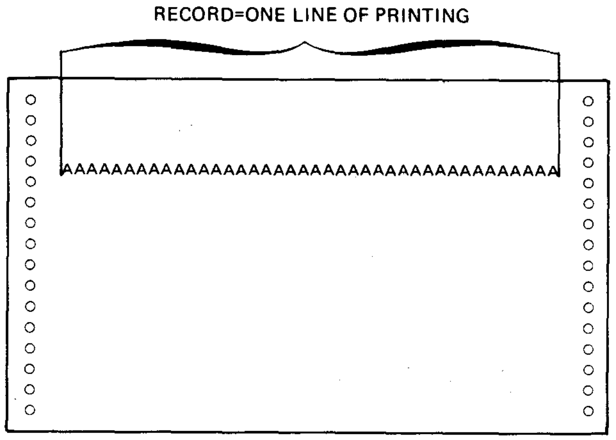
NOTE:

The set of tracks at a specific radius on all recording surfaces is called a cylinder.

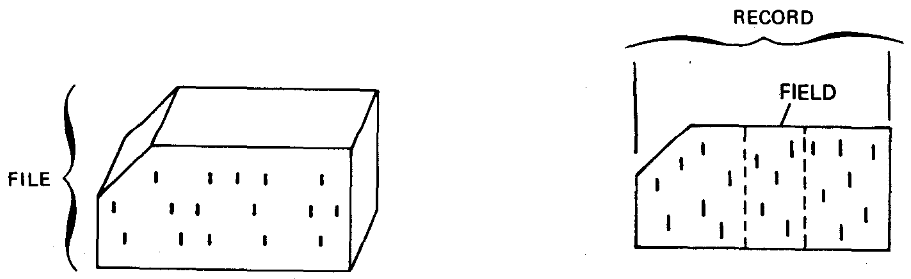
Figure 4-3. Organization of Data on Peripheral Devices (Part 1 of 3)



b. Diskette

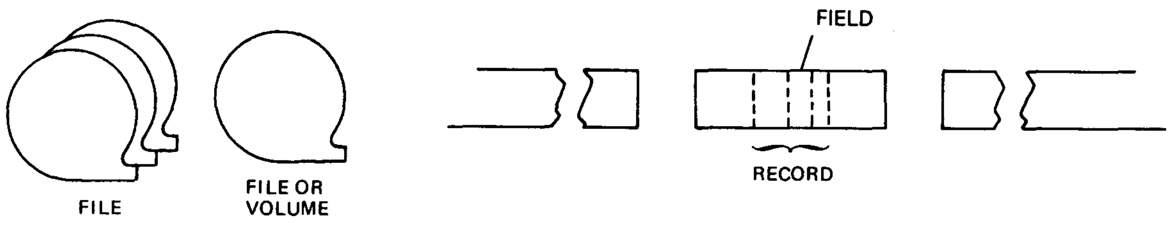

RECORD=ONE LINE OF PRINTING
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

c. Printer

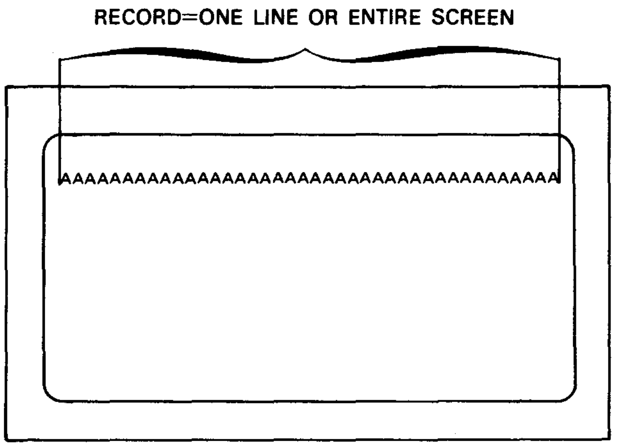


d. Punched card

Figure 4–3. Organization of Data on Peripheral Devices (Part 2 of 3)



e. Magnetic tape


RECORD=ONE LINE OR ENTIRE SCREEN
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

f. Workstation

Figure 4-3. Organization of Data on Peripheral Devices (Part 3 of 3)

4.5. AVAILABILITY OF CONSOLIDATED DATA MANAGEMENT

Consolidated data management is available to programmers using COBOL, FORTRAN IV, RPG II, BASIC, or ESCORT by means of the various input/output operations available in each of these languages. Basic assembly language programs can interface directly with consolidated data management through the data management macroinstructions. Whatever the language, the functions of consolidated data management are the same for all users. For more about consolidated data management, see the consolidated data management concepts and facilities, UP-8825 (current version).

4.6. DATA UTILITIES

Sperry Univac provides a system routine to transfer large amounts of data between the various peripheral devices configured into your system. This routine, called the data utilities, can be used to perform a number of data transfer functions including:

- Transferring files from one media to another
- Making copies of files either on the same media type or another type

- Performing certain compare, delete, and reformatting operations

More detailed information on the data utilities can be found in Section 9.

4.7. CARD FORMATS AND FILE CONVENTIONS

A punched card file consists of a card deck that is input via a card reader or output via a card punch. Records can comprise either a portion of a card or a complete card. The basic punched cards for the card subsystems are the standard 80-column cards. However, optional hardware features allow you to read 51- or 66-column cards. Refer to Appendix B for the functional characteristics of the card subsystems that are supported.

4.7.1. File Organization

Punched card files are sequential files; that is, the records are handled one at a time in sequential order. The card deck consists of a job control start-of-data card (optional), data cards containing one record each, and a job control end-of-data card.

Figure 4-4 shows a typical card file structure.

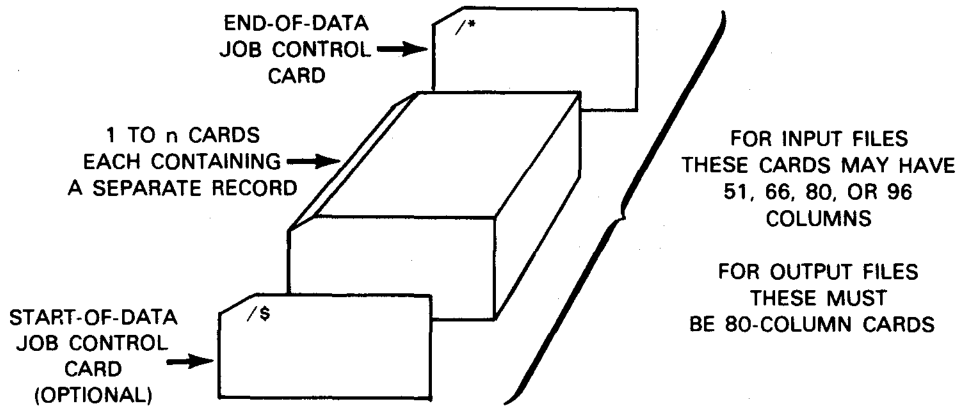
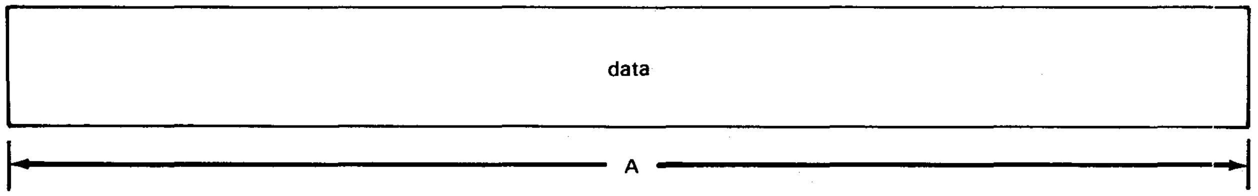

END-OF-DATA JOB CONTROL CARD
/*
1 TO n CARDS EACH CONTAINING A SEPARATE RECORD
START-OF-DATA JOB CONTROL CARD (OPTIONAL)
/$
FOR INPUT FILES THESE CARDS MAY HAVE 51, 66, 80, OR 96 COLUMNS
FOR OUTPUT FILES THESE MUST BE 80-COLUMN CARDS

Figure 4-4. Typical Card File Structure

4.7.1.1. Card Input Files

A card input file consists of fixed-length, unblocked records; that is, all records are the same size. When a record is read, it is placed in the input/output area. Figure 4-5 shows the record format for fixed-length, unblocked records.



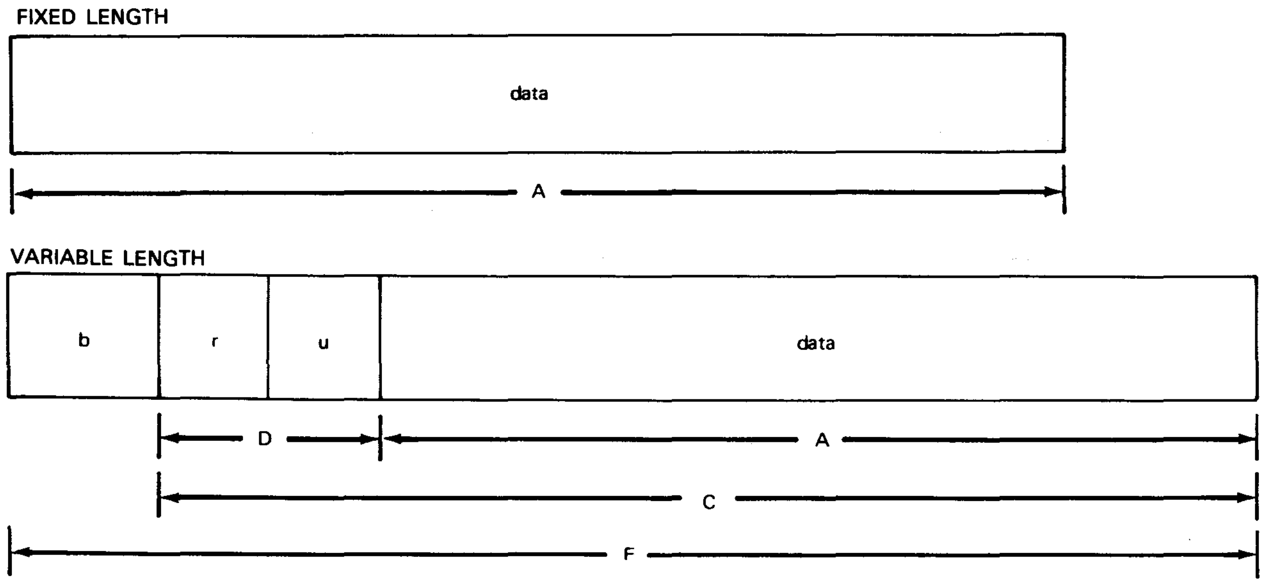
LEGEND:

A	Data record length. The I/O area must be at least the same size as the record length and must be an even number of bytes. If you are using 51-column cards, the I/O area must be specified as 52 bytes.

Figure 4-5. Fixed-Length, Unblocked Record Format for Card Input Files

4.7.1.2. Card Output Files

A card output file consists of data that is formed into records, either in the input/output area or a designated work area, and then is sent to the card punch, where the records are punched in the standard 80-column card format. The output records can be fixed length or variable length. These record formats are shown in Figure 4-6.



LEGEND:

b	Block size field, four bytes	A	Data record length
r	Record length field, two bytes	C	Variable record length
u	Reserved (two bytes); may be any two characters chosen by the user	D	Record Size field
F	I/O area layout. The I/O area must be an even number of bytes and its size must equal the maximum record size plus the block size and record size fields if you are dealing with variable-length records.		

Figure 4-6. Record Formats for Card Output Files

4.8. PRINTER FORMATS AND FILE CONVENTIONS

A printer file consists of data that you create in your program and cause to be printed, one line at a time, on a printer device. Each printed line can have up to 160 characters depending upon the printer subsystem you use. Refer to Appendix B for the functional characteristics of the printer subsystems that are supported.

4.8.1. File Organization

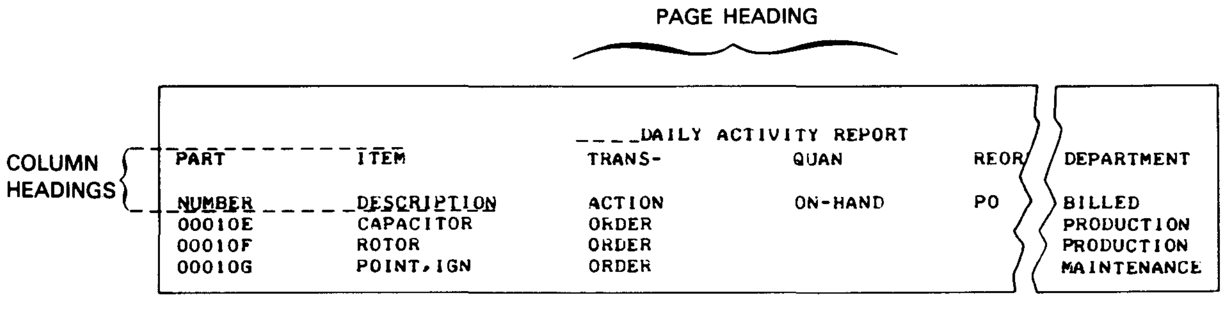
Printer files can be organized to produce text, tabular data, or data on preprinted forms. In each case you must set up the data you want printed on each line, must control the vertical separation between lines, and must provide for skipping to the next page when the space on the current page is exhausted. (See the applicable programming language user guide for details.)

4.8.1.1. Text

The simplest printer file is one that consists entirely of text. An example of this is the lines you are presently reading. If you had to write a program to produce these lines, each line would be a record and you would form each line in an I/O area or work area. Then you would cause it to be printed. This process is repeated for each line until the end of the page is reached at which point you issue an instruction to skip to the top of the next page.

4.8.1.2. Tabular Data

The records for tabular data and reports are formed in the same manner as in text files (in an I/O area or work area). In these cases, your program is more complex because of vertical and horizontal spacing requirements, page and column headings, and other repetitive items (Figure 4-7).



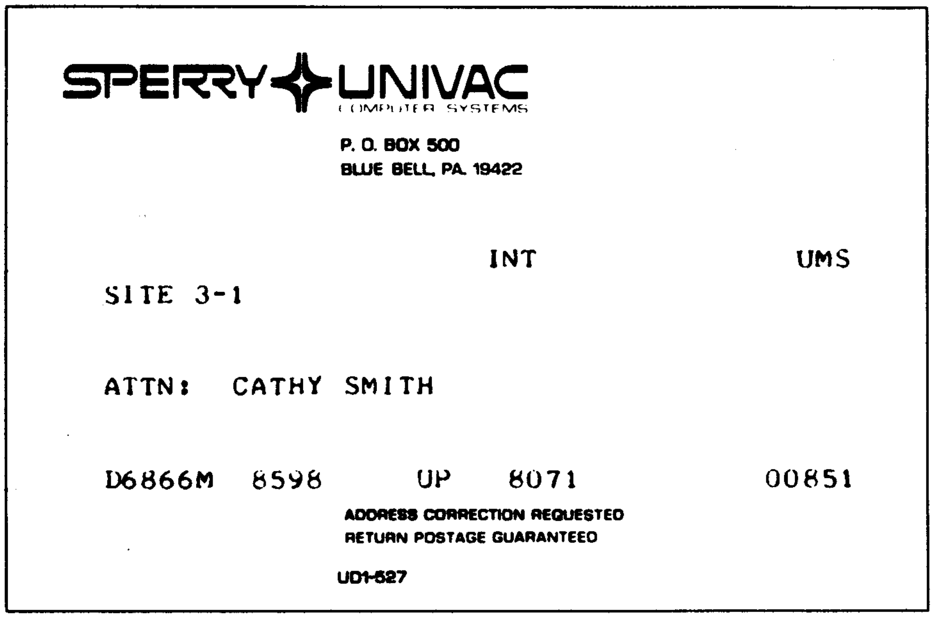
```
                                        DAILY ACTIVITY REPORT
PART          ITEM           TRANS-           QUAN          REOR    DEPARTMENT

NUMBER        DESCRIPTION    ACTION           ON-HAND       PO      BILLED
00010E        CAPACITOR      ORDER                                  PRODUCTION
00010F        ROTOR          ORDER                                  PRODUCTION
00010G        POINT,IGN      ORDER                                  MAINTENANCE
```

Figure 4-7. Sample Tabular Data

4.8.1.3. Data on Preprinted Forms

A printer file that places data on a preprinted form is easy to use once it is organized. You form your records in an I/O or work area as with text or tabular data. The difference, as you can see in Figure 4-8, is that you have to closely control the positioning of your data.



SPERRY UNIVAC
COMPUTER SYSTEMS

P. O. BOX 500
BLUE BELL, PA. 19422

```
                              INT                 UMS
SITE 3-1

ATTN:   CATHY SMITH

D6866M  8598       UP    8071                   00851
```

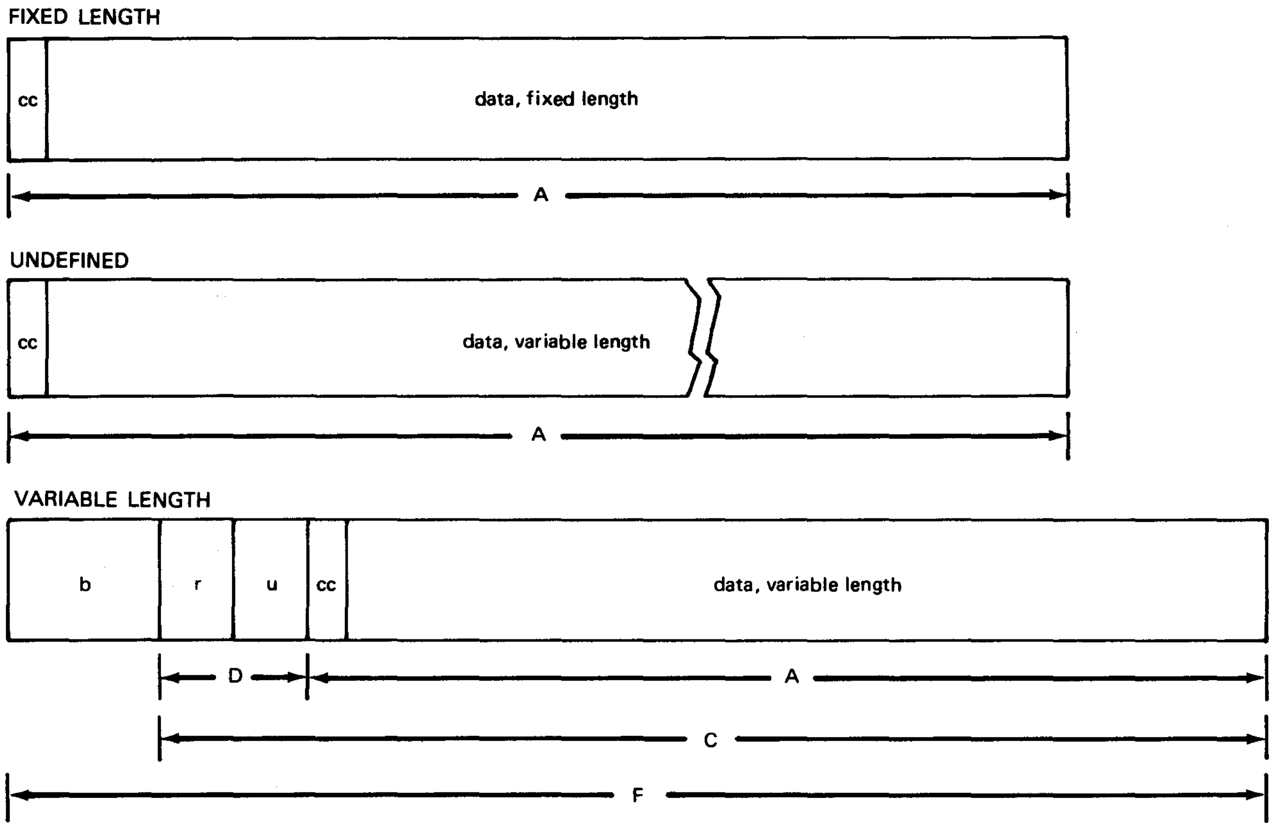
ADDRESS CORRECTION REQUESTED
RETURN POSTAGE GUARANTEED

UD1-527

Figure 4-8. Sample of Data on Preprinted Form.

4.8.2. Printer Record Formats

The printer record formats are shown in Figure 4-9. As you can see, a record may contain a control character that specifies line spacing or skipping when the file is printed. This character is not printed but is a part of the record in storage.



LEGEND:

b	Block size field, four bytes
cc	Control character, one byte, optional
r	Record length field, two bytes, binary
u	Reserved (two bytes); can be any two characters you choose.
A	Data record length
C	Variable record length
D	Record size field
F	I/O area layout

Figure 4-9. Printer Record Formats

4.8.3. Vertical Format and Load Code Buffers

All printers contain a vertical format buffer and a load code buffer. The vertical format buffer is used to define the printer page in terms of the number of lines per page, the density of the lines, the overflow line, and the codes you can use with your program instructions to skip to specific lines on a page.

The load code buffer specifies the 8-bit codes that are associated with the graphic symbols on the print cartridge, band, or drum. These cannot be changed.

The vertical format buffer and the load code buffer are set up at system generation time for each printer type. For details refer to the system installation user guide/programmer reference. You can use job control to override vertical format buffer parameters established at system generation time.

Normally, your only concern with these buffers is to know what standards are established for the printer you are going to use with your program. In most cases, these standards allow you to produce the printed output you require.

4.9. MAGNETIC TAPE FORMATS AND FILE CONVENTIONS

Magnetic tape files consist of data records that are recorded on one or more volumes (reels) of magnetic tape. These files are sequential files; the data records are recorded on tape in the order in which they are submitted, and they are read from the tape starting with the first record on tape and continuing with each successive record. The recording and reading is accomplished via a tape subsystem. Refer to Appendix B for the functional characteristics of the magnetic tape subsystems that are supported.

4.9.1. Tape Volume and File Orgranization

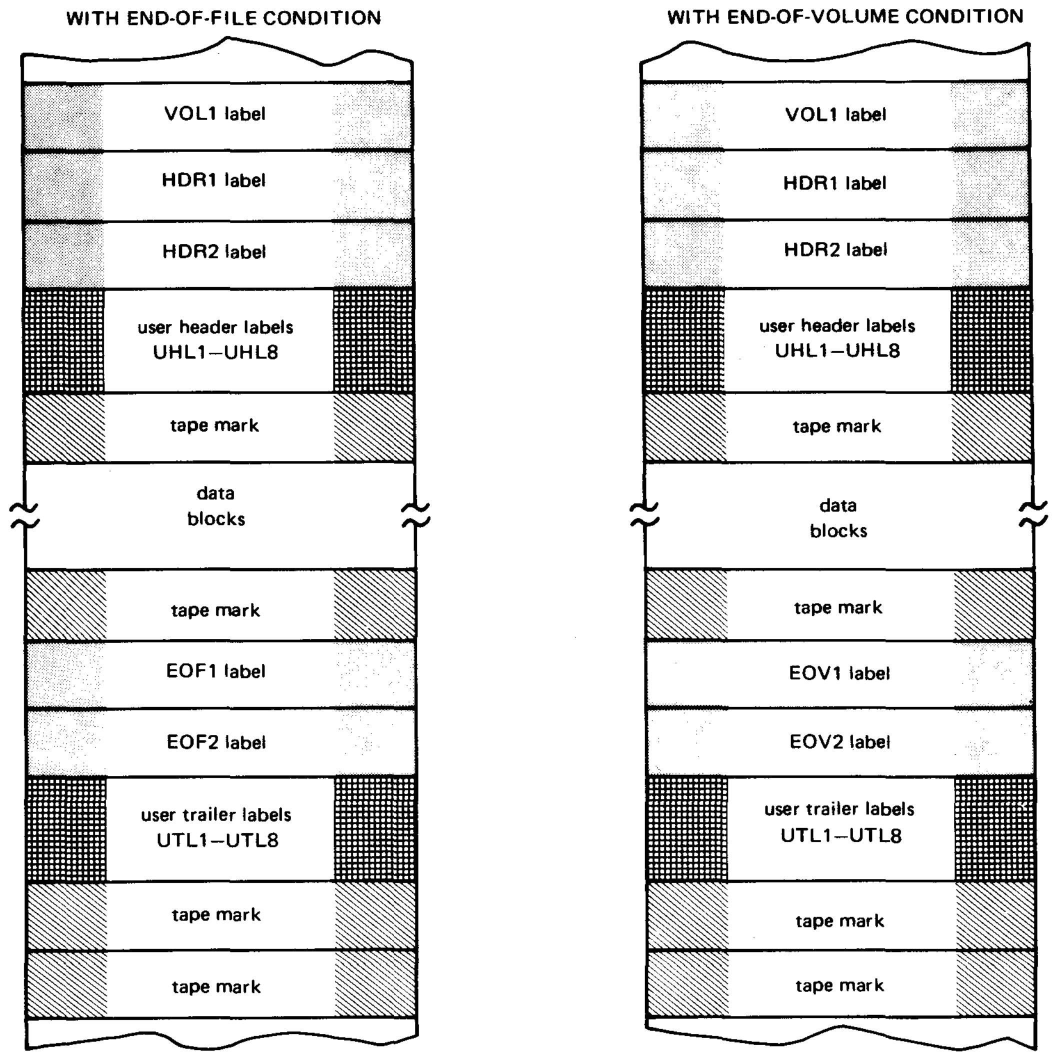
Consolidated data management allows you to process magnetic tape files encoded in Extended Binary Coded Decimal Interchange Code (EBCDIC) as well as in the *American National Standard Code for Information Interchange (ASCII) X3.4-1977.* The file structure, organization, and processing specifications for ASCII magnetic tape files is described in *American National Standard Magnetic Tape Labels for Information Interchange, X3.27-1969.* Both of these standards are followed when ASCII magnetic tape files are processed. The tape volumes (reels) can be organized as follows:

- EBCDIC
  - Standard labeled
  - Nonstandard labeled
  - Unlabeled
- ASCII
  - Standard labeled

The paragraphs that follow explain and illustrate the different types of volume organization.

4.9.1.1. EBCDIC Standard Volume Organization

A standard volume has system standard labels and required tape marks; it may also, at your option, contain standard user header and trailer labels (UHL and UTL). All standard tape labels are written in blocks of 80 bytes. Data management assumes that the labels appear on tape in the order shown in Figures 4-10, 4-11, and 4-12, which illustrate the reel organization for standard labeled EBCDIC volumes with an end-of-file (EOF) and an end-of-volume (EOV) condition. A standard labeled EBCDIC volume processed by data management ends in either an end-of-file or an end-of-volume label group, followed by two tape marks. The second tape mark indicates that no valid information follows. No provision is made for creating additional volume, header, or EOF/EOV labels on output files; if they exist on input files, data management bypasses them.



LEGEND:

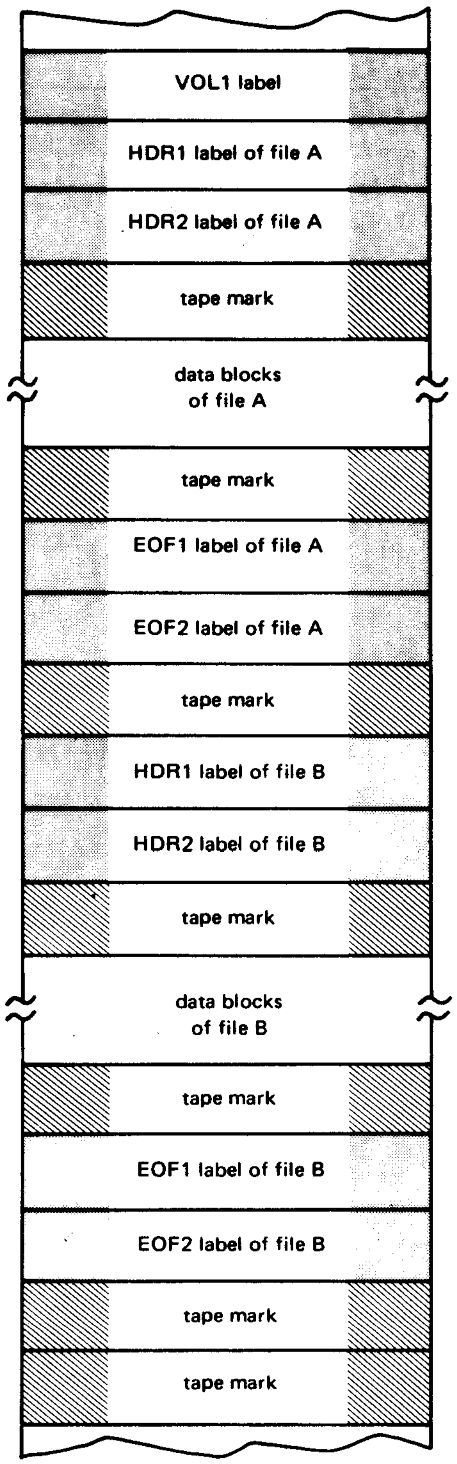
□ Content supplied by user.

▨ These bytes are required and generated by data management.

░ These bytes are generated by data management; user supplies content for certain fields.

▦ These bytes are generated by user's routine for processing these labels; content is at user's option except for content of 4-byte label ID fields. User is limited to eight UHL and eight UTL.

Figure 4-10. Organization for a Standard Labeled EBCDIC Magnetic Tape Volume (Single File)



LEGEND:

□ Content supplied by user.

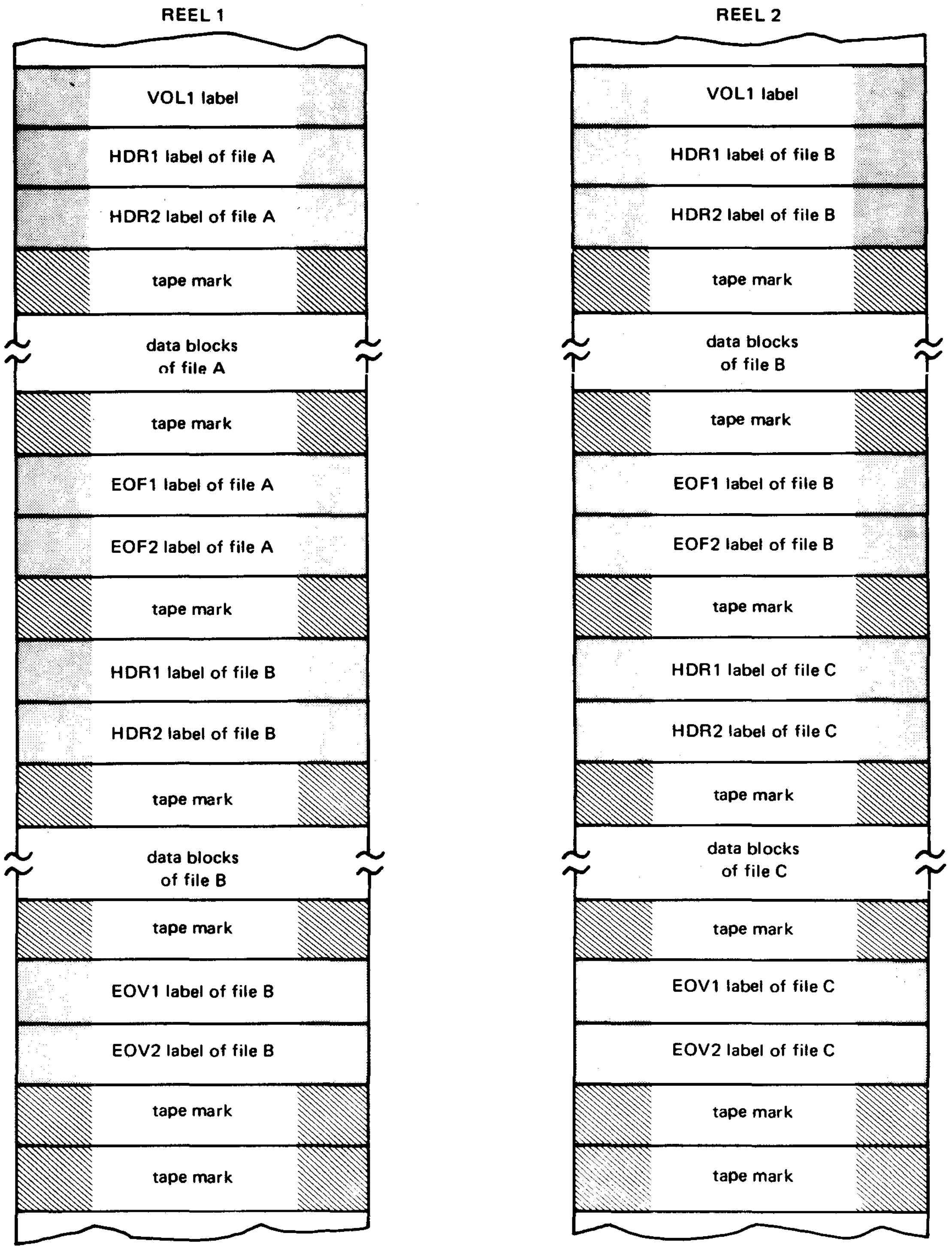
▨ These bytes are required and generated by data management.

□ These bytes are generated by data management; user supplies content for certain fields.

NOTE:

Assume that file B completes on this volume.

Figure 4-11. Organization for a Standard Labeled EBCDIC Magnetic Tape Volume (Multifile Volume with End-of-File Condition)



LEGEND:

- ☐ Content supplied by user.
- ▨ These bytes are required and generated by data management.
- ░ These bytes are generated by data management; user supplies content for certain fields.

NOTE:

Assume that file C is not completed on reel 2, but carries over (like file B) onto another volume. If file C were completed on reel 2, its EOV1 and EOV2 labels shown here would be replaced with EOF1 and EOF2 labels.

Figure 4-12. Organization for Standard Labeled EBCDIC Magnetic Tape Volumes (Multifile Volume with End-of-Volume Condition)

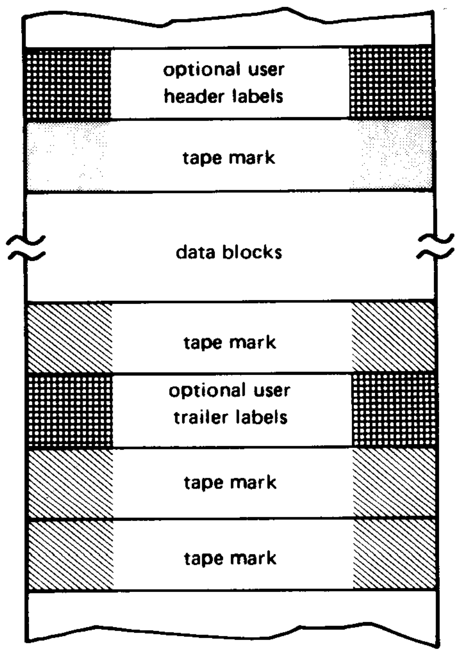
4.9.1.2. EBCDIC Nonstandard Volume Organization

A nonstandard volume is any volume that contains only nonstandard labels and certain required tapemarks. Figures 4-13 and 4-14 show the formats for EBCDIC nonstandard volumes.

The optional user header and trailer labels may be of any format, length, or number because they are handled by a label processing routine that you supply in your program.

The tape mark following the user header label is only required if you intend to use read-backward operations in your program or if you intend to omit label checking. It is not required and may be omitted if you intend to perform label checking. Normally, this tape mark is automatically generated by data management unless you specify otherwise.

The tape mark following the data blocks is required and is automatically generated by data management. If user trailer labels are present, data management automatically generates the two required tape marks that must follow these labels. If the user trailer labels are not present, data management writes only one additional tape mark after the one following the data blocks. This second tape mark is always present when a file is the only file or is the last file on the volume. If the volume is a multifile volume, this second tape mark is overwritten by the next file placed on this volume.



LEGEND:

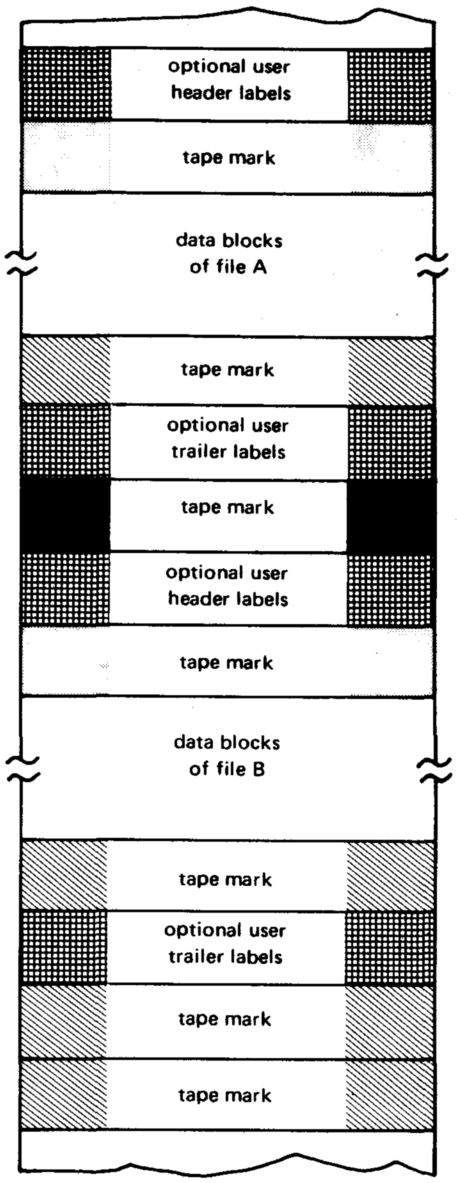
☐ Content supplied by user.

▨ These bytes are required and generated by data management; only two tape marks follow data blocks if UTL are not present.

☐ These bytes are generated by data management unless user specifies otherwise; required only if label checking is omitted or read-backward operations are specified.

▦ The presence, content, format, and number of these bytes are entirely at user's option.

Figure 4-13. Organization for a Nonstandard EBCDIC Magnetic Tape Volume (Single File)


optional user header labels
tape mark
data blocks of file A
tape mark
optional user trailer labels
tape mark
optional user header labels
tape mark
data blocks of file B
tape mark
optional user trailer labels
tape mark
tape mark

LEGEND:

☐ Content supplied by user.

▨ These bytes are required and generated by data management; only two tape marks follow data blocks of last file on volume if UTL are not present.

░ These bytes are generated by data management unless user specifies otherwise; required only if label checking is omitted or read-backward operations are specified.

▦ Presence, content, format, and number of these bytes are entirely at user's option.

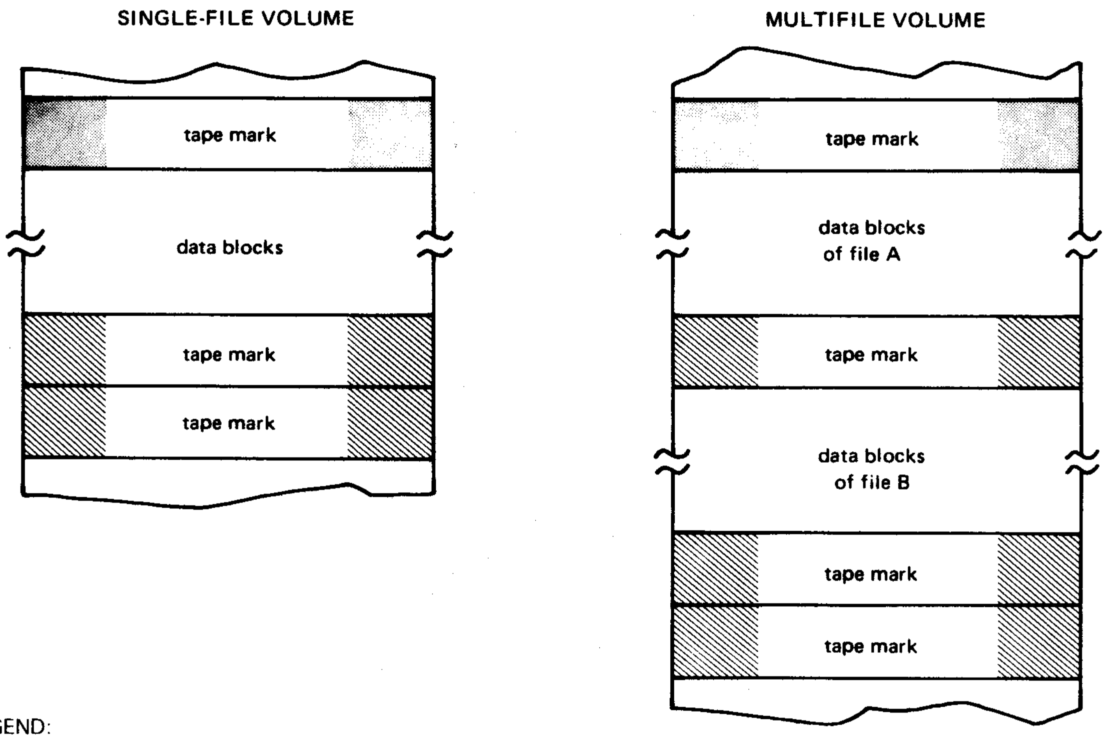
■ Always present; written by data management.

Figure 4–14. Organization for a Nonstandard EBCDIC Magnetic Tape Multifile Volume

4.9.1.3. EBCDIC Unlabeled Volume Organization

Consolidated data management can also process unlabeled tape volumes. Figure 4-15 shows the organization for unlabeled EBCDIC volumes.

A tape mark normally precedes the data block unless you specify otherwise. The tape mark following the data blocks is required on both single file and multifile volumes and is automatically generated by data management. A second tape mark is always written by data management following the last or only file on each volume. If the volume is a multifile volume, this second tape mark is overwritten by the next file placed on this volume.



LEGEND:

☐ Content supplied by user.

▨ These bytes are required and generated by data management; two marks follow data blocks of last file on volume.

░ These bytes are generated by data management unless user specifies otherwise; required only when user specifies read-backward operations.

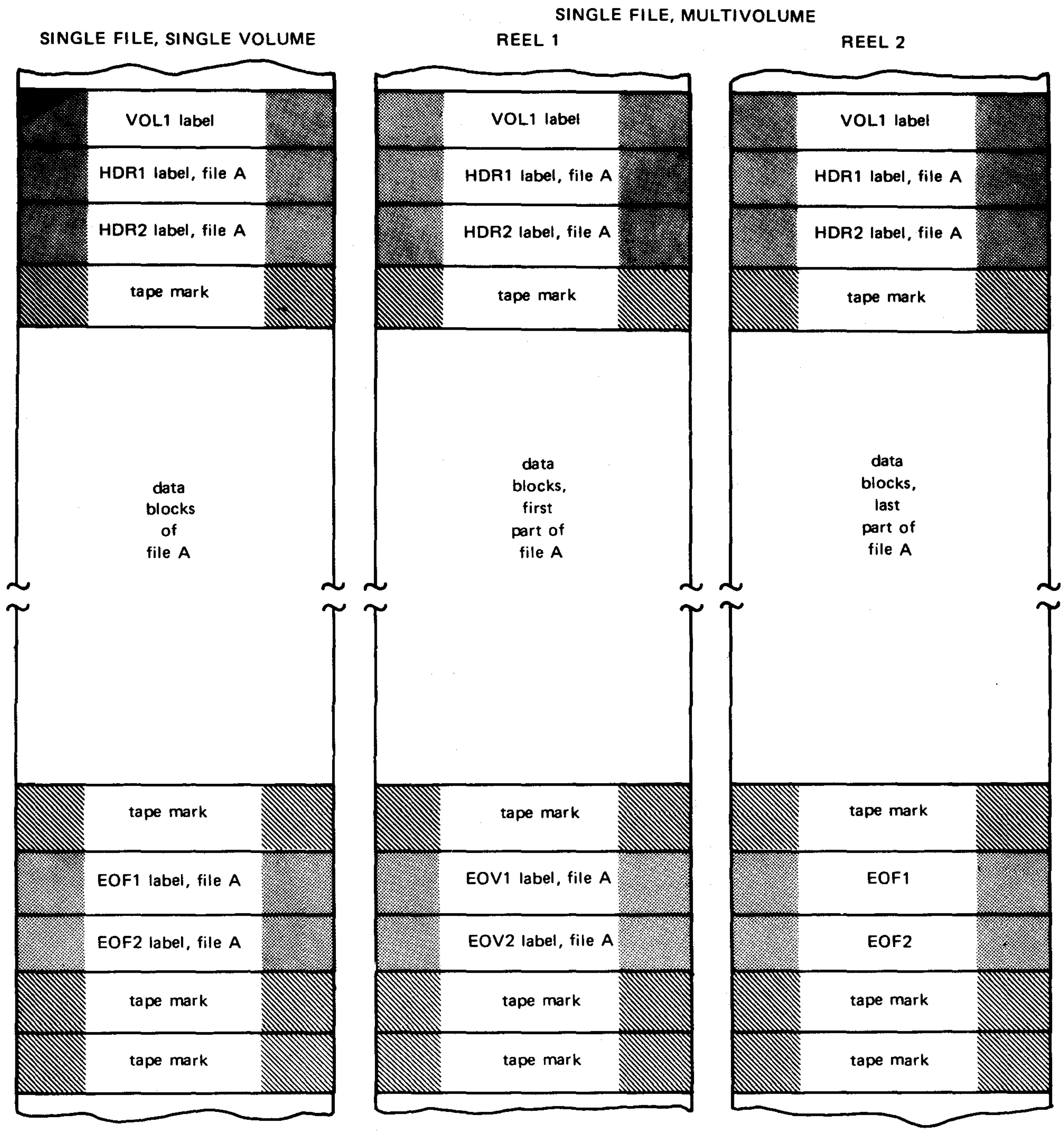
Figure 4-15. Organization for an Unlabeled EBCDIC Magnetic Tape Volume

4.9.1.4. ASCII Standard Volume Organizations

The *American National Standard X3.27-1969* provides for the following file sets (collections of one or more related files recorded on one or more volumes):

- Single file, single volume
- Single file, multivolume
- Multifile, single volume
- Multifile, multivolume

These volume organizations are shown in Figures 4-16 through 4-19. Note that all ASCII tape files contain standard labels. Since data management follows the standard, it expects to find these labels on ASCII input files and it generates them for ASCII output files.



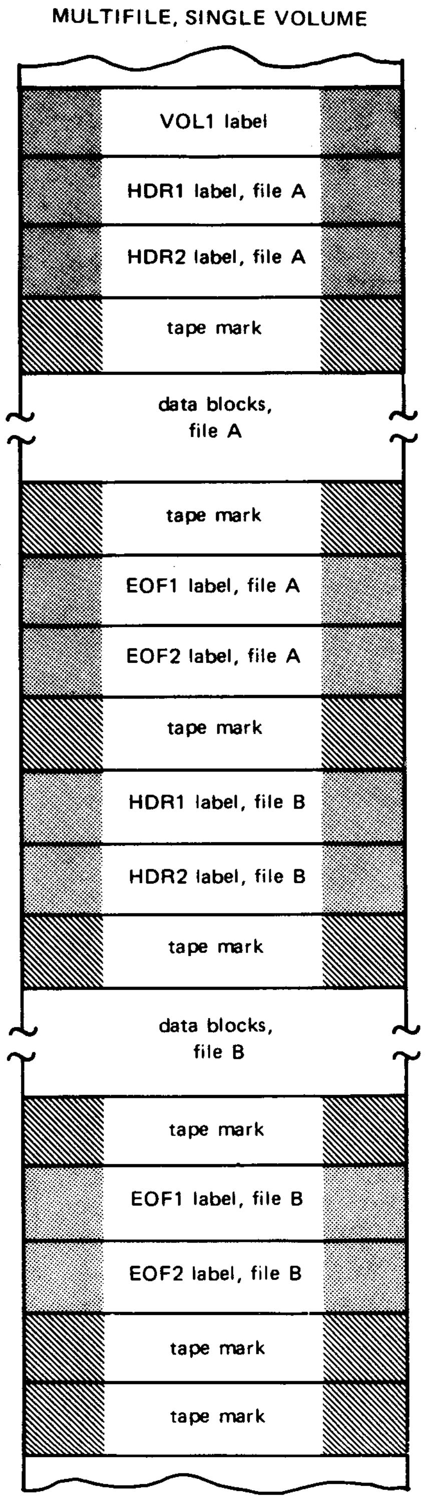
LEGEND:

☐ Content supplied by user.

▨ These bytes are required and generated by data management.

▒ These bytes are generated by data management; user supplies data for certain fields.

Figure 4-16. Organization of ASCII Single File, Single Volume, and Multivolume Sets



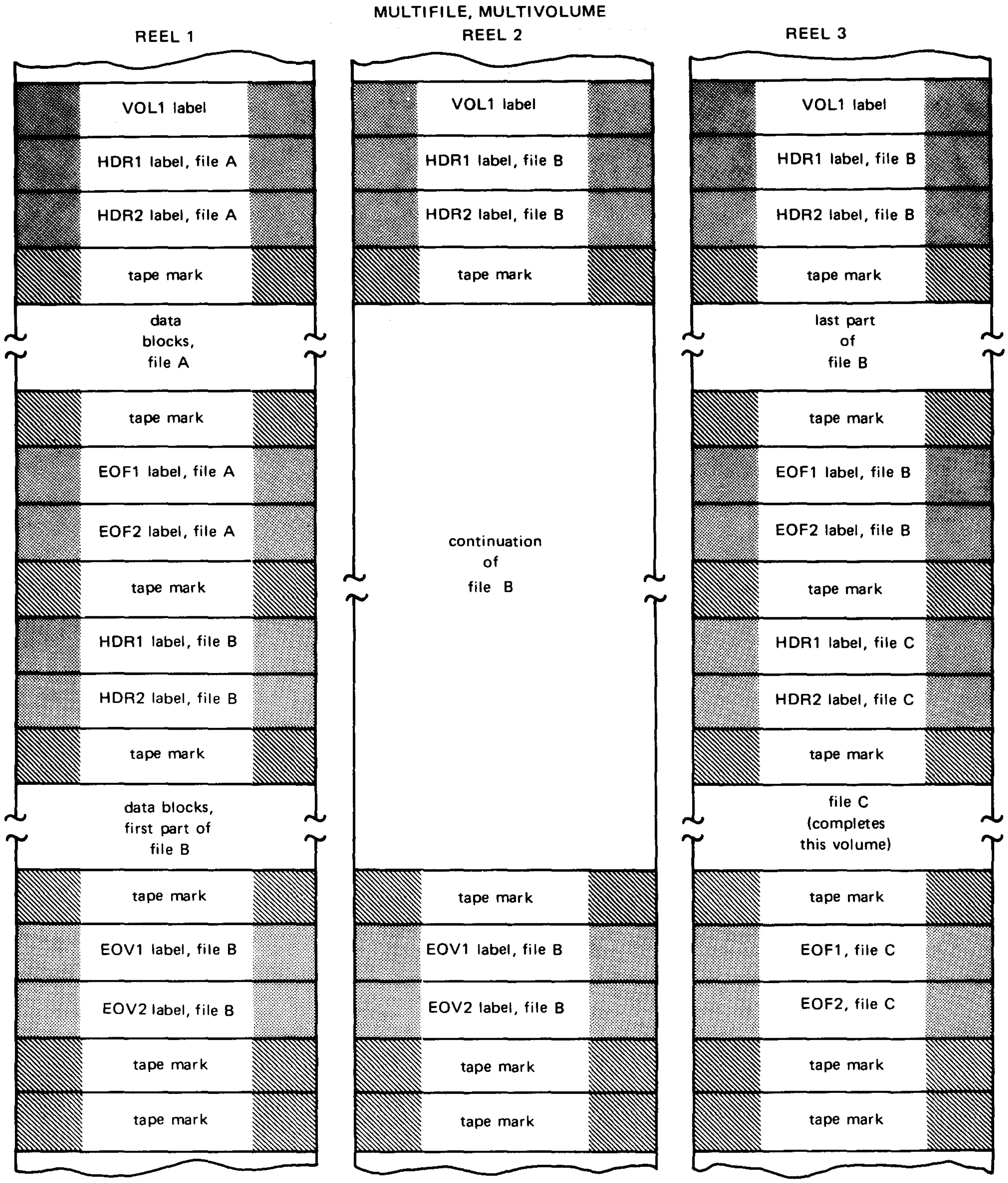
LEGEND:

☐ Content supplied by user.

▨ These bytes are required and generated by data management.

▩ These bytes are generated by data management; user supplies data for certain fields.

Figure 4-17. Volume Organization, ASCII Multifile, Single Volume Set



LEGEND:

☐ Content supplied by user.

▨ These bytes are required and generated by data management.

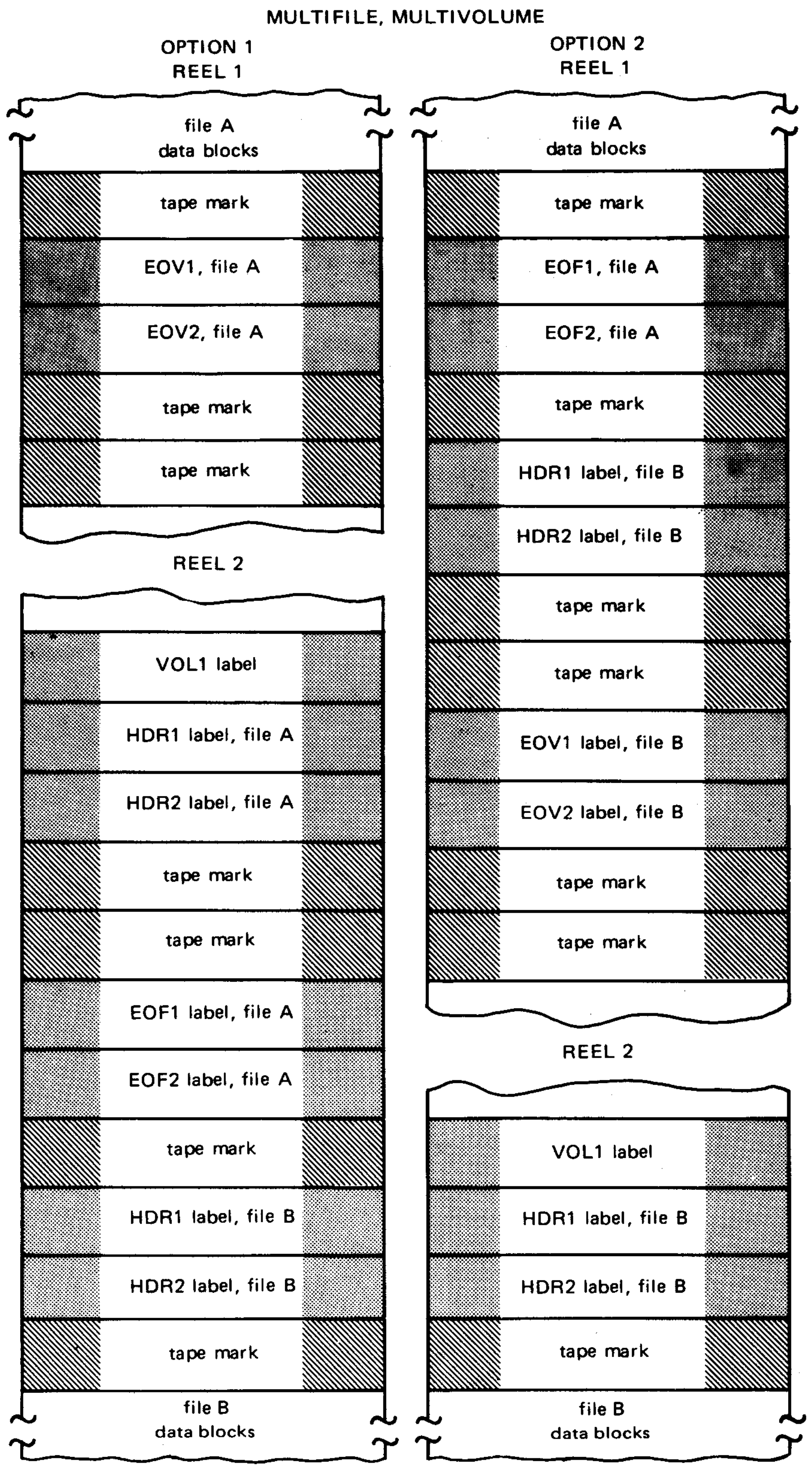
▩ These bytes are generated by data management; user supplies data for certain fields.

Figure 4-18. Volume Organization, ASCII Multifile, Multivolume Set

4.9.1.4.1. ASCII End-of-File and End-of-Volume Coincidence

The *American National Standard X3.27-1969,* provides that whenever a volume ends within a file, the last block of the file is followed by an end-of-volume label (EOV1). It also allows a second end-of-volume label (EOV2) which is standard in data management (an EOV1 label is always followed by an EOV2 label). A single tape mark precedes and two tape marks follow the EOV1 and EOV2 labels.

The standard also states that no file set may be terminated by end-of-volume labels; consequently, provision is made for those cases where the end-of-volume and end-of-file coincide. In these situations the standard provides that the labeling configuration shall be one of the two options shown Figure 4–10. Option 1 occurs when the end-of-tape warning mark is reached while the last block of a file is being written. Option 2 occurs when the end-of-tape warning mark is reached after the EOF1 and EOF2 label group has been started.



LEGEND:

□	Content supplied by user.
▨	These bytes are required and generated by data management.
▒	These bytes are generated by data management; user supplies data for certain fields.

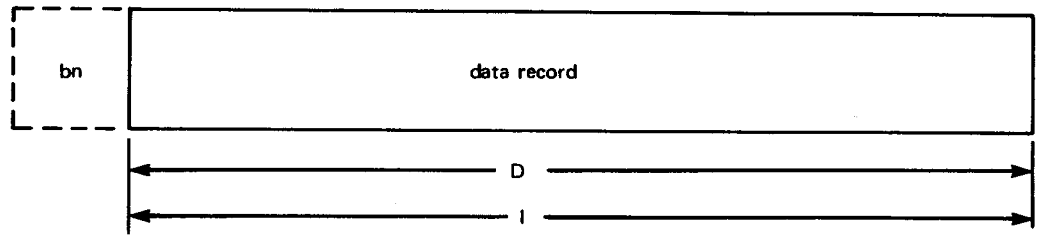
Figure 4-19. Label Configuration Options, ASCII Multifile, Multivolume Set (when End-of-Volume and End-of-File Coincide)

4.9.1.5. Magnetic Tape File Record Formats

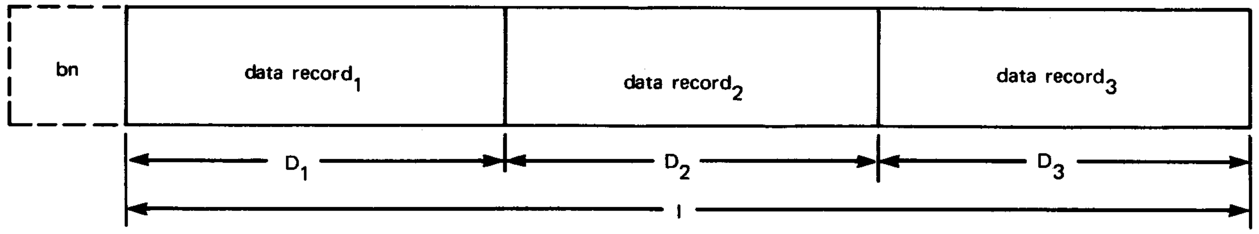
The data records on magnetic tape files may be fixed length, blocked or unblocked; variable length, blocked or unblocked; or undefined. Figure 4-20 shows these formats as they appear on EBCDIC and ASCII magnetic tape files. Note that the formats illustrated in Figure 4-20 do not show the optional use of padding because it is not supported. If your input blocks have been padded, the I/O area in your program must be large enough to accommodate this padding and your program should take care of detecting and removing the padding characters before it processes the data.

EBCDIC

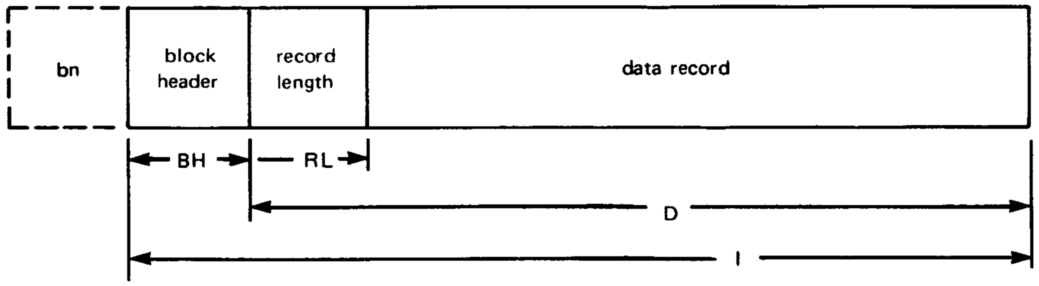
FIXED-LENGTH, UNBLOCKED RECORD



FIXED-LENGTH, BLOCKED RECORDS



VARIABLE-LENGTH, UNBLOCKED RECORD



VARIABLE-LENGTH, BLOCKED RECORDS

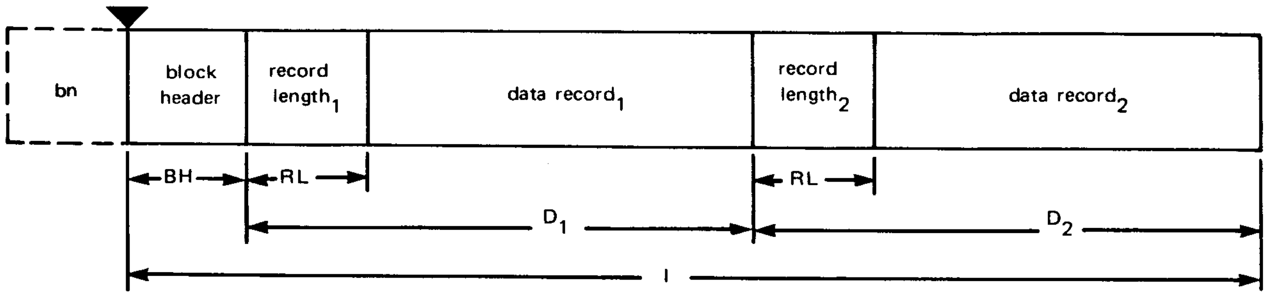
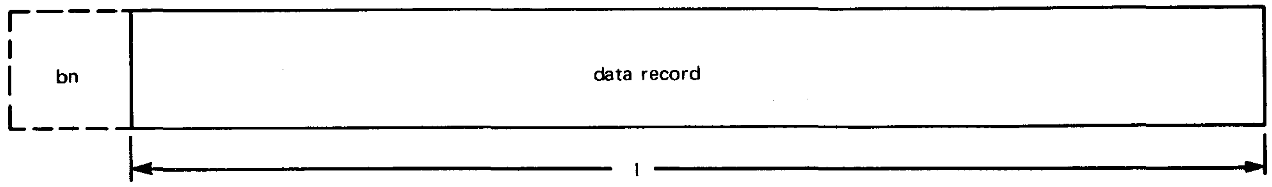


Figure 4-20. Record and Block Formats for Magnetic Tape Files (ASCII and EBCDIC) (Part 1 of 3)

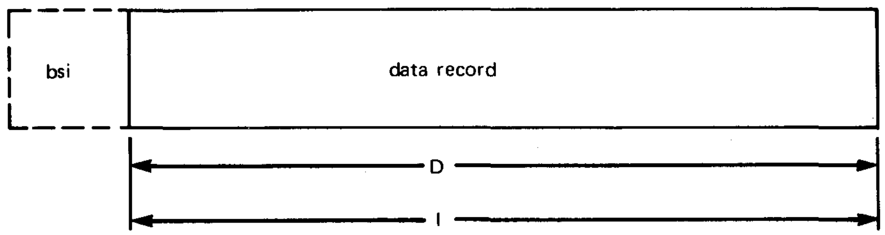
EBCDIC (cont)

UNDEFINED RECORD FORMAT

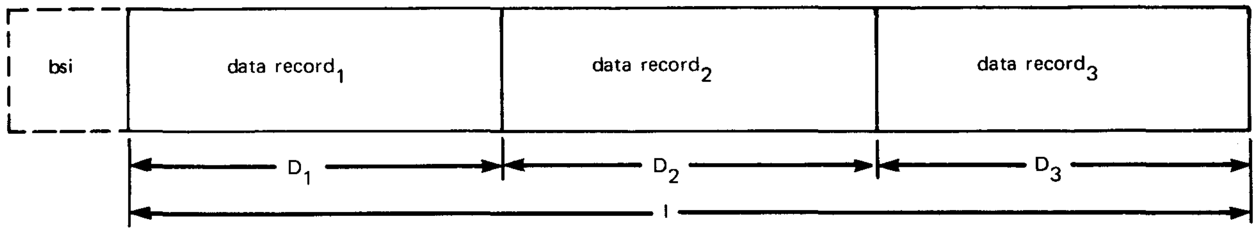


ASCII

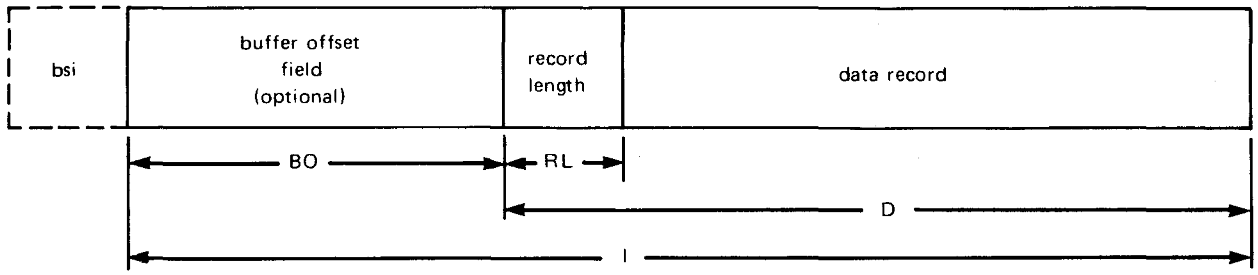
FIXED LENGTH, UNBLOCKED RECORD (FORMAT F)



FIXED-LENGTH, BLOCKED RECORDS (FORMAT F)



VARIABLE-LENGTH, UNBLOCKED RECORD (FORMAT D)



VARIABLE-LENGTH, BLOCKED RECORDS (FORMAT D)

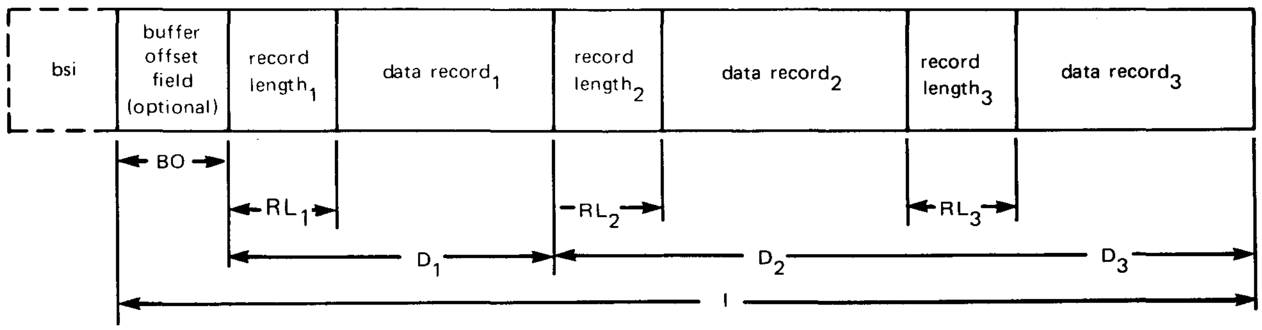
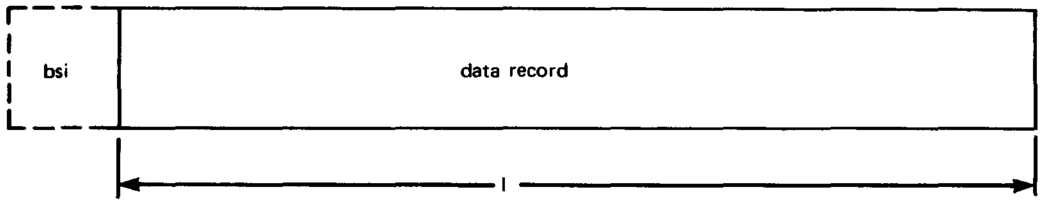


Figure 4-20. Record and Block Formats for Magnetic Tape Files (ASCII and EBCDIC) (Part 2 of 3)

ASCII (cont)

UNDEFINED RECORD FORMAT (FORMAT U)



LEGEND:

D Record length, measured in bytes. This measure is entered in the most significant two bytes of the 4-byte record length field; the two least significant bytes are reserved.

l Block length, measured in bytes. Minimum block length is 18 bytes. This measure is entered in the most significant two bytes of the 4-byte block header of EBCDIC variable-length records (block or unblocked); the two least significant bytes are reserved. When the buffer offset field of ASCII variable records is a 4-byte field, for output, data management writes the block length in it in ASCII. For input, data management assumes that it contains the length of the block in four bytes of ASCII.

RL Record length field of variable-length records; a 4-byte field in ASCII and EBCDIC records. Its own length is included in the measure inserted here. In EBCDIC records, record length is read and written in binary; in ASCII records, it is recorded on tape in ASCII, although you present it to data management in binary and process it in binary.

BH Block header, a 4-byte field at the head of the block format in which all EBCDIC variable-length records, blocked or unblocked, appear on magnetic tape. Most significant two bytes contain the length of the block, which includes the length of the header itself; the two least significant bytes are reserved.

bn Optional 3-byte block number in EBCDIC. May not be created for output files. Data management accepts the block number in EBCDIC input files, but does not process it.

BO Buffer offset field, an optional block prefix that may be placed at the head of each block of ASCII variable records. Its content is recorded in ASCII; its length ranges from 0 to 99 bytes for fixed and undefined records and is 0 or 4 for variable records. For variable records, when its length is four bytes, data management assumes that this field contains the length of the block (which includes the length of this field itself).

bsi Optional 1-byte block sequence indicator for ASCII files in ASCII numeric code. May not be created for output files. Data management accepts the block sequence indicator in input files, but does not process it.

▼ The index register specified by the IOREG keyword parameter points here, to the first byte of the record length field of variable-length records.

NOTES:

1. Although the *American National Standard X3.27-1969* also provides for a variable ASCII record with its record length specified in binary (the so-called "V-format" record), data management does not support this format.

2. Spanned records (those extending beyond one block) are neither allowed in ASCII magnetic tape files (in which there must be an integral number of records per block) nor supported by data management in EBCDIC tape files.

Figure 4-20. Record and Block Formats for Magnetic Tape Files (ASCII and EBCDIC) (Part 3 of 3)

4.10. DISK AND FORMAT LABEL DISKETTE FORMATS AND FILE CONVENTIONS

Whenever "disk file" is used in the following discussion, it refers to both disk and format label diskette files. A format label diskette file is treated exactly the same as any other disk file.

Disk files consist of data records that are recorded on one or more volumes (disk packs). These files differ from other types of files in that the data records can be retrieved not only sequentially but also randomly by relative record number (the position of a record in the file relative to the beginning of the file) or by the record key (a character string specified within each record to uniquely identify that record). The data records are retrieved or written on the disk volume via a disk subsystem. Refer to Appendix B for the functional characteristics of the disk subsystems that are supported.

Usually, there is more than one file on a disk volume. In order to keep track of where these files are located, each volume contains a volume table of contents (VTOC). Also, each file contains system standard labels that identify and delineate it.

4.10.1. How Disk Files are Organized

Disk files fall into two general categories: nonindexed and indexed.

A nonindexed file is organized consecutively. Its records are written on the disk in the order in which they are presented. The records are processed consecutively in the same order as they appear on the disk.

A nonindexed file can also be one that is organized relatively; each record in the file is written on the disk in a specific position relative to the beginning of the file. This allows any record in the file to be retrieved directly without processing any preceding records when the location (relative record number) of the record is specified.

An indexed file contains data records and an index of the record keys. The data records appear on the disk in the order in which you submitted them and the index is arranged in ascending key order. The records can be processed sequentially or randomly by record key. If the records are processed sequentially, the processing commences with the record that has the lowest key value. For random retrieval, you need only specify the key of the record you want retrieved.

4.10.2. Disk Access Method – MIRAM

Consolidated data management uses one access method for all disk files. This access method, called MIRAM (Multiple Indexed Random Access Method), provides the functions that were previously provided by separate access methods.

MIRAM has a number of features and concepts that distinguish it from other disk access methods.

- The data record slots in MIRAM files, for either fixed- or variable-length records, are of uniform size and may span physical blocks, sectors, tracks, and cylinders as required. They may even extend from one volume to another (unless the file was created for processing only a single volume on line at a time).
- Data records are written on disk compactly as a continuous string of bytes.
- The string of data records can always be accessed sequentially (consecutively) or by relative record number. In addition, the data can be indexed by up to five keys; this causes MIRAM to build a suitable index structure for each key type in a partition that is separate from the data.
- An indexed MIRAM file can be accessed by the additional random-by-key or sequential-by-key modes using a given key of reference, which can be changed.
- Indexed MIRAM files, either multivolume or single volume, may be created by means of an orderly load (records submitted in ascending key order) or a disorderly load (records submitted in no particular key order) and they may be extended by appending records in either manner. MIRAM does not sort the index at the completion of a disorderly load, but maintains the index current on a record-by-record basis.
- Duplicate keys are permitted.
- A new record is immediately available for retrieval whether it has been added to an indexed or nonindexed file.
- Multivolume MIRAM files may be created for processing with either one volume on line at a time or with all volumes on line. They must be processed in the same manner as they were created.
- All programs that access a MIRAM file need not use the same data buffer size for input/output as was used to create the file. Those that access an indexed MIRAM file, however, must use the same index buffer size.
- MIRAM allows you to logically delete records in your files; that is, it allows you to mark records so that in subsequent processing they will be ignored.
- The restrictions are:
  - the maximum key length is 80 bytes;
  - no byte of a record key may contain the hexadecimal value 'FF'; and
  - the minimum size for the index and data buffer is 256 bytes.

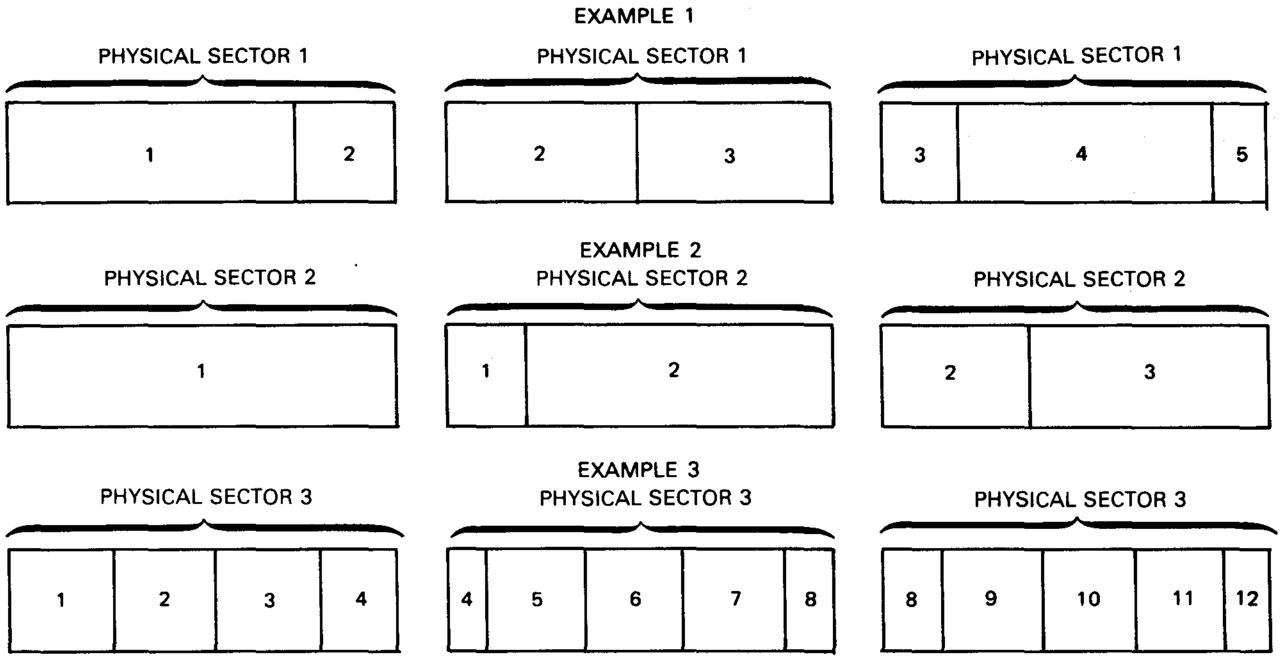
4.10.3. MIRAM File Organization

All MIRAM files contain two partitions: the data partition which contains the data records and the index partition which contains an index for each of the keys in your records. If the file is a nonindexed file, the index partition is not used; that is, no entries are placed in it and no space is allocated to it. If the file is indexed, entries are placed in the index partition and space is allocated to it. For indexed files, the data partition precedes the index partition, which begins on a separate cylinder.

4.10.3.1. Data Partition

The data partition is arranged in the same way for both nonindexed and indexed files. It is cylinder-aligned and consists of a single compact string of data records that may be keyed or unkeyed.

When data records are stored in a MIRAM file, the records are placed in uniform -size record slots and are arranged in the same order you originally presented them. These data records are stored in 256-byte physical sectors on your disk packs. Because the record slot size does not have to conform to the physical sector size, the records may span these physical boundaries as shown in Figure 4-21.



NOTES:

1. All physical sectors are 256 bytes.
2. 1, 3, 3 ... n represent record slots.
3. Record slots in Example 1 are approximately 190 bytes each.
4. Record slots in Example 2 are approximately 300 bytes each.
5. Record slots in Example 3 are approximately 70 bytes each.

Figure 4-21. Disk (MIRAM) Data Record Slots Spanning Physical Sector Boundaries

Your data records may also span track boundaries, cylinder boundaries, and volume boundaries (except when a multivolume file is created for processing with only one volume on line at any one time). When new records are added to a file, they are appended to the existing data record string; that is, they are added at the end as a continuation of the original string. The formats of the disk data records are shown in Figure 4-22.

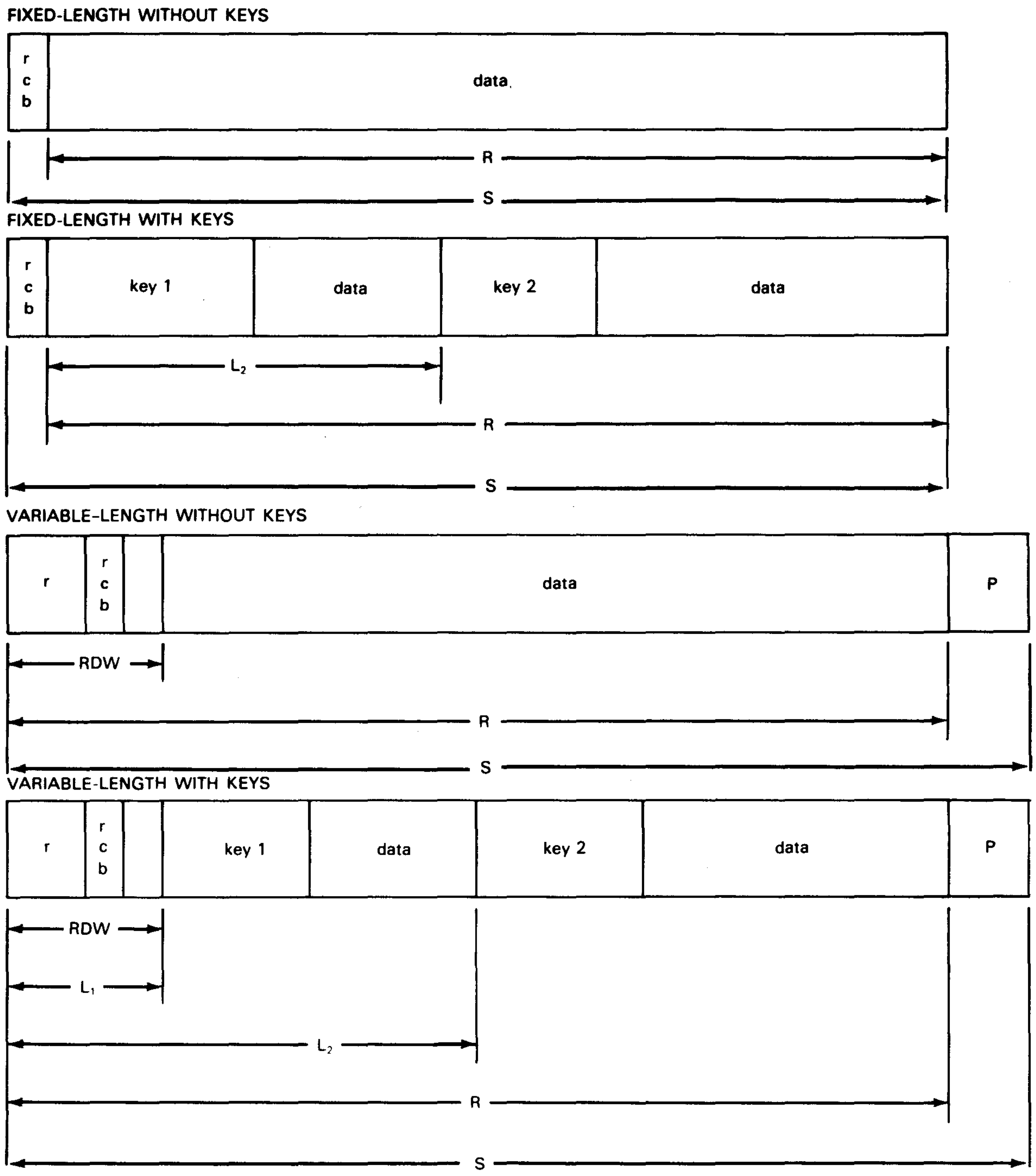


Figure 4-22. Disk (MIRAM) Data Record Formats (Part 1 of 2)

LEGEND:

rcb	=	Record control byte (optional). Used to indicate that a record has been logically deleted from the file. For MIRAM fixed-length records, this byte is placed at the beginning of each record. For variable-length records, the third byte of the record descriptor word (RDW) is used as the rcb.
R	=	Length of the logical record (RDW plus keys plus data). You specify this length as the number of bytes. For variable-length records, this value, expressed in binary, must be placed in the first two bytes of the RDW.
RDW	=	4-byte record descriptor word for variable-length records. The first two bytes contain the logical record length (r) expressed in binary; the third byte may be used as the rcb; the fourth byte is not used.
L n	=	The starting location of record key n (n = 1 through 5) of a MIRAM file data record when the key does not start in the first byte of the record. L n represents the number of bytes (RDW plus data) that precede key n. The starting location of key n must be the same in each record. Key n must have the same length in each record (a minimum of 1 byte and a maximum of 80), and no byte may contain the hexadecimal value 'FF'.
S	=	Slot size. All records are written into fixed-size slots. Slot size equals the record size + 1 for fixed-length records with a record control byte; otherwise, slot size equals the record size.
P	=	Padding.

Figure 4-22. Disk (MIRAM) Data Record Formats (Part 2 of 2)

4.10.3.2. MIRAM Index Structure

As you know, you can specify up to a maximum of five keys for a file. For each key that you specify, MIRAM builds a separate index structure. In those files where you have more than one key, these separate index structures allow you to use any of the key types as the key of reference to access your data records when you subsequently use the file in a program.

When MIRAM builds an index structure for your file, it creates a minimum of two levels of index: a fine-level index and a coarse-level index. If your file is very large, one or more mid-level indexes are created as needed.

The fine-level index consists of one entry for every record in the data partition of your file. The fine-level entries are filed in ascending key order until an index block (256 bytes) is filled. At this time, one coarse-level entry is made that contains the high key entry of that filled block of the fine-level index.

As each fine-level index block is filled, another coarse-level entry is made. This process continues until all your records are on file.

The coarse-level index is automatically allocated by MIRAM. If the coarse-level index is filled before all your records are on file, a mid-level index is created.

4.10.3.3. Entries in the Index Partition

If you have keyed records, entries are placed in the index partition as these records are loaded into the data partition. MIRAM extracts all the keys from each record (a maximum of five keys is permitted) and constructs a 3-byte pointer for each of the keys from the file relative record number of the position the record was written to. From these it forms an index entry for each of the keys in the record and stores them in the index partition.

The index entry for each key consists of the key plus three bytes (equal to the specified key length plus three bytes) and is stored in an area of the index partition, which is called a fine-level index. If you have three keys in each record, the index entry for each key is stored in a separate fine-level index; that is, the entry for key 1 is stored in the fine-level index for key 1, the entry for key 2 is stored in the fine-level index for key 2, the entry for key 3 is stored in the fine-level index for key 3, and so on.

A fine-level index is not formatted for hardware search, unlike the other levels of index that are described subsequently. It is treated as a chain of multisector blocks where each sector is 256 bytes long. All entries in a fine-level index are maintained in ascending key order. Figure 4-23 shows a typical fine-level index block of three sectors.

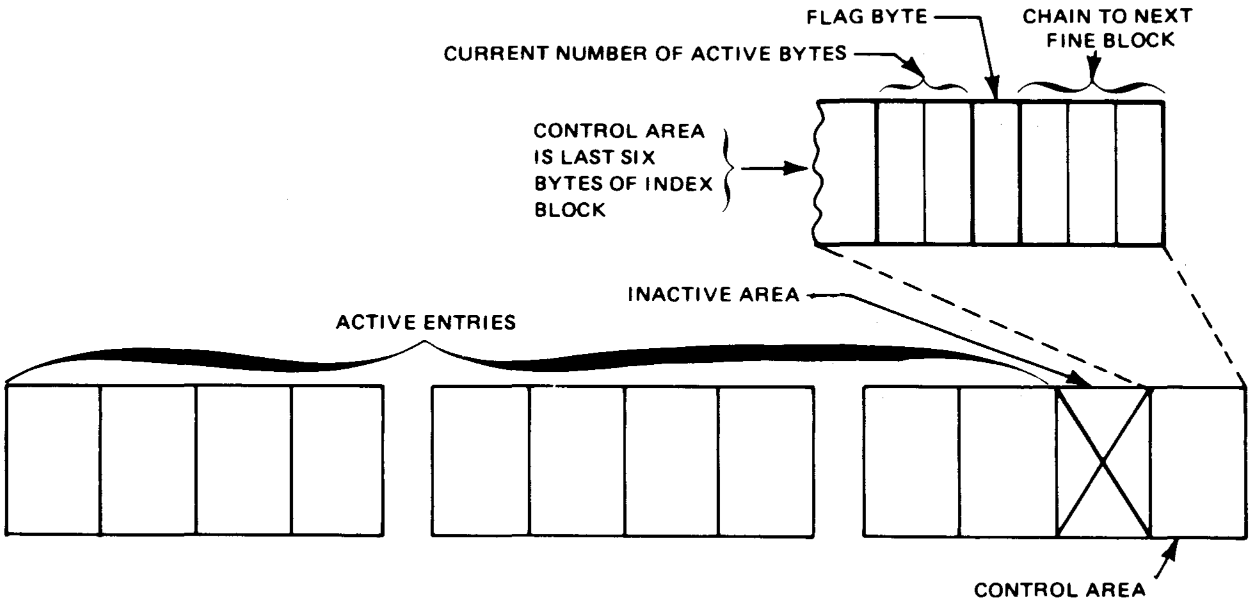

FLAG BYTE
CHAIN TO NEXT FINE BLOCK
CURRENT NUMBER OF ACTIVE BYTES
CONTROL AREA IS LAST SIX BYTES OF INDEX BLOCK
INACTIVE AREA
ACTIVE ENTRIES
CONTROL AREA

Figure 4-23. Fine-Level Index Block

When a fine-level index is created, another hierarchical level of index is always created – the coarse-level index. This is hardware searchable and is composed of 256-byte blocks that contain entries similar to those in the fine-level index. They differ, however, in that the 3-byte pointer in each coarse-level entry does not represent the file-relative number of a record in the data partition. Instead, it points to another index block at a lower level – either a fine-level block or a block in what is called a mid-level index. Another difference is that instead of having a 6-byte control area, each coarse-level block uses its final byte to indicate the number of active entries. The high key of the block is the first one encountered by the hardware search. Both the coarse-level and mid-level index blocks have the same format (Figure 4–24).

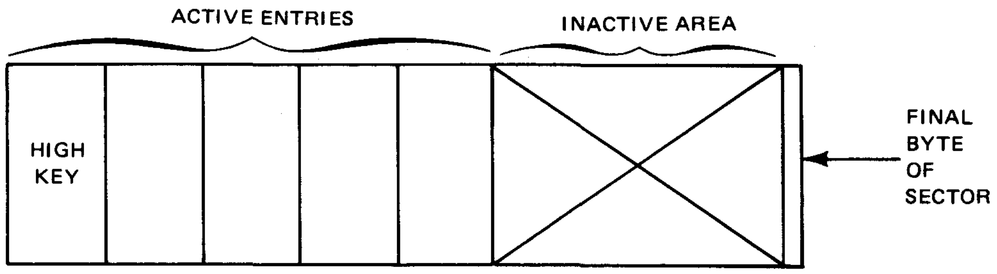


Figure 4–24. Coarse- or Mid-Level Index Block

4.10.4. Disk File Sharing

A data management file is a collection of related records stored on an external medium. If that medium is a disk storage device, then the individual records in the file are directly accessible. Any given reference to the file is independent of a prior reference to the file. This capability gives disk files the potential of being shared between programs. References to the file (from different programs) may be independent of one another, but they are dealing with a common set of records.

If multiple programs are sharing a file and at least one of the programs is writing (adding, updating, or deleting) to the file, then this may affect the other programs that are sharing the file. It is possible for one program to read a record and take an action based on the contents of that record, and then have another program update or delete that same record.

All programs that use a particular file are potential candidates to use the file at the same time (share the file), but this should only be done if the particular applications are suited to such an environment.

A determination must be made for each candidate program as to what its "share requirements" are. The share requirements reflect how a program intends to use the file (read use only or read/write use) and how other programs can concurrently use the file.

4.11. DATA SET LABEL DISKETTE FORMATS AND FILE CONVENTIONS

Data set label diskette files consist of data records that are recorded on one or more volumes (diskettes). The data records can be retrieved sequentially or randomly by relative record number (the position of a record in the file relative to the beginning of the file). The data records are recorded and retrieved via a diskette subsystem. Refer to Appendix B for the functional characteristics of the diskette subsystems that are supported.

4.11.1. Volume Organization

Data set label files are recorded on one or both sides of the diskette, depending on the diskette type used. The diskette type also determines the size of the fixed-length sectors on the diskette volume, the maximum number of files that the volume can contain, and the maximum number of data bytes the volume can contain. The effect of the diskette type is shown in Table 4-1.

Table 4-1. Data Set Label Diskette Characteristics

Diskette Type	Physical Sector Size in Bytes	Sectors per Track	Maximum Number of Sectors per Diskette Volume	Maximum Number of Data Bytes per Diskette Volume	Maximum Number of Files per Diskette Volume
Single Sided, Single Density*	128*	26*	1898* or 1924	242,944* or 246,272	19
	256	15	1110	284,160	19
	512	8	592	303,104	19
Single Sided, Double Density	256	26	1924	492,544	19
	512	15	1110	568,320	19
Double Sided, Single Density	128	26	3848	492,544	45
	256	15	2220	568,320	45
	512	8	1184	606,208	45
Double Sided, Double Density	256	26	3848	985,088	45
	512	15	2220	1,136,640	45

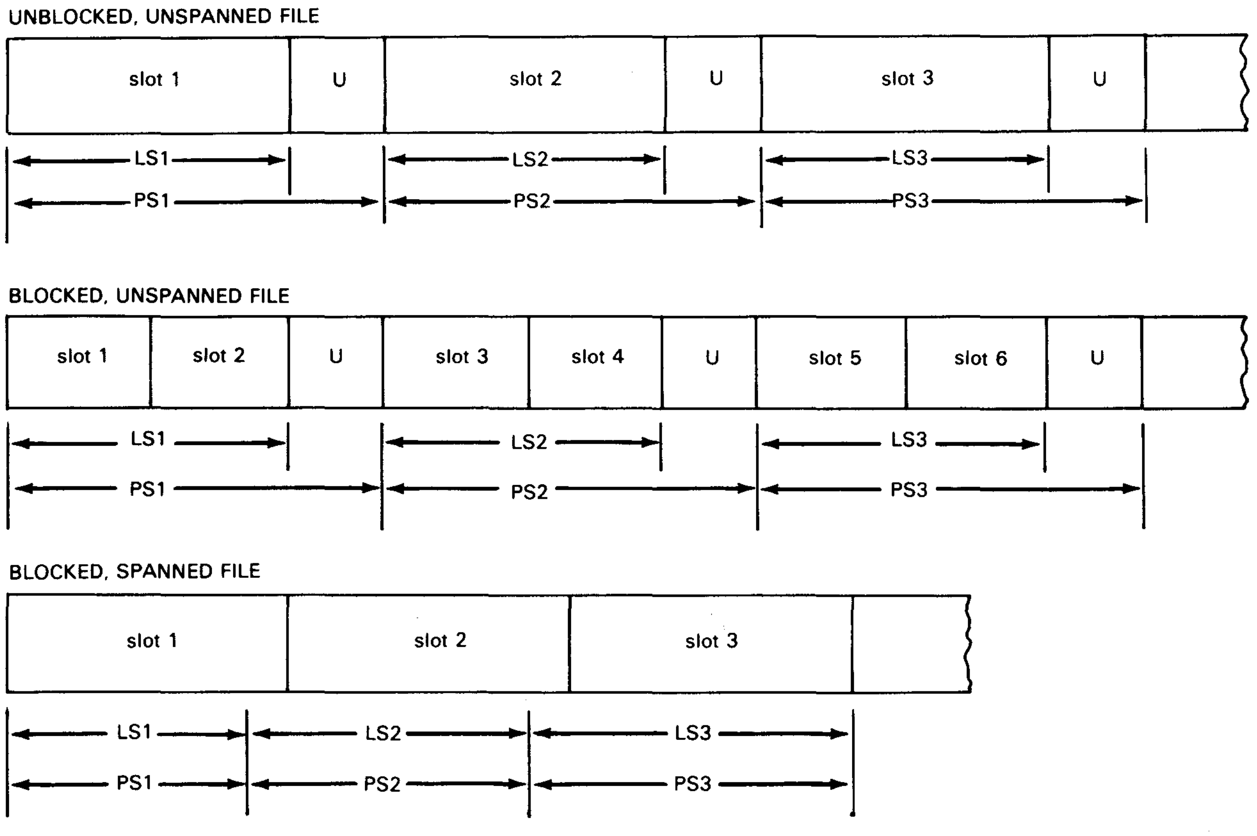
* Applies to files written in basic data exchange (BDE) mode – IBM System/3 and SPERRY UNIVAC 90/30 compatibility. The number of sectors available for BDE files is reduced to have compatibility between systems. Only tracks 1–73 are used in this mode. This is the only configuration available on the 8413 diskette subsystem. BDE has a logical record size ≤ 128, fixed-length unblocked-unspanned records, and a file name of eight characters or less. Tracks 1–74 are used for all other modes.

Regardless of the recording mode used, the information on a data set label diskette is organized into two areas: the index track (track 0) on which data management writes the file labels, and tracks 1-74 where the data records for the file are written.

Data set label files may be either single volume or multivolume files. In the latter case, the file can only be processed with one volume online at a time.

4.11.2. File and Record Organization

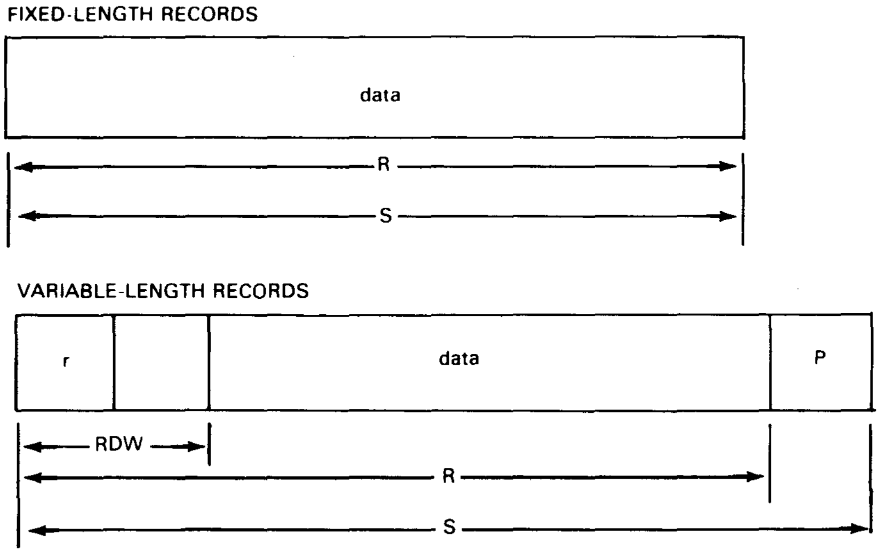
The file layout and the record formats for data set label diskette files are shown in Figures 4-25 and 4-26, respectively.



LEGEND:

LS	=	Logical sector
PS	=	Physical sector
U	=	Unused portion of physical sector (PS-LS=U)

Figure 4-25. Data Set Label Diskette File Layout



LEGEND:

R	=	Length of the logical record; record descriptor word (RDW) plus data. For variable-length records, this value, expressed in binary, must be placed in the first two bytes of the RDW.
RDW	=	4-byte record descriptor word for variable-length records. The first two bytes contain the logical record length (r) expressed in binary.
S	=	Slot size. All records are written into fixed-size slots.
P	=	Padding.

Figure 4-26. Data Set Label Diskette Record Formats

4.12. WORKSTATION FORMTS AND FILE CONVENTIONS

A workstation is an input/output device that contains a keyboard and a video screen. Workstation input files consist of data records that you type in via the keyboard and output files consist of data records, created by your program, that are displayed on the video screen. Refer to Appendix B for the functional characteristics of the workstation subsystems that are supported.

Workstation files can be either single volume or multivolume files. If a file is a single volume file, this means that one workstation (volume) is assigned to that file. If it is a multivolume file, this means that more than one workstation is assigned to that file.

4.12.1. File Organization

Workstation files differ from card, tape, printer, disk, and diskette files in that data cannot be permanently stored on them. This is true because a workstation data record exists on the input or output file only as long as it appears on the screen. Once the screen is cleared, the record is gone; that is, it ceases to exist physically. As you can readily see, a workstation file is a sequential file; that is, you present your input one record at a time and your output is displayed one record at a time.

4.12.2. Record Formats

Workstation records consist of alphabetic, numeric, or alphanumeric data. This data must consist of displayable characters. If you include any device control characters (hexadecimal equivalent 00 through 3F) this may cause hardware errors. A record can range from one character in length to the full extent of the screen. For example, if each line on a screen can contain 80 characters and there are 24 lines on a screen, the maximum record size would be 1920 characters.

4.12.3. Additional Workstation Programming Aids

If you want to use the full workstation screen, OS/3 provides several program products that let you easily create and manage entire screens. These include:

- Screen format services
- Dialog processing
- Menu services

To find out more about these products, see Section 7.

PART 3. PROGRAMMING FACILITIES

5. Job and Program Preparation

5.1. GENERAL

In Section 1.2, we discuss the concept of job processing. You should know that a job is a unit of work for the system to perform. In this section we discuss the preparation of jobs. You are introduced to the various job preparation facilities including the job control language, the job control dialog, and the interactive job control commands. In addition, the program preparation procedure is outlined from source code generation to final execution.

The facilities used to prepare a program, with the exception of the source languages discussed in a separate section, include the general editor, the RPG II editor, and linkage editor. Also discussed are the function and use of the system program libraries, as well as generating and using your own program libraries.

The first step when preparing a program for execution is to write the source code. You can write your source program on coding forms and have it keypunched, or you can enter it interactively from the workstation.

If you are writing an RPG II program, you can use the RPG II editor. If you are writing a COBOL, FORTRAN IV, or BAL program, then use the general editor. The general editor stores your source module in either your own program library or the systems source program library, $Y$SRC.

After generating your source program, you must have it compiled or assembled by the appropriate language processor. The language processors translate your source program into a nonexecutable object module. You can direct the processor to store the object module in the system object module library, $Y$OBJ, or in your own program library.

Once you have successfully compiled or assembled your program and have an object module stored in a library, you submit that object module to the linkage editor.

The linkage editor converts the object module into an executable load module. You can have the linkage editor place a copy of the load module into the system load module library, $Y$LOD, or in your own program library. Figure 5-1 presents a flowchart of the program preparation procedure.

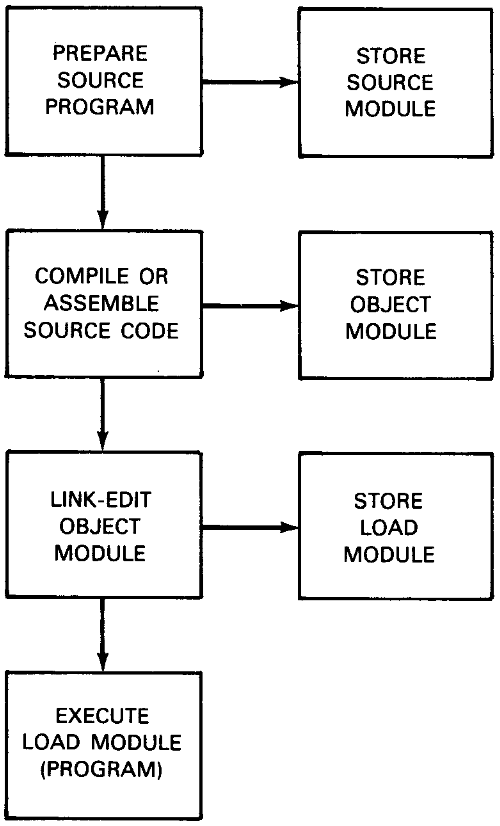


Figure 5—1. Program Preparation Flowchart

Each of the steps shown in Figure 5-1 can be performed as separate jobs or they can be combined into a single job. In either case, you must write a job control stream using the job control language. The job control stream tells the system what your job is going to do and what resources it requires.

5.2. JOB CONTROL LANGUAGE

To submit a job to the system for processing, you must prepare a job control stream using the job control language. The job control stream identifies the job to the system, tells the system what the job is going to do, and what resources are required. This section briefly describes the function and usage of the job control language. More detailed information can be found in the current version of the job control user guide, UP-8065.

The job control language consists of job control statements and job control procedures (jprocs). Job control statements are instructions to the operating system that tell it how to process your job. A job control procedure is a predefined series of job control statements that perform a specific function.

The jproc is called by a single job control statement. For example, the jproc for the COBOL compiler generates the job control statements that identify the files and devices needed and executes the processor to compile your COBOL source program. Jprocs are available for a number of command functions and use of the jprocs can significantly reduce your job control coding. In addition, you can generate your own jprocs to meet your particular job control requirements.

The OS/3 job control language offers an extensive repertoire of control statements and jprocs providing a wide range of job processing features. However, the discussions that follow focus on only those most commonly used and the job control statements required for every job.

5.2.1. Defining Your Job

The first job control statement that must be present in your job control stream is the JOB control statement. You use this statement to identify your job to the system by specifying a unique jobname. A typical JOB control statement looks like this:

```
// JOB PAYROLL
```

In addition to specifying the name of your job, you can also enter other information on the JOB statement. You can indicate the execution priority level of your job. The priority levels available are: preemptive, high, and normal.

If you do not specify a priority level, your job is scheduled and executed as a normal priority job. Minimum and maximum main storage requirements can also be entered on the JOB statement. Normally, you can allow the system to automatically determine your main storage requirement. However, there are instances when the system is unable to determine accurately how much main storage your job requires, such as if your job executes a load module that does not reside in the system load module library or executes a program designed to conserve main storage. In such cases, the system could allocate more or less main storage than your job requires.

After you identify your job through the JOB control statement, you must tell the system what resources your job requires. Each device (printer, workstation, etc), data file, and private program library needed by your job must be identified and assigned.

To assign a resource to your job you use a specific group of job control statements known as the device assignment set. The job control statements used to generate a device assignment set are: // DVC, // VOL, // EXT, // LBL, and // LFD.

Figure 5-2 shows the relationship of the device assignment control statements to the physical device. A resource assigned to a job is assigned for the duration of the entire job, even if it is used by only one job step.

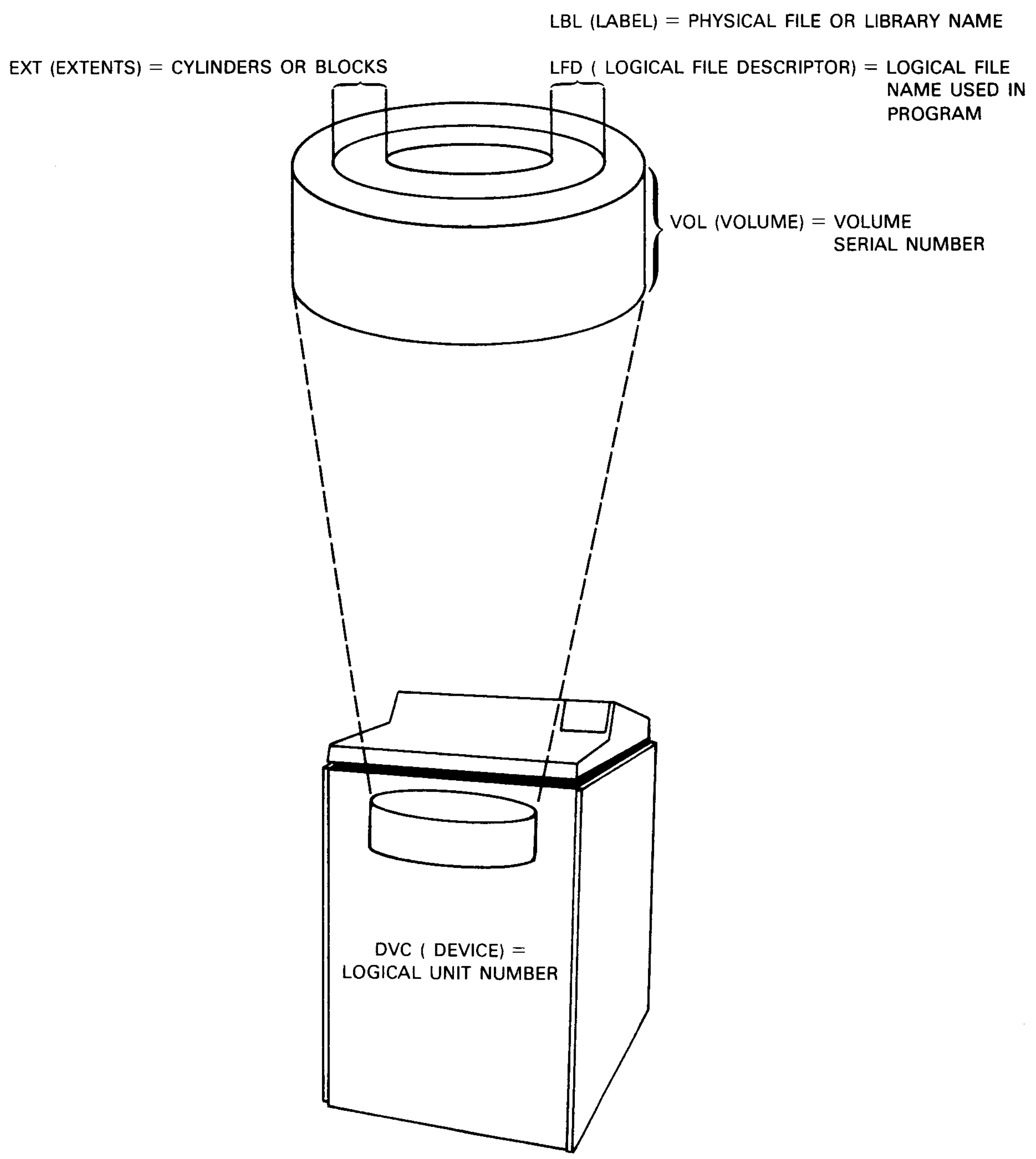

LBL (LABEL) = PHYSICAL FILE OR LIBRARY NAME
EXT (EXTENTS) = CYLINDERS OR BLOCKS
LFD (LOGICAL FILE DESCRIPTOR) = LOGICAL FILE NAME USED IN PROGRAM
VOL (VOLUME) = VOLUME SERIAL NUMBER
DVC (DEVICE) = LOGICAL UNIT NUMBER

Figure 5—2. Graphic Representation of Job Control Device Assignment Statements

You use the DVC job control statement to specify the logical unit number of a device required by your job. The logical unit number is often referred to as the device number and every device configured has a device number assigned to it. In addition, each device type has a general device number that is used when any available device of a particular type can be used. For example, if your job requires a particular printer, you would specify the device number for that printer. If, on the other hand, any available printer could be used, then you could specify the general device number for printers.

When assigning a disk file to your job, you would specify the device number of the disk drive that holds the disk containing your file. You could specify the specific device number or you could use the general device number for that disk type. If you use the general disk type number, the system locates your disk by the volume serial number you specify.

You use the VOL job control statement to specify the volume serial number of the disk, diskette, or tape volume used in your job. The volume serial numbers are assigned when you prep the devices for use. A typical VOL job control statement would be // VOL DSP028.

If you are initially allocating or expanding a disk or a diskette file, you would include the EXT job control statement in the device assignment set. A number of other parameter entries can be made on this statement including: the type of file, whether space is allocated by cylinders or tracks, if the space is to be contiguous, and information about dynamic expansion. For example, this statement:

```
// EXT MI,C,2,CYL,1Ø
```

would allocate a MIRAM file with 10 contiguous cylinders with a dynamic expansion of 2 cylinders.

Following either the VOL or EXT statements is the LBL job control statement. You use this statement to specify the file name of a disk, diskette, or tape file. This file name is often referred to as the file label and is the physical name of a data file or program library. Other parameters on this statement allow you to enter information for cataloging the file, multivolume file checking, file sequence numbering, and creation date. The file name you specify could be up to 44 alphanumeric characters in length. Thus, a typical LBL statement could be:

```
// LBL PAYFILE
```

or

```
// LBL SOURCE.PROGRAM.LIBRARY.ØØ231.COBOL.
```

The final job control device assignment statement is the LFD statement. If you are creating a device assignment set for a data file, this statement forms a logical bridge between the device assignment set and the file as it appears in your program. For example, if you have defined an input file in a COBOL program with an FD of RECRDIN, then you would specify that name on the LFD statement of the device assignment set for that file. If the device assignment set were for a program library, the name you specify on the LFD would be used wherever your job references the program library. Suppose these were the last two statements of a device assignment set for a program library:

```
// LBL COBOL.PROGRAM.LIBRARY
// LFD PROGLIB
```

A reference to the library in your job would be made using the specified LFD name. For example:

```
// EXEC PAYPROG,PROGLIB.
```

This statement causes the system to execute the load module named PAYPROG residing in the program library identified in a device assignment set as PROGLIB. In this case, it would be COBOL.PROGRAM.LIBRARY.

In addition, a number of system programs require specific LFD names for input and output files. For example, a file used as input to the sort/merge routine must be identified by an LFD name of SORTINn where n is a numeric value used for multiple input files.

Here are some typical device assignment sets for a variety of files and devices.

- To define any system printer:

```
// DVC 2Ø
// LFD PRNTR
```

- To define a data file:

```
// DVC 5Ø
// VOL DSPØ28
// LBL PAYROLL.FILE.SALARIED
// LFD PAYFIL
```

- To define a workstation:

```
// DVC 2ØØ
// LFD WRKSTN
```

5.2.2. Executing a Program

The EXEC job control statement is used to execute a program. As mentioned, a program must be compiled and then processed by the linkage editor to produce a load module before it can be executed. To execute the program, you specify the load module name on the EXEC statement. System programs are also executed by the EXEC statement. If the system program has associated control statements, they would immediately follow the EXEC statement as embedded data. Thus, to execute the librarian (which has a program name of LIBS) with control statements, you would use the following statements:

```
// EXEC LIBS
/$
 FIL DØ=FILEØ1,D1=FILEØ2
 COP DØ,S,MODØ1,D2
 LST D2,S
/*
```

The /$ and /* job control statements indicate the start and end of embedded data, respectively. Any embedded data entered in a job control stream must be bracketed with these two statements and the embedded data itself must be indented at least one column. The end of a job control stream is indicated by the /& end-of-job statement.

5.2.3. Running Your Job

You can generate your job control streams interactively by using the interactive job control dialog or the general editor, both of which are described later in this section. If you use one of these interactive features, you would store the generated control streams in the system job control stream library, $Y$JCS. To run a job stored in $Y$JCS, you issue a RUN command from the workstation or console specifying the name of the job.

To run a job control stream on cards, you place the cards in the card reader and press the run button. You could also run your control stream by placing the cards in the card reader and issuing a RUN/RV command from the console specifying the name of your job.

In addition, you can have job control streams currently on cards placed into $Y$JCS or an alternate library by using the FILE symbiont. The FILE symbiont is executed from the system console and places the control streams into the specified library without executing them.

When you initiate the running of a job, a component called the run processor reads your control stream into the system. The run processor scans your control stream to translate the job control statements and expands any jproc calls. At this point, your control stream is checked for order and syntax errors. If no errors are detected, the job is scheduled and a temporary run library is generated for the job. Generally speaking, jobs are scheduled within each priority level on a first-in, first-fit basis; preemptive jobs are scheduled first, followed by high, then normal priority jobs.

A translated job control stream normally disappears after the job is finished. But it may take a long time to translate and expand the control stream each time it's called, especially if it contains a number of jprocs. To save time, you can use the OS/3 save/restore facility.

With a // OPTION SAVE or // OPTION NOSCHED job control statement, you can save a copy of the translated stream, its jprocs expanded, in the $Y$SAVE system file. Later, you can call back (restore) the stream using the SC/SI system console or workstation command. The job runs just as if it were called by a RUN/RV command. However, by using a control stream that was previously translated, save/restore lets you bypass the translation process each time the control stream is subsequently called.

5.2.4. Program Libraries

The system includes a number of program libraries used to hold the various system programs included as part of your system and the program modules you generate during the normal course of system operation. The system program libraries are always prefixed with $Y$ and normally reside on your system resident disk pack, SYSRES. There is one system program library to hold each of the various module types. The program libraries included with your system vary depending upon the software configured in your system. However, every system includes the following five system program libraries:

- $Y$SRC – system source program library
- $Y$OBJ – system object module library
- $Y$LOD – system load module library
- $Y$JCS – system job control stream and jproc library
- $Y$MAC – system macroinstruction library

In addition to the permanent program libraries, the system maintains a temporary run library, $Y$RUN, for each job currently running. During the job scheduling operations, the $Y$RUN library is generated for a job and the job control stream is placed in it along with information relevant to the job's execution and resource requirements. In addition, $Y$RUN passes information from job step to job step and holds copies of all modules used or generated by your job.

For example, if your job involved the compilation and link-editing of a COBOL program, the object modules resulting from the compilation would be placed in your job's $Y$RUN library. The linkage editor would then locate the object module in $Y$RUN and process it. The contents of a job's $Y$RUN library are lost after the job terminates, so any generated modules to be saved must be permanently stored prior to the job's termination.

You can use the system's permanent program libraries to store your own program modules or you can generate your own program libraries. You can generate your own program libraries through job control or you can do it interactively by using the general editor. If using your own program libraries, you are responsible for ensuring that the libraries have sufficient space to hold your modules. On the other hand, the system handles space allocation and extension automatically for system libraries. In addition, system program libraries do not have to be identified to a job through a device assignment set. If using your own program library in a job, that library must be identified through a device assignment set.

The system provides utilities for maintaining program libraries. The utilities are the SAT librarian, used to maintain program libraries organized as system access technique (SAT) files and the MIRAM librarian used to maintain libraries organized as MIRAM files. A brief description of these two librarians can be found in 9.4.1 and 9.4.2. Detailed information for program libraries, including usage and format, and the librarians can be found in the system service programs user guide, UP-8841 (current version).

5.2.5. Job Control Dialog

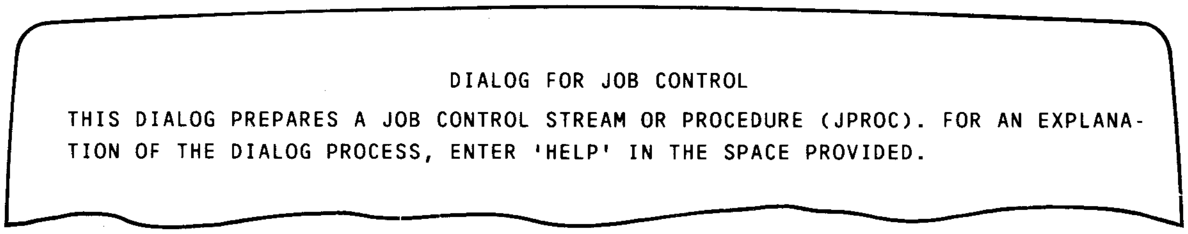
The job control dialog allows you to interactively build the control streams required to run your jobs and to store the generated control streams in the system's job control stream library or your own program library. The dialog guides you step by step through the process of building a control stream by presenting a series of queries to which you respond with the appropriate entries to build the desired control stream. You initiate the job control dialog by entering the following command at the workstation:

```
RV JC$BUILD
```

This command has a number of optional parameters used to:

- indicate audit files (discussed later in this section);
- specify the library the generated control stream is to be placed in; if not specified it is defaulted to $Y$JCS; and
- identify the printer that is to be used to produce the summary report of your dialog entries, which is defaulted to any available printer.

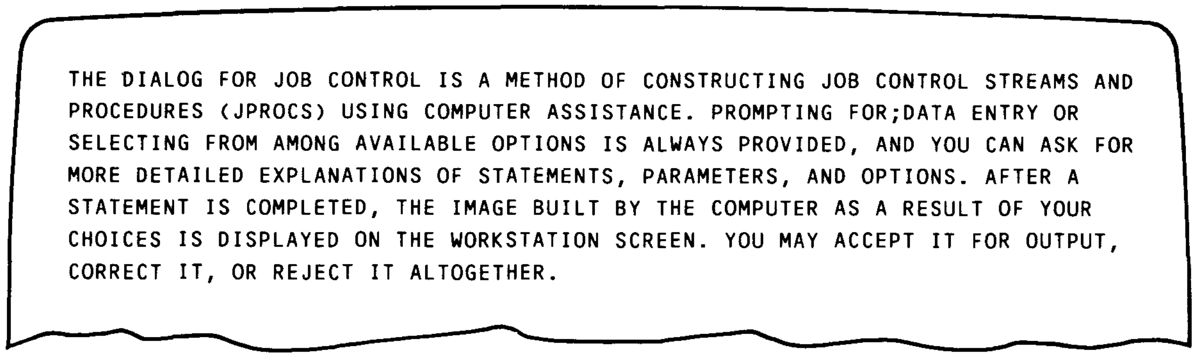
Once you have entered this command, the system responds by displaying the first screen of the job control dialog. Figure 5-3 shows the first screen.



```
                         DIALOG FOR JOB CONTROL
THIS DIALOG PREPARES A JOB CONTROL STREAM OR PROCEDURE (JPROC). FOR AN EXPLANA-
TION OF THE DIALOG PROCESS, ENTER 'HELP' IN THE SPACE PROVIDED.
```

Figure 5-3. Initial Job Control Dialog Screen Display

As you can see in Figure 5-3, the display offers a brief explanation of the job control dialog function. If you need further information, you can enter HELP as indicated. If you do so, the screen shown in Figure 5-4 is displayed.



```
THE DIALOG FOR JOB CONTROL IS A METHOD OF CONSTRUCTING JOB CONTROL STREAMS AND
PROCEDURES (JPROCS) USING COMPUTER ASSISTANCE. PROMPTING FOR;DATA ENTRY OR
SELECTING FROM AMONG AVAILABLE OPTIONS IS ALWAYS PROVIDED, AND YOU CAN ASK FOR
MORE DETAILED EXPLANATIONS OF STATEMENTS, PARAMETERS, AND OPTIONS. AFTER A
STATEMENT IS COMPLETED, THE IMAGE BUILT BY THE COMPUTER AS A RESULT OF YOUR
CHOICES IS DISPLAYED ON THE WORKSTATION SCREEN. YOU MAY ACCEPT IT FOR OUTPUT,
CORRECT IT, OR REJECT IT ALTOGETHER.
```

Figure 5-4. First Screen of System Explanation of JCL Dialog

You can enter HELP at any point in the dialog and receive an explanation of the part of job control you are at in the dialog. The HELP facility stays in effect only until you have successfully entered correct responses to the section of the dialog you were working on at the time you requested assistance. Once you have done so, the dialog returns to the standard display for the next portion of the dialog.

As mentioned earlier, the command to initiate the job control dialog has parameters that relate to the audit facility. The audit facility allows you to save your response to a dialog in an audit file, and then specify at the start of a subsequent dialog session which entries to the dialog are taken from the audit file and which you will change during the current session. You can also indicate that the new responses to the dialog, including those taken from the audit file and those you enter, are to be placed in the audit file.

Each time you use the job control dialog, you receive a printed summary that lists your entries to the dialog. This summary is useful in providing a listing of the entries and also showing what is presently in the audit file. Using the summary listing, you can determine which portions of the dialog are to be taken from that file and which require new entries. You can also use the job control dialog to build user-defined jprocs. These jprocs can be stored in your own user library or be placed in $Y$JCS by default. The HELP and audit facilities are available for building jprocs.

Embedded data can be entered through the dialog. When, in response to the dialog, you enter the /$ job control statement indicating the start of embedded data, a blank screen appears for you to begin entering data.

For more information on the interactive job control dialog, see the current version of the job control user guide.

5.2.6. Interactive Job Control and Job Processing Commands

You have available a number of workstation commands and job control statements that allow you to interactively control your job processing environment. You can place certain job control statements within your control stream to cause the job to query the workstation for variable data during job execution. In addition, there are available a number of workstation commands allowing you to interactively control the execution of your jobs.

The two interactive job control statements are OPTION QUERY and QGBL. The OPTION QUERY allows you to interactively alter job execution by skipping portions of the job control stream. The QGBL is used to change the value of prespecified global symbols used in the job control stream. When a QGBL statement is detected, a screen display requests that you enter the new value and all global symbols in the control stream are changed to that value.

The interactive job processing commands are issued from the workstation and allow you to:

- Initiate the execution of a presaved job control stream
- Stop the execution of a job, delete a job from the job queue, and stop the execution of a symbiont or transient routine
- Temporarily suspend job execution or reschedule a job
- Attach a workstation to a job
- Display the contents of a job queue and system status
- Perform job step main storage region dumps
- Generate spool file breakpoints

For more information on the interactive job control and job processing commands, see the current version of the job control user guide.

5.3. GENERAL EDITOR

The general editor, commonly known as EDT, is a user-oriented interactive program that enables you to create and update library and data files from your workstation. With EDT, you can interactively create and update your source programs, job control streams, and data files, plus update system files without using cards. In fact, you can use EDT to access any type of file.

You initiate the general editor via the workstation command EDT.

When you use EDT, it creates and maintains a temporary disk file where all the file manipulation takes place. This file is known as the workspace file and it lasts for the duration of the EDT session. When you create a file, the text is first placed in the workspace file, then written to your designated file. When you update a file, a copy of the existing file is written to the workspace file, then written to your designated file. By using this workspace file, you always have a backup copy of your file. Figure 5-5 shows how EDT operates when creating a file.

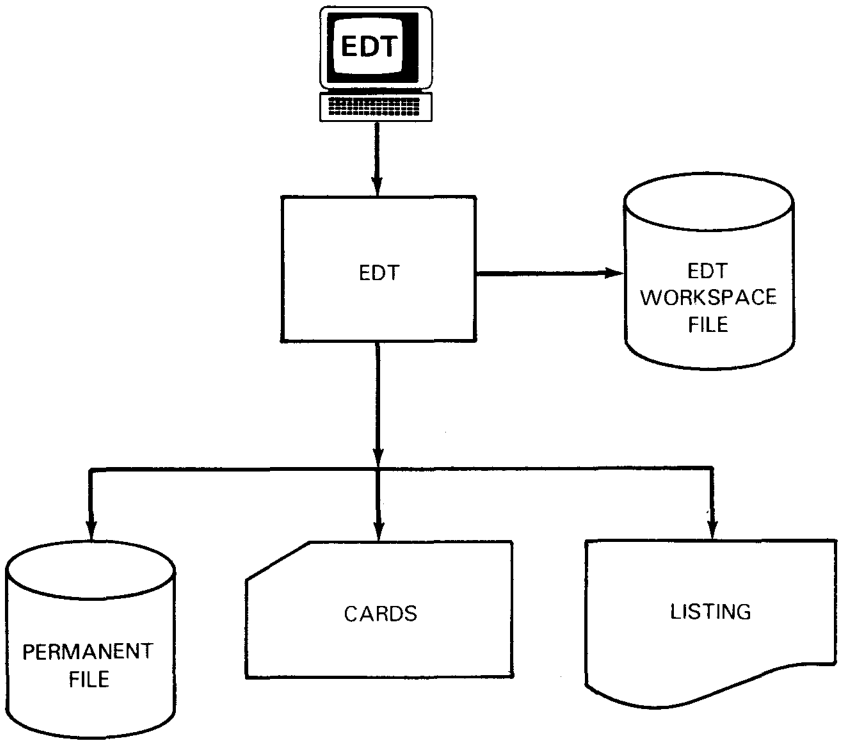
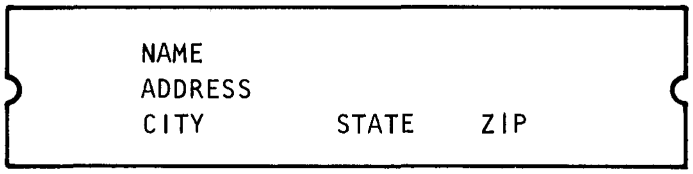


Figure 5-5. Creating a File Using the General Editor

5.3.1. Sample EDT Session

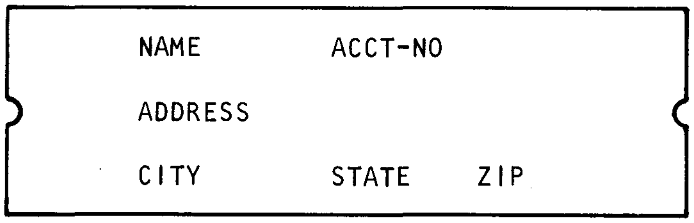
From the time you key in the workstation command EDT to the time you terminate EDT, you are engaged in what is called an EDT session. (Another way of looking at it is that you use the same workspace file throughout the entire session.) We show you a sample EDT session that should give you a general idea of what it is and how it works. What you see is the text of an existing COBOL program to be changed with EDT. Mixed in with the program text are a number of EDT commands which we use to read and write the text in the workspace file, change the text, display portions of it on the workstation, and output a copy to the system printer.

The existing program prints address labels for magazines like this:



```
NAME
ADDRESS
CITY          STATE     ZIP
```

By using EDT, we update that program to doublespace the lines and include an account number next to the name like this:



```
NAME          ACCT-NO

ADDRESS

CITY          STATE     ZIP
```

We begin this session by transferring a copy of the program LABELS into the EDT workspace file from the program library MAGAZINE on disk volume USER01. We then update it, using the available EDT facilities. The session ends when we return a copy of the workspace file (the program LABELS) to the library. The old version of the program LABELS will be replaced by the updated version.

Sample EDT Session:

Transfers our source program into the work-space file

```
 1.ØØØØ@READ MO=LABELS,FIL=MAGAZINE,VSN=USERØ1
```

Displays the contents of the work-space file (our source program) on the workstation screen

```
51.ØØØØ@PRINT &
```

Our source program displayed

```
 1.ØØØØ       IDENTIFICATION DIVISION.
 2.ØØØØ       PROGRAM-ID.  LABELS.
 3.ØØØØ       ENVIRONMENT DIVISION.
 4.ØØØØ       CONFIGURATION SECTION.
 5.ØØØØ       SOURCE-COMPUTER.UNIVAC-OS3.
 6.ØØØØ       OBJECT-COMPUTER.UNIVAC-OS3.
 7.ØØØØ       INPUT-OUTPUT SECTION.
 8.ØØØØ       FILE CONTROL.
 9.ØØØØ           SELECT CARDIN, ASSIGN TO CARDREADER-CARDIN-F.
10.ØØØØ           SELECT PRINTOUT, ASSIGN TO PRINTER-PRINTOUT-F.
11.ØØØØ       DATA DIVISION.
12.ØØØØ       FILE SECTION.
13.ØØØØ       FD  CARDIN
14.ØØØØ           LABEL RECORDS ARE OMITTED.
15.ØØØØ       Ø1  CARD-INPUT.
16.ØØØØ           Ø2  NAME          PIC X(25).
17.ØØØØ           Ø2  STREET        PIC X(25).
18.ØØØØ           Ø2  CITY          PIC X(15).
19.ØØØØ           Ø2  STATE         PIC X(2).
20.ØØØØ           Ø2  ZIP           PIC X(5).
21.ØØØØ           Ø2  FILLER        PIC X(8).
22.ØØØØ       FD  PRINTOUT
23.ØØØØ           LABEL RECORDS ARE STANDARD.
24.ØØØØ       Ø1  PRINTLINE         PIC X(29).
25.ØØØØ       WORKING-STORAGE SECTION.
26.ØØØØ       Ø1  CITY-STATE-ZIP-LINE.
27.ØØØØ           Ø2  CITY-OUT      PIC X(15).
28.ØØØØ           Ø2  FILLER        PIC X(1) VALUE SPACES.
29.ØØØØ           Ø2  STATE-OUT     PIC X(2).
30.ØØØØ           Ø2  FILLER        PIC X(2) VALUE SPACES.
31.ØØØØ           Ø2  ZIP-OUT       PIC X(5).
32.ØØØØ       PROCEDURE DIVISION.
33.ØØØØ       BEGIN-JOB.
34.ØØØØ           OPEN INPUT CARDIN, OUTPUT PRINTOUT.
35.ØØØØ       READ-CARD.
36.ØØØØ           READ CARDIN, AT END GO TO END-OF-JOB.
37.2ØØØ           MOVE SPACES TO PRINTLINE.
38.4ØØØ           WRITE PRINTLINE.
39.ØØØØ           MOVE NAME TO PRINTLINE.
40.ØØØØ           WRITE PRINTLINE.
41.ØØØØ           MOVE STREET TO PRINTLINE.
42.ØØØØ           WRITE PRINTLINE.
43.ØØØØ           MOVE CITY TO CITY-OUT.
44.ØØØØ           MOVE STATE TO STATE-OUT.
45.ØØØØ           MOVE ZIP TO ZIP-OUT.
46.ØØØØ           WRITE PRINTLINE FROM CITY-STATE-ZIP-LINE.
47.ØØØØ           GO TO READ-CARD.
48.ØØØØ       END-OF-JOB.
49.ØØØØ           CLOSE CARDIN, PRINTOUT.
50.ØØØØ           STOP RUN.
```

```
New line to program keyed        51.0000          02  ACCT-NO      PIC X(4).
in at the workstation
Moves the contents of line       52.0000@MOVE 51 TO 16.5
49 to line 16.5
Changes the '8' on line 21       52.0000@ON 21  CHANGE '8' to '4'
to '4'
New line to program keyed        52.0000          MOVE ACCT-NO TO PRINTLINE.
in at the workstation
Moves the contents of line       53.0000@MOVE 52 TO 39.5
51 to line 37.5
New lines to program keyed       53.0000          MOVE SPACES TO PRINTLINE.
in at the workstation            54.0000          WRITE PRINTLINE.
Copies the contents of line      55.0000@COPY 53 TO 40.2, 42.2, 46.2
52 to lines 38.2, 40.2 and
44.2
Deletes line 52                  55.0000@DELETE 53
Copies the contents of line      55.0000@COPY 54 TO 40.4, 42.4, 46.4
53 to lines 38.4, 40.4 and
44.4
Deletes line 53                  55.0000@DELETE 54
Displays lines 12 through        55.0000@PRINT 12:31
31 of our work-space file
                                 12.0000       FILE SECTION.
                                 13.0000       FD  CARDIN
                                 14.0000           LABEL RECORDS ARE OMITTED.
                                 15.0000       01 CARD-INPUT.
                                 16.0000           02 NAME         PIC X(25).
New line added                   16.5000           02 ACCT-NO      PIC X(4).
                                 17.0000           02 STREET       PIC X(25).
                                 18.0000           02 CITY         PIC X(15).
                                 19.0000           02 STATE        PIC X(2).
                                 20.0000           02 ZIP          PIC X(5).
Change made to existing line     21.0000           02 FILLER       PIC X(4).
                                 22.0000       FD PRINTOUT
                                 23.0000           LABEL RECORDS ARE STANDARD.
                                 24.0000       01 PRINTLINE   PIC X(29).
                                 25.0000       WORKING-STORAGE SECTION.
                                 26.0000       01 CITY-STATE-ZIP-LINE.
                                 27.0000           02 CITY-OUT     PIC X(15).
                                 28.0000           02 FILLER       PIC X(1) VALUE SPACES.
                                 29.0000           02 STATE-OUT    PIC X(2).
                                 30.0000           02 FILLER       PIC X(2) VALUE SPACES.
                                 31.0000           02 ZIP-OUT      PIC X(5).
Displays lines 37-46             55.0000@PRINT 39:48
of work-space file               39.0000           MOVE NAME TO PRINTLINE.
New line added                   39.5000           MOVE ACCT-NO TO PRINTLINE.
                                 40.0000           WRITE PRINTLINE.
New line added                   40.2000           MOVE SPACES TO PRINTLINE.
New line added                   40.4000           WRITE PRINTLINE.
                                 41.0000           MOVE STREET TO PRINTLINE.
                                 42.0000           WRITE PRINTLINE.
New line added                   42.2000           MOVE SPACES TO PRINTLINE.
New line added                   42.4000           WRITE PRINTLINE.
                                 43.0000           MOVE CITY TO CITY-OUT.
                                 44.0000           MOVE STATE TO STATE-OUT.
                                 45.0000           MOVE ZIP TO ZIP-OUT.
                                 46.0000           WRITE PRINTLINE FROM CITY-STATE-ZIP-
                                                   LINE.
```

(continued)

```
New line added                  46.2000          MOVE SPACES TO PRINTLINE.
New line added                  46.4000          WRITE PRINTLINE.
                                47.4000          GO TO READ-CARD.
                                48.0000      END-OF-JOB.
Lists the contents of the       55.0000@LIST &
work-space file (our entire
program)
Punches lines 12 through 45     55.0000@PUNCH 12:45
on cards
Writes our program to a         55.0000@WRITE MO=LABELS,FIL=MAGAZINE,VSN=USER01
permanent file on disk
Reminds us that this module     IS100 LABELS, MAGAZINE EXISTS; OK TO WRITE to IT? (Y,N) Y
already exists, and asks if
we want to overwrite it
We select Y, causing the
previous version of our program
to be overwritten
```

5.3.2. Screen Mode

The sample EDT session shows that EDT can operate in what we call "line mode" – you transmit one line at a time as you enter data or source code. But there is a second way of entering code or data through EDT, using what we call "screen mode." With screen mode, you can key in up to 14 lines on your workstation screen before transmitting them to EDT. This feature can save you time by having EDT accept larger blocks of text or data each time you transmit to it. And you can see more of your data at once as you enter it. You have the same range of EDT commands in screen mode as in line mode.

There are additional features available to you through screen mode. If you're a COBOL, RPG II, or FORTRAN IV programmer, you can ask the EDT screen mode to show you formatted screens for entering source code. Also, if you're programming in basic assembler language (BAL), or if you want to enter uniform data, screen mode provides you with a freeform screen. The freeform feature is especially useful for data entry personnel who can use it to create or update data files independently of any program. You then can use any of these data files with any program you write.

You can activate screen mode immediately upon entering EDT, and you can switch modes whenever you wish.

5.4. RPG II EDITOR

The RPG II editor is a specialized language editor used to create and update RPG II source programs interactively from a workstation. The RPG II editor is actually a subeditor of the general editor (EDT). Just like EDT, your program creation and changes are done in the workspace file. Thus, you always have a backup program. Figure 5-6 shows a flowchart creating an RPG II source program.

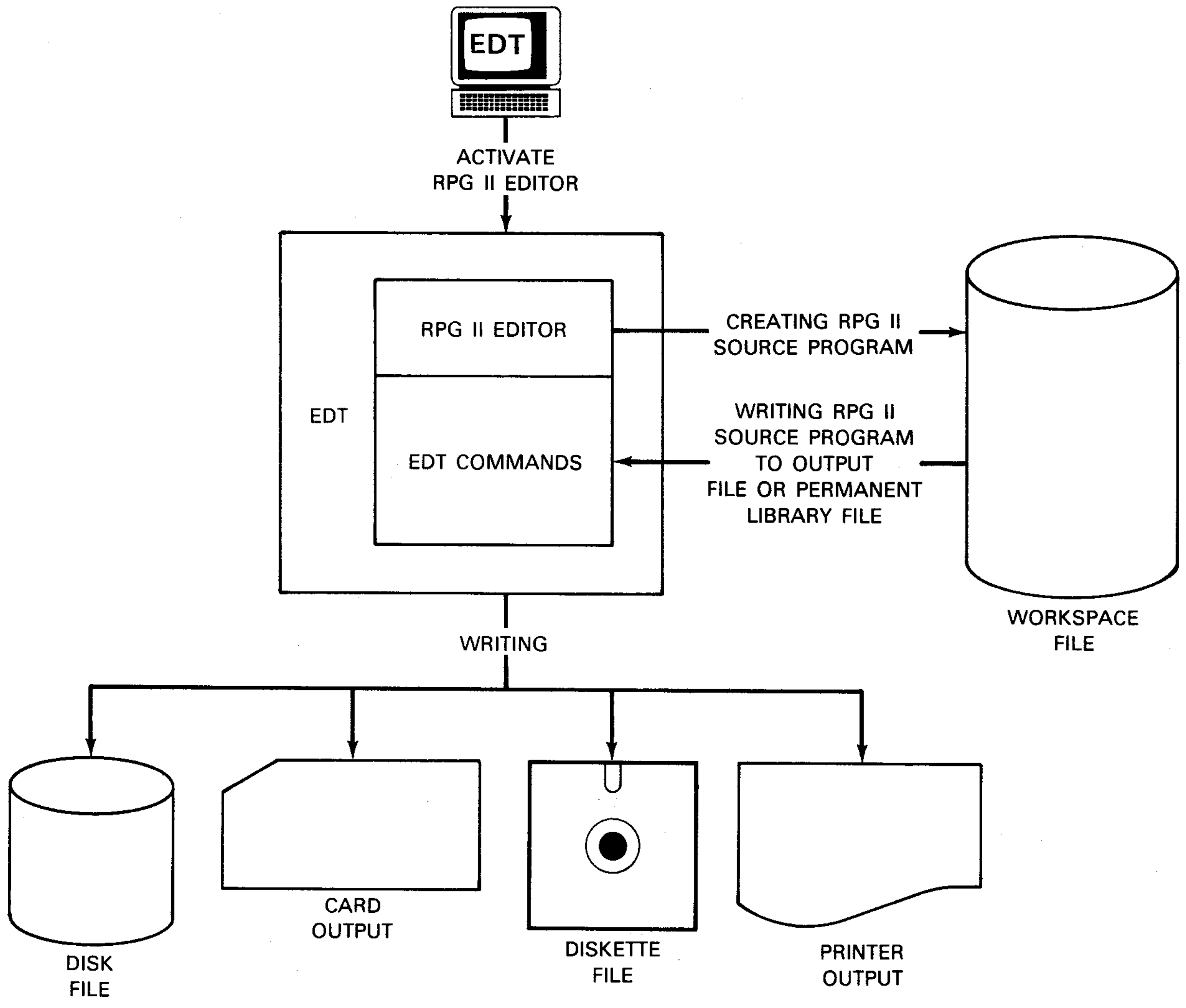
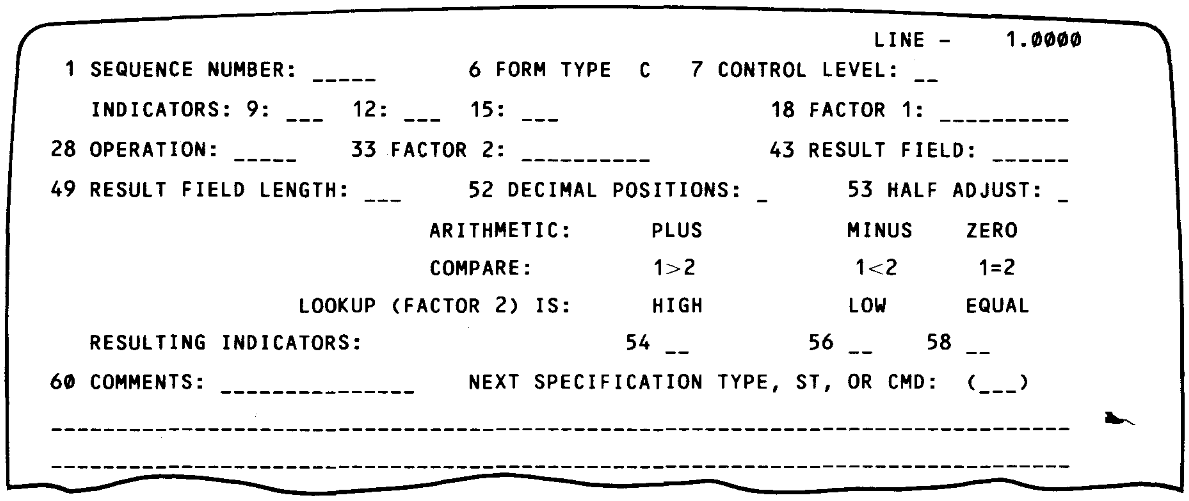


Figure 5—6. Creating an RPG II Program

In a noninteractive programming environment, you code your RPG II source program on RPG II specification forms. Well, using the RPG II editor interactively, it gives you a choice of the type of specification screens depending on your level of expertise.

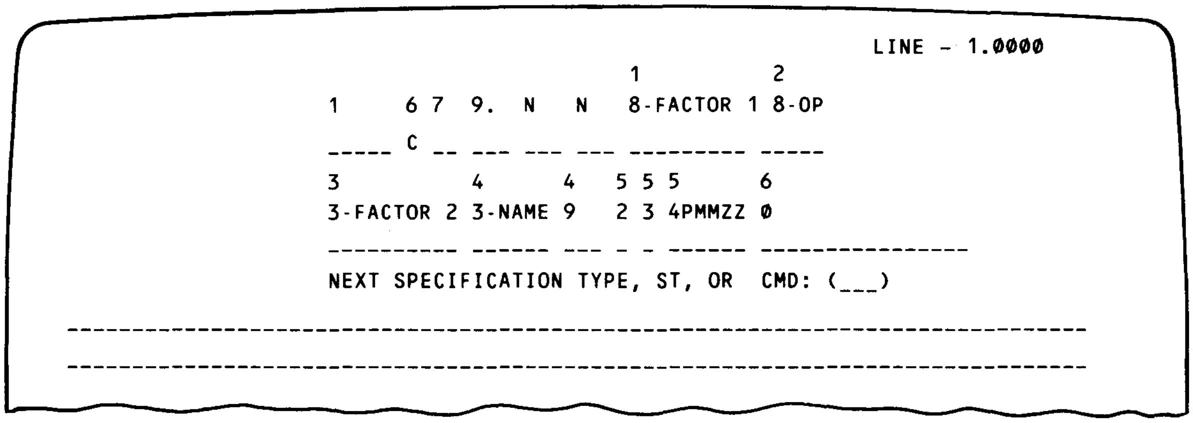
For the novice user, you are provided with a screen similar to the coding specification forms. This type of screen is called a formatted specification screen where all the column headings and the number of character positions are shown. Figure 5-7 shows a formatted calculation specification screen.



```
                                                         LINE -     1.0000
  1 SEQUENCE NUMBER: _____        6 FORM TYPE  C   7 CONTROL LEVEL: __
    INDICATORS: 9: ___  12: ___  15: ___              18 FACTOR 1: _________
 28 OPERATION: _____   33 FACTOR 2: __________        43 RESULT FIELD: ______
 49 RESULT FIELD LENGTH: ___      52 DECIMAL POSITIONS: _     53 HALF ADJUST: _
                              ARITHMETIC:      PLUS           MINUS     ZERO
                              COMPARE:         1>2             1<2       1=2
                   LOOKUP (FACTOR 2) IS:       HIGH            LOW      EQUAL
    RESULTING INDICATORS:                    54 __         56 __     58 __
 60 COMMENTS: _______________     NEXT SPECIFICATION TYPE, ST, OR CMD:  (___)
 ------------------------------------------------------------------------------
 ------------------------------------------------------------------------------
```

Figure 5-7. Formatted RPG II Calculation Specification Screen

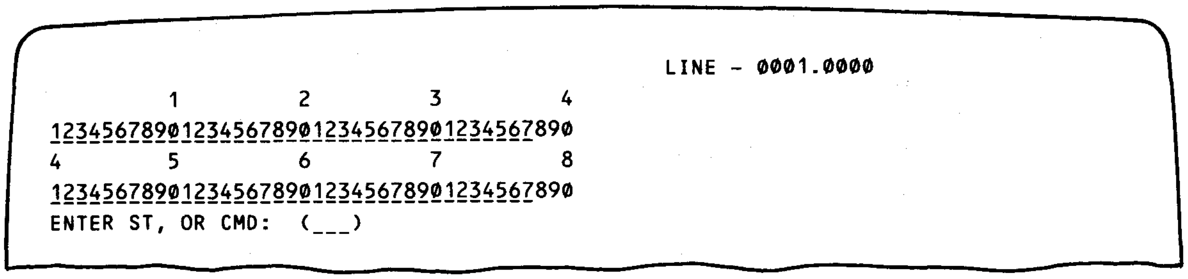
For the more experienced programmer, a positional specification screen is provided. This screen type is not as detailed as the formatted type because it only shows the starting field column numbers. Figure 5-8 shows a positional calculation screen.



```
                                                        LINE - 1.0000
                                        1          2
                         1     6 7  9.  N   N   8-FACTOR 1 8-OP
                         _____ C __ ___ ___ ___ _________ _____
                         3          4      4    5 5 5      6
                         3-FACTOR 2 3-NAME 9    2 3 4PMMZZ 0
                         __________ ______ ___ _ _ ______ ________________
                         NEXT SPECIFICATION TYPE, ST, OR  CMD: (___)
   ------------------------------------------------------------------------------
   ------------------------------------------------------------------------------
```

Figure 5-8. Positional RPG II Calculation Specification Screen

For the most experienced programmer, a freeform specification screen is provided. Here, only column numbers are displayed making you responsible for proper field locations and entries. Figure 5-9 shows a freeform specification screen.



```
                                                     LINE - 0001.0000
         1         2         3         4
1234567890123456789012345678901234567890
4        5         6         7         8
1234567890123456789012345678901234567890
ENTER ST, OR CMD:  (___)
```

Figure 5-9. Freeform RPG II Specification Screen

In conjunction with the various types of screens being provided, the RPG II editor also provides some syntax checking, thus preventing these errors from appearing in your compilation.

For more information about the RPG II editor, see the current version of the RPG II editor user guide/programmer reference, UP-8803.

5.5. COBOL EDITOR

The COBOL editor is a specialized language editor used to create and update COBOL source programs. Like the RPG II editor described in 5.4, it runs as a subeditor under EDT, which means you call EDT first, then the COBOL editor. Just like EDT, you create and change programs in the workspace file. Thus, you always have a backup copy of your program. The flowchart in Figure 5-10 shows how you go about creating a COBOL program.

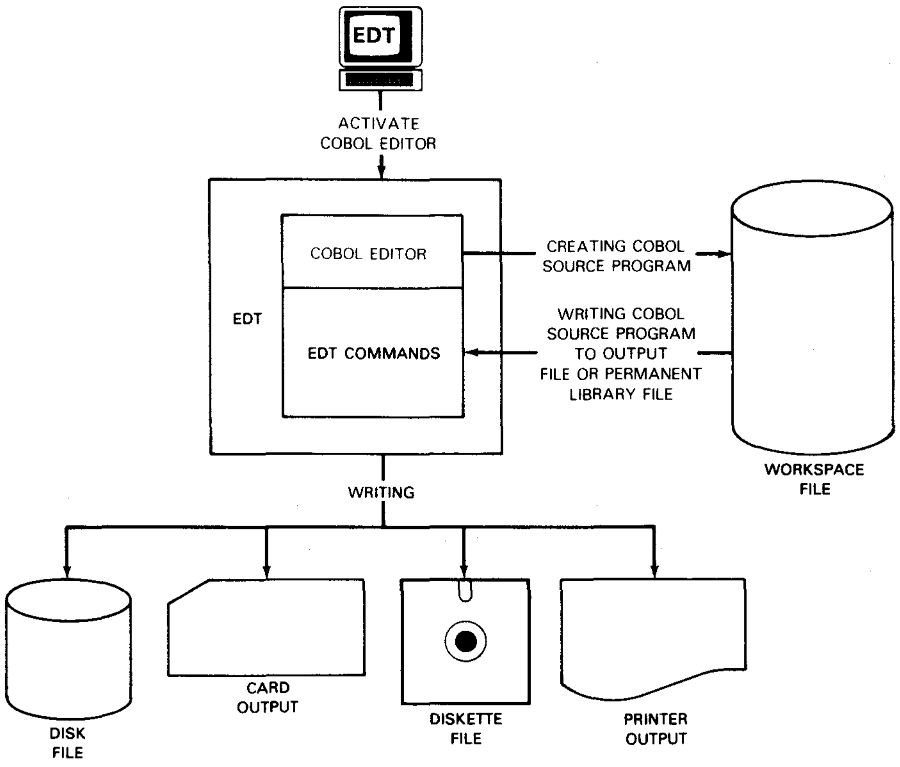
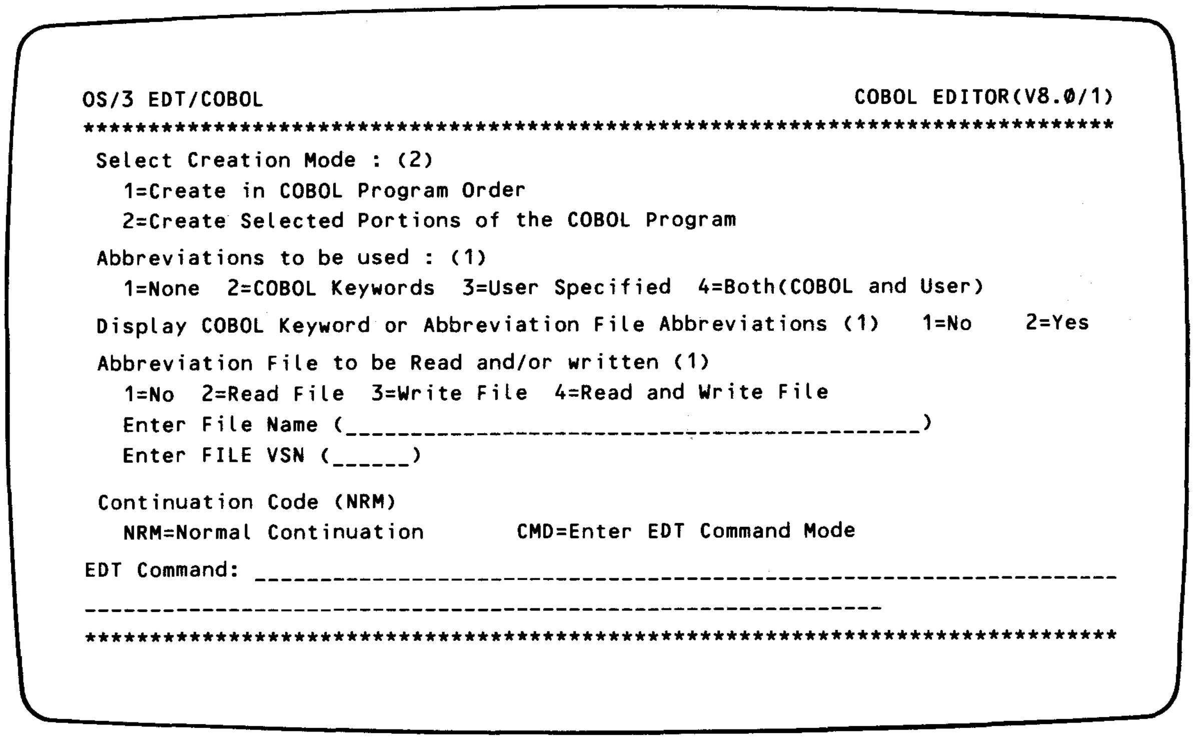


Figure 5-10. Creating a COBOL Program

In a batch programming environment, you code a COBOL source program line by line on coding forms. You then input the program to the COBOL compiler, using input media such as punch cards or diskettes. The COBOL editor, however, lets you do all this interactively. With it, you can see source lines displayed on the workstation screen as you key them in and therefore can check them immediately for coding and typographical errors.

A feature of the COBOL editor that is not available in batch environment is that when you finally transmit your source code, the COBOL editor does some syntax checking. Your COBOL source code will thus have fewer errors in it even before it is first compiled.

The COBOL editor works with screen formats that contain such things as required COBOL statements and directions for entering variable data. When you first call the COBOL editor, you see this screen:

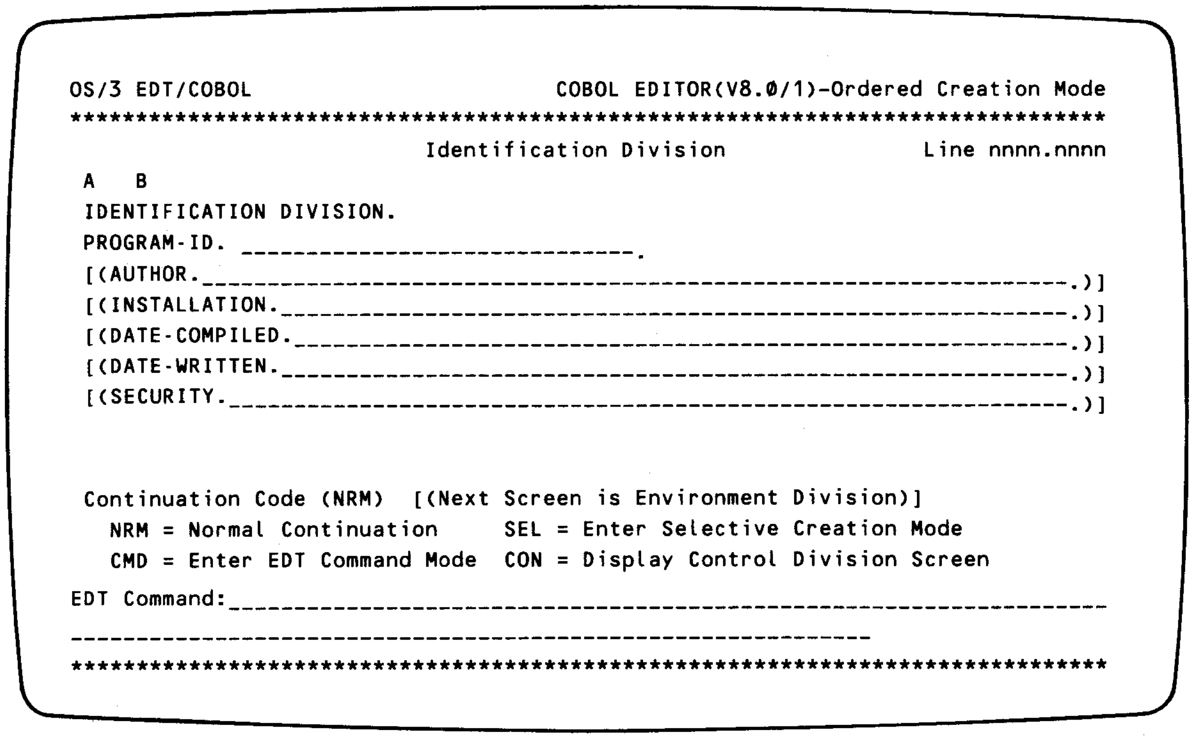


```
OS/3 EDT/COBOL                                              COBOL EDITOR(V8.Ø/1)
******************************************************************************
 Select Creation Mode : (2)
   1=Create in COBOL Program Order
   2=Create Selected Portions of the COBOL Program
 Abbreviations to be used : (1)
   1=None  2=COBOL Keywords  3=User Specified  4=Both(COBOL and User)
 Display COBOL Keyword or Abbreviation File Abbreviations (1)   1=No    2=Yes
 Abbreviation File to be Read and/or written (1)
   1=No  2=Read File  3=Write File  4=Read and Write File
   Enter File Name (___________________________________________)
   Enter FILE VSN (______)

 Continuation Code (NRM)
   NRM=Normal Continuation       CMD=Enter EDT Command Mode
EDT Command: _________________________________________________________________
____________________________________________________________
******************************************************************************
```

Figure 5-11. Initial COBOL Editor Screen

The screen in Figure 5-11 gives you a choice of how you enter your COBOL source code. If you are a novice COBOL programmer, the ordered creation mode of the COBOL editor will display screens in sequence to guide you through the creation of a COBOL program. Figure 5-12 shows one of the screens displayed in this mode.

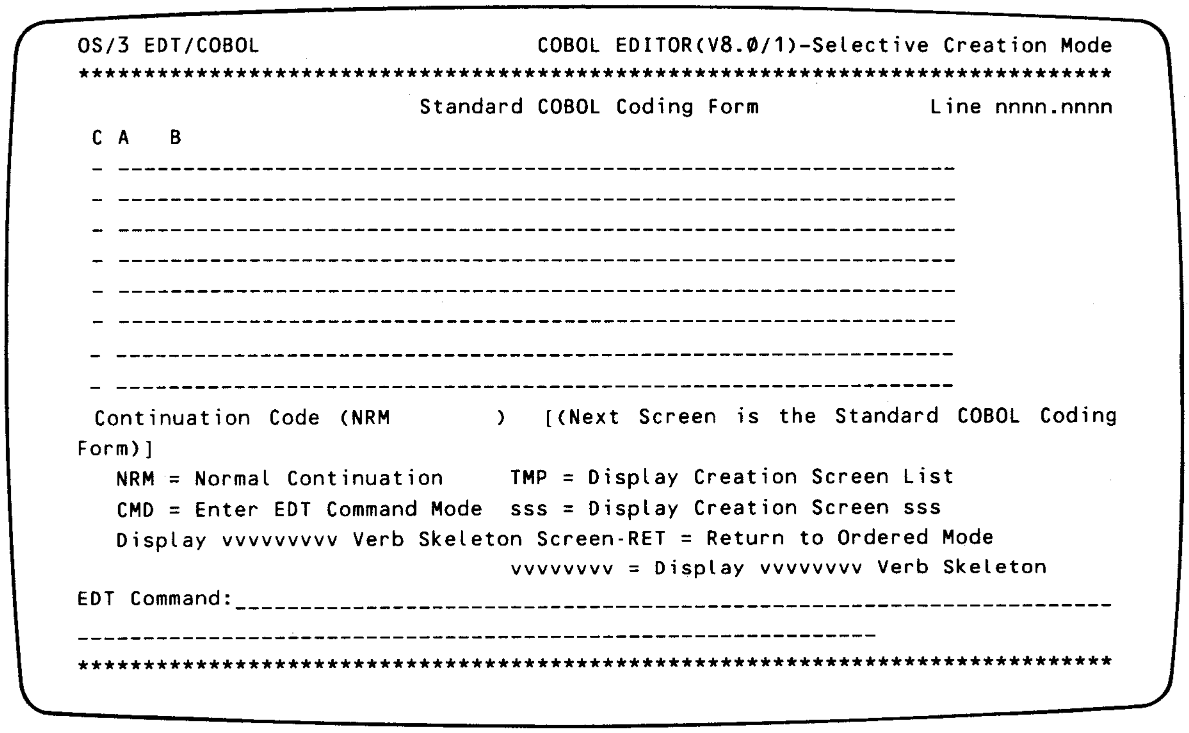


```
OS/3 EDT/COBOL                        COBOL EDITOR(V8.Ø/1)-Ordered Creation Mode
*******************************************************************************
                                 Identification Division             Line nnnn.nnnn
 A   B
 IDENTIFICATION DIVISION.
 PROGRAM-ID. ______________________________.
 [(AUTHOR.__________________________________________________________________.)]
 [(INSTALLATION.____________________________________________________________.)]
 [(DATE-COMPILED.___________________________________________________________.)]
 [(DATE-WRITTEN.____________________________________________________________.)]
 [(SECURITY.________________________________________________________________.)]


 Continuation Code (NRM)  [(Next Screen is Environment Division)]
   NRM = Normal Continuation     SEL = Enter Selective Creation Mode
   CMD = Enter EDT Command Mode  CON = Display Control Division Screen
EDT Command:_________________________________________________________________
_____________________________________________________________
*******************************************************************************
```

Figure 5-12. Typical Screen in Ordered Creation Mode

More experienced programmers may want to use the selective creation mode. In this mode, you may work on any part of the program you wish. Figure 5-13 shows one of the screens displayed in this mode.



```
OS/3 EDT/COBOL                      COBOL EDITOR(V8.Ø/1)-Selective Creation Mode
*******************************************************************************
                                 Standard COBOL Coding Form         Line nnnn.nnnn
 C A   B
 - ________________________________________________________________
 - ________________________________________________________________
 - ________________________________________________________________
 - ________________________________________________________________
 - ________________________________________________________________
 - ________________________________________________________________
 - ________________________________________________________________
 - ________________________________________________________________
 Continuation Code (NRM        )  [(Next Screen is the Standard COBOL Coding
Form)]
   NRM = Normal Continuation     TMP = Display Creation Screen List
   CMD = Enter EDT Command Mode  sss = Display Creation Screen sss
   Display vvvvvvvvv Verb Skeleton Screen-RET = Return to Ordered Mode
                                 vvvvvvvv = Display vvvvvvvv Verb Skeleton
EDT Command:_________________________________________________________________
_____________________________________________________________
*******************************************************************************
```

Figure 5-13. Typical Screen in Selective Creation Mode

For more information about the COBOL editor, see the current version of the COBOL editor user guide/programmer reference, UP-9106.

5.6. ERROR FILE PROCESSOR

So far we've shown you how useful EDT can be in helping you prepare source programs for compilation. Its usefulness doesn't stop here, however. Besides allowing you to update source code to eliminate errors, EDT has a component called the error file processor (EFP) that lets you see these errors on the spot.

EFP helps you in two ways. It lets you see compilation errors immediately after the language compiler you're using has compiled your program, rather than after the compiler prints an error listing. And it can display these errors along with the source lines containing them. You can then use all of EDT's facilities to correct the source code on the spot. This reduces the time you spend between compilations. You can use EFP with programs written in COBOL, RPG II, and FORTRAN IV.

EFP uses two input files. One is your source program on disk. The other is an error file created by your language compiler at the time the program was compiled. Error files can be created by the COBOL, RPG II, and FORTRAN IV compilers; each of these allows you to request an error file, and lets you direct the data to any file of your own. (See the appropriate language manual for more information on creating error files.)

When the compilation has finished, you activate the EFP through the general editor. Upon activation, EFP:

1. Asks you to identify the error file you had previously created.

2. Uses information from the error file to locate the source module, report an error summary at your workstation, and display a system message identifying all files and modules involved.

3. Directs EDT to write your source module to the EDT workspace.

From then on, you can control your source module through regular EDT commands, and you can control your error file through EFP commands.

EFP can't correct your source program if it's on cards, but it does give you the error summary. Nearly the same restriction holds for programs you compile using your compiler's option for correcting the module at the same time you're compiling it. In this case, the corrections you make don't correct your source module, but will give you a correct object module.

For more information on EFP, see the current version of the general editor user guide/programmer reference, UP-8828.

5.7. LINKAGE EDITOR

The linkage editor is used to convert the nonexecutable object modules produced by the various language processors into executable load modules. Every object module must be submitted to the linkage editor after compilation before it can be executed. The load modules produced by the linkage editor consist of all specified object modules, links to all sharable modules required, and information required to control the loading and execution of the load module.

You control the linkage editor through a set of control statements. These control statements specify alternative or optional processing and are inserted in the control stream immediately following the statement that executed the linkage editor.

The linkage editor can combine a number of separate object modules into a single load module. Unless otherwise directed, it will include all object modules in $Y$RUN at linkage editor execution time into the generated load module. It can construct load modules, called multiphase and multiregion load modules, that, by employing main storage overlay techniques, require less main storage to execute than the overall length of the job. It can also create sharable load modules.

The linkage editor produces a printed listing detailing the operations performed and the structure of the load module produced. This listing is called the link-edit map and also contains any error messages associated with the linkage editor operation.

More detailed information on the linkage editor can be found in the current version of the system service programs user guide, UP-8841.

6. Language Processors

6.1. GENERAL

OS/3 offers you flexibility in program development by supporting a number of programming language processors. The supported languages are:

- BASIC
- COBOL
- Report Program Generator II (RPG II)
- FORTRAN IV
- The Basic Assembly Language (BAL)
- ESCORT

Source programs for each of these languages can be entered through the workstation using the general editor or you can enter them through a card reader. Before the programs can be executed, they must be compiled by the appropriate language processor and processed by the linkage editor.

The compilation, linking, and execution of a program can be performed as a single operation or each step can be performed separately. In either case, you are required to enter your program as part of a job control stream. The following subsections describe the capabilities and usage of each of the supported languages as well as detailing some of the job control requirements for each language.

6.2. BASIC

The Beginner's All-purpose Symbolic Instruction Code (BASIC) language is an interactive programming language designed to be easy to use, yet meet the requirements of both business and scientific programming. The BASIC language available on the OS/3 operating system complies with the *American National Standard Minimal BASIC, X3.60-1978* and includes Dartmouth features and compatibility. It provides a powerful, yet simple set of commands allowing the novice to learn the language quickly, and yet gives the experienced programmer an extensive list of features for various applications.

BASIC is an interactive language and all source statements can be entered directly at the workstation with the results and error messages displayed on the screen. All source statements are checked for syntax errors as they are entered and a message appears on the screen if an entered line is in error. The BASIC source program can be compiled directly at the workstation and compilation errors can be corrected immediately. During an interactive BASIC session, you can input, modify, execute, and save programs.

The OS/3 BASIC compiler has facilities for arithmetic operations, data file processing, matrix generation and processing, and logical operations. Subroutines and string operations may be used in a BASIC program.

6.3. COBOL

The common business oriented language (COBOL) is a high-level, general purpose programming language designed to meet the programming needs of today's business. COBOL programs can be generated for a wide range of applications including payroll, accounting, billing, shipping and receiving, and so on. COBOL uses a syntax structure similar to that of English employing verbs and clauses that produce English-like statements to express programming operations. The following is a brief description of the COBOL language. More detailed information on the capabilities and use of COBOL can be found in the current version of the 1974 ANSI COBOL programmer reference, UP-8612.

OS/3 COBOL is compatible with the *American National Standard COBOL X3.23-1974* with extensions for more advanced programming and to meet the requirements of your Sperry Univac system.

COBOL programs are divided into segments called divisions. Each division performs a specific function within the program. The four COBOL divisions and their functions are:

- Identification division

  This division contains information identifying the source program and the output of a compilation. Additionally, other information may be included such as the program author, installation, and so forth.

- Environment division

  This division specifies the system's hardware characteristics, input/output control techniques, and other information of a similar nature.

- Data division

  This division defines the data that your program is using. This division is further divided into sections to facilitate the description of data contained in input or output files or developed during the program execution. Data constants are also defined in this section.

- Procedure division

  This division describes the logical steps that must be taken in the solution of the processing problem. The functions performed can be arithmetic operations; file opening and closing; record sorting and merging; record movement, deletion, and insertion; and program branching operations. Conditional test statements are also available to directly control the execution of your program.

The COBOL compiler can be initiated one of two ways. You can use the job control EXEC statement with the compiler name:

```
// EXEC COBL74
```

or you can use the jproc supplied by Sperry Univac:

```
// COBL74
```

The jproc is easier to use because it provides minimum job control requirements as default values that you have to specify if you use the EXEC statement. Additionally, the jproc can be issued to perform a compile and link operation:

```
// COBL74L
```

or a compile, link, and execute:

```
// COBL74LG
```

Again, these jprocs supply the minimum job control requirements for each operation.

When generating the job control stream for your COBOL program, you must provide device assignment sets for each of the files that you defined in your data division. The job control device assignment sets are related to the files in the COBOL program through the specified LFD name which must be the same as the name you specified on the assign clause of the select statement.

The COBOL compiler requires three disk work files during the compilation of your program. These files are provided automatically through the jprocs, but you must include a device assignment set for each one if you use the EXEC statement.

Your COBOL program can be compiled, linked, and executed from the workstation and error messages resulting from any of these operations can be returned to the workstation. In addition, your COBOL program can accept and display information on the workstation screen either directly through the available statements or through the screen format or dialog processing services. The method you select depends upon your requirements. More information on the use of screen formats and dialogs is included in Section 7.

6.4. REPORT PROGRAM GENERATOR II

Report program generator II (RPG II) is a nonprocedural programming language designed to meet business programming needs through simple, straightforward programming. While primarily designed as a means of generating printed reports from data records, OS/3 RPG II can be used for a wide range of business applications. It provides the processing required to:

- write programs to control inventories;
- process payrolls;
- provide accounting and billing facilities;
- develop sales and marketing analysis; and
- perform a variety of other business-oriented operations.

The following is a brief description of the RPG II language; for more information, see the report program generator II user guide, UP-8067 (current version).

To make the language as simple to use as possible, RPG II source statements are created by making entries on specific positions on fixed specifications formats. A specification format is available for each phase of program development. The formats are:

- Control specification
- File description specification
- File extension specification
- Line counter specification
- Telecommunication specification
- Input format specification
- Calculation specification
- Output format specification

RPG II programs can be developed interactively through the RPG II editor. The RPG II editor is initiated at the workstation and assists you in developing your programs by displaying the RPG II formats and explanations of available entries. You can also enter freeform RPG II statements to the RPG II editor. Once you have entered all the statements required for your program, you use the general editor to store the program as a source module in either a system program library or one of your own libraries.

You can also use the RPG II auto report facility to generate an RPG II source program. The auto report facility allows you to enter simplified specifications and standard RPG II statements to generate a complete RPG II source program from prefiled specifications.

6.5. FORTRAN IV

FORTRAN IV is a high-level, easy-to-use programming language best suited for applications requiring the performance of mathematical computations required for engineering and scientific applications. In addition, it is ideally suited for business applications that can make use of its extensive computational capabilities. OS/3 FORTRAN IV adheres to the *American National Standards FORTRAN X3.10-1966*. In addition, OS/3 FORTRAN IV contains many extensions to the standard that provide compatibility with IBM 360/370 DOS FORTRAN IV and SPERRY UNIVAC Series 70 FORTRAN. The following is a brief description of the capabilities and usage of FORTRAN IV. For more detailed information, see the current version of the FORTRAN IV programmer reference, UP-8814.

A FORTRAN IV program consists of one main program, containing the steps required to solve a given problem, and any number of required subprograms. The subprograms perform procedures that may be repeated several times, I/O operations, and the standard library functions supplied to perform a variety of common mathematical operations. The main program and the subprograms are compiled separately and included in a single executable load module by the linkage editor.

You must supply a separate I/O subroutine to interface the FORTRAN I/O operations with the consolidated data management routines. This subroutine is included in the load module by the linkage editor and consists of consolidated data management macroinstructions.

Your FORTRAN IV program can be interfaced with the screen formatting and dialog processing services for variable data input through the workstation. Additionally, you can interactively generate a FORTRAN IV program through the general editor and initiate the compilation, linking, and execution of your program from the workstation with error messages sent to the initiating workstation.

6.6. BASIC ASSEMBLY LANGUAGE (BAL)

The basic assembly language is a versatile and detailed symbolic language designed for the highly skilled programmer who requires system level control of the processing the program is to perform. The assembler language instruction repertoire consists of instructions, macroinstructions, procedural calls, assembly directives, and conditional statements.

The instruction is the basic control element of the assembler. Each instruction is assigned a mnemonic to denote the particular hardware function performed. Using the instructions set, you can perform:

- storage definition;
- various branching operations;
- logical operations;
- arithmetic operations; and
- miscellaneous special operations.

The assembler macro facility is a dual purpose facility that allows the inclusion in your program of macroinstructions supplied by Sperry Univac and the generation of your own macroinstructions and procedural calls. Macroinstructions and procedural calls (procs) are identical in concept and usage. A macroinstruction or a proc is a single statement coded into your program that, when the program is assembled, is expanded into a series of assembler instructions designed to perform a specific task or tasks. The use of macroinstructions and procs eliminates the need to write code required for frequently used operations. You can write your own macroinstructions or procs, assign names to them, and insert the names into the program wherever that particular set of coding is required.

Conditional statements control the processing of your assembler program by various testing operations. These statements respond to various conditions and can alter processing or cause additional processing to occur.

The assembler includes a set of directives used to specify instructions to the assembler itself. These directives allow you to control program sectioning, base register assignment, the format of output listings, sequence checking, and other auxiliary functions.

For more detailed information on the basic assembly language, see the assembler user guide, UP-8913 (current version).

6.7. ESCORT

ESCORT is an easy-to-use programming language that helps you solve common business problems. Unlike most programming languages, you don't have to be a programmer to use it. With ESCORT, you write (or let ESCORT create) programs to handle such common business tasks as those relating to payroll, personnel files, and inventory.

Much of its ease of use lies in its English-language program statements and its two operating modes:

- **Tutorial Mode**

  Tutorial mode introduces you to ESCORT and allows you to create programs even if you may know little or nothing about programming. In this mode, ESCORT displays a series of questions on the workstation screen. You respond to these questions by filling in blanks or choosing from a list of functions. ESCORT uses these answers to create a program tailored to your needs. To aid you when you don't understand a question, many screen displays include a choice called HELP, which provides an explanation.

- **Program Mode**

  As you become more familiar with ESCORT, you'll want to use program mode, which is faster and more versatile than tutorial mode. Program mode lets you create programs either through a series of screen menus or through direct entry of program statements. This mode also provides HELP screen displays to explain menu selections.

The ESCORT language has three components: programs, structures, and jobs.

- **Programs**

  Means by which you solve your business problems

- **Structures**

  Data definitions that tell ESCORT what your file records look like or describe a logical file for input or output devices (for example, workstation display or printer output).

- **Jobs**

  Not an OS/3 job, but a group listing of related programs that are executed together. For example, in a payroll job one program might calculate hours worked, another might calculate pay and taxes, and another might update a permanent payroll file.

For more information on ESCORT, see the ESCORT user guide, UP-8855 (current version).

7. Designing Your Own Interactive Software

7.1. GENERAL

As we mentioned earlier, you can use your System 80 workstation as an interactive device. An operator using a workstation can enter data to a user program, receive output from the program, or direct the program to take some action. While you can design and program workstation screens yourself, you can save yourself a great deal of programming effort by using these Sperry Univac features:

- Screen formats
- Dialogs
- Menus

Each of these manages your workstation screen on behalf of a user program (or interactive services in the case of some menus). Each consists of constant and variable fields. Constant fields contain data and empty space which cannot be changed. The data can consist of screen titles, captions, lists, etc. Variable fields are used to input data to or output data from a program. An input variable field is one into which the operator enters data for transmittal to the program. An output variable field displays data sent to the screen by the program. A bidirectional variable field can be used for both input and output.

You can develop screen formats, dialogs, and menus independently of the programs that call them. For example, one programmer can code a source program at the same time that another programmer is creating the screens that go with it. Screens also are interchangeable. You can use different screens for the same program or the same screen with different programs.

Another feature of screens is that you don't have to include extensive code in your source program to use any of these. In most cases, it takes no more programming to call a screen than it does to read a single card from a card reader. That's because consolidated data management treats workstation features as files like any other type of file available to System 80.

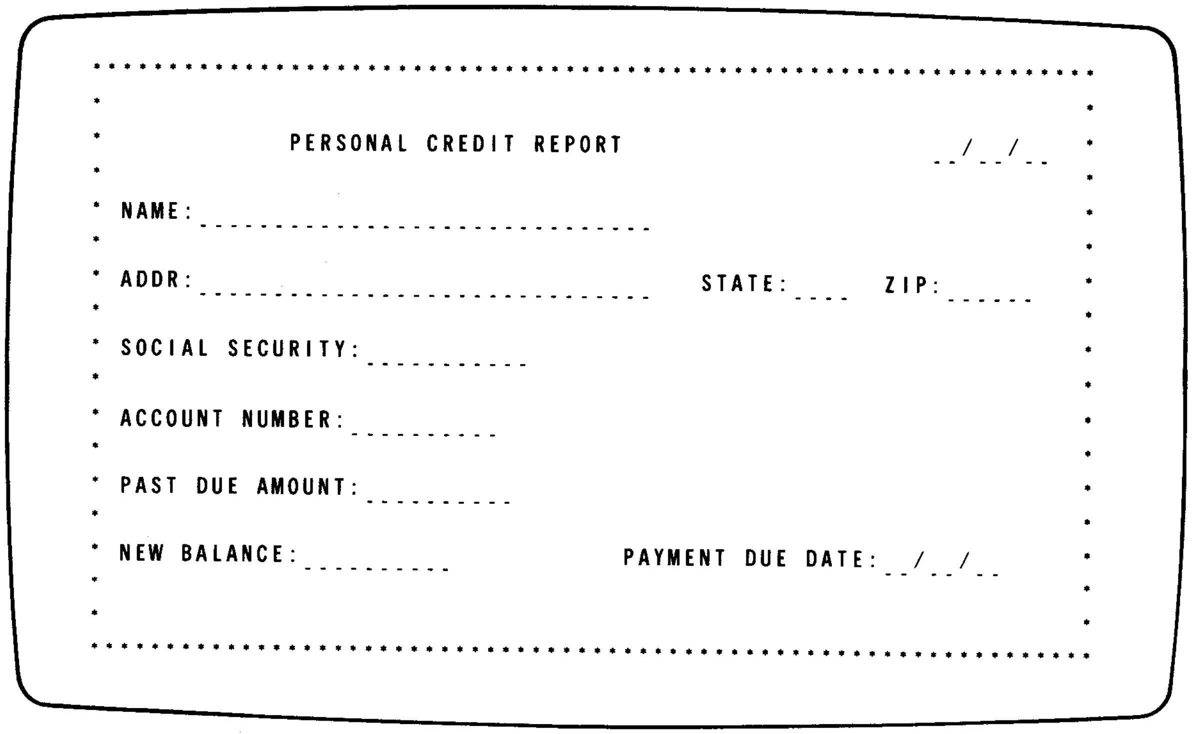
Use of screen formats, dialogs, or menus provides you with the following advantages:

- Uniform operator responses. You achieve this by limiting responses so that the operator can only fill in preset blanks or answer predetermined questions. You can ensure yourself that all data entered is uniform in content and format.
- Elimination of a number of data entry errors and omissions, thus reducing the number of inaccurate or incomplete data record
- Simple entry of complex data items
- Ready accessibility of the system to nonprogramming personnel

Following are examples of each feature:

7.1.1. Screen Formats

Screen formats are screens that present information to, and solicit information from, a workstation operator. As Figure 7-1 shows, a screen format resembles a form with blanks that the workstation operator fills in. The data thus entered is used to build records which are then passed to the user program.



```
*****************************************************************
*                                                               *
*            PERSONAL CREDIT REPORT                  __/__/__   *
*                                                               *
* NAME:______________________________                           *
*                                                               *
* ADDR:______________________________  STATE:____  ZIP:______   *
*                                                               *
* SOCIAL SECURITY:___________                                   *
*                                                               *
* ACCOUNT NUMBER:___________                                    *
*                                                               *
* PAST DUE AMOUNT:___________                                   *
*                                                               *
* NEW BALANCE:___________         PAYMENT DUE DATE:__/__/__     *
*                                                               *
*                                                               *
*****************************************************************
```

Figure 7-1. Typical Screen Format Display

In Figure 7-1, all variable fields (in underscores) are for input only. But you can also include output fields, and even bidirectional fields which first display output data then accept input data on the same field.

A screen format appears as a fixed screen. Each time it is called, the same variable and constant fields appear in the same places on the screen.

Screen formats are called from user programs. When the operator finishes a screen by transmitting it to the system, control returns to the user program calling it. A screen with no variable fields at all may also be created, for example, as a help screen.

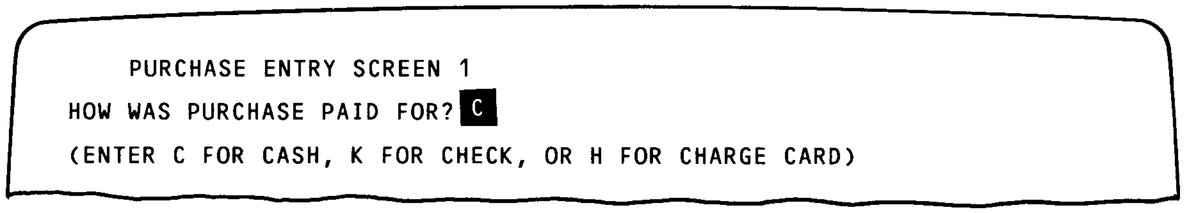
7.1.2. Dialogs

Like a screen format, a dialog is a screen or series of screens that present information to, and request information from, a workstation operator. Unlike screen formats or menus, dialogs can use non-fixed screens. You can not only program the sequence of dialog screens you want to appear, you can also program the layout of the screens themselves. Each dialog screen is built, at dialog execution time, from constant and variable fields. The selection of which fields to display and which to suppress can be based entirely on previous dialog replies.

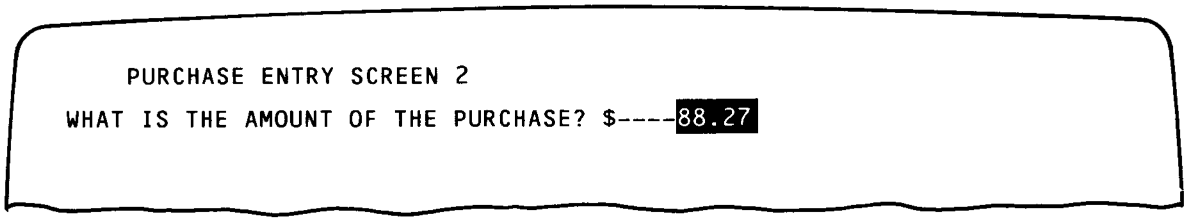
The two sample dialogs in Figure 7-2 (a and b) illustrate this important feature. They are called by a program that records department store purchases. The first screen of each dialog asks how a purchase was paid (cash, check, or charge plate), and in all three cases the second screen then asks for the purchase amount. But if the purchase was charged to a customer's account, the second screen asks for one additional piece of information, the account number (Figure 7-2b). If the purchase was by cash or check (Figure 7-2), that request does not appear.

Dialogs are input-only screens used for data entry. As Figure 7-2 shows, they may appear like either screen formats, with blank fields for the operator to fill in, or like multiple-choice screens. They can also display data entered previously in the dialog. (Although being input-only screens, they cannot display data output from the user program.)

When a dialog gets control of a workstation screen, it keeps control until the operator reaches the end of the entire dialog. After the dialog finishes, control returns to the user program, and the data collected by the dialog becomes available to the program.

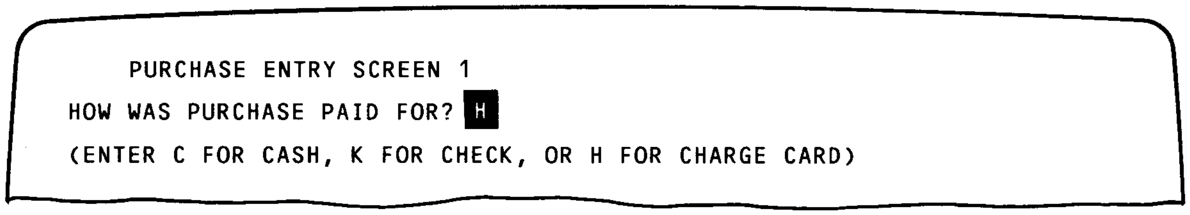


```
    PURCHASE ENTRY SCREEN 1
HOW WAS PURCHASE PAID FOR? C
(ENTER C FOR CASH, K FOR CHECK, OR H FOR CHARGE CARD)
```

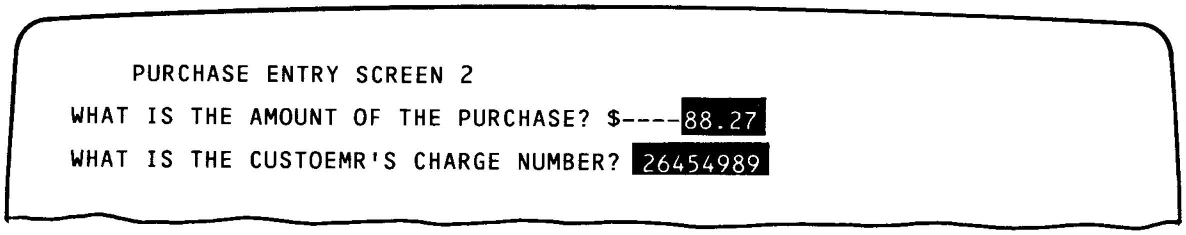


```
    PURCHASE ENTRY SCREEN 2
WHAT IS THE AMOUNT OF THE PURCHASE? $----88.27
```

a. Dialog when entering cash purchase



```
    PURCHASE ENTRY SCREEN 1
HOW WAS PURCHASE PAID FOR? H
(ENTER C FOR CASH, K FOR CHECK, OR H FOR CHARGE CARD)
```



```
    PURCHASE ENTRY SCREEN 2
WHAT IS THE AMOUNT OF THE PURCHASE? $----88.27
WHAT IS THE CUSTOEMR'S CHARGE NUMBER? 26454989
```

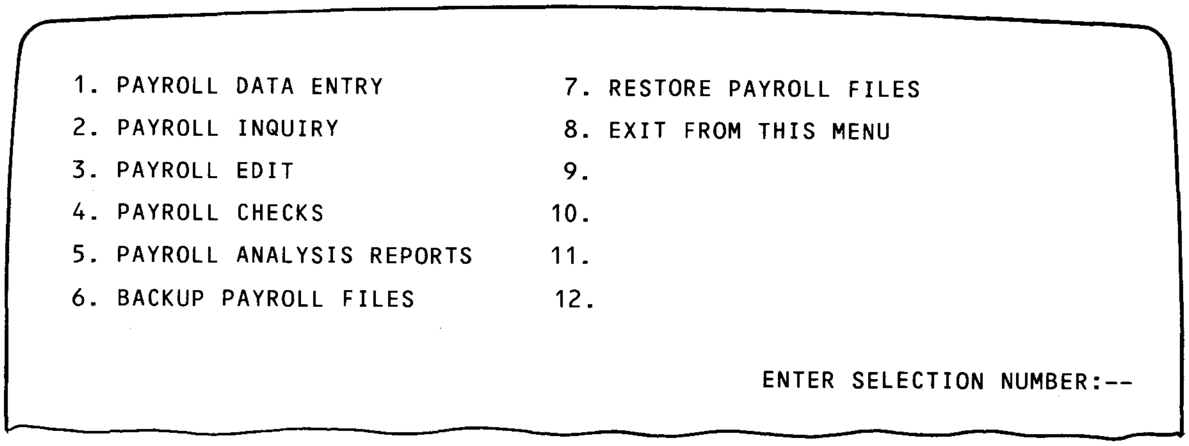
b. Dialog when entering charge plate purchase

Figure 7-2. Typical Dialog Session Screen Displays

7.1.3. Menus

The third interactive feature is a menu, an example of which appears in Figure 7-3.

As Figure 7-3 shows, a menu is a numbered, multiple-choice list of actions. The workstation operator chooses one action, enters its number in the space provided on the menu screen, and transmits the screen to the system. After the action takes place, the menu returns to the screen, ready to accept another choice.


```
1. PAYROLL DATA ENTRY              7. RESTORE PAYROLL FILES
2. PAYROLL INQUIRY                 8. EXIT FROM THIS MENU
3. PAYROLL EDIT                    9.
4. PAYROLL CHECKS                 10.
5. PAYROLL ANALYSIS REPORTS       11.
6. BACKUP PAYROLL FILES           12.

                                         ENTER SELECTION NUMBER:--
```

Figure 7-3. Typical Menu Display

Let's look more closely at how to use the menu of Figure 7-3. If the operator wants to enter payroll data, he chooses item 1 (PAYROLL DATA ENTRY) by entering the numeral 1 in the field after ENTER SELECTION NUMBER and pressing the transmit key.

The system then calls a job control stream to execute a program that allows the operator to enter payroll data. Upon termination of that job, the menu reappears so the operator can make another choice.

The same sequence of events applies to items 2 through 7. Item 8 (PAYROLL DATA ENTRY) allows the operator to end the menu session and return to system mode.

Menus are primarily used for multiple-choice applications, not for data entry. A menu may be called from a user program, for example, to ask the operator what action the program is to take next. As with a dialog, input from a menu appears to the program as merely another input file. This allows you easily to substitute a menu for other files and devices used by a program.

Unlike screen formats or dialogs, menus aren't limited to use with user programs. The MENU workstation command lets the operator call a menu which usually offers a choice of job control streams to run and workstation commands to perform related system functions. The menu of Figure 7-3, for example, could easily be called with MENU.

Like screen formats, menus use fixed screens. The same items appear each time the menu is called, and each item performs the same action when it's chosen.

Menu actions are usually keyed to the environment in which the menu is used. A menu developed for use with a program is generally not called with MENU, and vice versa. All menus, however, are capable of executing menu function commands, actions unique to menu processing that allow menus to call other menus, return to previously displayed menus, and perform other menu-related functions. It isn't necessary to use these commands, but an experienced programmer can find them quite useful in designing more sophisticated menus. Some of these commands let a menu call screen formats (input-only) on behalf of a user program. The menu relays any data entered on a screen back to the program.

When an operator chooses a menu option and transmits the menu to the system, the menu usually reappears, waiting for another operator choice. Only by programming the menu or by an external interrupt (usually depressing a function key) does it give up control.

7.1.4. Summary

We have given you an overview of how OS/3 interactive features can work for you. Following in 7.2 through 7.4, we introduce the OS/3 program products that make it possible to design your own interactive screens, and briefly discuss the job control and source program requirements for using them. After reading these subsections, you should have an idea of what is entailed in designing your own interactive software.

7.2. SCREEN FORMAT GENERATOR

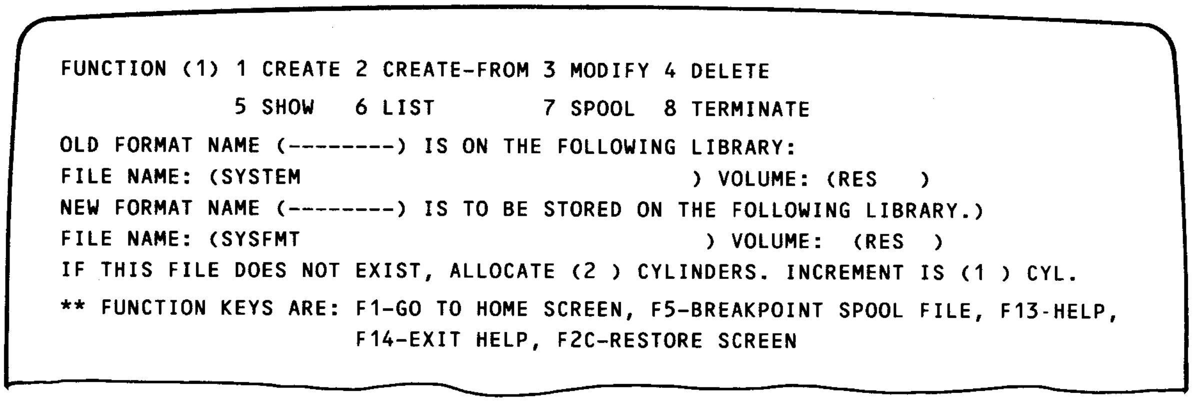
The screen format generator is used to generate screen displays to be used to input variable data to a program or to display output data through the workstation. It is an interactive feature allowing you to generate the screen formats right at the workstation. During generation you specify the format's layout, as well as other characteristics that best enable the format to reflect the input and output requirements of your program.

The actual process of laying out the screen formats involves first initiating the screen format generator via the RV SFGEN command.

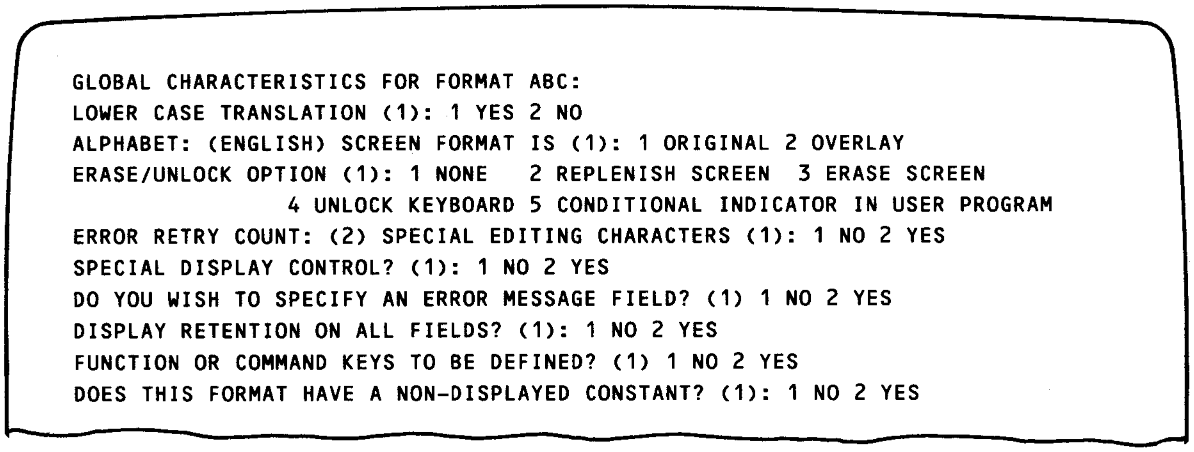
The system responds by displaying a sequence of two initial (home) screens. Figure 7-4 shows the home screens.

As you can see from Figure 7-4, the home screens request information concerning the nature of the screen or screens that you are going to generate. It is beyond the scope of this document to explain the various options available, but suffice it to say that the information you enter in response to these screens will depend on whether you are creating new screens, updating existing ones, or performing a maintenance function on the file holding your screens.

After you complete and transmit the home screens, the system will display a blank screen. Here, you fill in the information that you wish displayed. You indicate the location and size of the fields to be used for variable data and whether those fields are input, output, or both. After you complete a screen, the system returns you to the home screens. At that point you indicate whether you wish to generate more screens or terminate the screen format generator.



```
FUNCTION (1) 1 CREATE 2 CREATE-FROM 3 MODIFY 4 DELETE
             5 SHOW   6 LIST        7 SPOOL  8 TERMINATE
OLD FORMAT NAME (--------) IS ON THE FOLLOWING LIBRARY:
FILE NAME: (SYSTEM                              ) VOLUME: (RES    )
NEW FORMAT NAME (--------) IS TO BE STORED ON THE FOLLOWING LIBRARY.)
FILE NAME: (SYSFMT                                ) VOLUME:  (RES  )
IF THIS FILE DOES NOT EXIST, ALLOCATE (2 ) CYLINDERS. INCREMENT IS (1 ) CYL.
** FUNCTION KEYS ARE: F1-GO TO HOME SCREEN, F5-BREAKPOINT SPOOL FILE, F13-HELP,
                      F14-EXIT HELP, F2C-RESTORE SCREEN
```



```
GLOBAL CHARACTERISTICS FOR FORMAT ABC:
LOWER CASE TRANSLATION (1): 1 YES 2 NO
ALPHABET: (ENGLISH) SCREEN FORMAT IS (1): 1 ORIGINAL 2 OVERLAY
ERASE/UNLOCK OPTION (1): 1 NONE   2 REPLENISH SCREEN  3 ERASE SCREEN
                4 UNLOCK KEYBOARD 5 CONDITIONAL INDICATOR IN USER PROGRAM
ERROR RETRY COUNT: (2) SPECIAL EDITING CHARACTERS (1): 1 NO 2 YES
SPECIAL DISPLAY CONTROL? (1): 1 NO 2 YES
DO YOU WISH TO SPECIFY AN ERROR MESSAGE FIELD? (1) 1 NO 2 YES
DISPLAY RETENTION ON ALL FIELDS? (1): 1 NO 2 YES
FUNCTION OR COMMAND KEYS TO BE DEFINED? (1) 1 NO 2 YES
DOES THIS FORMAT HAVE A NON-DISPLAYED CONSTANT? (1): 1 NO 2 YES
```

Figure 7–4. Screen Format Generator Home Screens

To assist you in generating screen formats, the screen format generator provides a help facility. This facility allows you, at any point in the screen formatting procedure, to request prompting information on building a screen. If you request assistance at the beginning of the procedure, an explanation of the basics of screen formatting is displayed at the workstation. Once you understand the displayed explanation, the operation you were performing is redisplayed and you can continue generating your screens.

The screens that you create are automatically stored in the system screen format library, $Y$FMT, or you can store the screen formats in your own library. If you use your own library, that library must be a MIRAM library. If using your own library, you must include a device assignment set for that library in the job control stream for the program that uses the screens. You assign a name to each screen that you generate and it is this name that is used to access the screen format. Figure 7–5 shows the relationship of the components used during the screen format generation procedure.

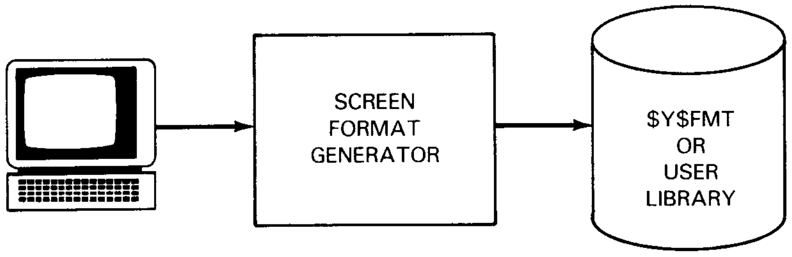


Figure 7-5. Relationship of Components during Screen Format Generation

If your program uses screen formats for data input, the program itself must include the appropriate commands, statements, or verbs to access and display the formats and to access the data records generated. In addition, the job control stream that executes that program must include the job control statements that relate to the screen formats. COBOL, RPG II, FORTRAN IV, and BAL include facilities for accessing and displaying formats and the data generated. You must also make certain that the data records generated by the screen formats match the record formats defined in your program.

The job control requirements for your program are:

- A device assignment set for the workstation or workstations that are inputting data to the program.
- The job control statement for using the screen formats:

  ```
  USE SFS
  ```

  This is included in the device assignment set for the workstation.
- A device assignment set for the program library holding the screen formats if it is not $Y$FMT.

During the execution of a program that uses screen formats, an internal system component, called the screen format coordinator, handles the transfer of data between the program and the screen format. The coordinator is responsible for accessing and displaying the requested screens and ensures that the data is in the proper format. The screen format coordinator is also responsible for detecting errors directly relating to screen format processing. When it detects an error, it displays a message so that you can take the appropriate corrective action.

Screen formats often prove to be the easiest and quickest way to have variable data entered into a program. This is especially so when those entering the data are nonprogramming personnel. The facilities provided by Sperry Univac make it easy for you to take advantage of this and include screen formats in your own programs. If you wish to find out more about screen formatting, see the screen formatting concepts and facilities, UP-8802 (current version).

7.3. DIALOG SPECIFICATION LANGUAGE

The dialog specification language allows you to generate dialogs to be displayed on the workstation screen. You can include dialog sessions for your own applications to simplify data entry, to eliminate data entry errors, to ensure uniform data entry, and to make the system accessible to inexperienced personnel. Not only can you write new programs to use dialog input, but your existing programs can take advantage of dialogs with little or no modification to the source programs themselves. The majority of changes when adapting existing programs occur in the job control associated with those programs. Figure 7-6 depicts the relationship of components used to generate dialogs.

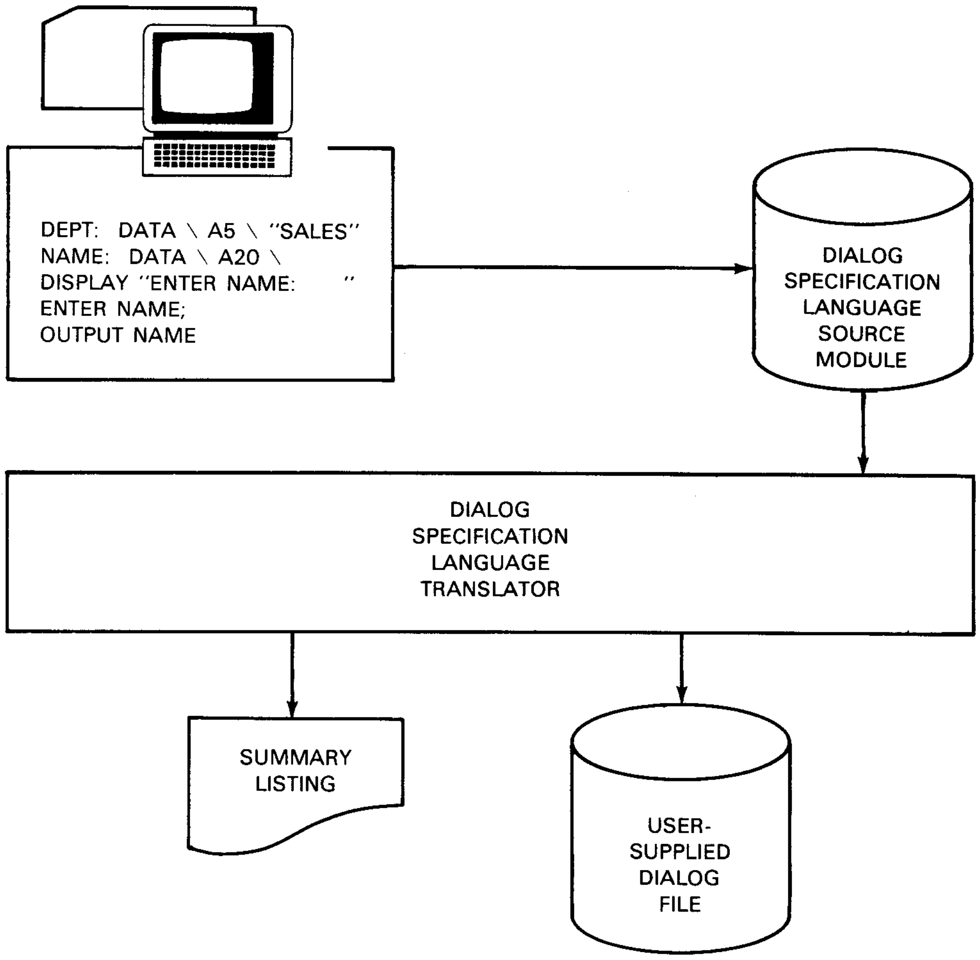

DEPT: DATA \ A5 \ "SALES"
NAME: DATA \ A20 \
DISPLAY "ENTER NAME: "
ENTER NAME;
OUTPUT NAME
DIALOG SPECIFICATION LANGUAGE SOURCE MODULE
DIALOG SPECIFICATION LANGUAGE TRANSLATOR
SUMMARY LISTING
USER-SUPPLIED DIALOG FILE

Figure 7-6. Relationship of Components during Dialog Generation

To generate a dialog, you first determine the information the dialog is to gather and how the dialog screens are to appear on the workstation. Having made this determination, you use the dialog specification language to write a program that generates the desired dialog screens. The dialog specification language program is written just like any other high level language program. A major difference is that the component that accepts your source code and translates it into machine executable code (the dialog specification language translator) only accepts input from a program library. Thus, you must place the source program into a library either using the general editor or the SAT librarian.

The translator places the translated source code, in the form of an encoded dialog, into a file that you supply. Once placed in the file, your applications program can access that dialog by specifying that file name on a job control statement. If your dialog is to output records to a program, you must format the output records so that they match the format used in the program.

In most cases, programs that utilize dialogs are executed in response to a workstation operator's request. When a request is issued, the system loads the program and a component called the dialog processor. The dialog processor locates and begins to display the dialog that the program uses. The dialog processor manages the dialog session by controlling the screen displays and passing the desired records to your application program.

One of the modifications you make to the job control is to substitute the dialog for the input file that the program normally uses. When your program issues an instruction to open the file and requests an input record, control is passed to the dialog processor which begins to display the dialog. The workstation operator's responses are used to construct a record and as soon as one record is complete, it is sent to the programs input buffer. At that point, control is passed back to the application program.

When another request for an input record is issued, the dialog resumes until another record is generated and sent to the input buffer. This process of generating and returning single records continues until the program issues an instruction to close the file. Figure 7-7 shows the relationship of the components used for dialog processing.

As shown in Figure 7-7, the dialog processor can produce a printed listing of each dialog session. This listing can include all user responses to dialog questions, the output records generated in response to the dialog, and other pertinent information. The printing and contents of this listing are both options.

The dialog processor includes a facility to store the responses made to a dialog in a file so that the next time that dialog is encountered, the dialog processor can be instructed to use the response stored in the file. This is called the audit facility and the file is referred to as the audit file.

The dialog processor can use all the responses or only selected responses stored in the audit file. The workstation operator specifies those portions of the dialog that are to use the audit file for responses and which are to be changed during the current session. You can also have the current changes included in the audit file. You indicate to the dialog processor that you wish to use the audit facility in the job control statement that executed the dialog processor.

If an audit file does not exist for that dialog session, your responses during the current session are entered in the audit file. If one exists, the dialog processor displays a screen to which you respond by indicating whether or not it should use the audit file and, if so, which portions of the dialog should be completed using the audit file. You are also asked at that point which responses to the current session are to be included in audit file.

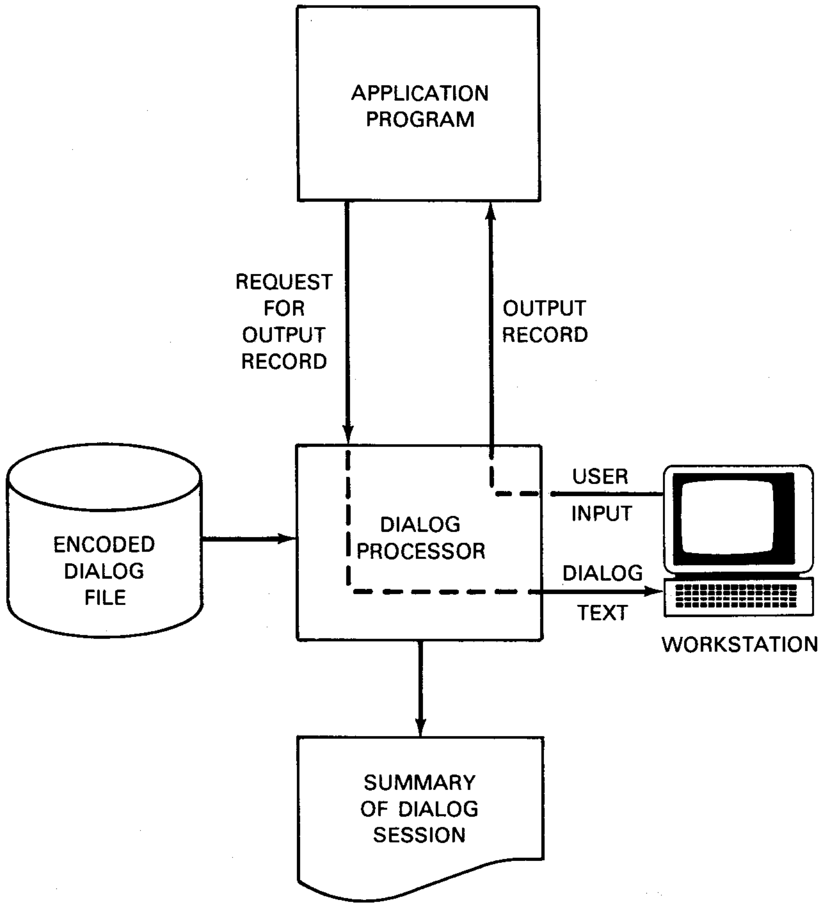


Figure 7-7. Dialog Processor Input and Output Flow

There are no source program requirements for using dialogs as input other than establishing a normal input file and matching the format for the input records to that of the records output by the dialog processor. The file name for the input file must match the LFD name specified in the device assignment set for the workstation or workstations to be using the dialog to input data. The job control stream that executes your program must include the following:

- A device assignment set for the workstation or workstations that input data to the program
- The // USE DP job control statement included in the device assignment set for the workstations
- A device assignment set for the dialog file. The LFD name specified in this device assignment set must match the dialog file name parameter specified on the USE statement.

Figure 7-8 shows the relationship between the job control stream and the applications program.

For more detailed information on using the dialog specification language to generate screen dialogs, see the dialog specification language user guide/programmer reference, UP-8806 (current version). For further information on the operation of the dialog processor, refer to the dialog processor user guide/programmer reference, UP-8858 (current version).

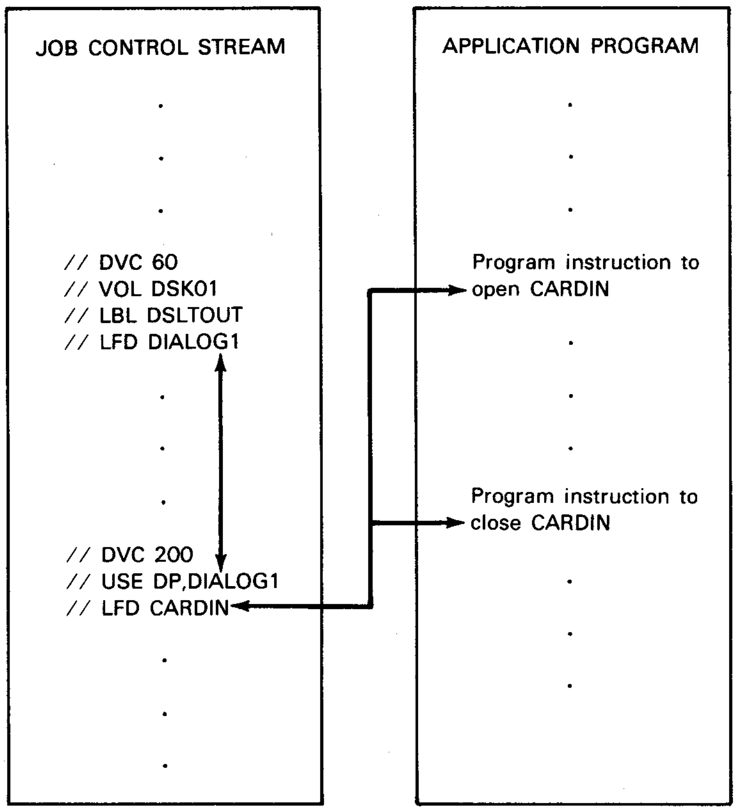


Figure 7-8. Relationship between the Job Control Stream and Application Program

7.4. MENU GENERATOR

The menu generator is an easy-to-use program product that lets you create and maintain menus interactively for operator use. With it you can create menus and store them in a menu library file for later use. With the menu generator, you can design the menu screen, specify exactly what action is to be taken for each item, and even create the help screens to go with the menu. And you can do all this in a single session that you begin by keying in the command MENUGEN. Figure 7-9 shows the relationship between the menu generator and menu library file.

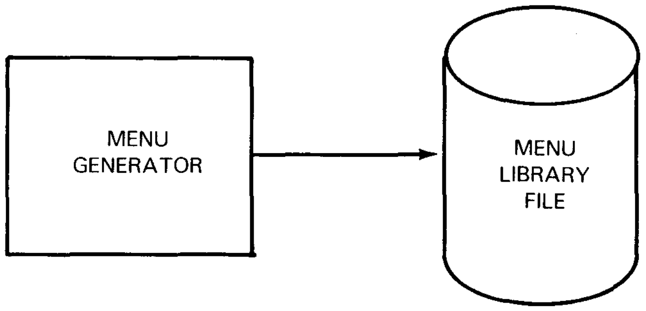


Figure 7-9. Relationship between Menu Generator and Menu Library File

After you create a menu, it's the job of an OS/3 component called the menu processor to display the menu, accept an operator reply, carry out the action intended by the reply, and display a help screen if the operator wishes it. When you call a menu from system mode with the MENU command, interactive services automatically calls upon the menu processor to display the desired menu. Figure 7-10 shows how the menu processor works in system mode.

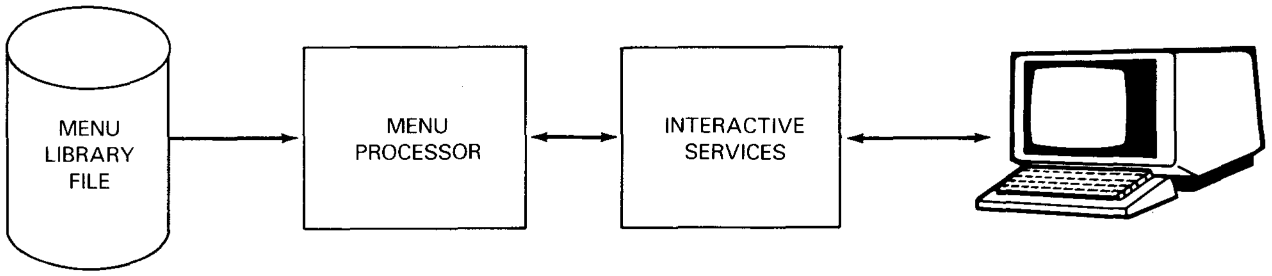


Figure 7-10. Menu Processor in System Mode

You can also adapt menus for use with programs. Typically, a program treats a menu as an input-only workstation file. When the program issues a request for input from that file, the menu processor automatically intercepts the request, displays the menu, and returns as input to the program a string of data that corresponds to the operator's choice. Figure 7-11 summarizes this process.

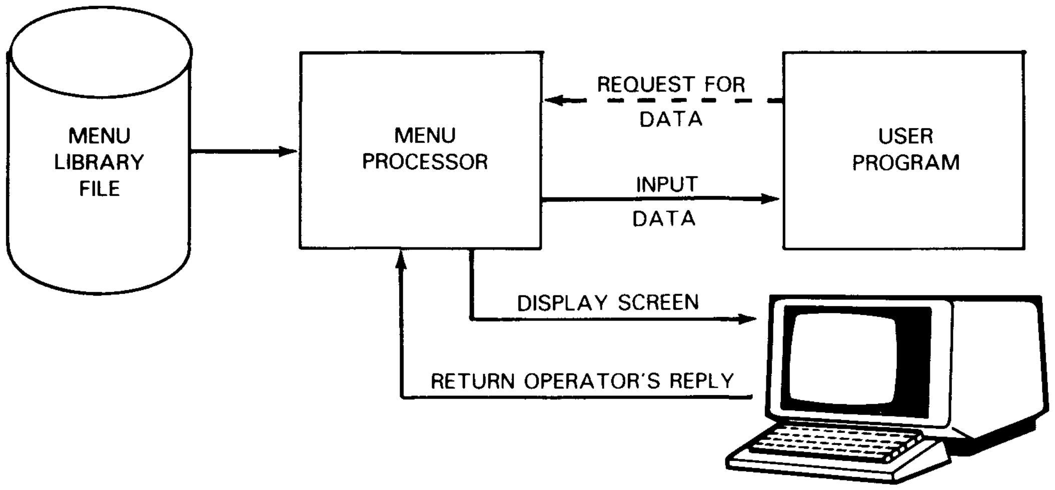


Figure 7-11. Menu Processor In Workstation Mode

It's easy to use menus with programs because the menu processor makes the menu file appear like a punch card file. So, if your program already accepts input from a file on punch cards, you won't have to change it. All you use is job control to link that file to the menu. The job control needed to do this includes:

- A device assignment set for the library file containing the menu if it is not $Y$FMT.

- A device assignment set for the workstation that is using the menu. The LFD name specified for this file is the same as the one used for the punch card file if one had originally been used.

- The job control statement for using menus:

  ```
  USE MENU
  ```

  This is included in the device assignment set for the workstation.

Unlike screen formats or dialogs, each menu item is programmed to send the same string of data to a program every time it is selected. The programming is done by the menu generator, and the data can only be changed by the menu generator. But since the menu returns only the data that you program into it, you can write your program to expect that same date without risk of unexpected input. This feature can help your system's security by guarding against wrong or unauthorized operator input.

For more information on creating menus and integrating them into your other OS/3 software, see the current verison of menu services concepts and facilites, UP-9317.

PART 4. SYSTEM UTILITIES

8. System Installation Facilities

8.1. SYSTEM INSTALLATION

System installation is the essential process of installing the SPERRY UNIVAC System 80 hardware, integrating with it the OS/3 software, and generating this software so that it fits your special needs.

8.1.1. Software Installation Facilities

Sperry Univac delivers your OS/3 software on release diskettes. Software installation involves transferring this delivered software to the integrated nonremovable disk pack in the System 80 processor complex. We call this disk pack your system resident volume, or SYSRES, because we designed it to contain all your system software and because it must be online when you operate the system. Appendix A lists the various system files contained in your system, along with descriptions of their type and use.

For software installation, Sperry Univac provides installation routines as part of your standard OS/3 release. These routines give you the ability to install:

- your initial release of OS/3 software;
- updated software as Sperry Univac releases major enhancements to OS/3; and
- any new software that you receive between major releases.

8.1.2. System Generation Facilities

System generation, or SYSGEN, is the process whereby you define your OS/3 hardware configuration and generate, or create, the control elements needed to satisfy your particular processing requirements.

Sperry Univac provides several facilities that simplify system generation:

- SYSGEN dialog
- SYSGEN parameter processor
- a set of SYSGEN job control streams.

The SYSGEN dialog is easy to use. It helps you prepare and process your SYSGEN parameters, or requirements, directly from your workstation. With it, you make SYSGEN parameter selections in response to queries displayed on your screen.

For parameters that you omit or specify incorrectly, OS/3 supplies default values that build a useful system. The dialog accepts your choices and supplies them to the SYSGEN parameter processor. The parameter processor, in turn, checks to see that your choices are correct and valid, generates a series of job control streams based on your selections, and lists these streams. When you execute the SYSGEN job control streams, using simple key-ins, they actually perform the system generation.

8.2. INSTALLATION VERIFICATION PROGRAMS

Sperry Univac supplies a series of installation verification programs to be run after SYSGEN is completed, ensuring the functional capability and operation of the various components included on the resident disk pack.

For more information on the installation verification programs, see the installation verification procedures user guide/programmer reference, UP-8820 (current version).

9. I/O Utilities

System 80 offers a number of utilities to assist you in using and maintaining the various devices and the files that are stored on these devices. The utilities range from the initialization utilities that prepare the devices for use to the librarians that maintain the various program libraries. The I/O utilities are:

- Disk, diskette, and tape preps
- Disk dump/restore
- 8419 disk copy
- Data utilities
- System librarians (SAT and MIRAM)

9.1. DISK, DISKETTE, AND TAPE INITIALIZATION

The various tape, disk, and diskette initialization, or prep routines check the condition of the magnetic storage media and prepare them for use. The disk and diskette prep routines respond to a set of keyword parameters inserted into the job control stream while the tape prep routine is executed with a tape parameter on the // VOL job control statement. Associated with the disk prep routine is a utility to automatically assign alternate tracks for each defective track identified by the disk prep routine. A complete discussion on the disk, diskette, and tape prep routines can be found in the system service programs user guide, UP-8841 (current version).

9.2. DISK DUMP/RESTORE

The disk dump/restore routine allows you to make backup copies of a disk volume. You can use dump/restore in either batch or interactive mode. To use it in batch mode, you run the program DMPRST and specify what you want it to do using parameters entered on punched cards. In batch mode, you can use dump/restore to:

- copy all or any part of a disk to a magnetic tape, diskette, or disk in sequential mode (a dump operation);
- copy a magnetic tape, disk (sequential), or diskette to a disk (a resotre operation);
- copy all or any part of a disk to another disk (a disk copy operation);
- copy a tape created by a previous dump/restore operation to another tape (a tape copy operation); or
- copy a diskette created by a previous dump/restore operation to another diskette (a diskette copy operation).

You can run the dump/restore routine interactively using the HU workstation command. What HU gets you is a series of menus and other types of screens with which you specify the operation you want performed, the files and volumes involved, and other information to guide the dump/restore routine in making the backup copies you want. Using dump/restore interactively, you can:

- copy some or all the files on a disk to a magnetic tape, disk (sequential), or diskette (a dump operation);
- copy some or all the files on a magnetic tape or diskette to a disk (a restore operation); or
- copy some or all the files on a disk to another disk volume (a disk copy operation).

More detailed data on the disk dump/restore routine can be found in the current version of the system service programs user guide, UP-8841.

9.3. 8419 DISK COPY

One dump/restore function lets you copy disks to other disks. Another routine for copying disk volumes is the 8419 disk copy routine (SU$C19). With SU$C19, you can create and verify up to six copies of a single 8419 disk, regardless of the disk's contents. Verifying ensures that your disk has been copied correctly by comparing the contents of the input volume with those of the output volume. You can also use SU$C19 in a separate operation to verify copies that have been made by a previous SU$C19 copy operation or by a disk copy operation performed using the dum/restore routine.

You can use SU$C19 in either batch or interactive mode. To use it in batch mode, you execute the SU$C19 program and specify what you want it to do with parameters entered on punched cards. To use it in interactive mode, you key in the HU workstation command, just as you do with dump/restore. You then see a series of menus and other screens that ask you what operation you want to perform, what volumes to use, and so on. Though you call SU$C19 interactively the same way you call dump/restore, the two routines operate apart from each other.

The SU$C19 routine is described in the current version of the system service programs user guide, UP-8841.

9.4. DATA UTILITIES

The data utilities program is a straightforward, easy-to-use method for reproducing and maintaining your data files. It provides the capabilities of transferring files between the various peripheral devices and editing or correcting data files. Figure 9-1 shows the data utilities operation.

You can use the data utility program via either a dialog presented on the workstation or through cards. When using a dialog, a series of questions is displayed on the workstation screen for you to answer. If you don't understand a question, a *help* screen is displayed which further defines the question. After you have answered all the questions, processing begins.

When using cards, there is a control statement for each type of operation (card-to-tape, disk-to-disk, etc.). Along with the control statement, there are parameters which further define the operations.

For more information on data utilities, see the data utilities user guide/programmer reference, UP-8834 (current version).

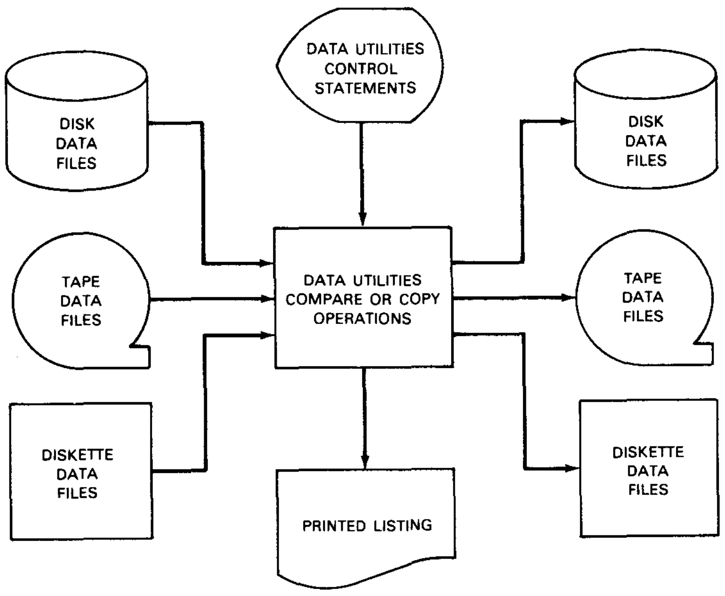


Figure 9—1. Block Diagram of Data Utilities Operation

9.5. SYSTEM LIBRARIANS

The system librarians are used to maintain your program libraries. Because there are two types of libraries, MIRAM and SAT, the system supports two librarians, one for each library type. By using the librarians, you can perform such functions as adding new program modules to a library, deleting unwanted program modules, or printing a listing of the contents of a program module.

Librarian operations are executed as jobs; therefore, you must supply a job control stream. Both librarians are controlled through control statements and associated parameters and these are embedded within the job control stream. In addition, any program libraries that you are going to manipulate, other than system libraries ($Y$LOD, $Y$OBJ, etc.), must be identified in the executing job control stream through the appropriate device assignment set. The LFD name you declare for each file must match the file name you specify in the file declaration librarian control statement.

The capabilities of each librarian differs. The following subsections describe the capabilities of each. For a more complete description of the SAT and MIRAM librarians, refer to the current version of the system service programs user guide, UP-8841.

9.5.1. SAT Librarian

The SAT librarian is used to maintain program libraries organized as system access technique (SAT) files. It can be used to perform such functions as:

- adding or deleting modules from a program library;
- copying, comparing, and renaming modules;
- correcting the contents of a module;
- printing the contents of a module; or
- placing sequence numbers on the records of a source module.

The librarian allows you to place modules within a program library into a group and assign a group name. Grouping allows you to process defined groups of modules as a single element. You can also perform librarian operations on modules of the same type within a library. This is called gang operation and allows you to process all the source, object, load modules, etc. within the program library as a single element.

You can optionally specify that the librarian is to produce a printed listing of all operations performed during the current librarian operation. This listing is called a librarian map and can contain such items as:

- the contents of specified program libraries;

- librarian control statements used during current session; and
- appropriate diagnostic messages.

The librarian executes as a job and, thus, requires a job control stream. A librarian control stream might look like this:

```
// JOB LIBJOB
// DVC 2Ø  // LFD PRNTR
// EXEC LIBS
/$
        .
        .  } librarian control statements
        .
/*
/&
```

The librarian requires a printer for execution and is executed by specifying the librarian program name, LIBS, on the EXEC job control statement. The librarian control statements are placed in the control stream as embedded data immediately following the execute statement.

9.5.2. MIRAM Librarian

The MIRAM librarian is used to perform maintenance functions on program libraries organized as MIRAM files. Program libraries containing screen format modules or expanded saved job stream modules and MIRAM program libraries generated by the general editor can be maintained by the MIRAM librarian. Using this librarian, you can perform the following functions:

- delete or change the names of modules;
- copy modules from one library to another; and
- print the contents of a module or a library directory.

The MIRAM librarian responds to a set of control statements. These control statements are placed within the control stream that executed the librarian. To execute the librarian, you specify the MIRAM librarian program name, MLIB, on the execute job control statement:

```
// EXEC MLIB
```

The appropriate control statements would follow immediately as embedded data.

10. Support Operations

10.1. FILE CATALOGING FACILITY

The file cataloging facility allows you to protect your files from unauthorized use by assigning passwords to the file. In addition, it can simplify the job control requirements for referencing a file. The file cataloging facility consists of:

- the system catalog file; and
- the catalog manipulation utility.

Files are cataloged by using the // CAT job control statement in conjunction with the device assignment set of the file to be cataloged. When you catalog a file, the device assignment set is placed in the system catalog file along with the passwords you are using to protect the file. Once cataloged with passwords, only those knowing the passwords can access it. In addition, cataloged files can be accessed by using only one job control statement (LBL) instead of the entire device assignment set. Files are decataloged through the // DECAT job control statement.

The file cataloging facility also allows you to maintain easy recording and processing of generation files. Generation files consist of the various update levels of the same file. The file identifier remains the same for each update level, or generation; however, a unique 2-digit generation number is appended to the file identifier for each generation. Generation files allow you to retain noncurrent versions of a data file as backup and eliminate the need for your computer operators to keep records of backup files.

The catalog manipulation utility (JC$CAT) allows the system administrator to obtain a printed listing of the catalog to copy the catalog to another file as backup. The system administrator can also assign a password to the catalog through JC$CAT. Once the password is protected, only those who know the password can catalog a file or access the catalog through JC$CAT.

For more information on file cataloging, refer to the current version of the file cataloging concepts and facilities, UP-8860.

10.2. SORTING

To meet your sorting needs, Sperry Univac provides two easy-to-use, yet sophisticated sort programs:

- Sort/merge
- SORT3

Sort/merge provides extensive file sorting and merging facilities; it can be run as an independent operation under the control of job control or as a subroutine to another program. SORT3 is functionally equivalent to the IBM System/3 sort routine.

10.2.1. Sort/Merge

Sort/merge is a system program that can be run as an independent operation under the control of job control or as a subroutine to another program. Either way, it provides the same capabilities:

- Establishing an interface that permits disk or magnetic tape to be used as workareas
- Handling input and output on disk, diskette, or magnetic tape
- Sorting of blocked or unblocked records
- Sorting of fixed-length or variable-length records
- Handling of seven types of key field formats:
  - Character
  - Binary (signed or unsigned)
  - EBCDIC data in ASCII collating sequence
  - Decimal (signed zoned or unsigned zoned)
  - Leading and trailing sign numeric
  - Overpunched leading and trailing sign numeric
  - Floating point (single and double precision)
- Specifying up to 255 key fields
- Sorting of noncontiguous key fields in ascending or descending sequence

- Specifying an alternate collating sequence
- Executing input and output own code
- Sorting of two or more different characters having the same collating value (multiple character sort)
- Using shared input and reserved output devices
- Performing data validity and data integrity checks during sorting
- Providing convenient restart procedure

Detailed information on using sort/merge as an independent routine can be found in the independent sort/merge user guide/programmer reference, UP-8819 (current version). For more information on using sort/merge as a program subroutine, see the current version of the sort/merge macroinstructions user guide/programmer reference, UP-9072.

10.2.2. SORT3

SORT3 is functionally equivalent to the IBM System/3 sort program and accepts control input in the same form. It is an independent program that runs under the control of job control. SORT3 can process disk, diskette, and tape files and card input. It performs the following functions:

- Rearranges the records in a file
- Selects specific records from a file
- Reformats the records in a file
- Summarizes fields in the records

SORT3 is capable of performing three different types of sorts:

- Full record sort
- Tag sort
- Summary sort

The output from the full record sort is 10-byte (binary) relative record numbers of the records in the input file. The tag sort output is a file of sorted records that can contain control fields and data, control fields only, or data only. And the output of the summary sort can be any of the following:

- Control fields, data fields, and summary data
- Control fields only
- Data fields only
- Data fields and summary data
- Summary data fields only
- Control fields and summary data fields

More detailed information on the operation and use of SORT3 can be found in the current version of the SORT3 user guide/programmer reference, UP-8836.

10.3. SPOOLING AND JOB ACCOUNTING

Spooling (simultaneous peripheral operations online) is an optional system feature that increases your system throughput. With it, your programs can read from and write to low-speed devices – like card readers and printers – while taking advantage of the greater speed of the disk.

To see how useful spooling is, consider a program that reads data from a card reader and writes output to a line printer. First, consider how the program works in a system without spooling. In such a system, all low-speed devices work directly with the programs using them.

Since a low-speed device cannot be shared among different programs, it's necessary to dedicate it to a single program for the entire time it takes to run that program. This means that the program has to wait for a printer to become available before it runs. Then when it does run, other programs have to wait until it finishes before they can get their turn on the printer.

What makes this arrangement doubly inefficient is that large time gaps are possible between consecutive operations on a device. For example, the program may print one line on a printer, process some data for 10 seconds, then print the next line. Those 10 seconds of idle time can be put to better use. And with spooling, they are.

Now consider the same program, using a card reader and printer, in a system that has spooling. First, before even running the program, you transfer the card data to a disk file called the spool file. Then you run the program. When it asks for card input, the program actually gets it from the spool file, at a speed far greater than that attainable with the card reader itself. Likewise, when it outputs data to the line printer, the program is really transferring data to the spool file, again at a speed greater than would be possible going directly to the printer. Later, the system automatically transfers the disk data to the line printer. Aside from speed, this spooling feature makes it possible to use a single printer to print the output of multiple programs, one after another.

What's important to remember is that all this proceeds without any knowledge on the part of the program. While input data to the program is actually coming from a disk, it seems to the program that the data is coming from a card reader. That's why we call such an input file a virtual card reader. And output to the spool file on disk seems to the program to be going to a printer; hence, we call such a file a virtual printer. You can use spooling for input or output without having to change any part of the program. Figure 10-1 graphically represents the spooling operation.

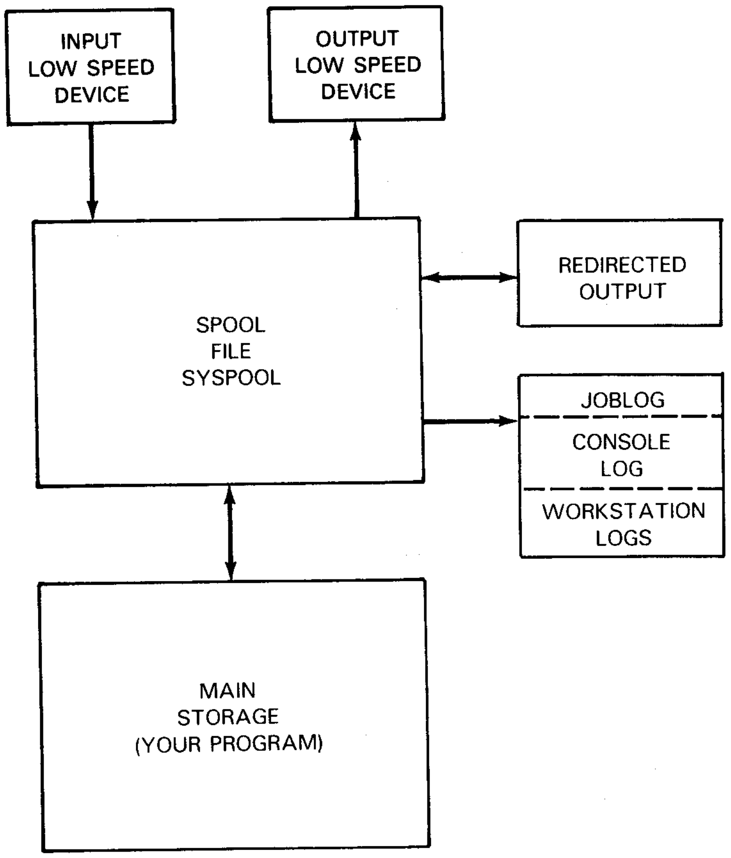


Figure 10—1. Flow of Information between Main Storage and Low Speed Devices in a System Configured with Spooling

Low speed devices supported by spooling include such local devices as the system printer, card reader, diskette in data set label mode, and card punch. Spooling also supports remote printers, readers, and punches. The transfer facility of distributed data processing (13.2.1) lets you transfer spool files from one system to another. And in addition, spooling can transfer program files to a workstation for output on an auxiliary printer attached to the workstation.

There is only one spool file on your system. It is divided into subfiles, each of which is a collection of data that acts as a file to a program. For instance, in our example above, each time your program reads an input file, it is reading a subfile that had earlier come from a card reader. And each time the program outputs data to the spool file, it is creating a subfile for later output to a low-speed device.

Your spool file is a permanent part of your system. For the most efficient use of disk space, its subfiles are dynamically created and destroyed as the need arises. Once the contents of a subfile are read by a program (on input) or written to the intended device (on output), the data normally disappears. There are two exceptions:

1. You can redirect output data to a disk, diskette, or tape. Later, you input that same data back into spooling for output to a low-speed device. Although this action destroys the output subfile, its data is preserved for later use.

2. You can also retain an input or output subfile in the spool file after its data has been read or written, when it would otherwise be destroyed. This enables you to use the same subfile over and over again.

The system operator has at his disposal a number of spooling commands that control input and output spooling. Many of these commands permit spooling to be selective about the files it processes; all files produced by a particular job, for example, or all files destined for a particular type of printer. A workstation operator has some control over spooling, too. If his workstation has an auxiliary printer attached to it, he can use it to print output files, produced by jobs he has run, right where he sits.

As we said, spooling frees you from the need to have a printer available on the spot when you run a program. You can redirect spooled output for later printing on your system printer; or you can send the output to a remote printer, or even to another system. This gives rise to another spooling feature - indirect printers.

You can design your System 80 without any printers at all physically attached to it. This feature is usedful in situations where you may wish to generate output at one System 80 but perform the actual printing at another System 80. You configure indirect printers at system installation time, a process described in the current version of the system installation user guide/programmer reference.

Along with spooling, you can also get job accounting information. This information includes such items as:

- number of I/O operations,
- CPU time used,
- number of transient requests, and
- supervisor interrupts.

You can use this information for billing purposes or to create a new job mix for more efficient use of the system. The information is listed via the system log accumulation routine and the job log report program.

More information concerning the usage and operation of spooling services is discussed in the spooling and job accounting concepts and facilities, UP-8869 (current version).

10.4. SOFTWARE MAINTENANCE PACKAGES

To enhance the reliability of your System 80, Sperry Univac periodically sends you software changes in diskette form which are called software maintenance packages (SMPs). You are responsible for installing each SMP within a specified period of time of receiving it.

These packages, thoroughly tested by Sperry Univac, enhance your system performance with no loss in productivity because you install them yourself with easy-to-use canned job control streams. The SMP installation procedure gives you several options:

- Apply an SMP to the system.
- Print the SMP document, which gives you guidelines for installing an SMP, such as the estimated time needed to complete installation.
- Remove a previously installed SMP. This option is useful if the SMP causes problems in the system after it has been installed. To make this feature possible, all software elements affected by an SMP are copied to a backup file so that they can be restored if the SMP fails.
- Display the correction log, which is a history of all SMPs previously installed in your system.
- Select and apply optional corrections.
- Automatically regenerate the supervisor or ICAM as required.

The SMP installation procedure displays menu screens to help you use many of these options. For more information on installing SMPs, see the system installation user guide, UP-8839 (current version).

10.5. SECURITY MAINTENANCE UTILITY

The security maintenance utility lets the system administrator control access to your system's interactive facilities through the security information in the system security file ($Y$SEC). The utility performs its tasks by creating user profiles and execution profiles. A user profile contains security and accounting information and execution profile names. An execution profile contains interactive commands that are automatically executed at logon time.

The user profile contains a user-id, which identifies the user to the system. It may also contain any of the following:

- A password that, with the user-id, controls the user's access to the system
- An account number that identifies the accounts being charged for computer time
- The name of a default execution profile to be used at logon time

When a user logs on, the system checks the user profile to determine whether the user is allowed on. If so, the system then looks for the execution profile name, either one specified at logon time or a default name in the user profile. If the system finds a name, the interactive commands in the execution profile are automatically executed. If, however, the system does not find a name, the workstation continues normal processing.

For more information on the security maintenance utility, see the current version of the security maintenance utility user guide/programmer reference, UP-8823.

10.6. SYSTEM ACTIVITY MONITOR

The system activity monitor (SAM) is an OS/3 product that lets you monitor and record your system's activity. It aids in the detection of production bottlenecks, optimizes production job mixes, and identifies and changes system variables that influence system performance. For instance, it helps you determine whether you are making optimum use of disks or whether your printer is keeping up with production.

SAM consists of two components: a data collection symbiont and a stand-alone report program called SAMRPT. The data collection symbiont continuously records statistical data about the operating system for output to the system console and, optionally, to a disk file. With program SAMRPT, you can print out the disk file data in formats tailored to your needs.

For more information on the system activity monitor, see the current version of the system activity monitor user guide/programmer reference, UP-8812.

11. Diagnostics

The operating system provides a number of diagnostic tools to help you determine the nature of abnormal conditions within the system itself or the jobs or programs that are running. These diagnostic routines can be implemented (1) prior to the execution of a job as a precaution, (2) after the unsuccessful completion of a job, or (3) dynamically as the abnormal condition persists in the system.

The diagnostic routines consist of:

- Dump routines
- Program error checking
- Error logging
- Hardware diagnostics

The following subsections describe the diagnostic routines available and detail their functions and use.

11.1. DUMP ROUTINES

A dump is a printed hexadecimal image of the system main storage at the time the problem occurred. By examining a dump, you can determine why a particular problem occurred and whether or not you need the assistance of a Sperry Univac representative. OS/3 provides three types of dumps. While all are basically hexadecimal images of main storage at the time they were taken, they differ in scope and purpose as follows:

- SYSDUMP

  Dumps all or part of main storage and is run in two phases: main storage write and dump printout. The printout provides a picture of your system in charts and text.

- JOBDUMP

  Dumps a user's job region upon abnormal termination of the job or execution of a DUMP or CANCEL macroinstruction. The dump is supplemented by charts and text interpreting the state of the job.

- EOJ dump

  The end-of-job (EOJ) dump dumps a user job region without the charts and text, but includes the contents of the registers and the program status word (PSW).

The following subsections briefly describe the various dumps and show how you can get the dump that best suits your needs. A complete discussion of the SYSDUMP, JOBDUMP, and EOJ dump routines is presented in the current version of the dump analysis user guide/programmer reference, UP-8837.

11.1.1. System Dump (SYSDUMP)

A system dump is a printout showing the complete contents of your system's main storage. The system dump listing is divided into several parts each corresponding to a system component. The sections are clearly labeled with the appropriate heading for ease of use. A typical system dump listing includes:

- the contents of low order storage;
- the physical unit, system information, and channel control blocks;
- the system switch list;
- translated job region;
- the supervisor;
- hexadecimal job region; and
- free region.

The system generates a system dump in two steps. First, it writes a copy of main storage on the $Y$DUMP file on the SYSRES volume. Then a system program uses the information in $YRDUMP to print the system dump listing.

There are several ways to obtain a system dump listing. You can request a listing using the // OPTION SYSDUMP statement in a job control stream. Then, if your job terminates abnormally or if it includes an assembler program that issues a CANCEL or DUMP macroinstruction, a SYSDUMP listing results. Also, if the system halts, you can perform the main storage write step from the console workstation, reload the system, and get a SYSDUMP listing by entering the RV SYSDUMPO command.

Another way to obtain a system dump listing is by entering the SYSDUMP command, which automatically performs the main storage write step and schedules a job to print the system dump listing. Finally, some supervisor routines automatically generate a system dump when a supervisor error occurs.

No matter how you obtain a system dump, the system dump program gives you control over the format and makeup of the printed output. It also lets you save the contents of the system dump by writing it to a file on diskette or tape and, later, copying that file back into the same system or one at another site. With this ability, you can move system dumps between sites in compact diskette or tape form.

11.1.2. Job Dump (JOBDUMP)

A job dump gives you a listing of the state of the job region at the time your job terminated abnormally or crashed. It consists of:

- a translated listing of the state of the job region presented as charts and text; and
- a label hexadecimal/character main storage dump.

To obtain a job dump, include the // OPTION JOBDUMP job control statement in your job control stream. You will receive a job dump if your job terminates abnormally or if it includes an assembler program that issues a CANCEL or DUMP macroinstruction.

An abbreviated job dump, called ABRDUMP, provides you with a shortened listing of the full job dump. Only the area in the vicinity of the last instruction executed is shown. The ABRDUMP is initiated by the // OPTION ABRDUMP job control statement.

11.1.3. EOJ Dump

An EOJ dump gives you a hexadecimal listing divided into four sections: problem program registers, job preamble, task control blocks, and your program region. In addition, the dump gives you the program status word (PSW) at interrupt time, the error code that caused the abnormal termination, and the next task control block (TCB).

11.2. PROGRAM ERROR CHECKING (UPSI BYTE)

The OS/3 system provides every job with a 12-byte communications region residing in the job preamble. The last byte of this region is the user program switch indicator (UPSI). The UPSI byte is used to pass information from one job step to the next job step and to indicate the presence of program errors. The librarian, the linkage editor, the utilities and dump routines, and other executable system components set the UPSI byte if errors are detected. You can test the UPSI byte during program execution to determine the nature and severity of any errors.

The UPSI byte can be useful in contingency error processing. For example, the byte can be examined and, if certain conditions prevail, can cause a branch to error handling routines. The SKIP job control statement is used to perform the test. For more information on using the UPSI byte, see the system service programs user guide, UP-8841 (current version).

11.3. ERROR LOGGING

Error logging provides the mechanism to record software detected hardware errors in the system error log file, $Y$ELOG. Information placed in $Y$ELOG can be subsequently retrieved and statistical reports prepared. The type of hardware errors that can be logged include:

- Peripheral device errors
- Machine check errors
- Communication errors
- User specified errors

These errors, when detected, are buffered in main storage and placed in the error log file by a supervisor transient routine.

The number of resident main storage buffers is determined during SYSGEN. At the beginning of each session, you may specify whether the current error log file is to be saved. If saved, detected errors are added starting after the last recorded error from the previous session.

At the end of each session, you should run a job to retrieve the logged errors from the error log file and either transfer them to another permanent file or produce a printed listing. Sperry Univac supplies the canned job control stream ONUERL for this purpose.

You can change the types of records that are to be logged during operation of the system by using the SET ELOG console command. By using this command you can turn off the error logging facility, stop the logging of previously logged errors, or commence the logging of previously unlogged errors.

11.4. HARDWARE DIAGNOSTICS

The hardware diagnostics are used by you and Sperry Univac personnel to isolate and identify system hardware faults. The hardware diagnostic system consists of:

- Resident diagnostics
- Offline diagnostics
- Online diagnostics

The hardware diagnostics routines are designed to operate in an interactive programming environment and make use of displayed dialogs and messages for initiation and entry of console information.

The resident diagnostic routines are an integral part of the system control software and perform basis error checking on the central processor complex. The central processor complex includes the central processor, control storage, and the resident disk control hardware. The resident diagnostics also check the paper peripheral, diskette, and workstation controllers along with the single line communications adapter. These checks are performed automatically whenever the system is powered on or reset.

The offline diagnostic routines consist of microdiagnostic and macrodiagnostic routines. The microdiagnostic routines isolate detected hardware faults to the particular component responsible. Initiated at the workstation, these routines diagnose problems in the central processor and control storage and main storage. The macrodiagnostics check the operability of the complete repertoire of system instructions. Initiated at the workstation, they can additionally be used to verify the basic operational soundness of the integrated 8417 disk subsystem.

The online diagnostic routines check all peripheral and communications devices to ensure proper functioning and operation. These routines operate under control of the operating system and run concurrent with user jobs. During normal system operation, errors are logged to allow interrogation and analysis by these routines. Error log edit and analysis programs are available to display individual error events and summary information.

PART 5. APPLICATIONS PROGRAMS

12. Applications Programs

Sperry Univac offers a wide variety of applications programs with the System 80 data processing system. These programs are designed to meet the data processing requirements of particular applications. The list of available packages varies as Sperry Univac institutes changes to meet the needs of its customers. The applications programs now available are:

- Univac Industrial System 80
- Information Collection System 80
- Univac Distribution Information System - Wholesale
- Order Entry 80
- Univac Financial Accounting System 80
- Accounting Management System
- Wholesale Applications Management System 80

12.1. UNIVAC INDUSTRIAL SYSTEM 80

The Univac Industrial System 80 (UNIS 80) is a comprehensive production and inventory control system. The modular construction of UNIS 80 allows you to implement those specific features needed to meet your requirements; UNIS 80 provides both online and batch features and uses database technology.

UNIS 80 includes the following functions:

- Bill of material processing provides the capability to add, delete, change, and copy product-defining records. Indented, summary, and single-level explosions and where-used lists are included.
- Standard routine processing provides the capability to add, delete, change, and copy manufacturing routing data. Retrieval capability is included.

- Tool data processing provides the capability to add, delete, and change tool master and tool reference data. A tool where-used report and tool lists are included.
- Standard cost buildup includes the ability to calculate the standard cost of a part based upon material cost and standard routings in both regenerative and net change modes.
- Inventory posting and control includes the ability to record stock issues and receipts, work and purchase order data, or vendor information.
- Forecasting utilizes exponential smoothing techniques and provides constant, trend, seasonal, and mixed models. Model analysis capability is available to determine the proper model and starting values based upon past history.
- Material requirements planning provides both net change and regenerative processing.
- Stock and order monitoring provides replenishment order recommendations selectively and for all parts.
- Customer order entry and control capability is provided. This includes: online maintenance of customer and customer order data; line item pricing; availability checking; pick lists; picking confirmation; back order generation and control; shipment notification; order status review and update; and order control reports.
- ABC analysis provides an ABC ranking of parts based upon usage at standard cost.
- Work order release both automatically and upon request is available.
- Work order scheduling provides the capability to use splitting, overlapping, queue time reduction, and backward and forward scheduling. UNIS 80 also provides network scheduling capability.
- Infinite capacity planning shows the workload by work center without regard to work center capacity. Detail and summary reports are provided.
- Finite capacity planning loads each work center to the limit of its capacity and reschedules work orders which exceed that capacity. Both detail and summary reports are produced.
- Master schedule load analysis determines the production capacity required by the master schedule.
- Shop reporting of time, quantity, operation, and job is provided.
- Dispatch lists are provided listing the jobs to be run in each work center in priority sequence.
- Work order status information may be obtained upon request.

12.2. UNIVAC Industrial System 80 – Extended

The Univac Industrial System 80 – Extended (UNIS 80-E) is a version of the UNIS 80 system offering all of the features of that system. In addition, UNIS 80–E allows the user to modify the system to meet particular needs. Included in the system are the following UNIS 80 modules:

- Production engineering data management
- Product costing
- Customer order processing
- Inventory status and control
- Forecasting and analysis
- Master scheduling
- Material requirements management
- Production planning
- Work order control

12.3. INFORMATION COLLECTION SYSTEM 80

The Information Collection System 80 (ICS 80) is a highly efficient online information collection system implemented as action programs to IMS. It offers a practical and economical solution to information and data collection problems. The system helps to ensure optimum utilization of computer resources, aids in the introduction of online information processing, and provides the following important capabilities:

- Online collection of data and information. The same workstation used for the collection of data and information may also be used for file inquiry and updating.
- A full range of data validation and checking routines. These may be application-dependent checks specified by you.
- Information and data collection handled simultaneously with other processing in a multiprogramming environment.
- A simple implementation language for specifying the formats to be used to enter data.

12.4. UNIVAC DISTRIBUTION INFORMATION SYSTEM - WHOLESALE

The Univac Distribution Information System - Wholesale (UNIDIS - WHOLESALE) offers a complete distribution control system designed to optimize cash flow, increase profits, streamline operations, and improve customer service. UNIDIS ordering strategies maximize inventory while holding down cost, and offers positive control over all goods flowing into and out of your organization. It is a real-time system allowing you to respond immediately to customer orders and retrieve billing information quickly and easily.

UNIDIS functions can be broken into order entry, stock control, and inventory management as follows:

- Order Entry

  UNIDIS offers real-time, online order entry with immediate response capability. In addition, order entry offers:

  - Online availability determination/reservation
  - Specialized delivery instructions and comments including item substitution
  - Standard ship-to and bill-to address
  - Automatic discount and pricing capabilities
  - Profitability control
  - Customer credit limit control
  - Automatic pick list generation and route optimization
  - Inventory and demand history updating
  - Pre- and post-billing accounting procedures
  - Blanket order, back order, and drop shipment processing
  - Invoice transactions
  - Automatic assignment of customer order identifiers

- Stock Control

  Stock control provides for the control of goods from their receipt on the shipping dock, through count verification, inspection, repair, and rework until the goods reach stock or scrap. Stock control offers the following:

  - Online processing
  - Receipt verification against purchase orders
  - Quantity tolerance verification
  - System control of goods movement
  - Inventory updating
  - Generation of financial transactions
  - Location control of all goods, stock, and nonstock

- Inventory Management

  Inventory management provides a sophisticated set of statistical features to analyze demand patterns and suggest replenishment strategies. The system provides the following features:

  - Demand models with automatic model analysis
  - Service level specification
  - EOQ calculation
  - Tracking signal/demand filters
  - Alarm reports
  - Graphic representation of demand patterns and forecast model
  - Product group of warehouse processing

12.5. UNIVAC FINANCIAL ACCOUNTING SYSTEM 80

The UNIVAC Financial Accounting System 80 (UNIFACS 80) is a group of packaged application programs, written in COBOL, that provides the user with all the standard features and functions of a business accounting package and gives the user added capabilities in the areas of personnel record-keeping and budgeting. Some of its function are:

- Accounts Payable

The accounts payable subsystem provides all functions needed to maintain accounts payable records, select invoices for payment, issue appropriate checks, and provide product reports needed for control and future planning. The subsystem performs such functions as invoice entry and validation, duplicate invoice monitoring, recurring payment processing, cash requirements forecasting, check printing, invoice aging, federal form 1099 reporting, and use tax reporting.

- Accounts Receivable

The accounts receivable subsystem provides all functions needed to maintain accounts receivable records. The subsystem is designed to allow maximum control and flexibility in processing cash payments and in controlling outstanding receivables.

General ledger transactions created by the accounts receivable subsystem are accepted directly by the UNIFACS 80 general ledger subsystem. The accounts receivable subsystem also creates the sales history necessary for sales analysis functions.

- Payroll/Personnel

The payroll/personnel subsystem provides a comprehensive, easy-to-use payroll processing system, with additional features for personnel record-keeping. Payroll/personnel supports a variety of pay categories and all federal and state tax calculations. It can process up to 20 regular deductions per employee, manually override pay rates, and process hand-written and voided checks.

The payroll/personnel subsystem also provides for labor distribution, payroll distribution, EEO reporting, and personnel history maintenace.

- General Ledger/Budgeting

The general ledger/budgeting subsystem maintains all necessary general ledger and budgeting information for many types of users, including those with multicompany requirements. The subsystem can process different companies, divisions, departments, etc., as well as any combination of accounting periods in the same run.

General ledger/budgeting is capable of handling current period transactions for financial systems ranging from 4 quarterly periods per fiscal year up to 13 four-week periods, including the standard 12 monthly periods. The general ledger/budgeting subsystem accepts automated input from the UNIFACS 80 accounts payable, accounts receivable, and payroll/personnel subsystems.

In addition to the general ledger functions, the subsystem provides the user with budgeting functions. Multiple budgets may be created and maintained. Customized budget reports and actual-versus-budget comparisons are available through the use of the financial reporter budgeting subsystem.

12.6. ACCOUNTING MANAGEMENT SYSTEM

The Accounting Management System (AMS) provides a group of packaged financial applications, written in RPG II, that provides all functions necessary for standard business accounting. AMS consists of the following subsystems:

- Accounts Payable

The accounts payable subsystem includes all functions needed to build and maintain the accounts payable files, select the invoices to be paid, issue the appropriate checks, and print reports reflecting the system operations. These files contain all the needed vendor information, such as name, address, telephone number, year-to-date purchases and payments, and discount-lost data.

- Accounts Receivable

The accounts receivable subsystem includes all functions needed to build and maintain the accounts receivable files. These files contain all required information on customers, such as name, address, telephone number, year-to-date sales figures, and credit limits. Daily accounts receivable transactions are summarized by the system to generate appropriate entries for the AMS general ledger subsystem.

- Payroll

The multistate payroll subsystem is modeled on the manual methods familiar to accountants and bookkeepers. Personnel working with the payroll subsystem need no data processing training or experience. The payroll subsystem is ready to use and provides all accounting records required by the Internal Revenue Service. It can accommodate multicompany and multidivisional payrolls. Special trade contractors will find that the AMS multistate payroll subsystem satisfies their particular needs.

- General Ledger

The general ledger subsystem provides control of accounting records, including an audit trail of entries, and the balancing and validation of all bookkeeping entries. Balance sheets and income statements are produced. This subsystem accepts input from the accounts payable, accounts receivable, and payroll subsystems.

12.7. WHOLESALE APPLICATIONS MANAGEMENT SYSTEM 80

The Wholesale Applications Management System 80 (WAMS 80) is an online, interactive system that provides the basic wholesale distribution management functions required in today's business environment. It contains the following four subsystems:

- Inventory/Sales Analysis

The inventory/sales analysis subsystem includes the functions necessary to create and maintain the inventory file. The file includes required information on a product such as pricing levels, stock levels, reorder levels, sales data, vendor number, and substitute product designation.

- Order Entry/Billing

The order entry/billing subsystem, an online interactive product, allows you to keep inventory and customer information instantly available for the user, while accurately performing all of the required order processing operations. These goals include identifying products, determining stock availability and pricing, identifying the shipping customer, discounts applicable, commission allocations, applicable taxes, producing invoices and pick slips, producing billing records for accounts receivable, maintaining all sales records, and producing management reports.

- Credit Return

The credit return subsystem, an online interactive system, allows you to keep inventory and customer information readily available while accurately performing all of the required returned merchandise operations. This includes identifying products, discerning pricing and customer discounts, identifying the shipping customer, determining commission, allocations and tax liabilities, producing credit invoices, producing credit records for accounts receivable, maintaining credit records and information, and producing various management reports.

- Expanded Sales Analysis

The expanded sales analysis subsystem provides expanded sales reports to the user. The reports produced by this subsystem are detailed and tailored to the needs of each user. This subsystem is used in conjunction with the WAMS 80 order entry/billing and credit return subsystems to extract input data. The extracted data is then reformatted, stored, and utilized in the various reports. Files from the WAMS 80 accounts receivable subsystem and inventory/sales analysis are accessed to obtain supporting data for these reports.

This subsystem provides monthly reports on product analysis, customer analysis, territory analysis, and monthly and year-to-date sales reports by customer/product class, as well as by salesman/customer/product class. Comparative analysis by customer/product class and salesman/customer/product class is also provided.

PART 6. COMMUNICATIONS AND DATA BASE FACILITIES

13. Communications

13.1. INTEGRATED COMMUNICATIONS ACCESS METHOD

The integrated communications access method (ICAM) is an optional software component that provides for:

- the inputting of data from a network of remote terminals into a program for processing;
- the distribution of messages to the terminals within the network; and
- the transfer of messages from terminal to terminal.

The following discussions offer brief descriptions of the components that make up the integrated communications access method. For more detailed information, refer to the ICAM concepts and facilities, UP-8194 (current version).

Generally, you will have only one network defined and operating on your system. However, ICAM permits you to define several distinct networks and to have these networks operating simultaneously. Each network is defined during system generation (SYSGEN) by submitting a set of macroinstructions that define the terminals, lines, buffers, and queues for each network. In addition, you must specify which of the four available communications interfaces each network is to use. A terminal can be included in more than one network definition; however, only one of those networks can be running at a time.

ICAM supports two types of networks: dedicated and global. A dedicated network can be accessed by only one program at a time. A global program permits several programs to access the network concurrently. Only one global network can be operating at any given time.

The four interfaces your networks can use are:

- Communications physical interface (CPI)
- Direct data interface (DDI)
- Standard interface (STDMCP)
- Transaction control interface (TCI)

You are also provided with an extended set of macroinstructions you can use to optionally specify message handling procedures including functions such as date and time stamping. This extended set of macroinstructions is called the message processing procedure specification (MPPS).

Once your network is configured, you can code your programs to interface with ICAM. These programs, interfacing with ICAM, retrieve data for processing from the remote terminals and send messages to them. You do this by including ICAM interface macroinstructions in your programs. Your programs normally initiate message transfers through the ICAM macroinstructions. However, through the use of the MPPS macroinstructions, you can have message transfers performed automatically.

Based on the information you specify in your network definition, the OS/3 system generation process includes and links all of the modules required to support your particular configuration into a single module. This module is loaded into low order storage as a symbiont. Once in main storage, it is referred to as the message control program and functions essentially as an extension of the supervisor.

ICAM provides the following internal services:

- Internal services to control communications lines and terminals.
- Queueing of messages in main or disk storage. A network may contain one or more message queues associated with lines and terminals.
- Multiple destination routing to allow up to 255 destinations for a single message.
- Activity scheduling and priority control of scheduled activities.
- A centralized timing service for control of active data buffers and activity scheduling.
- Restart procedures to reconstruct message queues after a system hardware or software failure.
- Accumulation of statistics including totals on messages received and transmitted, input and output retransmission requests, poll messages, and no-traffic responses.

In addition, ICAM provides the following major external services:

- User programs
  - Support of assembly language programs is provided in all ICAM interfaces.
  - COBOL programs are supported using the ICAM COBOL message control system (CMCS) feature.

- RPG II Telecommunications

  The ICAM system interfaces with the telecommunications facilities of RPG II through the direct data interface to provide RPG II remote input/output capability.

- Remote batch processing

  ICAM supports the entering of jobs through remote batch devices. The remote batch devices may request job output to be printed at the remote site or at the central site. Commands are available for the terminal operator to request status of jobs.

- Information management system

  ICAM provides support for the information management system (IMS) through the transaction control interface. IMS is an interactive, transaction-oriented communications data management system.

- ICAM device emulation system

  This system permits System 80 to emulate the capabilities of certain large-scale data communications terminals and permits System 80 to connect to another processor to perform remote batch processing.

- Nine thousand remote (NTR) system utility

  ICAM interfaces with the NTR system utility to permit a System 80 to be connected to a SPERRY UNIVAC 1100 operating system as a remote communications device.

- ANSI 1974 COBOL Communications

  Enables COBOL programs using the communications facilities of ANSI 1974 COBOL to send and receive messages.

- Remote terminal processor

  This data communications program permits your System 80 to work as a remote job entry terminal for one or more IBM host processors.

- IBM 3270 remote terminal handler

  This facility lets you use an IBM 3270 terminal system with your System 80. It also lets you use any other terminal that supports 3270 protocol. The 3270 terminal system includes the 3271 control unit, 3277 display station, and the 3284 and 3286 printers.

- IBM 3270 Emulator

  Like the remote terminal handler, the IBM 3270 emulator links System 80 to IBM hardware. Just as the handler connects an IBM terminal system to a System 80 host, the emulator connects a System 80 acting as a terminal to an IBM host system. This allows System 80 workstation users to access applications and IBM program products running on an IBM host. To do this, Sytem 80 operates in emulation mode, pretending to be a 3270 terminal system. The emulator uses the standard interface.

- Public data network support

  ICAM supports the capability to connect Sytem 80 to several popular packet-switched and circuit-switched public data networks using the X.25 interface. This support provides the user with an alternative to using costly leased lines. It provides high reliability and low cost because of packet-switched efficiencies and the ability to share links through the use of virtual circuits.

13.1.1. Message Control Program Structure

The message control program generated from your network definition and loaded into main storage consists of:

- an activity control module;
- either channel control routines or a set of remote device handlers;
- the particular system interface; and
- the communications control area and message processing procedure specification, if any.

The makeup of the message control program is based upon the interface you choose and the network definition you submitted. Figure 13-1 depicts the message control program structure for each interface type and shows the relationship of the message control programs with the operating system.

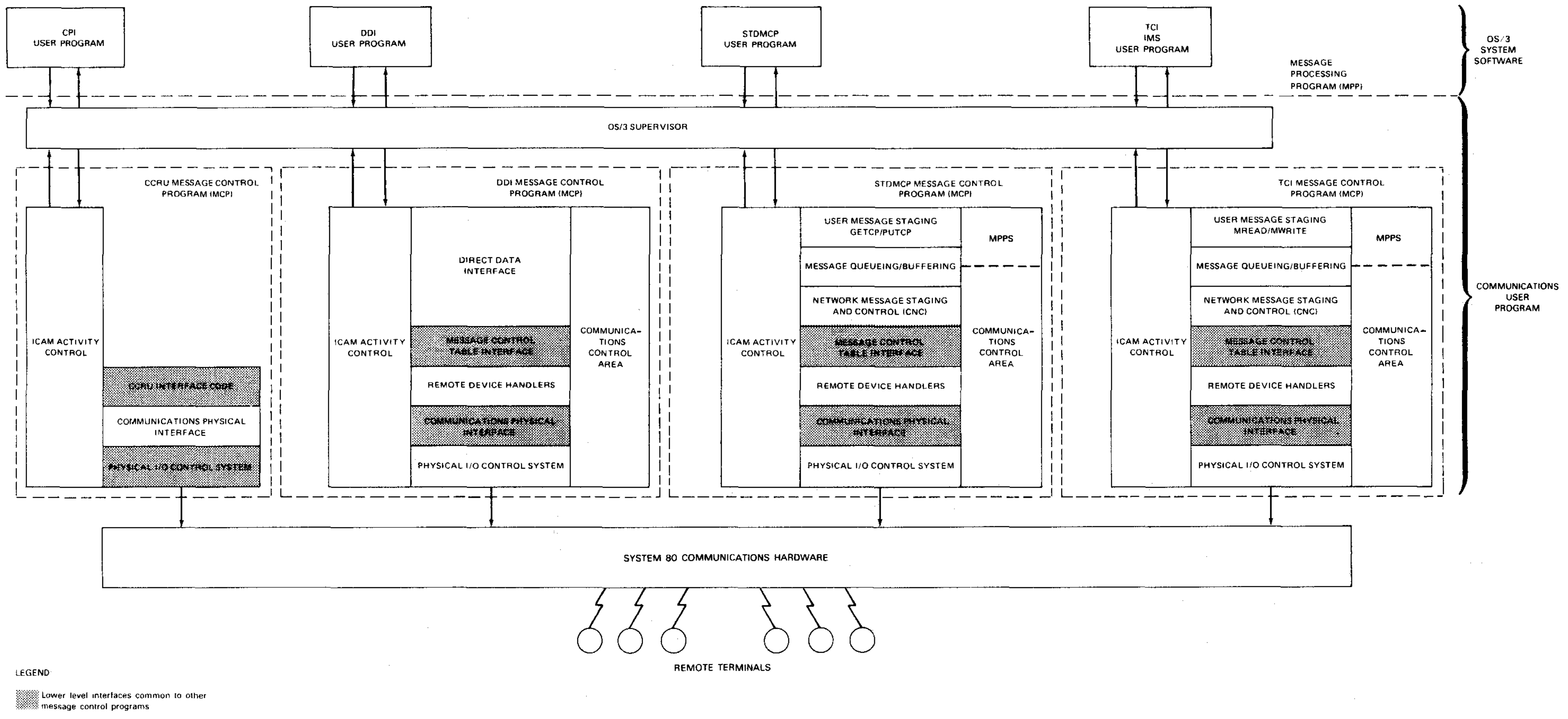

CPI USER PROGRAM
DDI USER PROGRAM
STDMCP USER PROGRAM
TCI IMS USER PROGRAM
MESSAGE PROCESSING PROGRAM (MPP)
OS/3 SYSTEM SOFTWARE
OS/3 SUPERVISOR
CCRU MESSAGE CONTROL PROGRAM (MCP)
DDI MESSAGE CONTROL PROGRAM (MCP)
STDMCP MESSAGE CONTROL PROGRAM (MCP)
TCI MESSAGE CONTROL PROGRAM (MCP)
COMMUNICATIONS USER PROGRAM
ICAM ACTIVITY CONTROL
CCRU INTERFACE CODE
COMMUNICATIONS PHYSICAL INTERFACE
PHYSICAL I/O CONTROL SYSTEM
DIRECT DATA INTERFACE
MESSAGE CONTROL TABLE INTERFACE
REMOTE DEVICE HANDLERS
COMMUNICATIONS CONTROL AREA
USER MESSAGE STAGING GETCP/PUTCP
USER MESSAGE STAGING MREAD/MWRITE
MPPS
MESSAGE QUEUEING/BUFFERING
NETWORK MESSAGE STAGING AND CONTROL (CNC)
SYSTEM 80 COMMUNICATIONS HARDWARE
REMOTE TERMINALS
LEGEND
▨ Lower level interfaces common to other message control programs

Figure 13—1. ICAM Structure and Interface Organization

The ICAM activity control module performs the central control functions for all activity within the system. It is this module that receives and interprets all activity requests and interfaces with the supervisor to have all requests performed. The channel control routines and the remote device handlers are the interface between the system software and the terminals configured into the network.

The four interfaces each support a different level of ICAM programming and offer different levels of support and capabilities.

Each interface is defined via specific sets of declarative macroinstructions issued during the generation of the message control program. The transfer of messages through each interface is initiated by interface macroinstructions issued by the communications user program (CUP). Each interface responds to a different set of macroinstructions.

The operation and capabilities of each interface are described in 13.1.1.1 through 13.1.1.4.

13.1.1.1. Communications Physical Interface

This interface (CPI) provides physical level definition and control of the communications data transfer. It requires the least amount of main storage of any of the available communications interfaces, but offers the fewest number of services. You must supply your own device and line protocol, polling, error detection and recovery, and message management.

You are not required to supply a network definition using the CCA macroinstructions, but you must code a control structure called the communications physical input/output control packet. This packet provides the interface between your CUP and the supervisor's channel control routines. The packet is used to pass software command codes generated by your program and software status codes generated by the system. These codes are standardized so that the message processing program need not recognize or generate specific hardware codes required by the communication system hardware.

Specific macroinstructions are provided to generate the communications input/output control packet. A set of macroinstructions, unique to the communications physical interface, is used to control data transfers. These macroinstructions are issued by your communications user program.

13.1.1.2. Direct Data Interface

The direct data interface (DDI) provides communications capability and services for small configurations of the system while offering a reasonable amount of device independence. This interface has many advantages for the user who is willing to invest more programming effort to gain a more flexible data communications capability. This interface also allows you to implement special applications that could not be efficiently supported by the standard message control program interface. This interface can only be used for a dedicated network.

At this level of support, your program interfaces with the remote device handlers that, in turn, interface with the physical input/output control system through the communications physical interface. Your program's interface with the remote device handler routines is through the message control table. This table resides in your program area and is generated by a supplied declarative macroinstruction. The message control table, along with specific macroinstructions, allows you to open, connect, disconnect, assign, and release a communications line, send and receive data, and set either the interactive or batch (card) mode.

When configuring the DDI you must supply a network definition. A simple method of line and terminal identification is provided to reduce the complexity of network definition. The interface provides a demand type operation in which data transfers to and from remote devices are directly into and out of buffer areas in your CUP. Demand operation means that each message must have a response issued before another message is issued. No message queueing, network buffering, or message processing procedure specification functions are supported by the message control program with a direct data interface. Your CUP must handle these services directly.

13.1.1.3. Standard Interface

The standard interface (STDMCP) provides a general telecommunications capability offering queued message processing and optional automatic processing control of message transfers. You must code a network definition by using the set of supplied network definition macroinstructions. Your program accesses the standard interface by means of logical standard interface macroinstructions. This interface can be used for either a dedicated or global network.

This interface requires more main storage, but it offers greater services as well as the advantage of logical level control and complete device independence once the message control program is operational. This is made possible through the inclusion of a communications network controller in your message control program. This component interfaces your network and CUP queue interface packets with the remote device handlers by automatically generating the required message control table.

Within your network definition, you include queues to hold messages being sent through the system – input queues for message input to your program from the network and output queues for messages from your program to the network. You must also include interface packets within your program. The user communications interface packets you defined within your user program are accessed when you issue a message transfer macroinstruction. The information within the packet is used to transfer the message to the appropriate destination.

You can optionally specify special message handling procedures through the inclusion of the message processing routine macroinstructions within your network definition. Use of these macroinstructions provides your network with ability to automatically handle message transfers. You can specify message switching routines, date and time stamping, certain tests, and a variety of other operations that can be performed without a CUP being accessed.

13.1.1.4. Transaction Control Interface

The transaction control interface (TCI) is a modified version of the standard message control program designed to meet the unique requirements of transaction programs including the IMS. This interface provides for automatic scheduling of your program as each message arrives in the system. Your program has the ability to examine the leading text field identifiers of each incoming message and to selectively retrieve the message for immediate processing or defer the message for later processing. Thus, you may specify concurrent processing of transactions without being involved with complex message send and receive procedures. This interface can be used for either a dedicated or global network.

You must provide a network definition and you may also include the MPPS functions. You must also define a transaction control area within your program. This area (defined through a supplied declarative macroinstruction) and the transaction control interface imperative macroinstructions provide the interface between your user program and the message control program. The transaction control area declarative macroinstructions generate a transaction control table and a series of transaction terminal tables. A transaction terminal table must be generated for each terminal defined in your network.

Among the services provided by this interface are:

- asynchronous and concurrent message processing;
- optional disk nonqueued disk buffering for both input and output messages;
- buffering of unsolicited output to a terminal during multiple message transactions between your program and the terminal; and
- dynamically controlled multiple destination routing.

13.2. DISTRIBUTED DATA PROCESSING

The SPERRY UNIVAC distributed data processing (DDP) system allows a number of separate processing systems to be tied together in a network so that all systems can share the processing load of the entire organization. This interchange is based on a standard SPERRY UNIVAC DDP command language implemented within each of the supporting operating systems. In such a distributed data processing network, a person at one site can control the operations of another site, and can perform such functions as:

- Site-to-site data file and program library transfers
- Operator console control over remote site and routing of messages to remote operator console
- Initiation of jobs at the remote site
- Inputting of data through the local site to the remote site for processing
- Program-to-program communications

The software required to support a DDP network includes a configured communications network to physically link the included sites and distributed data processing processors at each site. The DDP processors interpret all DDP related commands and perform the requested functions. The DDP processors are designed to operate in an interactive environment and respond to commands issued from a workstation.

The DDP software can be used to copy data files and program libraries from one system to another, or to add data to an existing data file in a remote system. Data files can be deleted from a remote system. Data files and program libraries can be transferred between OS/3-based systems.

You can utilize the DDP commands to submit and initiate a job to a remote system through the local system. Output from the job can remain at the processing site or can be directed back to the initiating site. Jobs running at another site can also be cancelled from the initiating system.

Messages can be sent to the system console of a remote system and the answer routed back to the initiating device. A device on a system can, in a limited way, operate as the system console for a remote system.

The capability to distribute the processing workload among systems in a DDP network can provide improved business operations and management control. Jobs can be decentralized and given to the location responsible for gathering and using the data. Distributed data processing also provides greater control over work priority, improved response time, and a recovery system in the event of local system failures.

The OS/3 DDP consists of:

- a transfer facility;
- a program-to-program communications facility; and
- an IMS-DDP transaction facility.

13.2.1. Transfer Facility

The DDP transfer facility allows you to view each system in the DDP network as an available resource for scheduling and executing your work. Using simple commands, you can initiate job distribution and file transfer within the system without concern for the requirements of the hardware and software of each system or the communications protocols needed to initiate and monitor the distribution of a job. The facility has two functions: job distribution and file transfer.

13.2.1.1. Job Distribution Function

The job distribution function allows you to initiate execution of a job on any system within the DDP network and to monitor the execution from the initiating site. Any printed or punched output generated by the job is normally routed to the initiating system. However, the initiator can request routing of this output to any available system in the network.

The job distribution function provides commands to:

- submit jobs to the DDP network;
- monitor the execution of submitted jobs;
- cancel a submitted job;
- communicate with a remote operator's console;
- issue instructions to a remote operating system; and
- respond to messages issued by a job executing on a remote system.

13.2.1.2. File Transfer Function

The file transfer function provides the capability to transfer sequential files from one system to another or to duplicate file structures between systems.

Files can be transferred between systems that contain the same file handling facilities, in which case no data transcription is required. Files with serially-accessible records may be transferred between systems of dissimilar architecture with or without character conversion. That is, a file can be transferred as a bit stream to be manually reformatted or it can be converted (character translation only) during the transfer process to the internal code recognized by the receiving system.

The file transfer function does not support automatic reformatting of *items* within a file, or converting items to their equivalent form in the destination system, when transfer is between systems of dissimilar architectures. In this case, the contents of the file are treated as all character data or bit string data. No record sensitivity, record sequence, or numeric field characteristics are recognized or adjusted during the transfer operation.

File transfer can be directed in the file area of the destination system or, if the destination file area is in use, to a temporary file area in the destination system. Transfer from the temporary file to the destination file is made when the destination file becomes available.

Files transmitted between systems can include:

- Data Files

  Data formatted in the OS/3 MIRAM format. Files transmitted from OS/3 are MIRAM files; files transmitted to OS/3 will be created in MIRAM format.

- Program Libraries

  Any directly-accessible module in a program library can be transferred between systems or an entire library can be transferred.

The file transfer function permits the user to:

- generate a file directory to catalog the characteristics of files;
- transfer copies of data files and program libraries;
- delete a file from the file directory; and
- obtain a listing of a file's characteristics from the file directory.

13.2.2. Program-to-Program Communications Facility

The DDP program-to-program communications facility (P-T-P) enables your executing program to communicate with another executing user program and controls activity between the programs. The programs can be on the same host or different hosts.

Two programs tied together by P-T-P converse with each other. The program that initiates the conversation is called the primary user application program (primary UAP) and the other program is called the surrogate UAP. For certain operations, the two programs can exchange status, the primary UAP becoming the surrogate UAP and vice versa. There is no limit to the number of times the UAPs can exchange status. And you can also establish P-T-P communications among more than two independent hosts, with as many as 255 workstations connected to each host computer.

13.2.3. IMS-DDP Transaction Facility

The IMS-DDP transaction facility lets you process information management system (IMS) transactions at a remote OS/3 computer. (For an overview of IMS, see 13.4.) In a DDP transaction, a terminal operator at one IMS system, called the primary IMS, initiates the transaction. The primary IMS, through the transaction facility, routes the transaction to a remote system where a secondary IMS processes the transaction and sends back a response.

The remote transaction may be processed by user action programs (written in COBOL, RPG II, or basic assembly language) or by UNIQUE. There is little difference between the way action programs process a remote transaction and the way they process a local transaction. Most IMS features are available, including the use of screen format services.

There are three ways in which IMS can route a transaction to a remote system:

1. Directory routing
2. Operator routing
3. Action program routing

These differ mainly in how each initiates a transaction. All three allow messages to be sent to the remote system and back.

13.3. UTS SUPPORT

OS/3 offers the following software components to support the SPERRY UNIVAC Universal Terminal System 40 (UTS 40) and the SPERRY UNIVAC Universal Terminal System 400 (UTS 400):

- UTS COBOL
- UTS edit processor
- UTS load/dump facilities

13.3.1. UTS COBOL

UTS COBOL is a high-level, business-oriented international language with features to complement the capabilities of the programmable UTS terminals. The UTS COBOL compiler meets the ANSI X3.23-1974 standards and the ISO recommendations for COBOL. The compiler also contains extensions to provide for interactive data entry, program control, and screen management. The UTS COBOL compiler executes under control of OS/3 and produces a compiled program that can be downline loaded to the terminal or placed on a diskette.

For more information on UTS COBOL, see the current version of the UTS COBOL programmer reference, UP-8481.

13.3.2. UTS Edit Processor

The UTS edit processor allows you to create and manipulate text data on a diskette file. It provides an easy and efficient way to create and update line-oriented files of data. Lines may be inserted, replaced, deleted, or changed in any order.

The edit processor permits functions such as string searches, insert or delete lines, moves, and print. Error messages advise the user of mistakes made in entering data. Lines up to 118 characters in length may be inserted in an edit processor file.

For more information on the UTS edit processor, see the current version of the universal terminal system 4000 (UTS 4000) edit processor user guide/programmer reference, UP-8932.

13.3.3. UTS Load/Dump Facilities

The UTS load/dump facilities provide two ways of exchanging data between your System 80 and a local or remote terminal. One way uses two workstation commands, DLOAD and ULD.

- The DLOAD command extracts a UTS load module from the system $Y$LOD load library and transmits this module to a UTS terminal (downline load).

- The ULD command dumps the main storage of a terminal to an OS/3 dump file (upline dump). Additional ULD options format and print a copy of the dump and save or erase the file after the print operation has finished. This capability is available only with the UTS 400 terminal, not the UTS 40.

The other method of exchanging data involves communications software which performs the same upline dump/downline load operations as DLOAD and ULD. We provide this software for compatibility with previous releases.

For more information on DLOAD and ULD, see the interactive services concepts and facilities. Refer to the current version of the interface UTS 400-OS/3 user guide/programmer reference, UP-8611, for more information on loading UTS terminals using communications software.

13.4. INFORMATION MANAGEMENT SYSTEM

The information management system (IMS) is a transaction-oriented, file processing system operating in a communications environment that utilizes a simplified inquiry/update language for manipulating data files. IMS allows nonprogramming personnel to access and update a data base from a workstation or terminal. This discussion of IMS offers a brief look at the capabilities, use, and components of the system. For more detailed information, see the IMS applications user guide/programmer reference, UP-8614 (current version).

IMS is a transaction-oriented processing system. Stated simply, this means that the system responds to each request you make on a one-to-one basis. Each entry you make results in some type of system response: the information that you seek, verification that the request action was performed, or an error message. You cannot make another request until the system responds to a prior request. This one-to-one message processing simplifies the system for the nonprogramming individual by preventing the issuing of multiple requests that can lead to confusing results.

You configure a tailored IMS system to handle your particular file processing applications. An IMS configuration consists of data files, a network of workstations or terminals, action programs to process the inquiry messages issued from the terminals, and resident information management routines to control the system.

IMS offers a simplified terminal language for the nonprogramming personnel to use to access and update data files. This language, called UNIQUE, is actually a set of IMS action programs each designed to perform a specific function and initiated by an entered UNIQUE command. The use of UNIQUE is optional, as you can write your own action programs in COBOL, RPG II, or BAL to process your files. If you choose to use UNIQUE, the data files that are to be part of the IMS system must be defined using a supplied data definition language. This language lets you structure logical file and record formats that differ from the actual format of the data files. If you write your own action programs, you can use the data definition language or your programs can access the data files directly.

Data bases generated by the data base management system can also be accessed through IMS. You can either use the IMS data definition language to define the files to IMS or you can code IMS action programs in COBOL and include DMS data manipulation language statements in the program.

The information management system supports concurrent users. The number of concurrent users is limited only to the number of workstations and terminals configured into the system. File locks are used to prevent destructive interference among concurrent users accessing the same data file.

If you use UNIQUE, a password protection capability is provided to facilitate security measures in the interactive environment of IMS. Using this feature, you can limit system access to authorized personnel and also limit those persons to only selected elements within those files.

IMS also provides commands for a terminal or workstation designated as a master terminal to assist in monitoring the system. There are additional commands that can be issued from any terminal that can be used for educational purposes or to resolve various operations or administrative problems.

Extensive file recovery features guard against permanent destruction of data files accessed by IMS. An automatic rollback feature is automatically initiated if a transaction is abnormally terminated or cancelled by the workstation operator. Any file modified by the terminated transaction is returned to its logical state as it existed before the transaction was initiated. IMS also provides an offline recovery facility that can reconstruct data files adversely affected by a system failure.

14. Data Base Management

14.1. GENERAL

The data base management system (DMS) is a system facility that supports the development and usage of integrated data bases. It consists of a series of routines that define the data included in the data base, describe the physical structure of the data base, and manipulate the data. Also included are a number of recovery, maintenance, and audit routines. The following is a brief description of the data base management system. For more detailed information on DMS, see the DMS system support functions user guide/programmer reference, UP-8272 (current version).

14.2. DMS OPERATIONS

In DMS, the description of data is separate from the manipulation of that data by applications programs. This results in data independence from applications programs and eliminates redundancy in the data base. Once a data base is defined (using the supplied data description language) and included in the system, it may be accessed by several programs concurrently.

Programs that can access the data base include:

- COBOL programs using the DMS data manipulation language
- Information management system (IMS) action programs
- IMS UNIQUE terminal language.

In addition to allowing concurrent access of a single data base by several programs, DMS permits a single program to access several data bases during the course of a single execution.

The logical structure and data retrieval methods for an entire data base are described using the schema data description language. The description, called the schema, includes the name and description of all areas, records, and sets in the data base. For purposes of data base security, a subschema must be defined for each application program or group of programs that access the data base.

An application program can only access that portion of the data base defined in its associated subschema. However, the entire data base may be included in a subschema. Subschemas are defined using the subschema data description language. The schema and subschema data description languages are derived from the CODASYL data base specifications.

The physical structure of the files that hold your data base is defined using the device media control language. Using this language, you define the media characteristics of the data base files such as page and area sizes and the number of required buffers.

Your data base is divided into segments called areas. Areas represent the first level of logical division of the data base. All occurrences of a record of a given type must be stored in the same area. However, an area can also hold records of various types. Each area is assigned a specific number of pages, each page being a unit of storage that is a multiple of 2048 bytes in size. Space within an existing page is allocated to a record when it is stored. When a record is deleted, the allocated space within the page containing the record is released and made available for reallocation.

Records are related in sets in which a record of one type is designated as an owner and records of one or more other types are designated as members. Sets are implemented as record occurrences linked into chains by data base keys. A given record type may be an owner of multiple set types and a member of multiple set types. The set mechanism can be used to relate records in sequential, hierarchical, or network structures.

The physical placement of records can be controlled to optimize performance. The physical location of a record is referred to as its location mode. The location mode of a given type of record can be specified in several ways. It can be specified as direct to allow programs to dynamically specify where each occurrence is to be stored. The location mode can be specified as calculated to cause records to be stored as a function of an algorithm. It can also be defined by set to cause each member record occurrence to be stored in or near the page containing the corresponding owner record. When a record is entered into the data base, a unique direct address, called a data base key, is assigned to it. This key is never altered throughout the lifetime of the record.

Each data base that you define will have a data dictionary containing descriptions generated by the schema and subschema data description language and the device media control language compilers. The data dictionary is referenced by the compilers to generate reports describing the data. It can also be accessed by programs written by the data base administrator to generate specialized data dictionary reports.

A data base is accessed through data manipulation language statements included in the procedure division of a COBOL program. COBOL programs containing data manipulation language statements are processed by a DMS preprocessor to produce a standard COBOL source module that can be subsequently compiled by a COBOL compiler. These programs can be batch-oriented or serve as IMS transaction-oriented action programs.

Data manipulation statements are used to establish contact with the data base management system, access a specific subschema, and open and close areas of the data base. You can also locate records and place them in working storage; store, modify, and delete records; and insert and delete records from set occurrences. Locks are used to prevent destructive interference by concurrent active users and they can be applied on the area, page, and record level.

The data base management system provides extensive facilities for recovery of data base records in the event of abnormal conditions or the inadvertent destruction or alteration of record information. When you are defining your data base, you also define recovery files called the QBL and journal files.

The quick before look (QBL) file is used by automatic backward recovery (ABR) to rollback a file to its state prior to an abnormal termination of an application program or system failure. The journal file is used by offline recovery to restore the data base.

Each time information on a page in the data base is to be altered, a copy of the page before it was altered is placed in the QBL and journal files. A copy of the altered page is also placed in the journal file. If, for some reason, the information on the data base page is incorrectly altered or lost through abnormal conditions, you can, through the use of one of several recovery utilities, restore the data base to its original state or to the correct altered state.

The information in a journal file is saved for the duration of a data base session. Utilities also exist to audit the journal file and produce reports on selected information in the journal file.

Other utilities are available to dump the contents of a data base from disk to tape or disk and to perform such functions as editing and printing the contents of a data base page and verifying and altering the contents of a data base.

PART 7. CONVERSION

15. Conversion Aids

15.1. GENERAL

In the rapidly changing world of the computer industry, the need is constantly arising to upgrade the present computing system to one that is more powerful, economical, or that meets changing needs. Because of its advanced design, offering power and versatility with economical operation, System 80 is seen as being the growth path for a number of other systems including:

- SPERRY UNIVAC 9200/9300 and SPERRY UNIVAC 9400/9480
- IBM System/3, System/32, and System/34
- Honeywell 100 series, 200/2000 series, and 60 Series (Level 62/64 systems.)

As a result, Sperry Univac supplies a number of conversion aids if you decide to convert from one of these systems to System 80. In addition, there are a number of areas of compatibility between System 80 and the other systems that make the conversion a fairly straightforward process.

15.2. SPERRY UNIVAC SYSTEMS

Sperry Univac supplies conversion aids for the SPERRY UNIVAC 9200/9300 and 9400/9480 (OS/4) Systems.

15.2.1. SPERRY UNIVAC 9200/9300 System

The following areas of compatibility exist between the SPERRY UNIVAC 9200/9300 System and System 80:

- The OS/3 RPG II compiler provides a 9200/9300 mode that permits direct compilation 9200/9300 RPG programs on System 80 without source code translation.

- 9200/9300 sequential tape files developed on the UNISERVO VI-C or UNISERVO 12 tape devices can be mounted on System 80 UNISERVO 10 tape drives and processed directly by OS/3 programs.

In addition, Sperry Univac supplies a number of conversion aids for those areas of incompatibility between OS/3 and the 9200/9300 system. These aids are:

- 9200/9300 Data File Transcriber (UNLOAD/DATA)

  The data file transcriber (UNLOAD) supplied by Sperry Univac is executed on the 9200/9300 system to copy 9200/9300 disk data files to a tape file. This tape file can be used as input to the OS/3 data utility, which, in turn, generates the appropriate disk file from the tape.

- 9200/9300 Assembly Language Translator (TRASM3)

  The 9200/9300 assembly language source statements can be translated into OS/3 basic assembly language statements through the 9200/9300 to OS/3 assembly language source translator (TRASM3).

- COBOL and COPY Translator (COBTRN305)

  The 9200/9300 COBOL source programs and COPY library elements can be converted directly into OS/3 compatible ANSI 1974 COBOL through the COBTRN305 translator.

- 9200/9300 Library Transcriber (COPY93)

  The 9200/9300 library files can be converted to OS/3 format through the OS/3 COPY93 library transcriber. COPY93 accepts a 9200/9300 formatted tape as input and produces an OS/3 formatted disk file.

15.2.2. SPERRY UNIVAC Operating System/4 (OS/4)

OS/3 offers a high degree of compatibility with SPERRY UNIVAC 9400/9480 systems operating under OS/4. OS/4 RPG and FORTRAN source programs can, for the most part, be recompiled by the OS/3 compilers. Any changes required will be minor. A conversion guide that details all the steps required to migrate from OS/4 to OS/3 is available. For those areas of incompatibility, Sperry Univac supplies the following conversion aids:

- OS/4 Job Control Converter (JCON1)

  The OS/4 to OS/3 job control language converter (JCON1) supplied by Sperry Univac converts OS/4 job control source statements to OS/3 compatible job control statements. Input to the JCON1 utility can be a magnetic tape containing only control streams (no procs) produced by an OS/4 FILE command, cards, or an OS/3 disk file created by an OS/3 FILE command, the COPY94 utility, or the OS/3 librarian. JCON1 outputs to cards, the printer, or to a disk file.

- OS/4 Assembly Language Translator (ASMTRN)

  OS/4 basic assembly source statements can be translated into OS/3 assembly statements through the OS/4 to OS/3 assembly translator (ASMTRN).

- COBOL and COPY Translator (COBTRN301)

  OS/4 COBOL source programs and COPY library elements can be converted directly into OS/3 compatible ANSI 1974 COBOL through the COBTRN301 translator.

- Disk Data File Converter (DCON4)

  Disk data files can be converted to OS/3 format by using the disk data file converter (DCON4) to dump the files onto tape and then inputting the tape to the OS/3 data utility that, in turn, builds the appropriate data file.

- OS/4 Library Transcriber (COPY94)

  OS/4 library files can be converted to OS/3 format through the OS/3 COPY94 library transcriber. COPY94 accepts an OS/4 formatted tape as input and produces an OS/3 formatted disk file. The input tape must be generated through the OS/4 tape and disk librarians.

15.3. IBM SYSTEMS

Sperry Univac supplies conversion aids for those who are migrating from the System/3 and System 32/34 IBM systems.

15.3.1. IBM System/3

SPERRY UNIVAC OS/3 software provides a significant amount of compatibility with the IBM System/3. Among the major areas of compatibility are:

- A System/3 compatible sort, SORT3, that accepts System/3 parameters

- An access method, MIRAM, that is functionally compatible with the System/3 disk access method

- A System/3 mode on the OS/3 RPG II compiler that permits direct compilation of System/3 RPG II source programs

- Parameter specifications for $DELET, $COPY, $KCOPY, $DCOPY, and $MAINT that can be used directly on System 80 to duplicate the utility functions

- An OCL processor that accepts and processes System/3 OCL control streams

In addition, Sperry Univac supplies a number of conversion aids for those areas of incompatibility between OS/3 and System/3:

- Disk Data File Conversion

  System/3 data files must first be dumped to a magnetic tape using one of the System/3 utilities such as $KCOPY. The tape is then submitted to the OS/3 data utilities to reload the files to disk storage devices.

- Models 10, 12, and 15 Source and Proc Transcriber

  You can transcribe IBM System/3 source and proc modules to your System 80 using the source and proc transcriber COPYS3. This utility accepts tape or diskette input that you create as follows:

  - For diskette input, you copy the source and proc modules directly to diskette using the IBM $MAINT utility.

  - For tape input, you follow a 2-step procedure: first, you use $MAINT to copy your source and proc modules from a source library on disk to another disk file; then, you copy the second file to tape using the IBM $COPY utility.

15.3.2. IBM System 32/34

If you are migrating from an IBM System/32 or IBM System/34 data processing system, you will find a high degree of compatibility in OS/3 program products. The RPG II programming language used in OS/3 is highly compatible with System 32/34 RPG II, and we even have RPG II auto report, a feature that should be familiar to System 32/34 users. Other compatible programming tools include:

- A sort program (SORT3)

- The COBOL and FORTRAN languages

- A general editor for creating language source code interactively.
- A conversion program that accepts System 32/34 S & D specifications to create formatted screen displays.
- A system program that enables you to create menus and their associated help screens.

When the time comes to carry your IBM software to OS/3, we provide conversion aids that let you:

- Transcribe System 32/34 data files to OS/3 disk files by first running the IBM TRANSFER or $COPY procedure to produce a diskette. This diskette is then input to the OS/3 data utilities to produce a user-tailored OS/3 file.
- Transcribe System 32/34 source and proc files to OS/3 disk files by first copying the files to a diskette with the IBM $MAINT utility, then running the SPERRY UNIVAC COPYS3 source and proc transcriber (also a System/3 conversion aid).
- Convert System 32/34 operation control language (OCL) to OS/3 job control language (JCL) with the JCLCON802 utility program.

15.4. HONEYWELL SYSTEMS

Sperry Univac provides you with conversion aids if you are converting from the following Honeywell computer systems: 100 Series, 200/2000 Series, and 60 Series, Level 62 and Level 64.

15.4.1. Honeywell 100 Series

If you are migrating from the Honeywell 100 Series data processing systems, you will find a high degree of compatibility with System 80. For those areas of incompatibility, Sperry Univac offers the following conversion aids:

- COBOL Translator (COBTRN304)

  Honeywell 100 Series COBOL source programs can be converted directly into OS/3 compatible ANSI 1974 COBOL through the COBTRN304 translator.

- Data File Translator (TAPCON)

  Honeywell data files can be converted to OS/3 format by using the data file translator (TAPCON). The Honeywell data files must be copied to tape or card, then submitted to TAPCON for reformatting. The resultant tape file can be converted to the ultimate intended file media.

15.4.2. Honeywell 200/2000 Series

Sperry Univac provides the following conversion aids if you are converting from the Honeywell 200/2000 Series systems to System 80:

- COBOL Translator (COBTRN302)

  Honeywell 200/2000 Series COBOL source programs can be converted directly into OS/3 compatible ANSI 1974 COBOL through the COBTRN302 translator.

- EASYCODER Converter (ETC3)

  Honeywell EASYCODER source programs can be converted directly into OS/3 compatible ANSI 1974 COBOL through the ETC3 translator.

- Data File Transcriber (TAPCON)

  Honeywell data files can be converted to OS/3 format by using the data file translator (TAPCON). The Honeywell data files must be copied to tape or card, then submitted to TAPCON for reformatting. The resultant tape file can be converted to the ultimate intended file media.

15.4.3. Honeywell 60 Series, Level 62 and Level 64

The following conversion aids are available if you are converting to System 80 from the Honeywell 60 Series, Level 62 and Level 64 systems:

- Program Library and Data File Transcriber

  Honeywell data files and program libraries can be transcribed to OS/3 format through the program and data file transcriber (TAPCON). The Honeywell data files and program libraries must be copied to tape or card, then submitted to TAPCON running on OS/3 for reformatting. The resultant tape files can be converted to the ultimate intended file media.

- COBOL Translator

  Sperry Univac provides a translator to convert Honeywell 60 Series COBOL to OS/3 compatible ANSI 1974 COBOL.

PART 8. APPENDIXES

Appendix A. System File Descriptions

The following is a list of system file descriptions.

File Name	File Type	Description
SMPMIRAM	MIRAM	Backup file for SMP applications. This file is allocated at run time.
$Y$ESUM	MIRAM	System 80 error log summary
$Y$SAVE	MIRAM	Saved run-library modules
$Y$CAT	SAT	Used by job control to record cataloged files made by the customer
SMCAUDIT	SAT	Audit trail of SMP application. This file is allocated at run time.
$Y$ELOG	MIRAM	A storage area that the supervisor uses to record the I/O error history
$Y$FMT	MIRAM	Contains the system screen formats and the menu generator screens and help screens
$VTOC	MIRAM	Used to allocate and deallocate disk space on the SYSRES
SMCBSAT	SAT	Used to back up SMC applications. This file is allocated at run time.
SG$XXX	SAT	Used by SYSGEN for separately priced products
$Y$SCLOD	SAT	Shared code load library
$Y$JCS	SAT	Contains stored job control streams and job control procedures

File Name	File Type	Description
IVPLIB	SAT	Installation verification program library
$Y$SJF	MIRAM	System journal file, used to record changes in hardware status
$Y$MIC	SAT	System microcode library
$Y$OBJ	SAT	Contains the object module subroutines for data management and run-time library for each compiler
SG$JCS	SAT	SYSGEN-related job control procedures
$Y$HELP	MIRAM	Contains help screen modules
$Y$SHR	SAT	Holds information about files that are currently open in the system and their ability to be shared
$Y$MAC	SAT	Contains the interface procedures and macros for the components that may be referenced from assembled code
SG$LOD	SAT	Contains the SYSGEN load modules required to perform a RESGEN
$Y$SDF	MIRAM	Contains the system defintion that permits down-line loading of microcode to peripherals such as workstations
$Y$LOD	SAT	Contains the system load modules
$Y$SRC	SAT	Contains the source and copy modules provided
SG$MAC	SAT	Contains the procedures and macros for generation of supervisors and ICAMS
$Y$DIALOG	MIRAM	Contains system dialogs for SYSGEN and JCL
$Y$TRAN	SAT	Contains the OS/3 transient modules and the canned messages for console and printed display
$Y$DUMP	MIRAM	The storage area used by SYSDUMP processing
$Y$SMCLOG	MIRAM	Used for the software maintenance correction log (SMCLOG)

File Name	File Type	Description
$IPL	MIRAM	Contains the OS/3 software that initially performs a load of the desired supervisor or utility program
$IMPL	MIRAM	Microcode load file
SMCFILE	SAT	Contains the software maintenance corrections (SMC) processed for the system
$Y$TRANA	SAT	Alternate $Y$TRAN file
$Y$SYSTEMTABLES	MIRAM	Contains system tables
$Y$DDP	SAT	DDP recovery file. This file is allocated at run time.
SMCBMIR	MIRAM	Backup file for SMC. This file is allocated at run time.
SMCBTRAN	SAT	Backup file for SMC transient file. This file is allocated at run time.

Appendix B. Functional Characteristics of Input/Output Devices

The tables in this appendix summarize the SPERRY UNIVAC System 80 functional characteristics of the input/output devices that are supported by consolidated data management.

Table B-1. 0719 Card Reader Subsystem Characteristics

Characteristic	Description
Card orientation (80-, 66-, and 51-column cards)	Face down, column 1 to left and row 9 facing away
Card rate	300 cpm
Read technique	Two columns of photosensitive sensors and light-emitting diodes Dual redundant column amplifier checking
Read modes	Image mode: 160 six-bit characters per card Translate mode: 80 characters per card
Read station sensing	Column by column
Hopper capacity	1000 cards
Stacker capacity Normal Reject	 1000 cards 1000 cards

Table B-2. 0608 Card Punch Subsystem Characteristics

Characteristic	Description
Media	80-column cards
Punch mode	2-column serial
Check mode	Punch motion check
Feed mode	On demand
Punch rate	75 cpm (full card) 160 cpm (28 columns only) 120 columns/second advance speed
Input capacity	700 cards
Output capacity	700 cards (primary stacker) 100 cards (auxiliary stacker)
Reading	Optional
Read rate	160 cpm

Table B-3. 0776 and 0789 Printer Subsystem Characteristics (Part 1 of 2)

Characteristic	Description		
0776 Printer Subsystem			
Print speed	210 to 1250 lpm depending on character contingencies:		
	Available character sets (characters/set)	Number of sets per band	Nominal print rate (lpm)
	384	1	210
	192	2	395
	128	3	560
	96	4	710
	64	6	980
	48	8	1200
	32	12	1250
	24	16	1250
Line advance timing	Advance and print (number of lines)	Time (ms)	
		6 lpi	8 lpi
	1	16.7	14.1
	2	24.6	20.9
	3	30.9	25.9
	4	35.0	30.9
	5	38.9	34.1
	6	42.6	37.1
	7	45.9	39.9
	8	49.3	42.6

Table B-3. 0776 and 0789 Printer Subsystem Characteristics (Part 2 of 2)

Characteristic	Description		
	0776 Printer Subsystem (cont)		
Number of print positions	136 print positions (columns)		
Forms advance control	Vertical format buffer		
Forms advance rate	50 inches (127 cm) per second		
Form dimensions	4 to 18.75 inches (10.16 to 47.62 cm) wide 1 to 18 inches (45.72 cm) long		
Horizontal spacing	10 characters per inch		
Vertical spacing	6 or 8 lines per inch, operator-selectable		
	0789 Printer Subsystem		
Print speed	180, 300, and 640 lpm depending on character contingencies		
	Available character sets	Number of sets per band	Nominal print rate (lpm)
	48 64 96 128	4 (plus 16 char) 3 (plus 16 char) 2 (plus 16 char) 1 (plus 80 char)	317 306 246 177
Line advance timing	Advance and print	Time (ms) 6 lpi	8 lpi
	1 line 2 lines 3 lines n + 1 lines	40 52 64 76 + 12	40 52 64 76 + 12
Number of print positions	120 print positions (columns) by standard printer; 132 columns by feature		
Forms advance control	Controlled from host processor		
Forms advance rate	15 inches (38.1 cm) per second		
Form dimensions	3 to 15 inches (7.62 to 38.10 cm) wide 1 to 22 inches (55.88 cm) long		
Horizontal spacing	10 characters per inch		
Vertical spacing	6 or 8 lines per inch, operator-selectable		

Table B-4. Disk and Diskette Subsystem Characteristics

Characteristics	Description		
	8417 Disk Subsystem	**8419 Disk Subsystem**	**8420/8422 Diskette Subsystem**
Data capacity (8-bit bytes)	118.2 million	72.39 million	Single density[3] Double density[3] 1 side 303,104[1] 1 side 563,320 2 sides 606,208 2 sides 1,136,640[2]
Number of disk units	1 to 8	1 to 8	1 to 4
Disk/diskette speed (rpm)	3400	2800	360
Rotation period (ms/rotation)	17.6	21	166
Data bit rate (MHz)	9.05	6.2	-
Bit density (ppi)	6366	5050	-
Track density (tracks/inch)	476	-	-
Track capacity (bytes/track)	15,360	12,800	3328 to 7680[3]
Number of tracks	550 + 10 spare tracks per disk surface	808 + 7 spare usable tracks per disk surface	77 total, 75[4] for data use per diskette surface
Number of surfaces per disk unit	14	7	2
Positioning time (seek time) Minimum (ms) Average (ms) Maximum (ms)	 7 35 70	 10 33 60	 3 15 35
Transfer rate (kilobytes/second)	1130	784	Dependent on sector sequence arrangement

NOTES:

[1] 242,944 for data set label BDE (basic data exchange) diskette file.

[2] 971,776 for format label diskette file.

[3] Maximum value. Actual value is dependent on diskette type (single sided, single density; single sided, double density; double sided, single density; double sided, double density), physical sector size (128, 256, or 512 bytes) and file type (format label or data set label).

[4] 73 for format label diskette file and data set label BDE (basic data exchange) file. 75 for other data set label non-BDE files.

Table B-5. UNISERVO 10 Magnetic Tape Subsystem Characteristics

Characteristic	Description
Tape units per subsystem	1 to 8
Data transfer rate (maximum)	40,000 frames per second
Tape speed	25 inches per second
Tape direction Reading Writing	 Forward or backward Forward
Tape length (maximum)	2400 ft.
Tape thickness	1.5 mils
Block length	Variable
Maximum block size (bytes)	65,535
Minimum block size (bytes)	18
Interblock gap	0.6 in.
Interblock gap time Nonstop Start-stop	 24 ms
Pulse density	1600 ppi on 9-track phase encoded 800 ppi on 9-track NRZI
Recording mode	Phase encoded or NRZI
Reversal time	16 ms
Rewind time	3 min.
Simultaneous operation	Optional

Table B-6. Workstation Subsystem Characteristics

Characteristic	Description
Type of display	Cathode ray tube (CRT)
Number of display lines	24 plus 1 indicator
Characters per line	80
Number of units	1 to 8
Keyboard arrangements	Typewriter layout, typewriter layout with numeric and function pads, or Katakana/English
Character sets	Domestic, United Kingdom, Germany, France, Spain, Denmark/Norway, Sweden/Finland, Italy, or Katakana/English

Index

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

____
(Document Title)

____________________ ____________________ ____________________
(Document No.) *(Revision No.)* *(Update No.)*

Comments:

From:

____
(Name of User)

____
(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

ong line.

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD

CUT

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

(Document Title)

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

Cut along line.

From:

(Name of User)

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

CUT

FOLD

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

(Document Title)

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

From:

(Name of User)

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

Cut along line.

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD

CUT