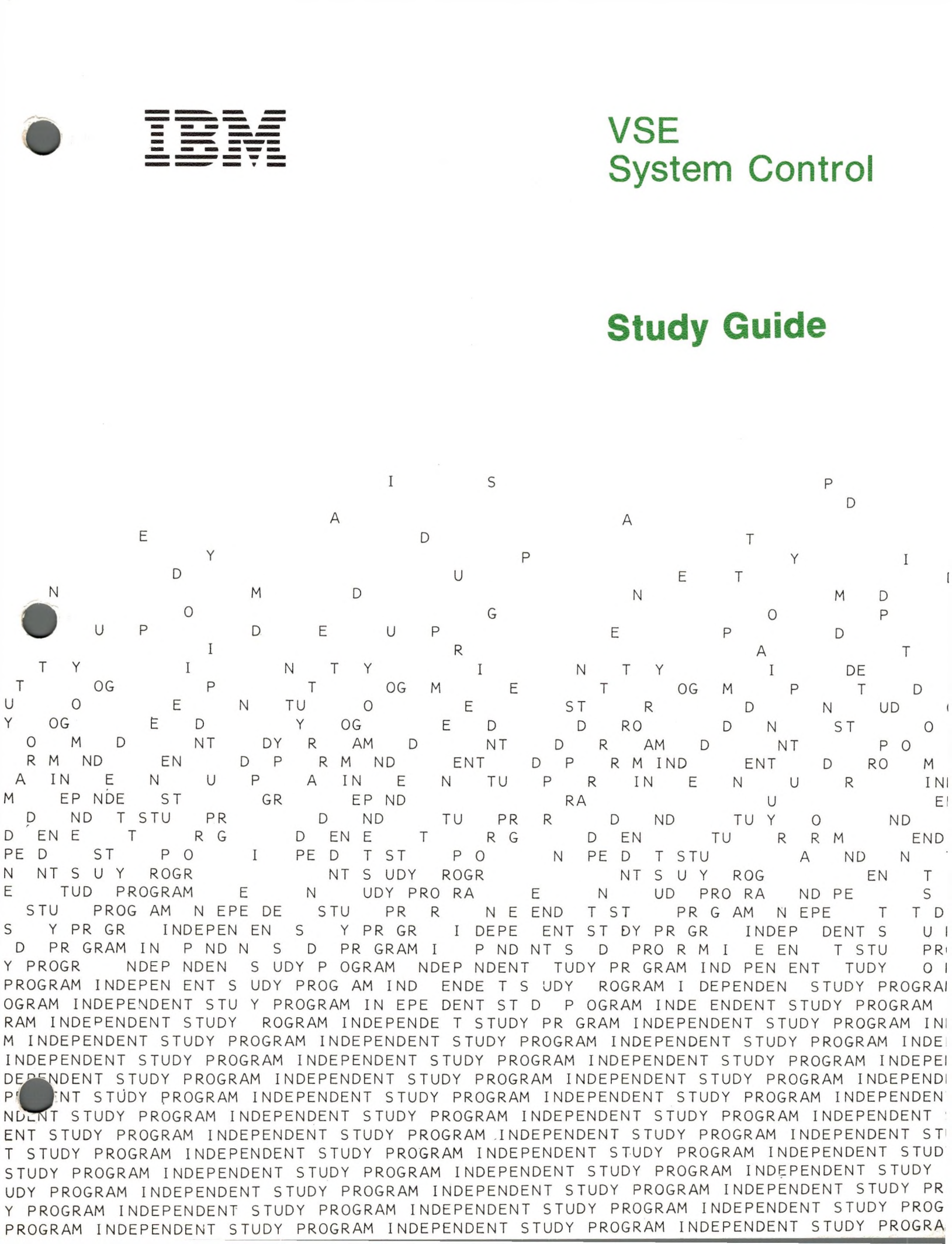


VSE
System Control

Study Guide

IBM

VSE
System Control
I0072

Study Guide

Independent
Study
Program

Major Revision (August 1980)

This publication is a major revision and obsoletes all previous editions.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available outside the United States.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. Address comments concerning the contents of this publication to IBM Corporation, Publications Services, Education Center, South Road, Poughkeepsie, New York 12602

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

INDEPENDENT STUDY PROGRAM
UPDATE SERVICE

Welcome to the IBM Independent Study Program.

To help you maintain this education material at the most current level, we have available an Update Service designed to provide you with Education Newsletters (ENLs). These newsletters contain updates and corrections that are brought to our attention through the Reader's Comment Form and various other sources.

If you wish to receive these ENLs, you may do so by subscribing to our System Library Subscription Service (SLSS). This can be accomplished by contacting your IBM Marketing Representative.

Course Introduction and Orientation

How To Start

On the next several pages you will find:

A Course Description -

Read it to make certain this is the course you want or need.

A list of Recommended Prerequisites -

Examine it to be sure you are prepared to take this course.

A list of Materials and Equipment -

Compare the list to the materials you have received or should have on hand. Be sure you have everything you need before you start the course.

A General Outline -

Read the outline to gain an overview of the course and to see what the relationship of the parts are to each other.

A description on How to Use this Study Program -

This section describes the features of the course. It will help you to find your way through the course and describes how you can use the materials to maximize their learning effectiveness.

Course Description

Audience

This course is for persons who want to be able to compile and test programs for batch mode implementation under VSE. The course is primarily intended for application programmers. It will be helpful to others such as system programmers, programming managers, system analysts, and operators who need to know the features and functions of the VSE System that relate to the implementation of programs in the VSE environment.

Overview

This course teaches how to control a VSE system when compiling and testing programs for batch mode implementation, including the coding of the following control statements:

- Job control language statements
- Linkage editor control statements
- Librarian program control statements

Objectives

Upon completing this course, you should be able to:

1. Describe the component programs of the VSE System and the interaction of system control and user programs.
2. Describe the characteristics of multiprogramming and virtual storage organization.
3. Code control statements to:
    a. execute a production program
    b. compile, link-edit, and execute a program
    c. catalog phases permanently into a Core Image Library
    d. create and check disk and tape labels
    e. catalog, access, and alter items in the various VSE libraries, including both system and private libraries
    f. execute a cataloged procedure and use the overwrite function to temporarily modify procedure statements for a particular execution

Duration

Thirty-six to forty study hours over a period of ten days should be allowed for taking the course, assuming a student obtains three or four machine runs per day for doing the assigned computer exercises.

Prerequisites

Persons taking this course are assumed to have:

1. Successfully completed the Programming Fundamentals ISP or had equivalent training.
2. Studied the System/370 Fundamentals student text (SR20-4607) or equivalent.
3. Completed the Introduction to 4300 and DOS/VSE Facilities self study (SS057).

4. Received training or experience in using a programming language. This requirement is not mandatory, but individuals completely unfamiliar with programming concepts will have more than average difficulty with the course material.

No previous knowledge of VSE or DOS/VS is assumed or required.

Materials and Equipment Required

The following materials and equipment are required for successful completion of this course. Before you begin to study, take the time to check and make certain that you have everything you will need.

DP Equipment

The following system facilities are required if the computer exercises are to be run:

- Minimum VSE system
- Four cylinders of disk storage space or 250 FBA blocks per student
- One tape drive (if computer exercise five is run)
- One card reader and card punch (if card decks are used)

Learning Materials

Study Guide (SR20-7300) - Appendix B contains a listing of the Computer Exercise Card Deck (SR20-7302)

Computer Exercise Card Deck (SR20-7302) or equivalent

Computer Exercise Solutions (SR20-7301)

Audio and Video

None.

Reference Material

In order to complete the course, students will need access to the following VSE reference material:

Introduction to the VSE System (GC33-6108)

VSE/Advanced Functions System Management Guide (SC33-6094)

VSE/Advanced Functions System Control Statements (SC33-6095)

VSE/Advanced Functions Messages (SC33-6098)

Each student must have convenient access to the reference manuals, as they are an important part of the course material.

General Outline

How to Use This Study Program

Administration

The materials and equipment you will use to learn the subjects covered in this course have been designed for self study. Although an advisor is not required to administer the course, it would be helpful if you had someone you could refer to for technical advice should the occasion arise. Someone should be available to help you interface with your installation's operations department when running the computer exercises.

Study Plan

The Independent Study Program environment allows you to create your own study plan, but it is important that you bring the same level of attention to the material presented here as you would to a classroom lecture. Try not to let your attention become distracted during your studies, and do the readings and exercises when they are assigned, exactly as you would in a regular classroom.

You or your manager may want to set up a study schedule with progress checkpoints to help you plan your work-study time and monitor your progress. The purpose of the checkpoints is to provide you with a daily or weekly objective and to afford you the opportunity to confer with a more experienced person on an area in which you may have questions.

A Student Progress Form has been provided to make it convenient for you to do this. It is found in the back of this book. You may also remove the form and use it as a place marker.

Sequence

The author has organized the materials in a logical sequence. Taking things in the order they are presented will help to make your study of the course easier.

Exercises

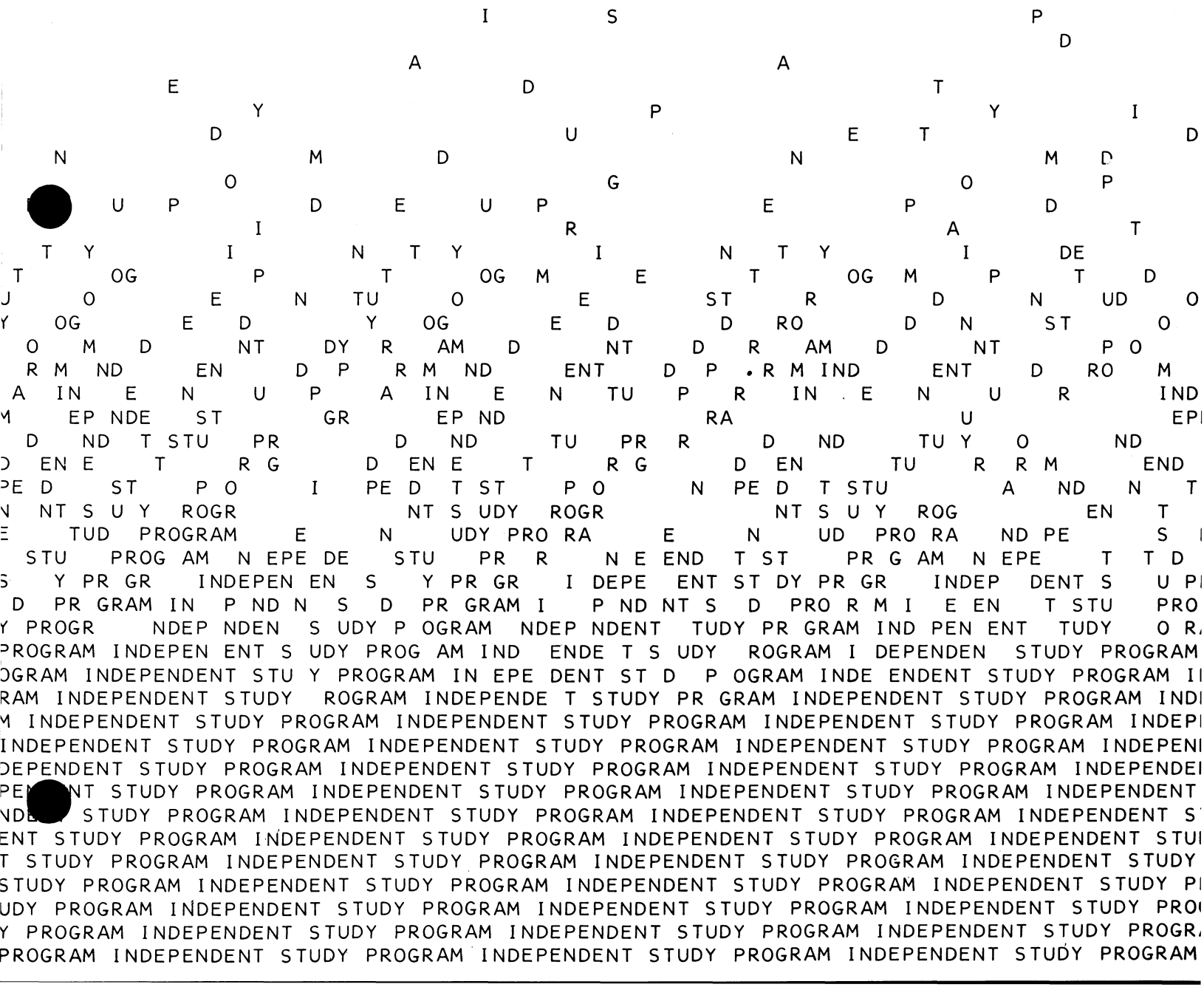
At points throughout the course you will be asked to take a quiz, solve a problem, or run a computer exercise. These exercises have been included to test your understanding, give you practice, and help you to remember the material. Follow the instructions in each case, and do not skip an exercise or look ahead to its solution until you have applied your best effort toward developing an answer. Except where otherwise specified, quizzes and tests are to be done without the aid of reference manuals.

If you desire to leave this text in a reusable condition, mark your answers to quiz questions on separate scratch paper instead of entering them in this book.

Solutions will be found on the pages following the exercises. In the case of computer exercises, solutions are located in the Computer Exercise Solutions book (SR20-7301).

Try not to allow large time gaps in your study of the material presented here. If this happens you will tend to lose the thread of the presentations and your learning curve will suffer. On the other hand, do not try to take it in all at once. Short breaks at the logically situated points provided (between Assignments within a Unit, and between Units) will give you the rest you need without destroying the continuity of your learning.

Unit 1



Unit 1
An Introduction to the VSE System

Introduction

VSE (Virtual Storage Extended) is a set of programs and libraries that makes efficient use of the resources of a data processing system. VSE, through a process of system generation, can be tailored for the specific hardware configuration of an installation and relieves the user of the burden of developing the wide range of programming support required by today's data processing applications.

The VSE System consists of the DOS/VSE System Control Program, the VSE/Advanced Functions Release 2 program product, and other related program products. It provides the support needed for processing in a multi-programming environment. It includes the potential for build-up of additional computing power and for adaptation to changing data processing requirements. For example, by including the VSE/Interactive Computing and Control Facility (VSE/ICCF), you change the installation's operating characteristics from a batch system to an interactive system. (This will be explained in Unit 8).

The concept of virtual storage is an important difference between the VSE system and its early predecessor, DOS (Disk Operating System). Users of DOS were restricted to a main memory address space limited by the storage physically contained in the central processing unit. Users of the VSE System, through a combination of processor hardware and programming support, have an address space that extends beyond the machine storage physically present. With virtual storage, the constraint imposed on program size by physical storage limitations is not absolute. Programmers no longer need to write programs to fit entirely within a computer's machine storage. This increases programmer productivity, since effort previously expended to make programs conform to size constraints can be diverted to other productive work such as developing new applications.

As an application programmer, system programmer, or manager, your job will bring you into contact with VSE in a wide variety of ways. The material presented here will give you what you need to know to use VSE in its most common manner. If you require further education in VSE, it is suggested that you consult the DOS/VSE pages in the current IBM Customer Education Catalog and Schedule (G320-1244).

Objective

Upon completing this unit, you should be able to:

Assignment 1

- Describe the three control programs of VSE.
- Describe the sequence of events required to initialize VSE for execution of processing programs.
- Describe a simple case of job-to-job transition.

- Describe how the job control language is used to communicate program requirements to VSE.
- Name the VSE libraries and describe the purpose of each.
- Define the three types of processing programs.

Assignment 2

- Define the term "multiprogramming" and be able to justify its use in a data processing system.
- Describe the functions of the shared virtual area.

Assignment 3

- Describe in a general way the means by which the VSE system controls storage utilization.

Materials Required

Study Guide (SR20-7300)

The following VSE reference material:

Introduction to the VSE System (GC33-6108)

VSE/Advanced Functions System Control Statements (SC33-6095)

Assignment 1 - System Components

The Control Programs

The capability of disk devices to access stored data directly makes them ideal for program storage, so the programs and libraries that form the VSE System are kept on disk. Figure 1.1 shows that the disk volume containing the most important operating system programs and libraries is called the system residence device, or SYSRES.

The three control programs of VSE are the initial program loader (IPL), the supervisor, and the job control program. They reside in a portion of SYSRES known as the system Core Image Library.

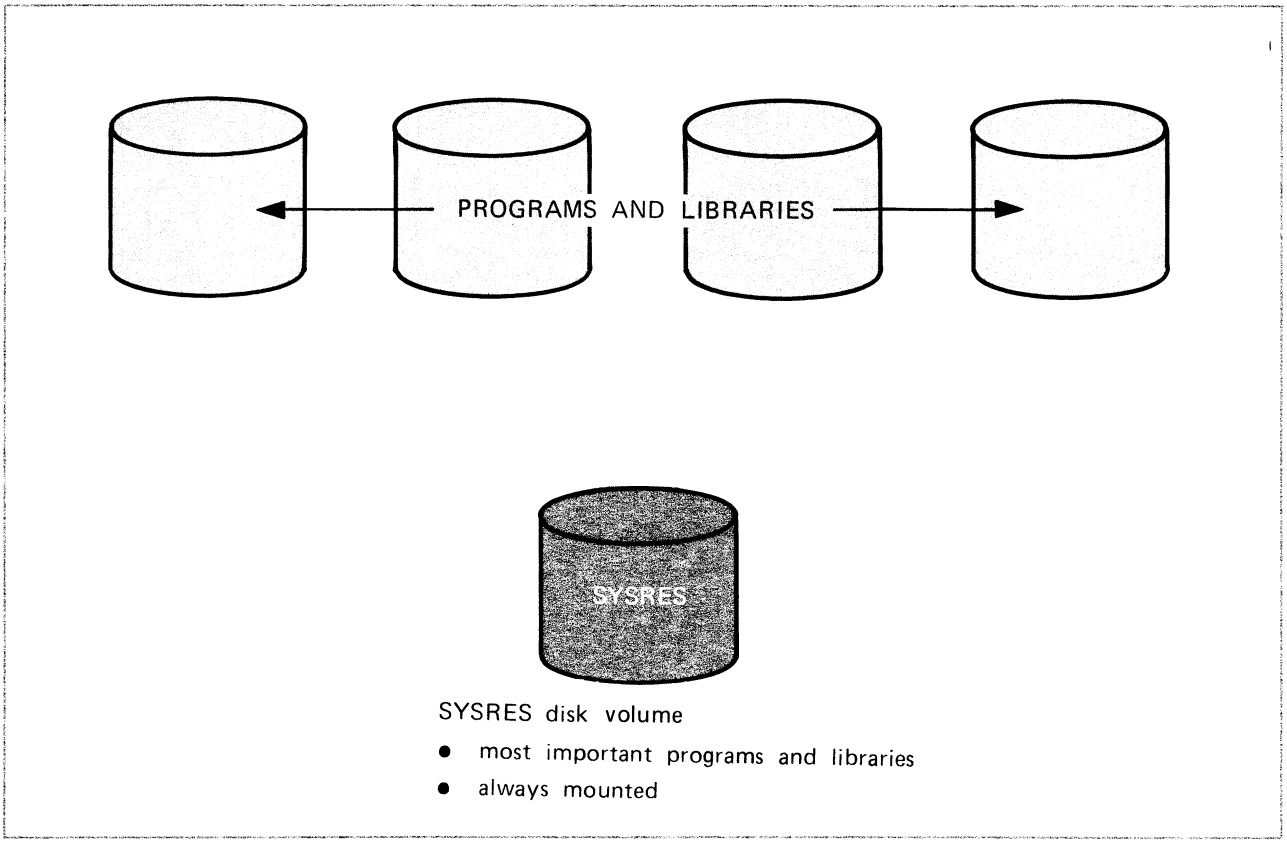


Figure 1.1 - VSE is Disk Resident

The IPL Program

This program is used to start your VSE system. Figure 1.2 shows the elements involved.

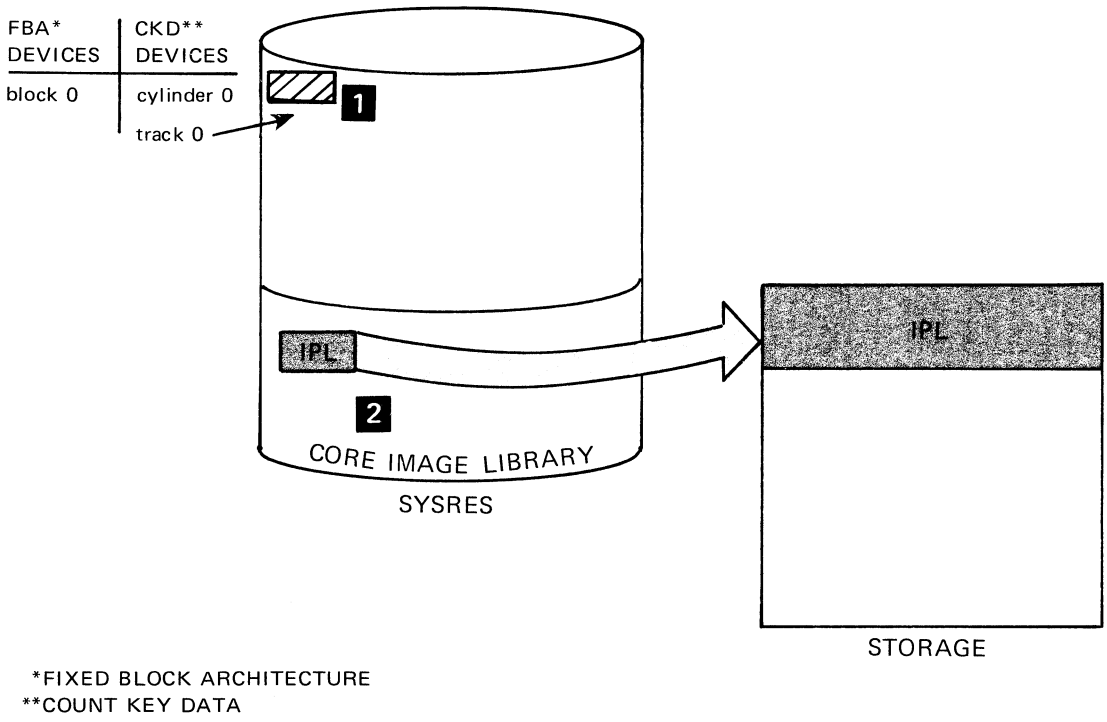


Figure 1.2 - The IPL Function

Sequence:

1. A small program, the "bootstrap," is located at a fixed address on the SYSRES pack. The bootstrap is brought into processor storage and executed when the operator performs the console load operation.

2. The bootstrap loads the IPL program. IPL performs certain required system initialization functions that are usually of no concern to the application programmer. These functions are described in the section "Initial Program Load" in the *VSE/Advanced Functions System Control Statements* manual.

The last thing IPL does is to load the supervisor and give it control of the system. Figure 1.3 shows the supervisor being loaded into main storage.

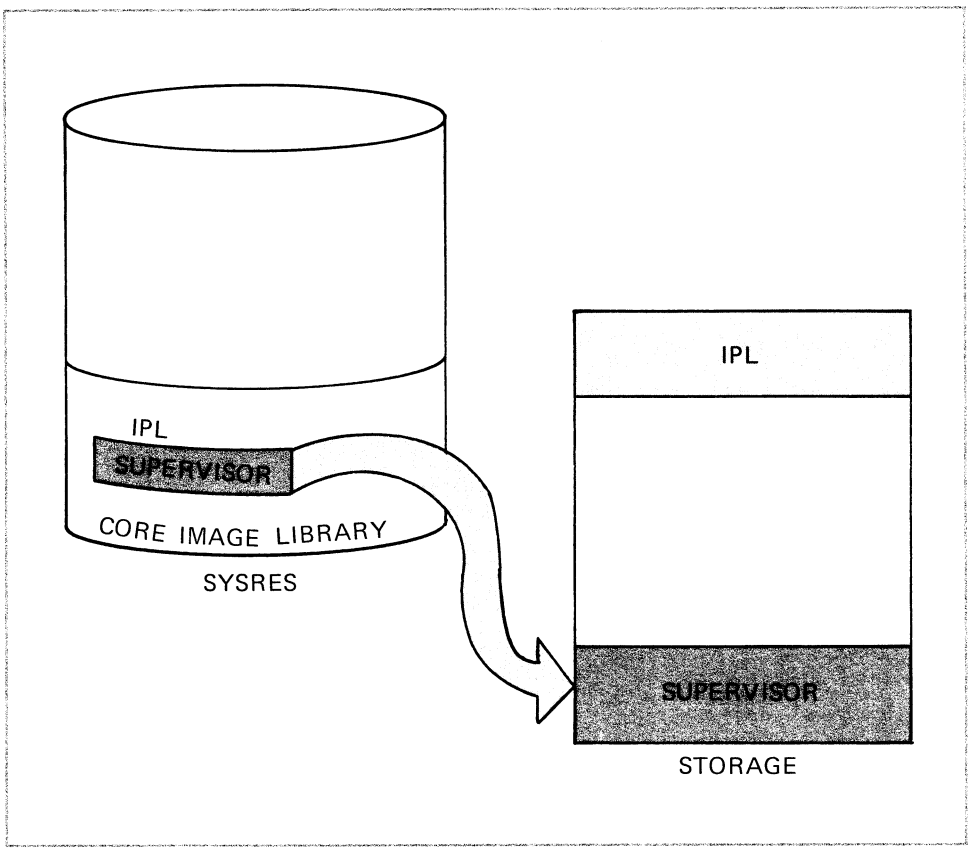


Figure 1.3 - The VSE Supervisor

The Supervisor

The supervisor loads into storage beginning at location 0. Once loaded, it remains in storage continuously during system operations. The supervisor initiates the loading and execution of all other programs and handles the input and output of data that is processed by any program executed in the system. It contains tables of control information and maintains communication regions that allow you to communicate with your program at execution time. It holds transient areas for certain supervisor routines that are only needed for specific functions, such as OPEN and CLOSE file processing.

The supervisor controls main storage utilization and the use of CPU cycle time. It handles communications to and from the operator, error recovery, and input/output operations. In a word, the supervisor controls the system.

The Job Control Program

Job control is loaded by the supervisor to begin execution of user programs. Job control acts on information you supply, such as program name and main storage and device requirements, to allocate resources and prepare VSE to execute your programs. The interaction between job control and a user program is shown in Figures 1.4 to 1.7.

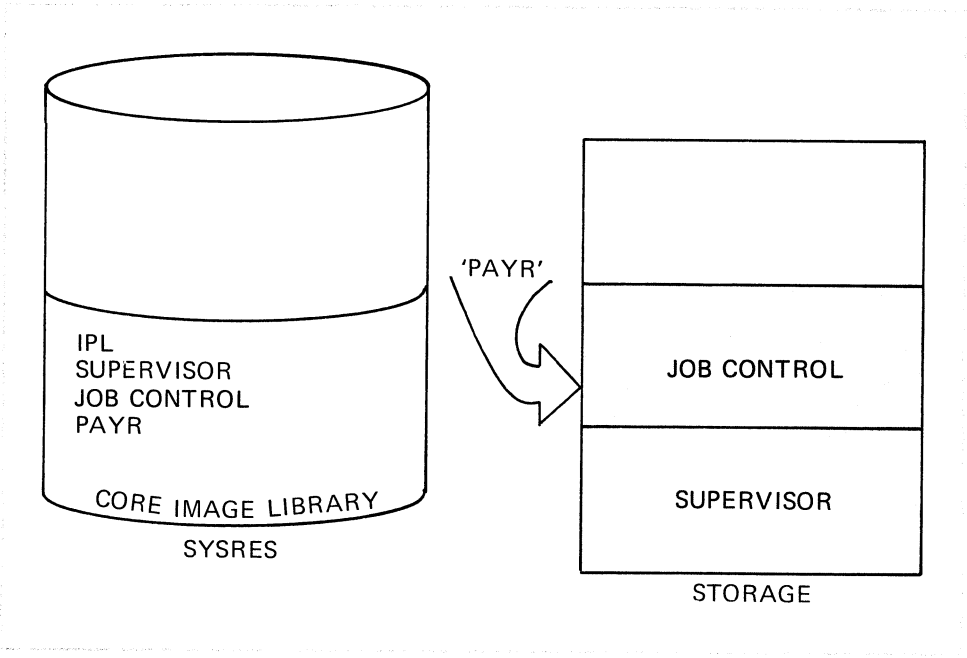


Figure 1.4 - Initial State

1. Job control is informed that we wish to run a program named PAYR. This done by specifying the execution of PAYR on an EXEC job control statement. Notice that PAYR exists in the Core Image Library.

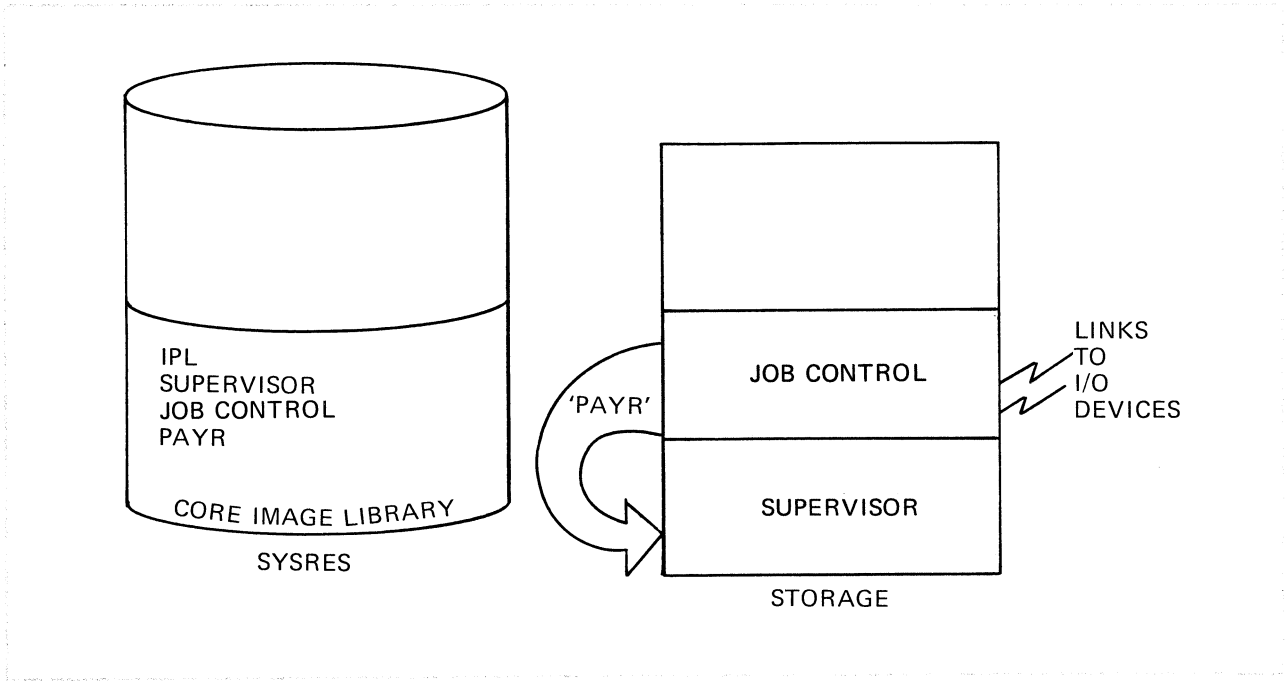


Figure 1.5 - Job Control Prepares for PAYR

2. Job control makes sure that the storage and device requirements of PAYR are satisfied, then informs the supervisor that PAYR is to be executed.

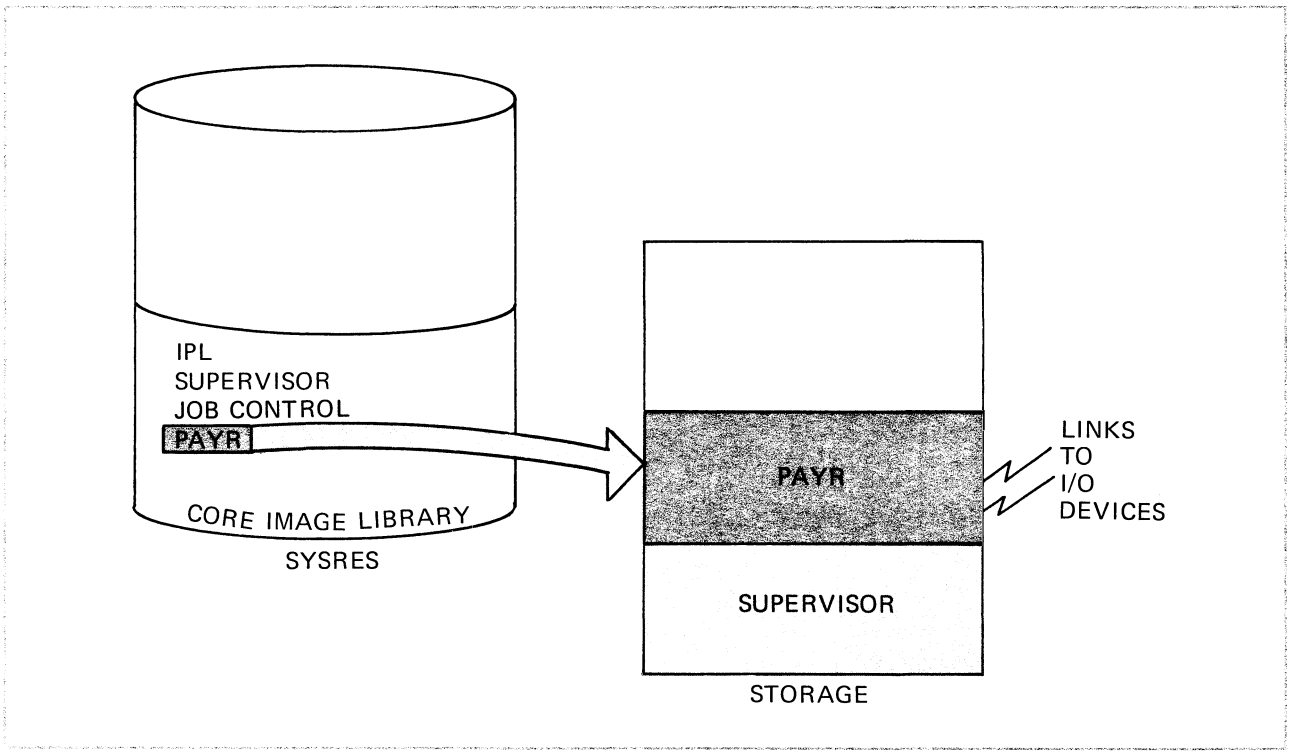


Figure 1.6 - PAYR is Loaded for Execution

3. The supervisor loads PAYR and allows it to begin execution. Figure 1.6 shows that PAYR overlays the job control program. This is possible because job control is not needed in this area of storage during the time that PAYR is executing.

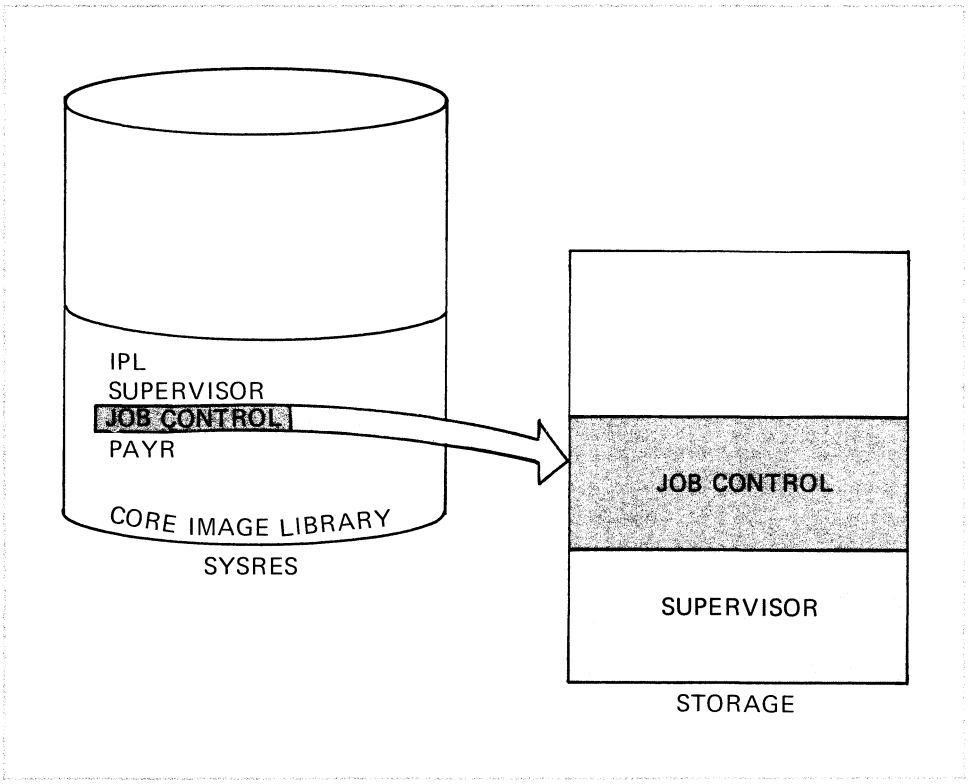


Figure 1.7 - Final State = Initial State

4. The supervisor knows when PAYR has completed its processing. It reloads job control, overlaying PAYR. The cycle is complete and will be repeated to set up for execution of the next user program.

Notice in this last diagram that the I/O links are gone. This is because they were needed only for PAYR. The next program will probably have a different set of requirements which will in turn be set up by job control. This sequence of events is called job-to-job transition.

Basic Job Control Concepts

In the document:

Introduction to the VSE System

Under the heading:

"Resource Management and Job Control"

Read:

Up to but not including "Loading Programs for Execution." When you have completed the reading assignment, do the review Exercise that follows.

(Note that this reading assignment does not specify actual page numbers in the referenced manual. This is because the reference manuals are frequently updated, and pages are sometimes renumbered. To avoid any possible confusion, all references made here to the manuals will be to topic headings.)

Exercise 1.1

After completing the Exercise, compare your answers with those provided, then continue in this Unit.

1. When writing a program to run in a virtual storage system, the programmer need not be concerned with _____.

   a. program size

   b. program correctness

   c. program speed

   d. program efficiency

2. The IPL program activates VSE. The _____ causes IPL to be loaded.

   a. job control program

   b. operator

   c. supervisor

   d. SYSRES device

3. A program named PROGA is loaded into storage for execution by the _____.

   a. job control program

   b. operator

   c. supervisor

   d. SYSRES device

Questions 4 to 7 refer to the following job stream:

```
1. // JOB  A
2. // EXEC PROG1
3. /*
4. // EXEC PROG2
5. // EXEC PROG3
6. /&
7. // JOB  B
8. // EXEC PROG4
9. /&
```

4. There are _____ jobs in this job stream.

   a. 1

   b. 2

   c. 3

   d. 4

5. There are _____ requests for program execution in this job stream.
   a. 1
   b. 2
   c. 3
   d. 4

6. How many job steps are there in Job A?
   a. 1
   b. 2
   c. 3
   d. 4

7. When all the statements of this job stream have been processed, what program is in storage along with the supervisor? Explain your reasoning.

8. The _____ statement associates a physical device with a symbolic device name.
   a. EXEC
   b. ASSGN
   c. JOB
   d. COMMENT

9. The designation SYS023 is a _____.
   a. tape drive
   b. disk drive
   c. system logical unit
   d. programmer logical unit

10. The function of SYSRDR is to read _____.
    a. system data
    b. VSE/VSAM catalog data
    c. job control statements
    d. error records

1. a
2. b
3. c
4. b
5. d
6. c
7. The job control program is in storage along with the supervisor, ready to do the required set-up operations for the next job read into the system.
8. b
9. d
10. c

The VSE Libraries

There are four types of libraries used in a VSE System. They are called the Core Image, Relocatable, Source Statement and Procedure libraries.

Every DASD volume used by VSE must be identified by a six character name called a volume serial number or a volume ID. Typically the DASD volume which contains the libraries devoted to system functions has a volume ID of DOSRES. This is also the volume used to IPL the system.

The contiguous area on the DOSRES volume that contains libraries dedicated to the system is referred to as the system residence file (SYSRES).

By definition any library within the SYSRES extent is called a SYSTEM library and any library outside that area is called a PRIVATE library. Otherwise the structure, organization, and format of system and private libraries is identical.

There can be only one system library of each type within the SYSRES extent. There can be as many private libraries of each type as required by the installation.

Figure 1.8 shows how private libraries can exist on DOSRES as well as on other packs whose volume IDs are SYSWK1, SYSWK2, or SYSWK3.

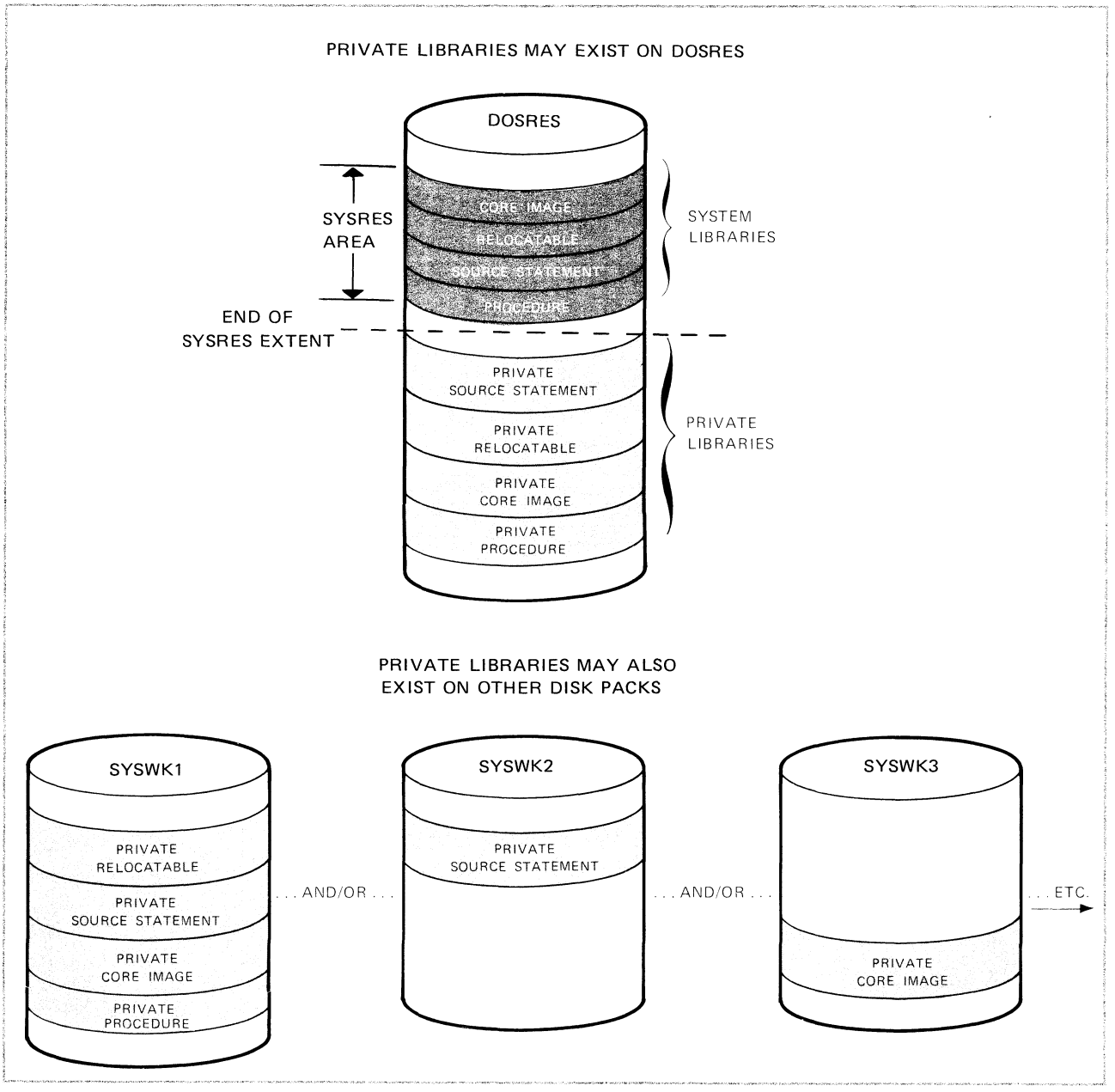


Figure 1.8 - System and Private Libraries

Core Image Library

The system Core Image Library (CIL) is the only library that is always required. It contains the IPL, supervisor, and job control programs, as well as all the other programs that make up your VSE system.

Programs in a CIL are in executable format and are called *phases*. Any program to be executed under VSE must first be stored in a CIL. The supervisor will load programs for execution only from a CIL, be it system or private.

When a source program is processed by a language translator (any of the compilers or the Assembler), a machine language version of the program called an *object module* is produced. Under VSE these modules may reside in a Relocatable Library (RL).

Object modules are processed by the linkage editor program to construct the executable phases that reside in the CIL.

When you write a new program, you can place the source code in the Source Statement Library (SSL). Programs stored in the SSL are called *books*. Books can be copied from the SSL into source programs, and individual statements may be added to, deleted from, or updated within books in the SSL. This means that a source program may be maintained in the SSL and that maintenance of the program in card deck form (or on diskettes) is not necessary.

Frequently used sets of job control statements may be stored in a Procedure Library. They are then called *procedures*. A procedure can be invoked from the procedure library with a single job control statement. Thus, the use of procedures can reduce the volume of control cards to be read into a system. This reduces card handling by the system operator and improves availability of your system's card readers.

The four libraries are structured as illustrated in Figure 1.9.

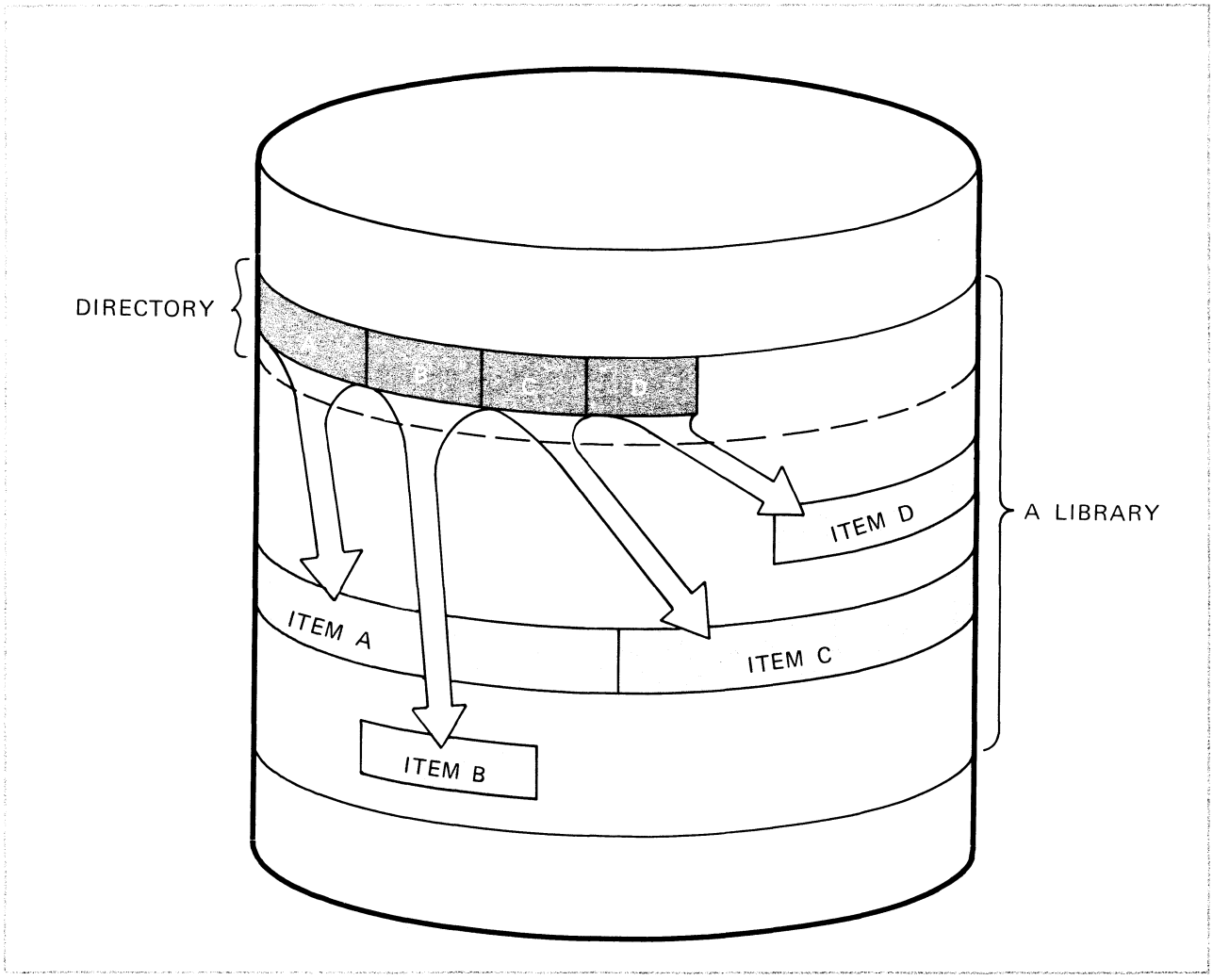


Figure 1.9 - Structure of the Libraries

A directory at the beginning of each library contains an entry for each member in the library. Thus, the CIL directory contains an entry for each phase in the Core Image Library, the RL directory contains an entry for each module in the Relocatable Library, and so forth. Directory entries contain the name of the member they reference and the location of that member in the library proper. A directory entry is made automatically when a member is added to a library.

Private Libraries

Private libraries can be viewed as extensions of the system libraries. Individual users or user departments can maintain phases, modules, or books in separate private libraries that are independent of each other and of the system libraries. In this way there is no threat of running out of room in the system libraries.

Programs being written or revised and undergoing testing can be kept in private libraries without interfering with any production programs that reside in the system libraries. Once development is completed and the programs are ready for operational status, they may be retained in the private library, or they may be copied into the system library. Whether a given program is to reside in a system or private library is determined by your installation.

Processing Programs

The VSE system control programs (IPL, the supervisor, and job control) control the execution of another class of programs called processing programs. These consist of application programs, service programs, and language translators.

Application Programs

An application program (user program) is a program that applies to the work in your installation. Most application programs are written by VSE users, while some are obtained from software vendors, such as IBM, on a purchase or lease basis.

Service Programs

Service programs assist in the use of VSE without contributing directly to the control of the system or to the production of results. Examples are the librarian programs, the linkage editor, and the system utility programs.

Librarian Programs

Librarian programs are used to maintain the VSE libraries and to display their contents.

Linkage Editor Program

The linkage editor processes object modules to produce phases, which it places in a Core Image Library.

System Utility Programs

These programs perform functions such as copying data from one volume to another and preparing volumes for the storage of data.

Data Management Routines

These routines are available for use by your programs whenever they need to write or read data to or from an external storage device. They consist of Physical Input/Output Control System (PIOCS) routines and Logical Input/Output Control System (LIOCS) routines.

PIOCS

The PIOCS routines are located in the supervisor. They work in conjunction with the LIOCS routines to perform the work associated with transferring data between external storage (tapes, disks, printers, etc.) and main storage. It is the PIOCS routines that actually issue the commands that cause this data transfer to take place.

LIOCS

These routines do the blocking and de-blocking of records, and pass requests to PIOCS to perform data transfers to and from I/O devices. The LIOCS routines supplied by IBM may be maintained in the system Relocatable Library, and automatically included as part of your program by the linkage editor that prepares your program for execution.

In addition to these service programs there are several programs available on a lease basis from IBM to perform specialized functions. These include Sort/Merge, VSE/DITTO, VSE/ICCF, and VSE/POWER.

The Sort/Merge programs are used to arrange records into some sequence or to combine (merge) two or more files that are already in a desired sequence into a single file.

The VSE/DITTO program provides a quick and easy way for the programmer or operator to perform a variety of unit record functions. These include the duplication of card decks and tape or disk utility operations, as well as more sophisticated tasks.

The VSE/ICCF (Interactive Computing and Control Facility) transforms the VSE batch system to a system that can operate in both batch and interactive modes. In interactive mode, you are able to communicate directly with the VSE system from a display terminal.

VSE/POWER improves overall system performance by reducing the CPU's dependence on the relatively slow speeds of its unit record equipment (card readers, punches, and printers). By intercepting requests for these devices and simulating their functions on direct access storage, VSE/POWER allows unit record operations to proceed at disk speeds.

These products and others are available only to users of the VSE/Advanced Function feature.

Language Translators

Language translators convert source language programs to machine language programs called object modules. The Assembler language translator is included with the DOS/VSE system. Other translators are available on a lease basis.

Exercise 1.2

Complete this Exercise before going on to the next Assignment. If you miss any of these questions, review the appropriate material in the text.

1. Private libraries are _____.
   a. viewed as extensions of system libraries
   b. used to store programs undergoing testing
   c. the same as system libraries in their structure, organization, and format
   d. all of the above

2. To be loaded for execution a program must reside in a _____.
   a. Core Image Library
   b. Relocatable Library
   c. Source Statement Library
   d. Procedure Library

3. The output of a language translator is a module that could be placed in the _____.
   a. Core Image Library
   b. Relocatable Library
   c. Source Statement Library
   d. Procedure Library

4. Frequently used sets of job control statements may be stored in the _____.
   a. Core Image Library
   b. Relocatable Library
   c. Source Statement Library
   d. Procedure Library

5. The linkage editor and the librarian are examples of _____.
   a. Application Programs
   b. Service Programs
   c. Control Programs
   d. Language Translators

Solution

1. d
2. a
3. b
4. d
5. b

Assignment 2 - Multiprogramming Concepts

Single Program Execution

The flow chart in Figure 1.10 represents a single program running under VSE in your CPU. Examine the logic flow and see if you can determine if there is any point in the program's activity where there is a "hidden" time delay. Once you have made your determination, look at Figure 1.11.

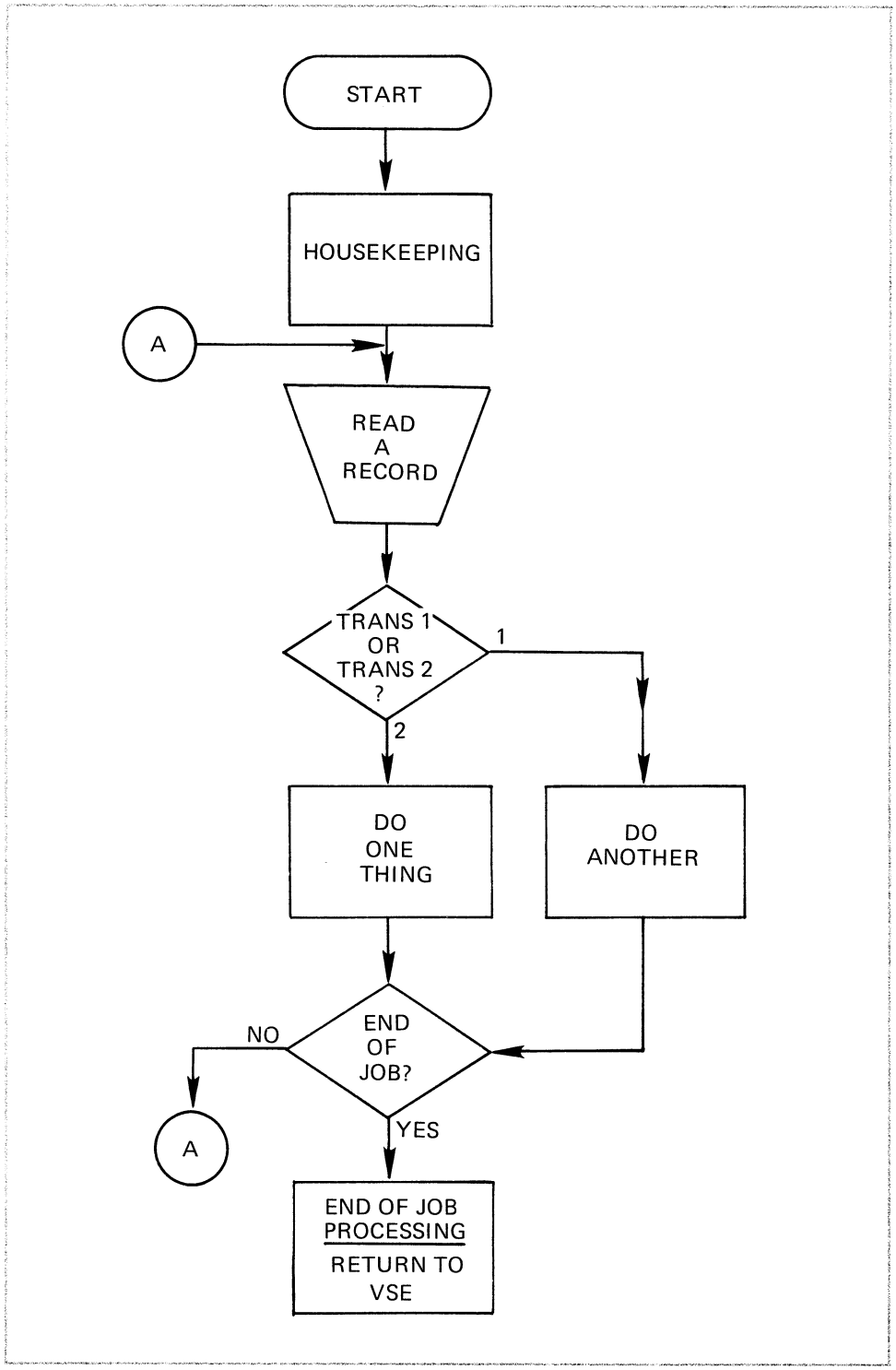


Figure 1.10 - Where is the "Hidden" Delay?

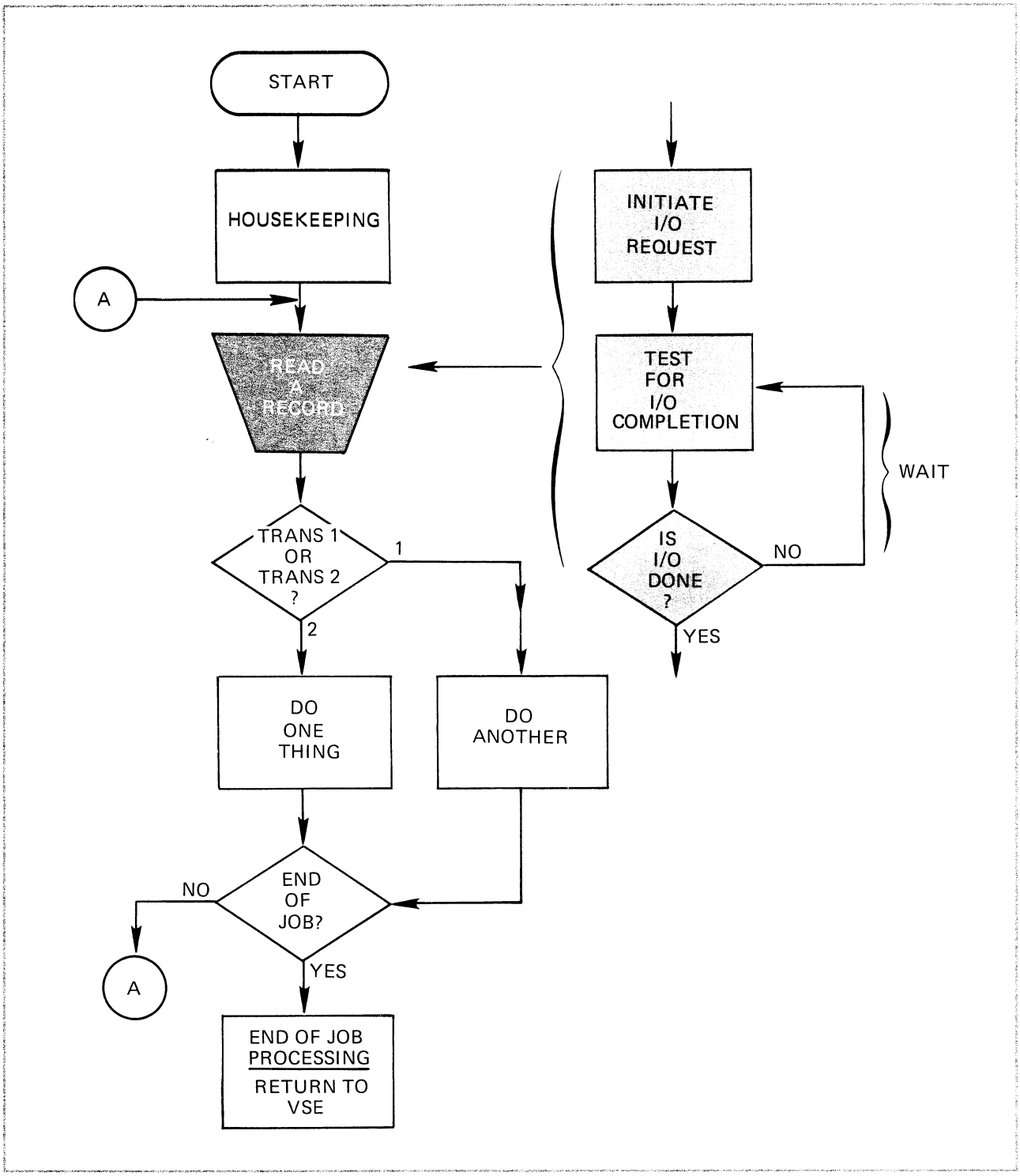


Figure 1.11 - We Must Wait For the Read to Complete

The delay involved is the "wait" time resulting from the read I/O request. In order for this program to process a transaction record, it must have that record available in processor storage. The read I/O operation will retrieve the record from where it resides on cards, tape, or disk, but countless CPU cycles (the smallest periods of time in which CPU activities take place) are allowed to go by unused while the program waits for its I/O request to be fulfilled.

Compared to the CPU's internal speed, the rate of data transfer between an I/O device and processor storage is very slow. The reading or writing of data involves mechanical as well as electronic actions. Positioning of disk access mechanisms, punching cards, and the like are time consuming operations that slow down a program's performance. If a given program is the only user of the CPU, then the CPU is essentially idle while that program is waiting for completion of its I/O operations.

Rather than waste the expensive resource of CPU time, it is preferable to allow more than one program to be active simultaneously. These programs are independent of each other. While Program A, for example, is waiting for an I/O operation to complete, Program B can be doing its processing.

Remember, only one thing can take place in the CPU at any one time, so Program A and Program B cannot both be using CPU cycles at the same instant. Either one of them, however, can process "in the gaps" created by the wait time involved for the other's I/O activity.

Storage Organization

In the document:

Introduction to the VSE System

Under the topic:

"Resource Utilization"

Read:

Up to but not including "Multitasking."

When you have finished the reading, do the review Exercise that follows, then continue in this text.

Exercise 1.3

1. Which of the following operations take the longest time to complete?
   a. executing a multiply instruction
   b. executing an add instruction
   c. reading data from a disk drive
   d. retrieving data from internal storage
2. Multiprogramming is designed to make efficient use of _____.
   a. I/O devices
   b. CPU cycles
   c. program interrupts
   d. operator interventions
3. VSE allows the user to divide the problem program area into as many as _____ partitions.
   a. 5
   b. 7
   c. 14
   d. 12
4. Explain in your own words the reason for a priority system in a multiprogramming environment.

5. By default, the highest priority partition under VSE is the _____ partition.
6. An I/O bound program is one in which there is a great deal of I/O activity, while a CPU bound program is one in which there is a minimum of I/O activity and a great deal of CPU calculations.

   In a multipartition system would you choose to put an I/O bound or a CPU bound program in the highest priority partition? Explain your answer.
7. The default priorities may be changed by _____.
   a. an operator command
   b. a partition override
   c. the supervisor
   d. a problem program

Solution

1. c
2. b
3. d
4. In an environment where more than one program may be active simultaneously, a priority system is necessary to resolve situations when more than one of these programs is ready to use the CPU at the same time.
5. F1
6. An I/O bound program should go into the highest priority partition. The I/O bound program allows for interrupts to occur as it waits for completion of its input/output requests, and the lower priority partitions can get control of the CPU at these times.

   If a CPU bound program were in the highest priority partition, it would not allow for a sufficient number of interrupts to take place so that other partitions sharing the CPU could gain control within reasonable periods of time.
7. a

The Shared Virtual Area

In addition to the partitions specified at system generation time, the user must also specify the mandatory shared virtual area (SVA). This is done by the user at IPL time. This area of storage is used for three purposes:

1. To hold reenterable program phases. These routines are available for use by any program (or programs) active in any number of partitions at any time. SVA routines must be reentrant, that is, they must allow for simultaneous use by one or more programs. Thus, even though only one copy of the routine exists in the SVA, it can be shared by any number of programs. In this way, the SVA routine does not have to be physically duplicated in each program where it is needed.

   This capability allows the IBM supplied modules of the Virtual Storage Access Method (VSE/VSAM) to be loaded and executed from the SVA rather than from the user's partition. This means that if several programs in different partitions are using VSE/VSAM at the same time, only one copy of the VSE/VSAM code (in the SVA) will be needed to service these users.

2. To hold the system directory list (SDL). This directory contains entries (consisting of phase names and locations in the SVA) of each SVA routine, as well as entries (consisting of phase names and locations in the CIL) for selected CIL members that require rapid loading when requested for execution. When one of these CIL phases must be loaded, the supervisor locates it by using the SDL entry rather than the CIL directory entry. Program loading is sped up by avoiding the usually required access to the CIL directory for these phases. The SDL used in this way can be thought of as an index to selected phases. The SDL is created at IPL time.

3. To hold the system GETVIS area. One of the uses of this area is to contain the IBM supplied Rotational Position Sensing (RPS) routines. These routines increase the efficiency with which certain disk storage devices can be accessed.

Assignment 3 - Virtual Storage Concepts

A Case Study

One of the constraints that until recently was imposed on the computer user has been the amount of machine storage available for the execution of programs. The user has had to be aware of the amount of machine storage associated with his particular computer, and to take that size limit into account when developing application packages. This size limitation often complicated the entire development process. Assume, for example, the environment pictured in Figure 1.12.

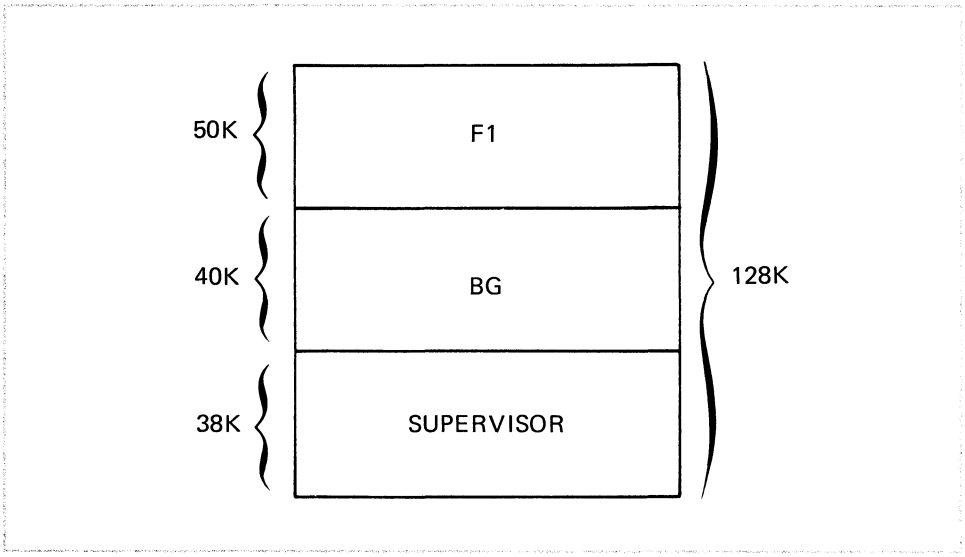


Figure 1.12 - Hypothetical Environment

Note: The sizes of the supervisor and partitions as shown in Figure 1.12 are for illustrative purposes to explain the virtual storage concept. They do not represent actual VSE system storage requirements.

Here, 128K of machine storage is available, organized as the diagram illustrates. The 128K figure is an absolute constraint on the user--at any given time the sum of the sizes of all programs running in the machine cannot exceed this value.

Assume that the F1 partition must have 50K allocated to it for an online data retrieval application. That leaves 40K for the BG partition. Your job is to write a Sales Analysis program to fit within that area. You do your best, but the program you develop exceeds available storage by 15K. See the next illustration.

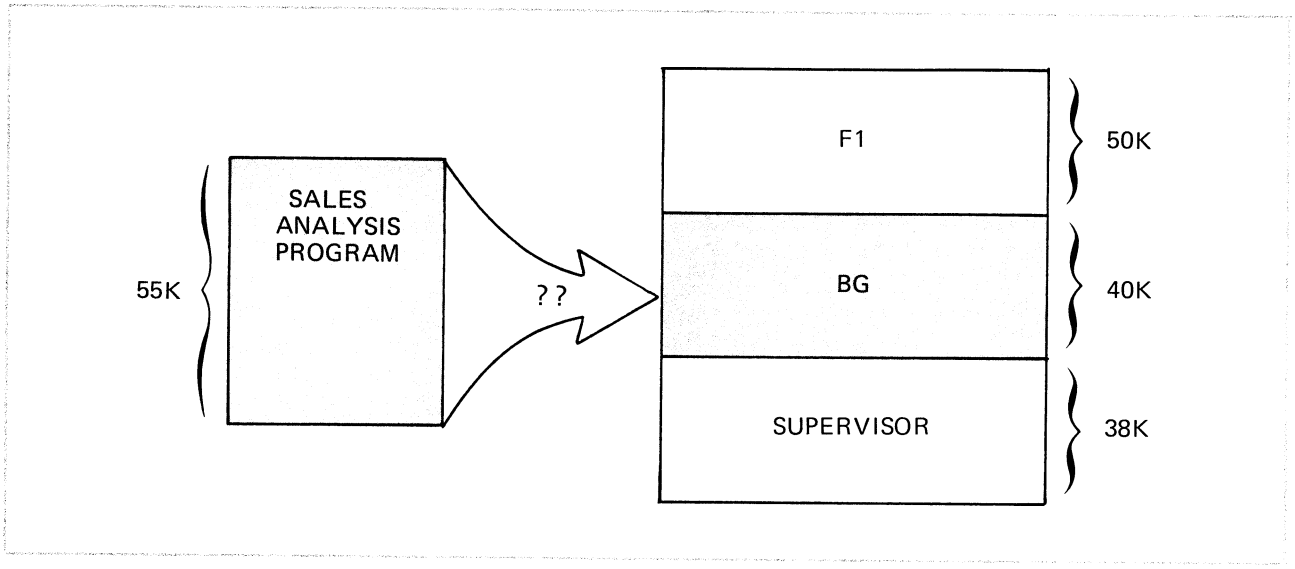


Figure 1.13 - A Problem of Size

You are not able to reduce the size of your program and have it perform according to specifications. Your only alternative at this point is to divide the program into logical segments, so that the only piece of it occupying storage at any time is what is needed at that time. If you have done your job correctly, your program is constructed in a modular fashion to begin with, that is, it is made up of functional pieces (subroutines) that fit together to form the whole program. See Figure 1.14.

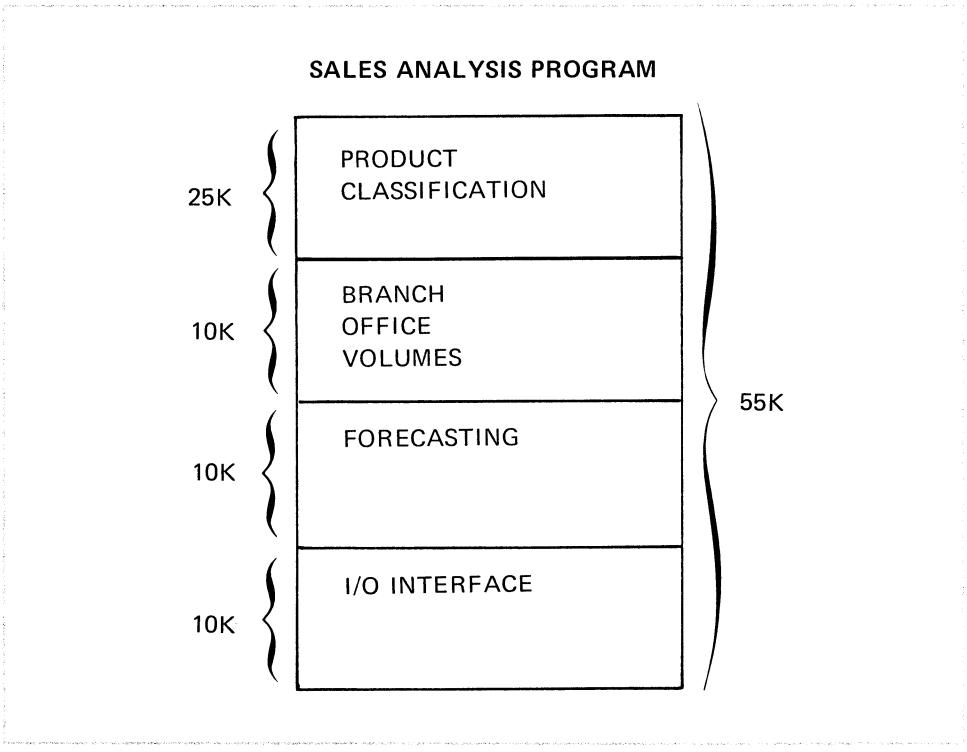


Figure 1.14 - Program Components

Now assume that the 10K I/O Interface must be present when any other co:nponent of the Sales Analysis program is active. Take a piece of scratch paper, draw some boxes to represent

the 40K of machine storage you can use, and draw in the various combinations of parts of the Sales Analysis program that can be active together. Use the component sizes given in Figure 1.14. When you are done, look at Figure 1.15.

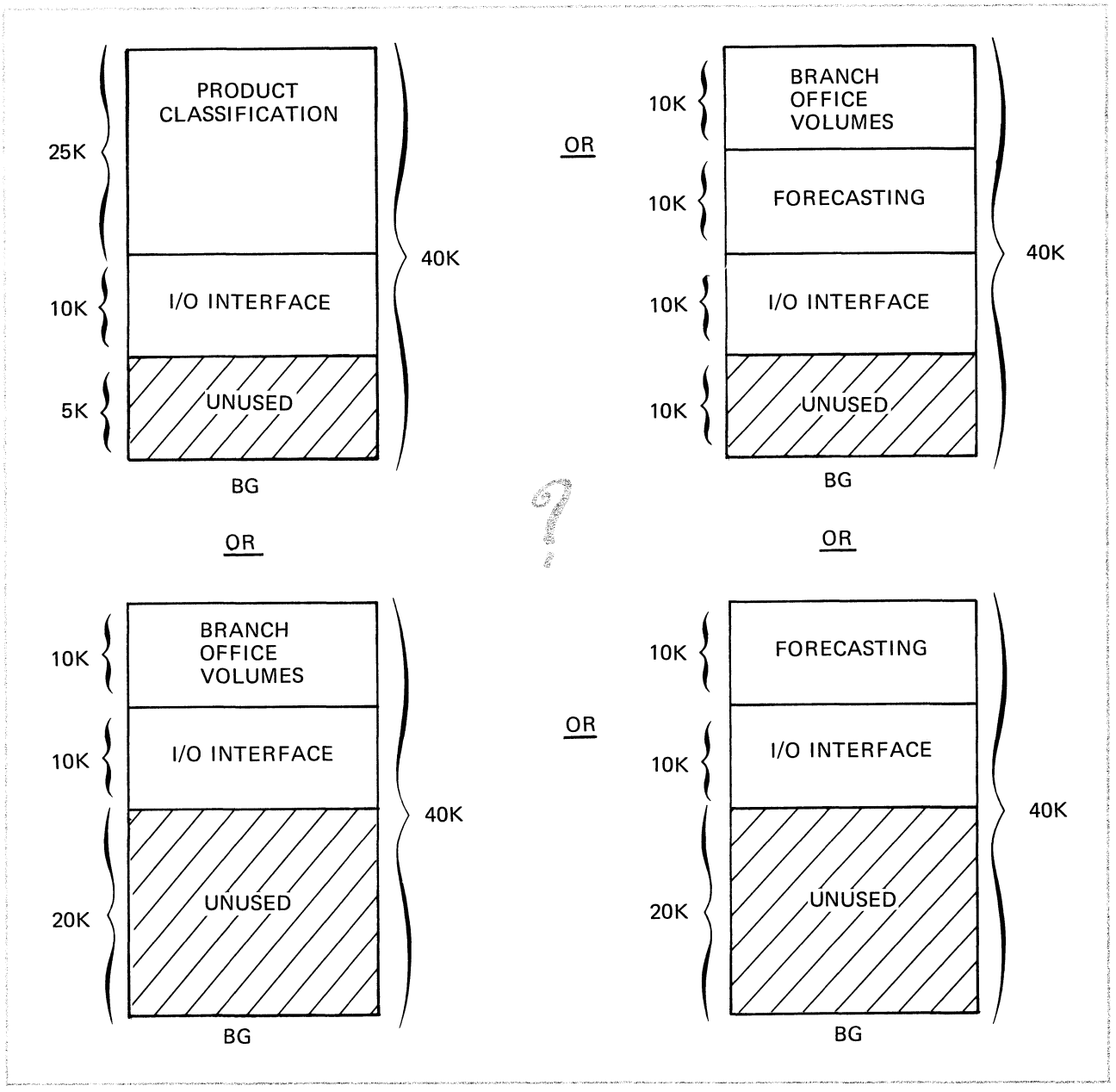


Figure 1.15 - The Possible Combinations

The most efficient use of the partition (from the point of view of storage utilization) occurs when the Product Classification segment is active, but even here 5K is left unused. The other possible cases leave 10K or 20K unused. Remember, the I/O interface must always be present (an assumption for this example), and this prevents the 25K Product Classification segment from being active along with the Forecasting or Branch Office Volume segments.

Thought has to be given to developing and maintaining areas in machine or external storage for communication between parts of the program. The I/O Interface portion might be designed to hold this common communication area, but this is not the point. The point is that time, energy,

and human resources have to be expended to develop an implementation technique. If the program could be written *as if* it had available to it as much storage as it needed, there would be no need to be concerned with:

1. The most efficient use of machine storage.
2. How to divide the program.
3. Communication between parts of the program.
4. Additions to the program that might cause available storage to be exceeded again.

But this is not the end of it. Take a look at one of the combinations as illustrated in Figure 1.16.

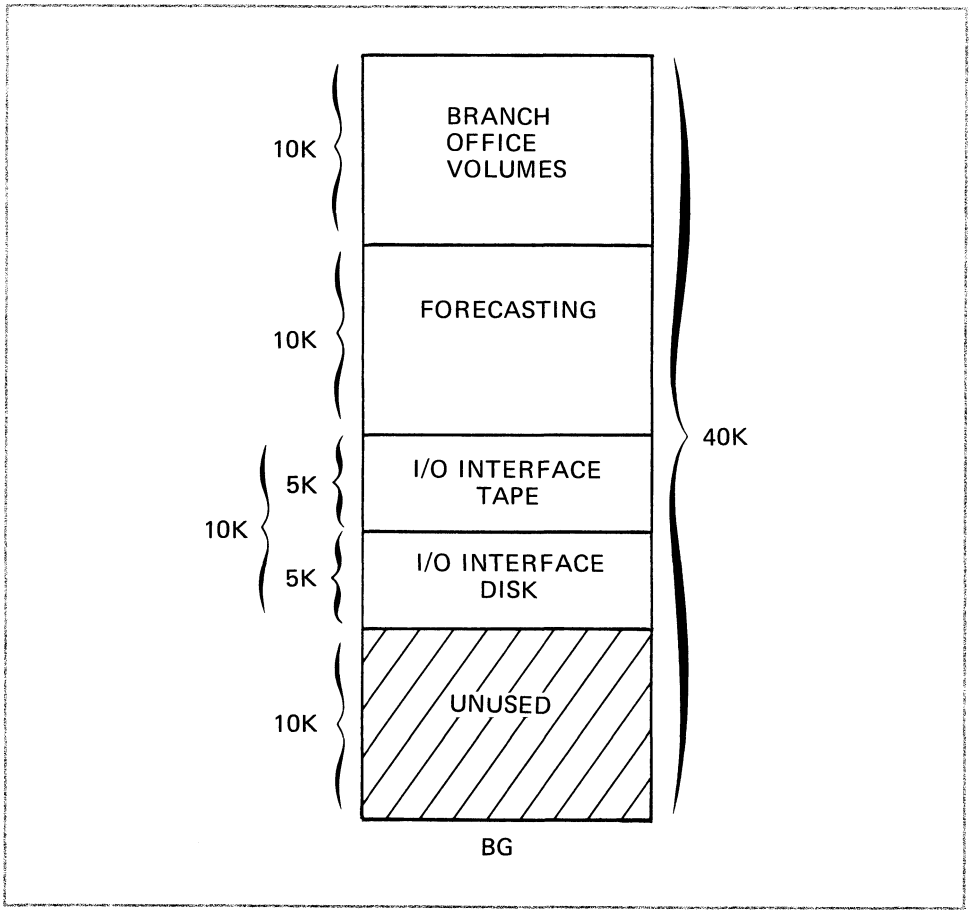


Figure 1.16 - What is in Use?

Notice that the I/O Interface portion has been divided into its two components, tape I/O and disk I/O. (Remember, these are arbitrary choices for this example). At any given time, either the Branch Office Volumes segment or the Forecasting segment will be actively executing instructions, and will be accessing either tape or disk. In fact, the program might not be accessing any device--it might be just doing processing that does not require external storage. Assume for now that the Forecasting code is active and needs information stored on disk. See Figure 1.17.

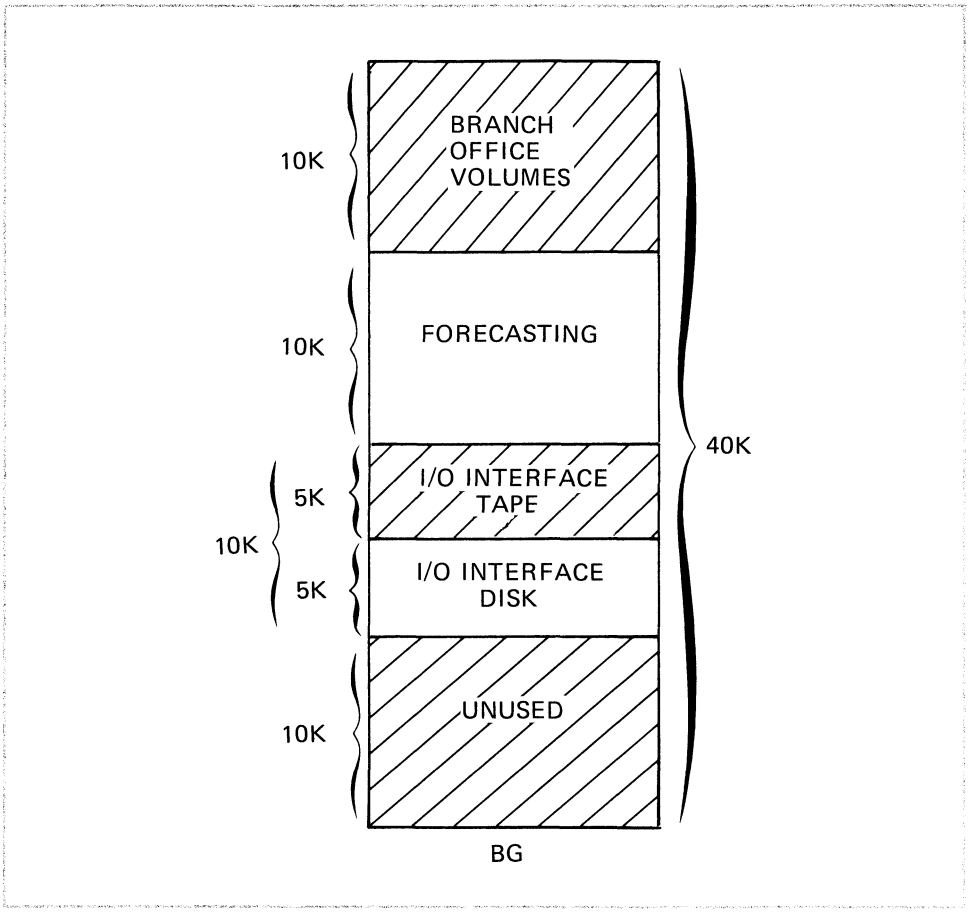


Figure 1.17 - How Much Storage is Really in Use? (Shaded Areas Inactive)

You can see that less and less of the background partition is being used productively. Indeed, the point has been reached where more space is inactive than active: 25K vs. 15K. Naturally, there will come a time when the Branch Office Volume code and the I/O Interface Tape code will be required--it is then that these pieces of the application are considered to be active while the other pieces are inactive.

Suppose you can determine how much time is spent in each of these four routines during a typical run. See Figure 1.18.

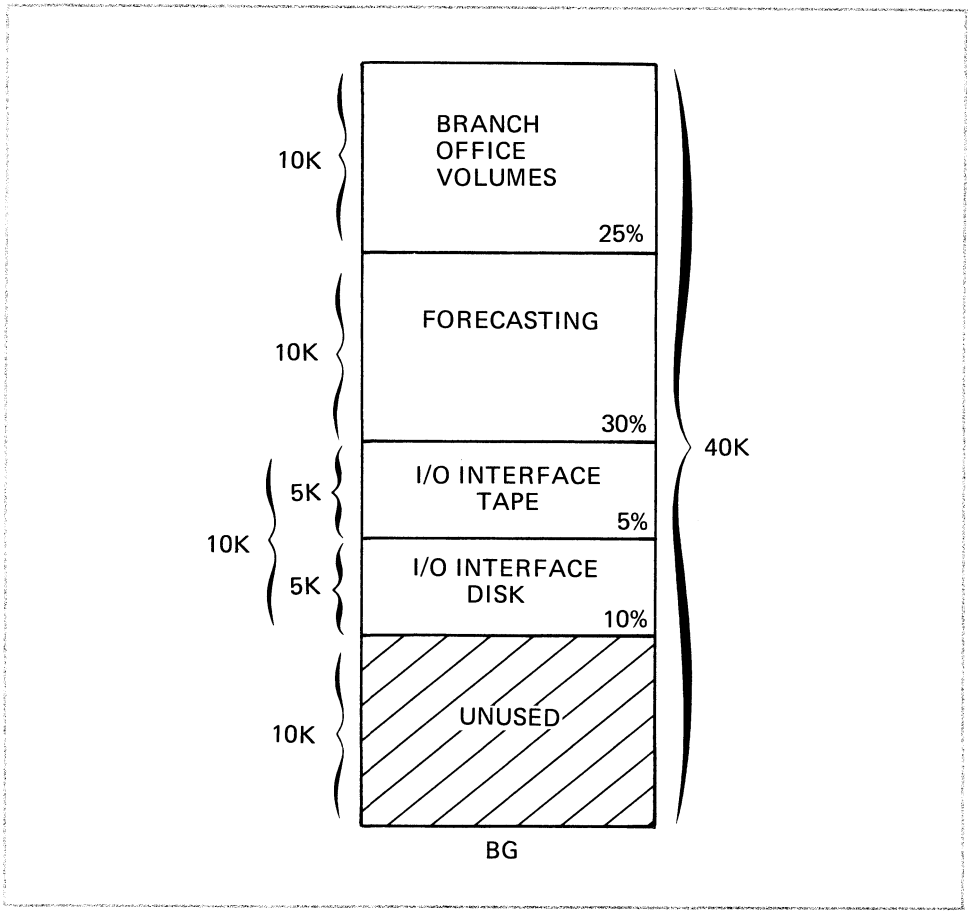


Figure 1.18 - Per Cent of Utilization

All kinds of games can be played with these numbers, but the point to be recognized is that for most of the time most of the code is inactive. Only a small part of any program is required to be active at any given point in time. In this example it is easy to see that if there were only 15K available, it would be enough to hold the active program portions. Figure 1.19 illustrates this idea.

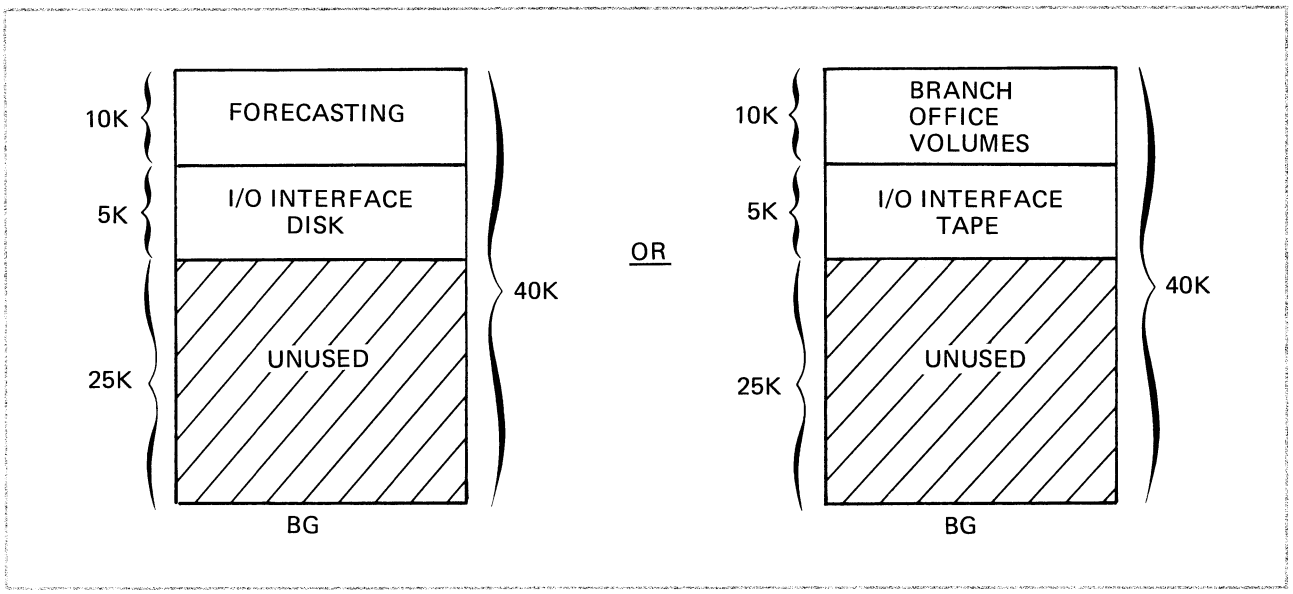


Figure 1.19 - The Active Portions

The way the program is now organized leaves 25K of the BG partition unused. This is enough to hold the biggest segment, the 25K Product Classification code. If you wanted this code to be available in storage at the same time as either of the two combinations shown in Figure 1.19, it could be done as illustrated in Figure 1.20.

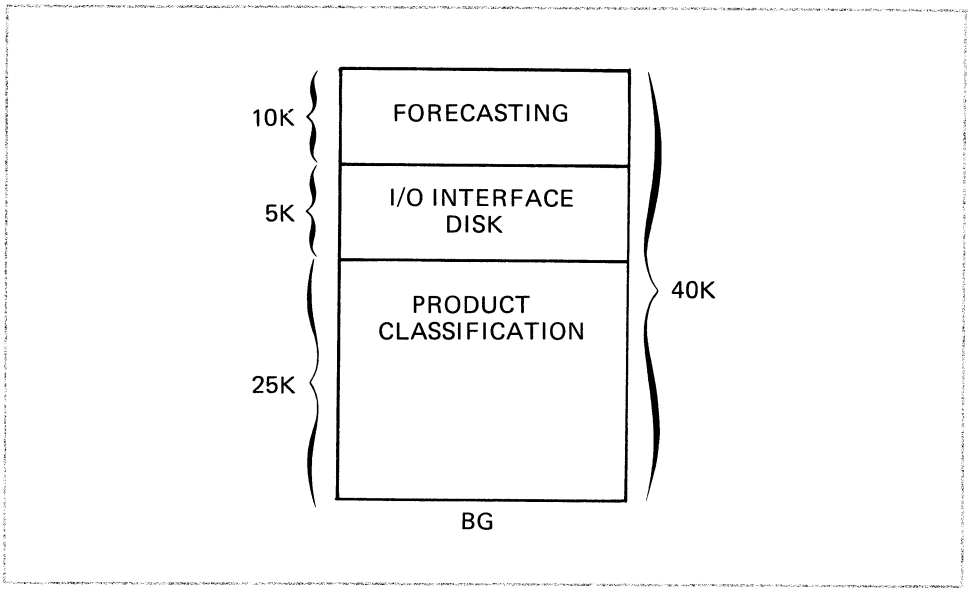


Figure 1.20 - Full Utilization--The Problem Solved?

We seem to have come full circle, and simply by breaking the I/O Interface code into two 5K pieces have achieved full utilization of the BG partition. Or have we? Take a look at Figure 1.21.

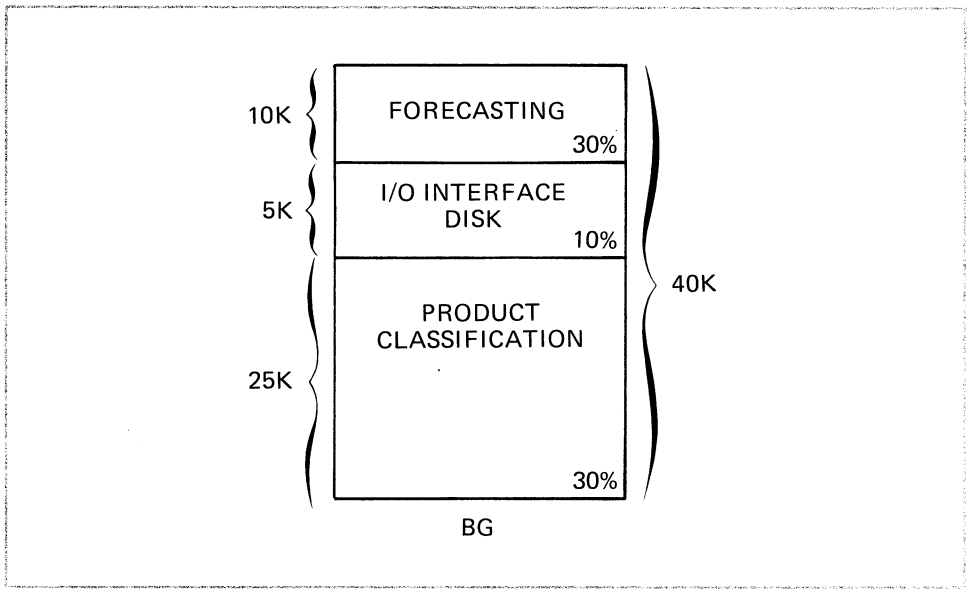


Figure 1.21 - What is Missing?

The percentages indicate the amount of time each piece is active. But, as with Figure 1.18, the numbers do not add up to 100%. That's because for 30% of the time the Branch Office Volume and I/O Interface Tape code is required to be in storage, just as in Figure 1.18, 30% of the time is required for the Product Classification code to be in storage.

What happens when the Product Classification code in Figure 1.21 needs the I/O Interface Tape module to fulfill a tape I/O request? See Figure 1.22.

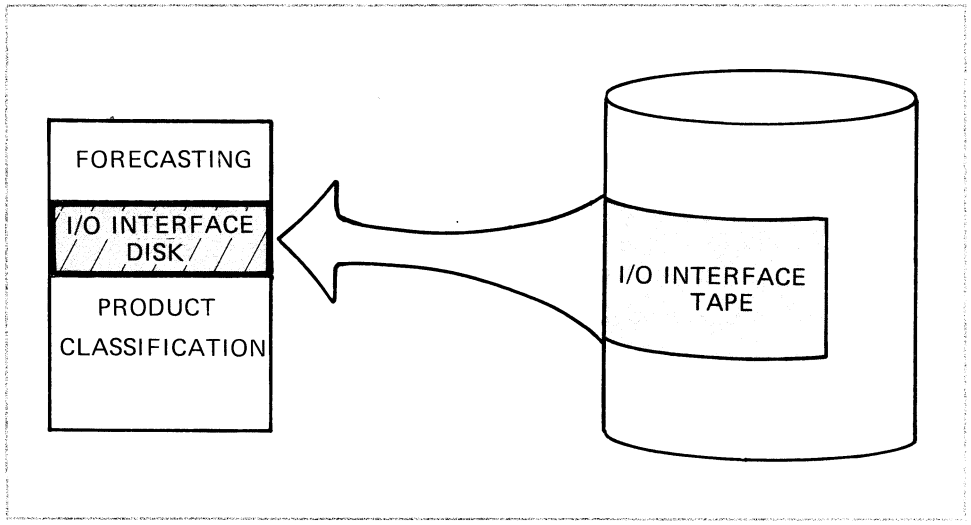


Figure 1.22 - Block Replacement 1

The block of code called the I/O Interface Disk module must be replaced by the block of code called the I/O Interface Tape module. The incoming block could fit into any 5K area in the BG partition not currently active--the disk code seems the most convenient to be replaced in this illustration.

Assume that the tape I/O request is fulfilled, and that control is eventually passed to the Forecasting module. What happens when the Forecasting code needs some calculations done

that are handled by the Branch Office Volume segment? Where should this 10K segment be put? See Figure 1.23.

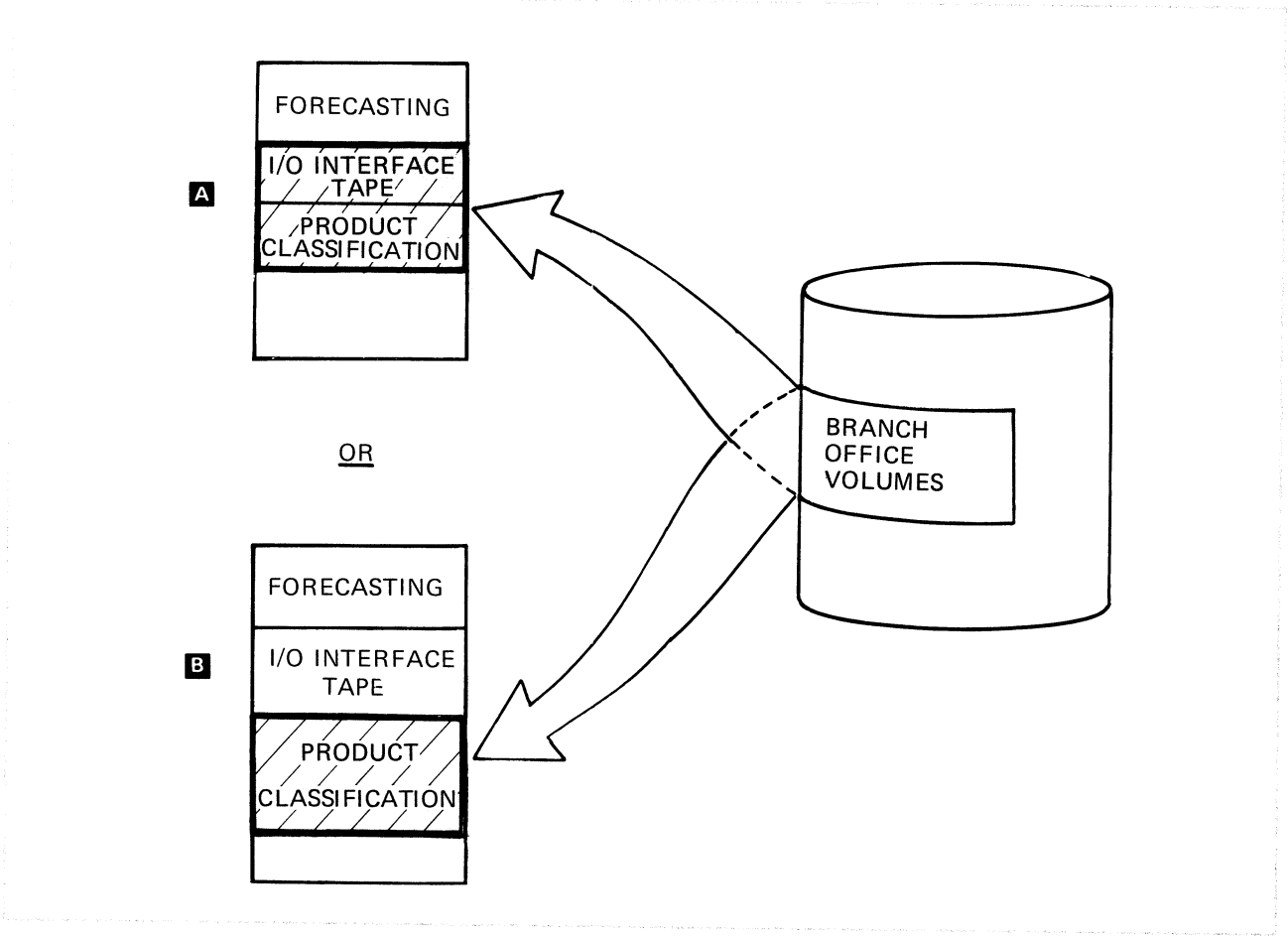


Figure 1.23 - A Matter of Choice

The Branch Office Volume code could overlay 5K of the Product Classification module along with the 5K devoted to an I/O module, or it could overlay 10K that is entirely within the Product Classification module.

The path of least resistance seems to be choice **B**. This means the I/O Interface Tape code will not have to be brought back in the next time there is a request for a tape operation--it will be there. But whether **A** or **B** is chosen, part of the Product Classification module is going to be overlayed by the incoming code. This presents another problem.

What if the Product Classification code about to be overlayed has been *modified,* that is, changes have been made (updated tables, added to counters, incremented indices, etc.) that will influence its action when next it gets control? This part of the program cannot be simply destroyed, so a copy of it must be made on disk for later retrieval. When this code is needed later, it can be loaded from disk to resume its processing functions. See Figure 1.24.

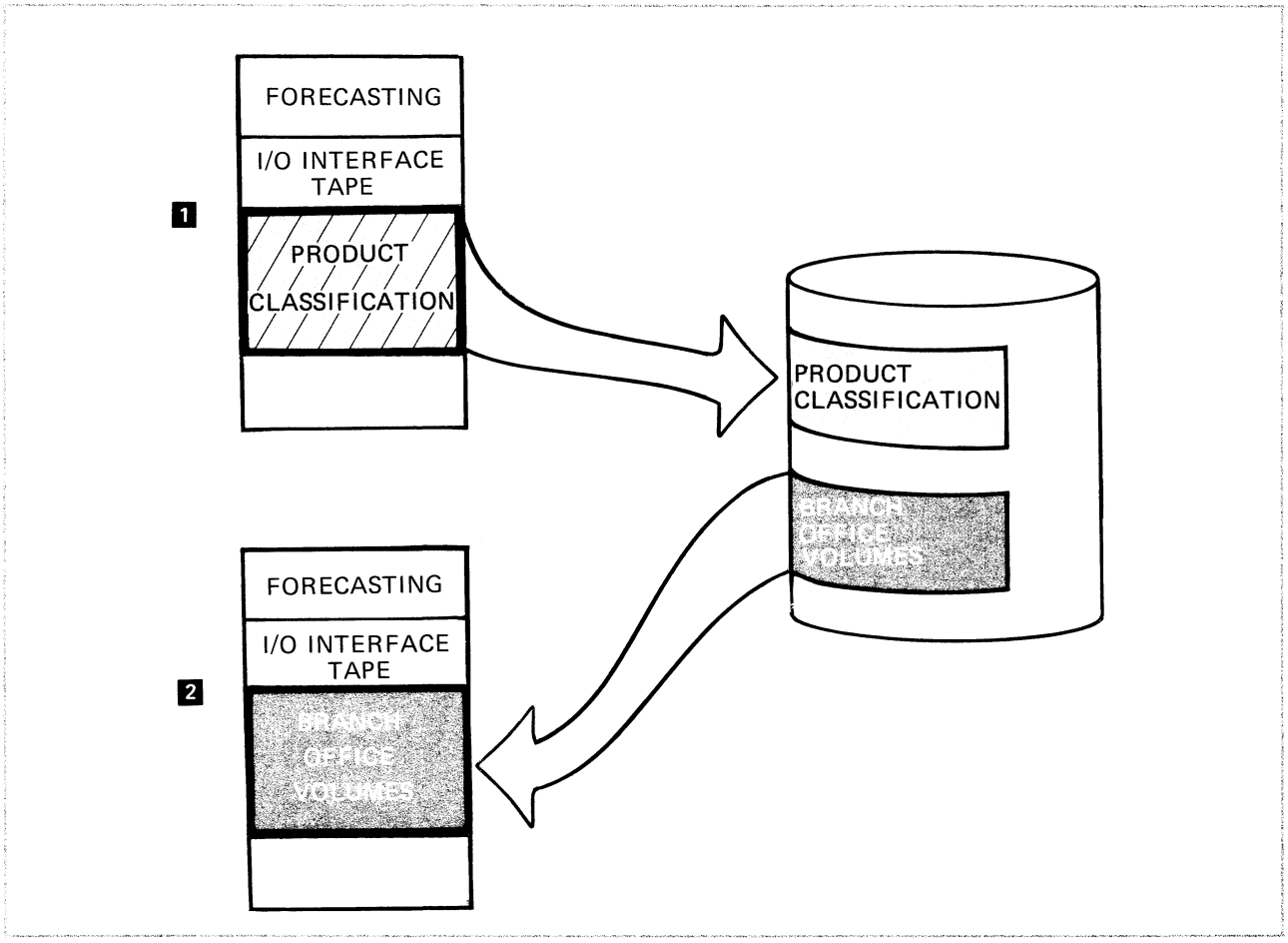


Figure 1.24 - Block Replacement 2

1. The code that resides in the area of storage to be overlayed is saved.
2. The code required to be active is loaded for execution.

The Virtual Storage Technique

Figure 1.25 shows what has happened to disk intermediate storage. 40K of machine storage has been "turned into" 55K of effective storage by using a disk to hold program portions not needed at any given time. This is exactly the concept of the Virtual Storage method of storage organization.

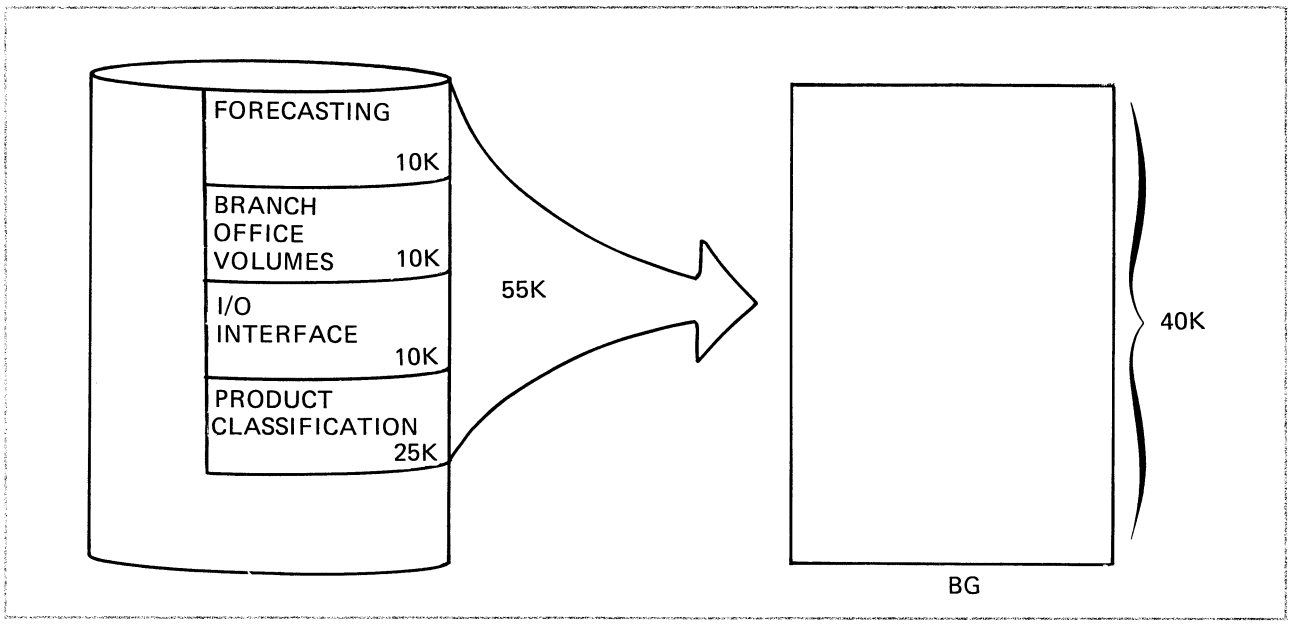


Figure 1.25 - 40K Appears to be 55K to the User

Implementation

Prior to the advent of Virtual Storage organization, the system and application programmers at DOS installations were responsible for the mechanics of managing storage in situations like the one in this case study. Through careful design and judicious use of the linkage editor and various disk I/O macros, or through special language translator features such as the ANS COBOL compiler's segmentation option, programs were developed that managed their way in and out of machine storage as has been done here. This approach has several disadvantages:

- It gets very complicated.
- It requires a lot of time and energy to implement.
- It does not solve the whole problem.

The whole problem is that all of your application programs should be treated this way. At any given time only the active parts of programs in various partitions should be actually using machine storage. Virtual Storage organization handles the resource of machine storage in the way you have seen it done in the Sales Analysis application just presented.

Part of machine storage is "turned into" a larger size of effective storage by using the disk file known as the Page Data Set to hold program portions not needed at any given time. This process is called *mapping,* and is accomplished by a combination of hardware (electronic) and software (VSE supervisor) functions.

The units of storage mapping, corresponding conceptually (but not in size) to what we have termed "blocks of code" in our case study, are called *pages.* The programs you write occupy some number of these pages when they execute under VSE. The supervisor ensures that only the currently active pages are present in machine storage at any given time. The term *page pool* is used to represent the portions of machine storage available for mapping operations.

The effective storage can have an upper address limit of 16 million bytes. Application programmers can code as if they have as much storage available to them as their programs require.

Machine Storage Allocation

The available machine storage is used for two purposes:

1. Partition and SVA mapping. This is the portion of machine storage where blocks of active program code execute. Each virtual partition in your system, as well as the SVA, will be mapped onto the Page Data Set and associated with part of the available machine storage.

2. Dedicated machine storage programs. Certain applications will demand that there be no mapping of code between storage and the Page Data Set. Remember, this transfer of program information takes time, and highly time-dependent applications may require a dedicated block of machine storage that is not subject to the rules of program mapping. Only a minimum of application programs in most installations are of this type.

In general, Virtual Storage organization is meant to be "transparent" to the user, which means you act and code as if you had much more storage available than you really do and let VSE handle the implementation. The application programmer does not usually need to be concerned with any greater level of detail than has been presented here.

Reading Assignment

Although you may have read portions of this reading material, now you should read the whole sections.

In the manual *Introduction to the VSE System* read the sections "Initial Program Load" and "Resource Management and Job Control".

Unit Summary

VSE is a Disk Operating System for Virtual Storage that can be tailored through the process of system generation to run your IBM System/370 or 4300 Processor in an efficient manner. The programs and libraries which comprise VSE reside on disk.

Three system control programs - IPL, the supervisor, and job control - are major components of VSE. The system is started by the IPL program; operation is controlled by the supervisor; and processing programs are prepared for execution by the job control program.

The SYSRES extent contains the system libraries. There must be at least one system library (Core Image) and there may be as many as four system libraries: Core Image, Relocatable, Source Statement, and Procedure. The libraries respectively contain "phases," "object modules," "books," and "procedures." Any of these libraries located beyond the SYSRES extent on that volume or on any other volume are private libraries.

The job control language is used to communicate processing requirements to VSE. Each job and its required resources must be defined by job control statements. This enables VSE to perform the basic functions of automatic job-to-job transition, assignment of I/O devices, and loading programs for execution.

Application programs, service programs, and language translators comprise a classification called processing programs; all may be efficiently controlled by VSE.

Your valuable CPU resource is handled efficiently by the facilities of multiprogramming and virtual storage organization. Multiprogramming allows multiple users (multiple programs) to share the CPU at the same time--VSE permits up to twelve partitions to execute programs concurrently--while the virtual storage technique frees the application programmer from constraints previously imposed by the amount of machine storage physically present on the computer.

Take the Mastery Test that follows.

Mastery Test

1. Name the three control programs of VSE.
2. Name the four libraries and what a member of each library is called.
3. What differentiates a system library from a private library?
4. Name the three types of processing programs and define or give an example of each.
5. Job-to-job transition is handled by
   a. the supervisor
   b. the IPL program
   c. job control
   d. none of the above
6. A job to be executed overlays which of these programs in storage?
   a. the supervisor
   b. the IPL program
   c. job control
   d. none of the above

Solution

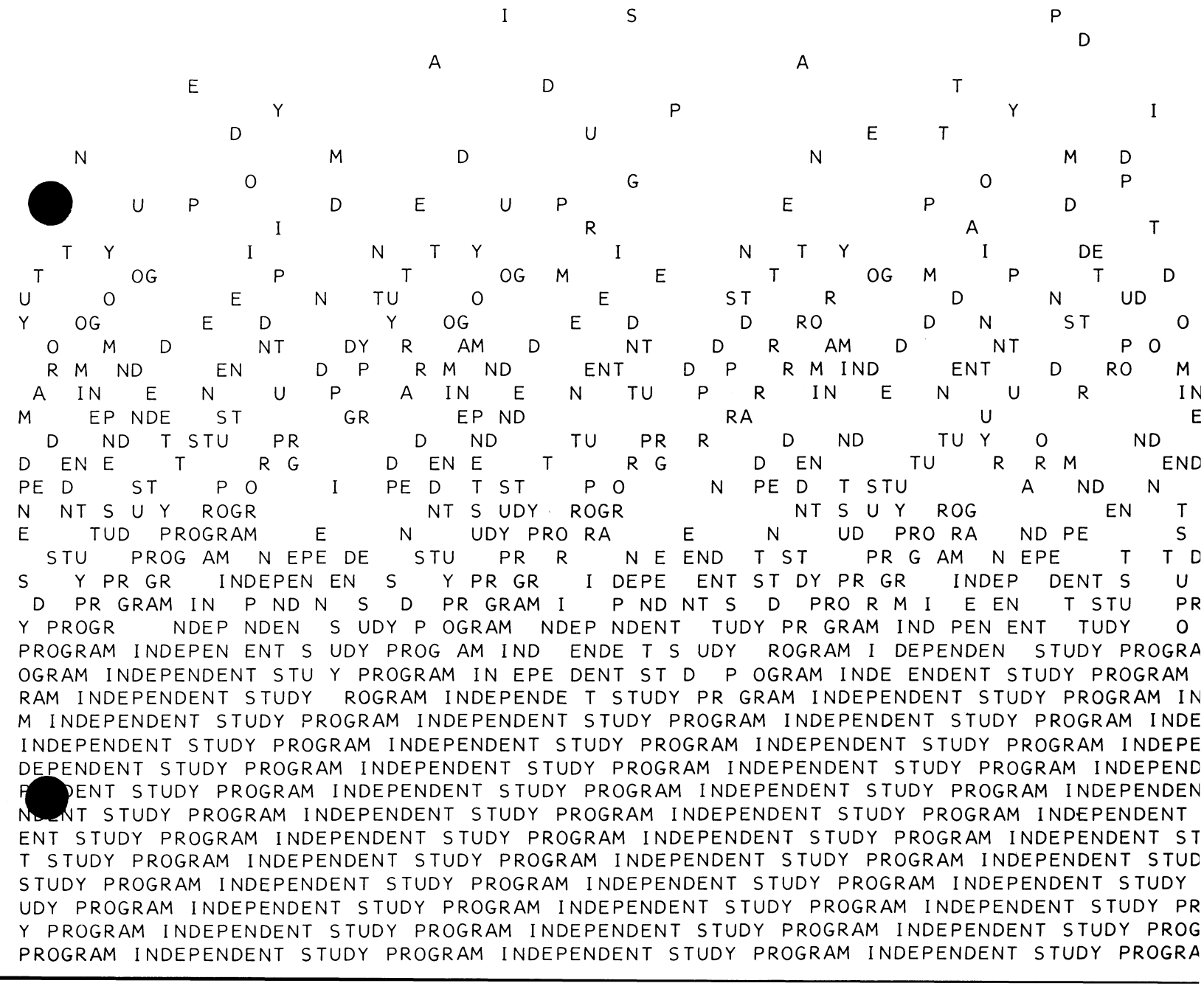
1. Supervisor, Job Control, Initial Program Load
2. Core Image - Phase
   Relocatable - Module
   Source Statement - Book
   Procedure - Procedures
3. A system library resides within the SYSRES extent; a private library resides outside that extent on same volume or any other volume.
4. Application programs - apply to work to be done at the installation

   Service programs - assist in using VSE. Service programs include the librarian programs and the linkage editor program.

   Language translators - convert source language programs to machine language programs. Language translators include the Assembler program, COBOL, PL/I, RPGII, and others.
5. c.
6. c.

Unit

2



Unit 2:

Job Flow and Job Control

Introduction

In order to submit jobs intelligently to the VSE system, you must be familiar with the use and functions of the job control language. This unit explains the basic concepts of job control. You will learn to code six important job control statements, and will learn the relation between jobs and job steps. In addition, you will be introduced to console messages, and will see how cataloged procedures are used.

Objective

Upon completing this Unit, you should be able to:

Assignment 1

- Use the *VSE/Advanced Functions System Control Statements* manual to find and code specific job control statements and commands.
- Define and justify multistep jobs.
- Describe the parts of a basic console message.
- Code the statement required to invoke a cataloged procedure.

Assignment 2

- Define device independence.
- Identify system as opposed to programmer logical units.
- Define the Logical and Physical Unit Block tables (LUBs and PUB) and describe their use.
- Code a basic ASSGN statement.
- Code the LISTIO statement to determine device status.

Materials Required

Study Guide (SR20-7300)

The following VSE reference material:

Introduction to the VSE System (GC33-6108)

VSE/Advanced Functions System Management Guide (SC33-6094)

VSE/Advanced Functions System Control Statements (SC33-6095)

Assignment 1 - The Job Control Language

Statements and Commands

VSE provides you with the capability of using job control statements (JCS) and job control commands (JCC). Both statements and commands are included in the overall concept of the job control language (JCL). Most statements have equivalent command types and vice versa.

The way that VSE knows whether it is dealing with a statement or a command is the presence or absence of slashes in positions one and two. A job control statement will have these slashes, while a job control command will not. Statements are generally used by the programmer and are entered via cards (or in card image format from tape or diskette), while commands are generally used by the operator and are entered via the system console.

Since the slashes are the only way VSE discriminates between statements and commands, you could easily submit a job control command in your job stream instead of the corresponding statement. The JCC format, however, will often cause VSE to do something different than the JCS format. You must be aware of what you are doing when you create your job control statements because of the significance of the slashes.

Four Basic Statements

In the document:

VSE/Advanced Functions System Control Statements

Under the heading:

"Introduction"

Read:

"Control Statement Conventions"

All the VSE job control statements and commands are described in the *VSE/Advanced Functions System Control Statements* manual, where they are listed in alphabetic order. When you have finished the above reading, find and read the material on:

- the JOB card
- the EXEC card (read only the PGM= and REAL parameters at this time)
- the /* card
- the /& card

When you have finished the reading take the Review Exercise that follows, then continue with this assignment. You may use the System Control Statements manual to help you with your answers.

Exercise 2-1

1. Identify any errors on each of the following job control cards.

   a. // JOB TEST.ONE$

   b. // JOB SAMPLE D

   c. // JOB SMITH'S

   d. // JOB CASE2

   e. // EXEC PGM=/TEST

   f. // EXEC PGM=SAMPELS

   g. // EXEC TEST14,REAL=YES

2. A batch of jobs as illustrated in Figure 2.1 is called a job stream. A job stream consists of a collection of single and multiple step jobs to be run in a given partition of your system.

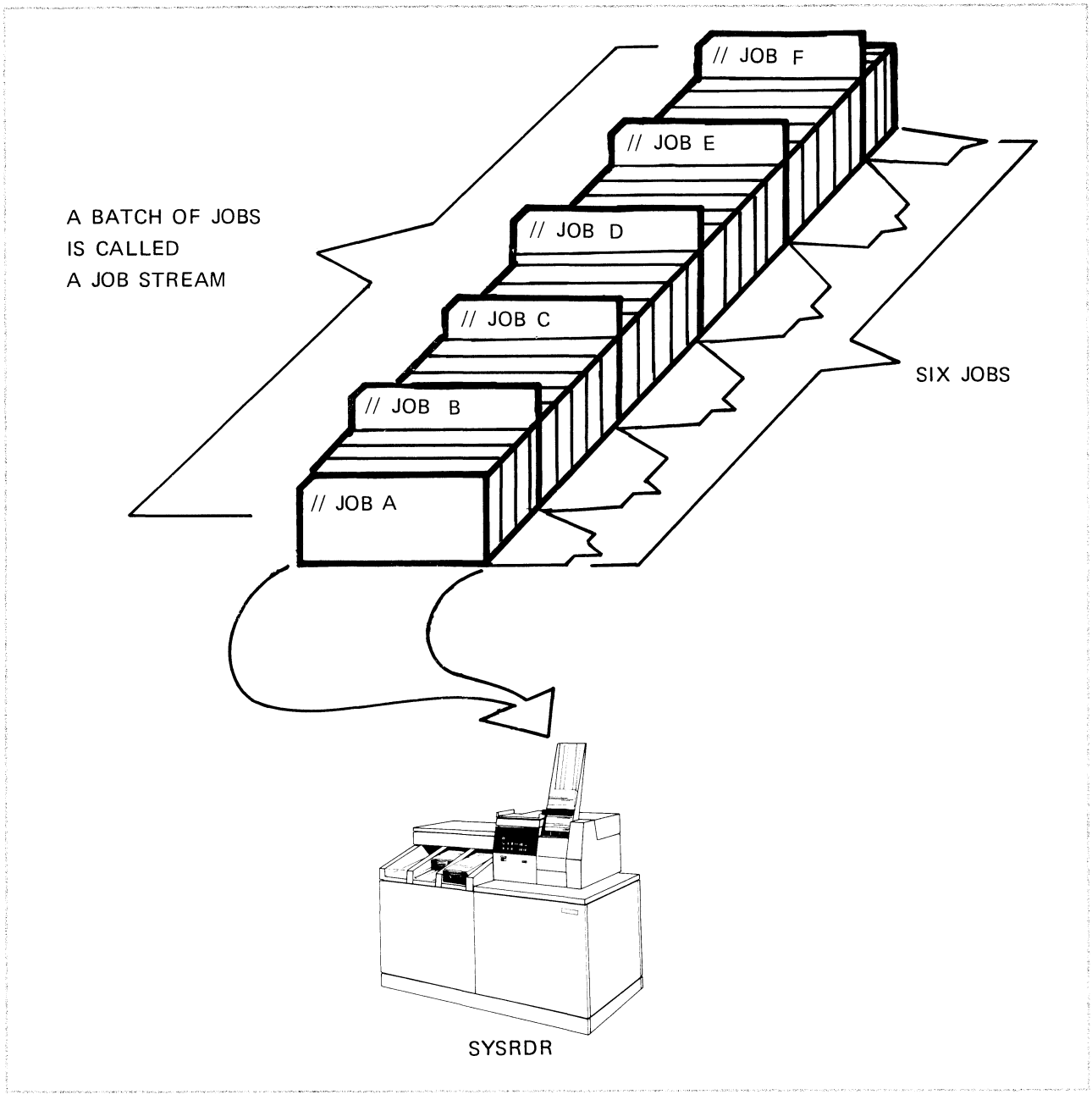


Figure 2.1 - A Job Stream

Construct the job stream necessary to run the following jobs. Each job requires the execution of the program names shown. End of data cards are not required in this problem, but do include end of job cards where needed.

Job Names	Program Names
Case1	Alpha, Beta
Case2	Gamma (Must run REAL)
Case3	Delta, Iota, Epsilon

3. Code the JOB, EXEC, and /& statements for the execution of the six programs described below. Some of these programs are dependent on other programs for their input. Arrange the JCL to take these dependencies into account.

Program Name	Function	Dependent On
A	Calculate Payroll	--
B	Print Checks	A
C	Print Cost Report	A,B
D	Print Inventory Status	E
E	Update Inventory Master File	--
F	Print Work Schedule	--

Ignore specifications of input and output files, and pick your own job names.

Solution

1. a. Job name too long.

   b. D will be taken as accounting information, not as part of the job name.

   c. Apostrophe (special character) not allowed.

   d. No error.

   e. Slash (special character) not allowed in program name.

   f. No error.

   g. REAL parameter incorrectly coded. When properly specified, this parameter indicates that the program to be executed is to reside completely within machine storage during its execution, and is not to be subject to the ordinary rules of program mapping onto the Page Data Set.

   Only a few highly specialized, time-dependent programs must be run in this fashion, and your system programmers or operations department will be able to tell you which ones they are at your installation.

2.
```
//  JOB  CASE1
//  EXEC ALPHA
//  EXEC BETA
/&
//  JOB  CASE2
//  EXEC GAMMA,REAL
/&
//  JOB  CASE3
//  EXEC DELTA
//  EXEC IOTA
//  EXEC EPSILON
/&
```

3.
```
//  JOB  MY1
//  EXEC A
//  EXEC B
//  EXEC C
/&
//  JOB  MY2
//  EXEC E
//  EXEC D
/&
//  JOB  MY3
//  EXEC F
/&
```

If your solution had the three jobs running in a different sequence, that would be perfectly all right. The important thing is that within each job you observed the required dependencies of job steps.

Single vs. Multistep Jobs

In two of the problems you just did, there were cases of multistep jobs. Multistep jobs are used whenever the execution of a particular job step is dependent on the completion of a previous step. Your decision at any point as to whether to put together a single or a multistep job is whether you want the completion or lack of completion of any given step to influence the execution of subsequent steps.

Suppose, for example, that you have two programs, CALC and PRINT, to execute. The CALC routine develops amounts that the PRINT program prints out. This makes execution of PRINT dependent on the successful completion of CALC. See Figure 2.2.

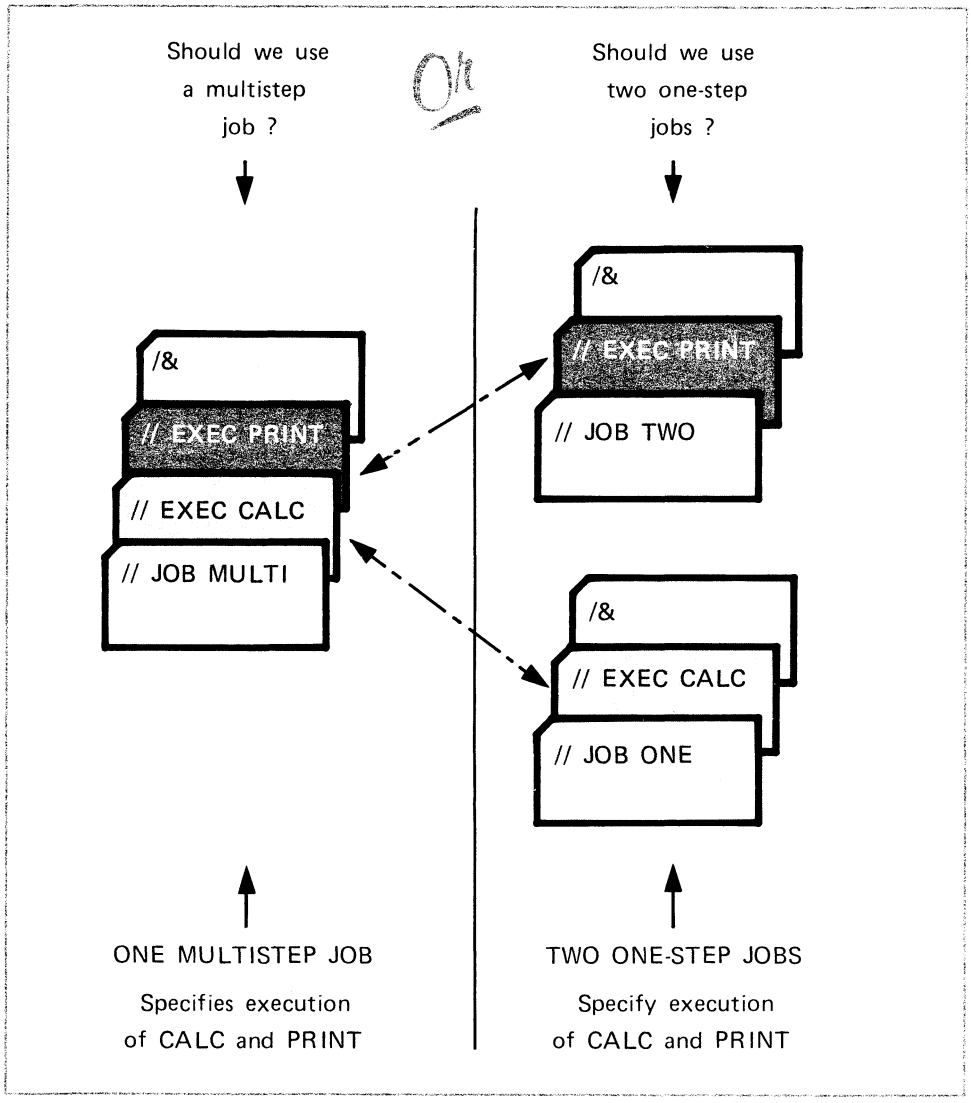


Figure 2.2 - One 2-Step vs. Two 1-Step Jobs

The decision of how to construct the job stream is a function of job dependencies. If you submit these as two separate jobs, you are telling VSE that they are independent of each other. They might well be scheduled to run in different partitions, which would make it difficult for PRINT to get its data from CALC.

Even if they run in the same partition, they will execute independently of each other. This means that if CALC abnormally terminates, PRINT will nevertheless attempt execution. This is a waste of time, as PRINT has nothing to process in this case. Since PRINT is dependent on CALC in this example, you would want to make this a single job with two steps.

When any step of a multistep job is cancelled, all remaining steps will be bypassed by job control until a /& or a new JOB card is found. If CALC abnormally terminates (or is cancelled for any reason), PRINT will not be run. Since PRINT needs CALC's output to run properly, and there is no output in this case, this is exactly what should be done.

Compile, Link-edit, and Execute

The program development sequence of compile, link-edit, and execute is probably the most common example of a multistep job in any installation. Each step in this sequence is completely dependent on the successful completion of the previous step or steps.

If the compile step fails, it would be fruitless to attempt to link-edit and execute, as your program is probably too full of errors to run properly. If the link-edit step fails, then there is no program phase in the Core Image Library to be retrieved at execute time.

The only occasion when VSE will attempt execution of your program is when both the compile and the link-edit steps have successfully completed their functions.

Operator Interaction

Console Messages

The point of batching jobs in a job stream is to permit VSE to do its processing with a minimum of operator intervention. When one of the jobs in the stream completes, VSE automatically looks for another one. This is automatic job-to-job transition.

The operator's job is to respond to system requests, and to mount required volumes or act on system or program messages.

Messages from the component programs of VSE are output to the operator on the system console which has the logical unit name of SYSLOG. Figure 2.3 illustrates a console display of typical system messages.

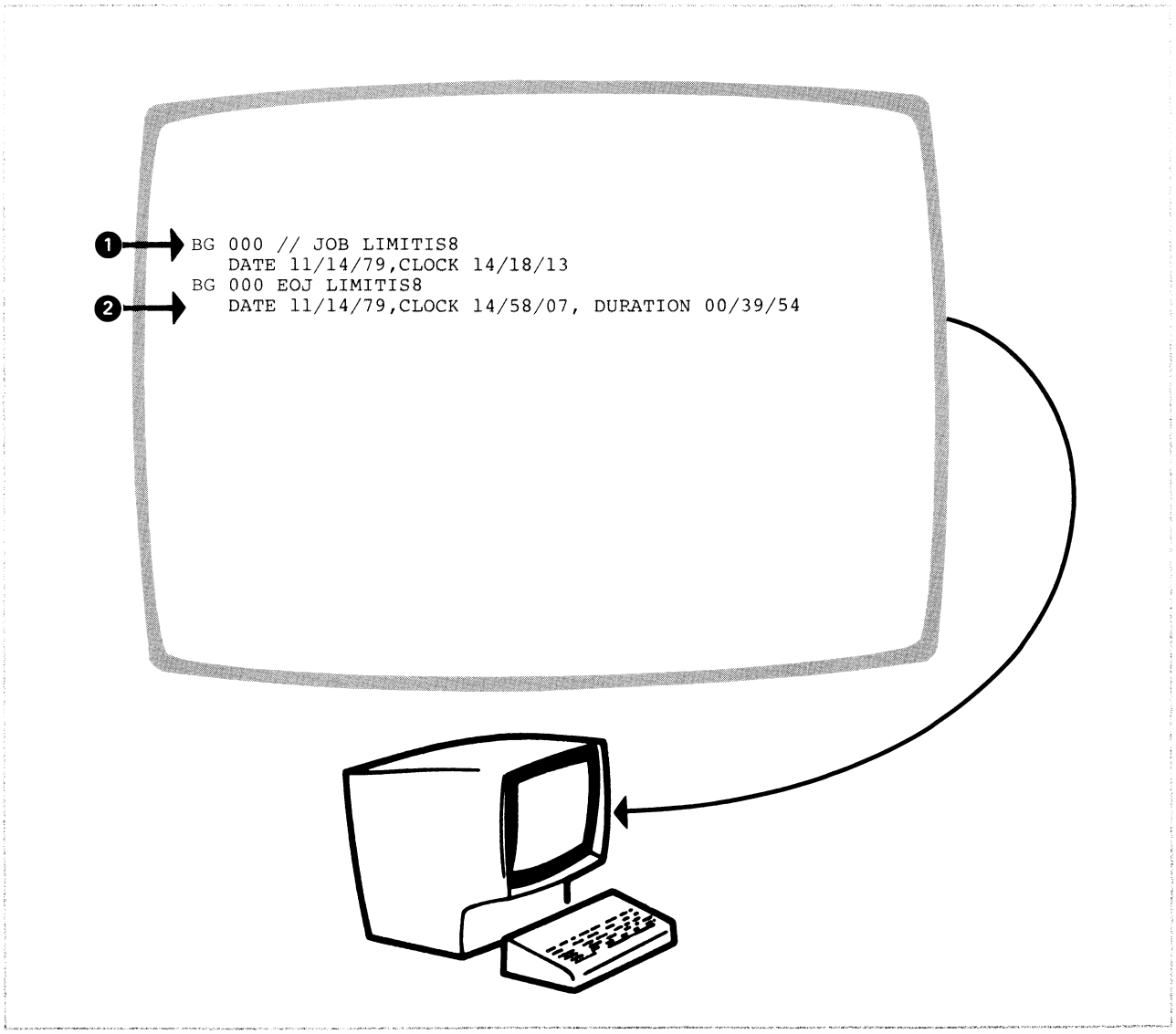


Figure 2.3 - Console Messages

1. BG is the partition identifier, and means this job is running in the background partition. The next three-digit number (000) is the reply-ID. The reply-ID is assigned by the system. To enter a reply to a message, you must key in the reply-ID that the system had assigned to the message. Following the reply-ID is the contents of the job card. Date and clock are the day and time that execution of the job began and are provided by the system.

2. "EOJ LIMITIS8" is displayed when this job completes. Date and clock are the time of completion. Duration is elapsed or "clock time" for the execution: 39 minutes, 54 seconds.

In addition to information messages of this type, VSE will generate warnings concerning errors or actions the system has taken. Your JCL, for example, may contain syntax (format) errors or keypunching errors. When VSE encounters a JCL error, the operator is informed via a console error message. See Figure 2.4.

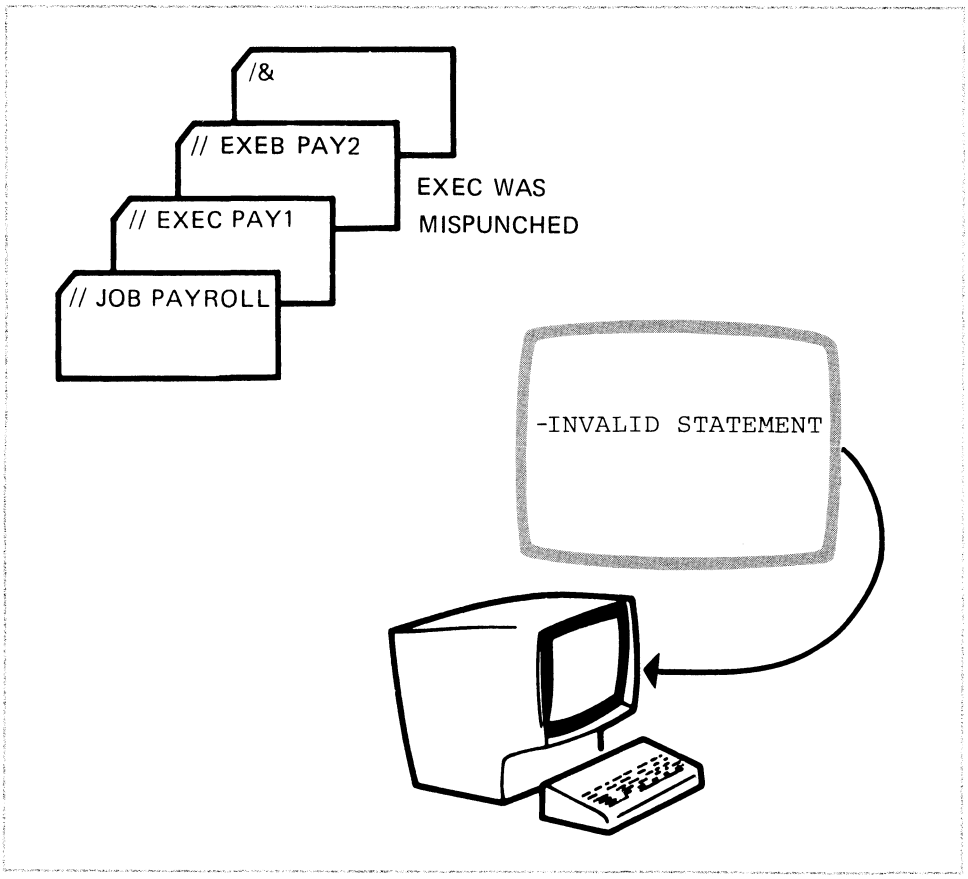


Figure 2.4 - JCL Errors Result in Error Messages

The operator can cancel the job or enter a corrected statement from the console to allow processing to continue. Cancellation of the job is the choice made when the operator does not know the required correction.

Jobs may also be cancelled because of errors detected by VSE or by the processing program during execution. See Figure 2.5. No matter how the job is cancelled, the results are the same: any remaining job steps are bypassed.

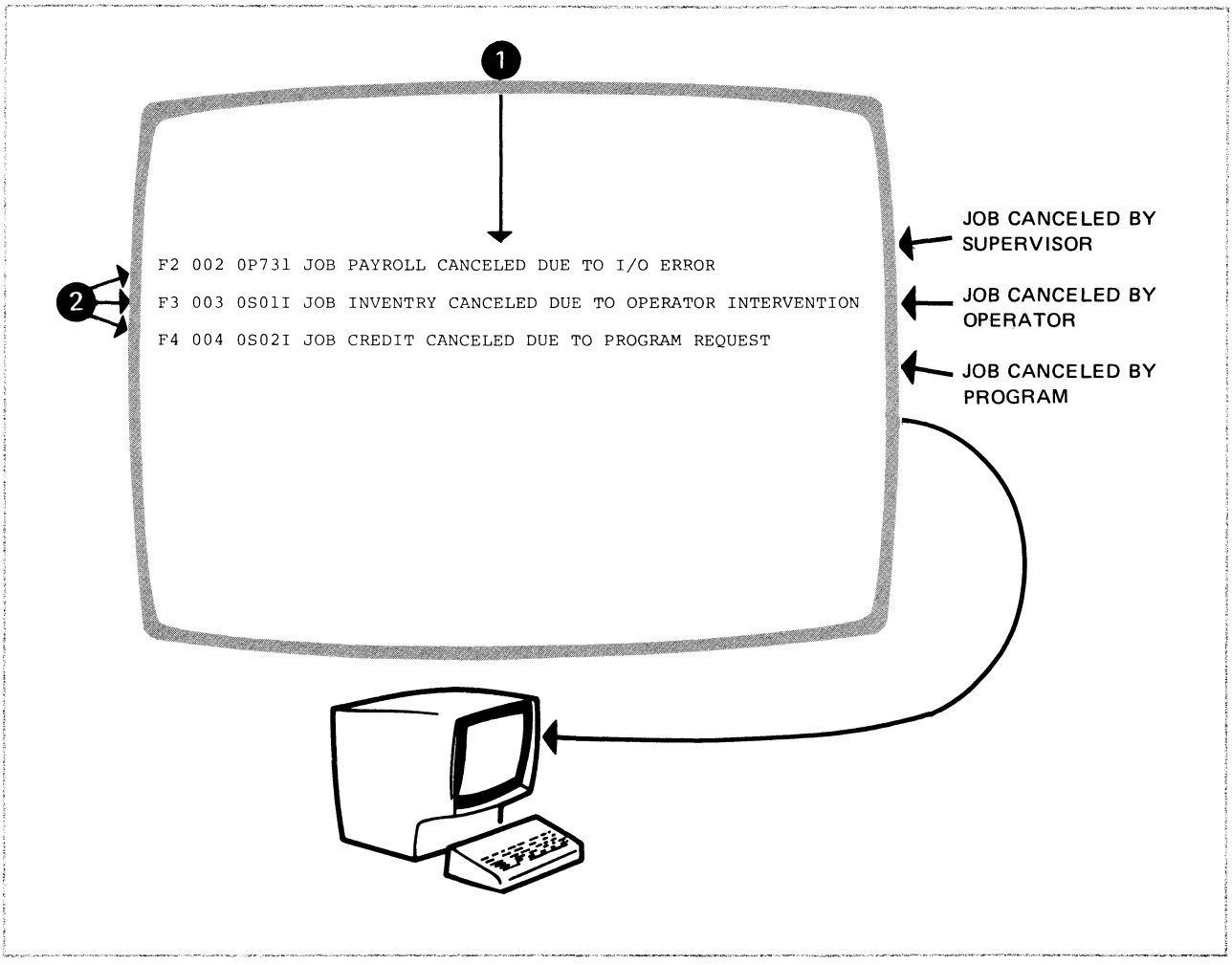


Figure 2.5 - Three Sources of Job Cancellation

1. The reason for cancellation is identified by a message number such as 0P731, 0S011, 0S021. Messages are described in the *VSE/Advanced Functions Messages* manual.

2. F2, F3, and F4 identify the partition in which the jobs were running.

Cataloged Procedures

The scores of jobs an installation has to run each day could require several thousand control statements. These must be stored for retrieval as needed. To eliminate most of the manual retrieval and replacement of control statements, they can be stored in a Procedure Library.

Figure 2.6 shows control statements for a four step job being placed in a Procedure Library. The VSE librarian service program MAINT is used to do this. MAINT will be covered in Unit 7.

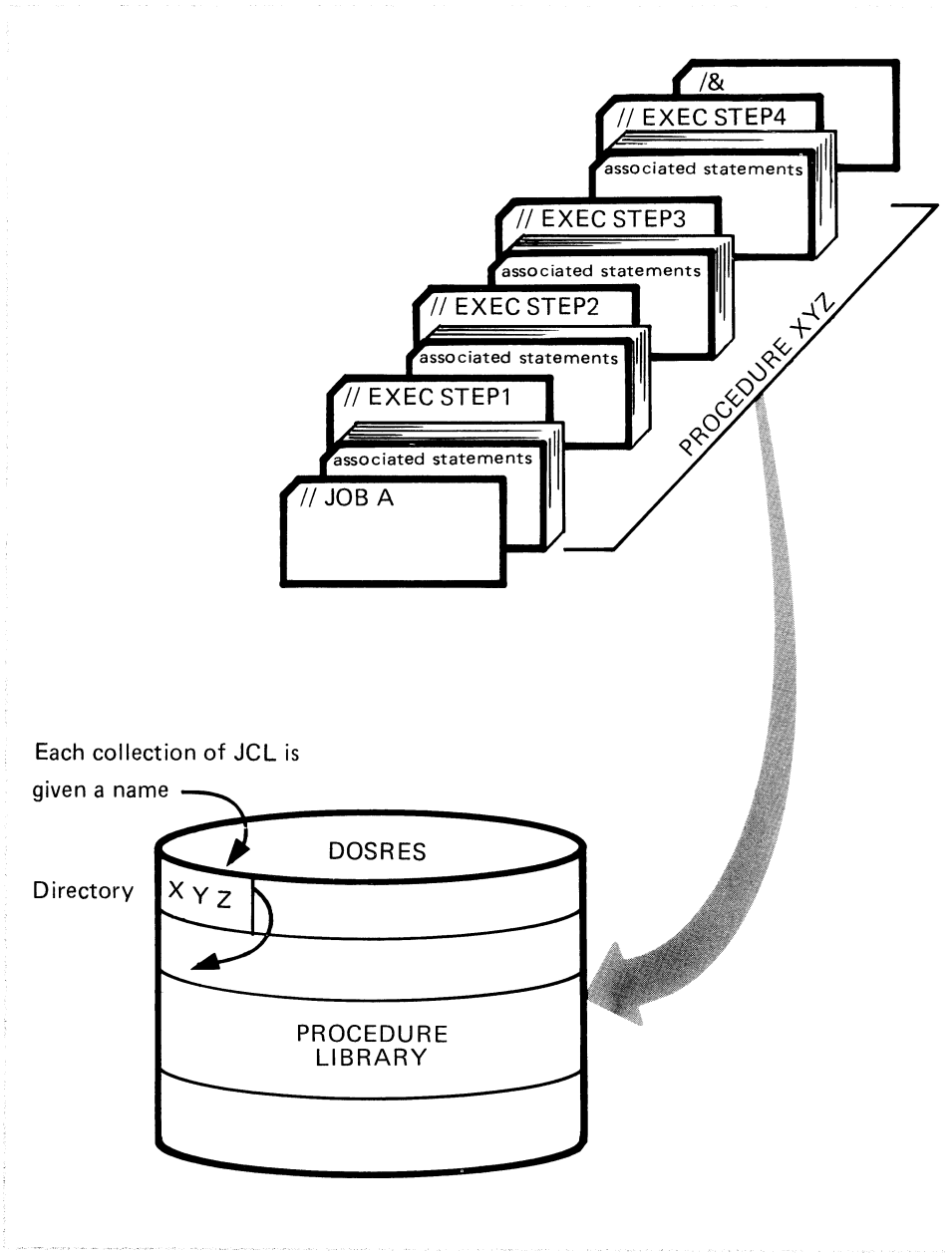


Figure 2.6 - Storing Control Statements

Rather than being manually retrieved from a file drawer every time they are needed, these statements can be obtained from a Procedure Library. Figure 2.7 shows the coding required to invoke, or use, a procedure.

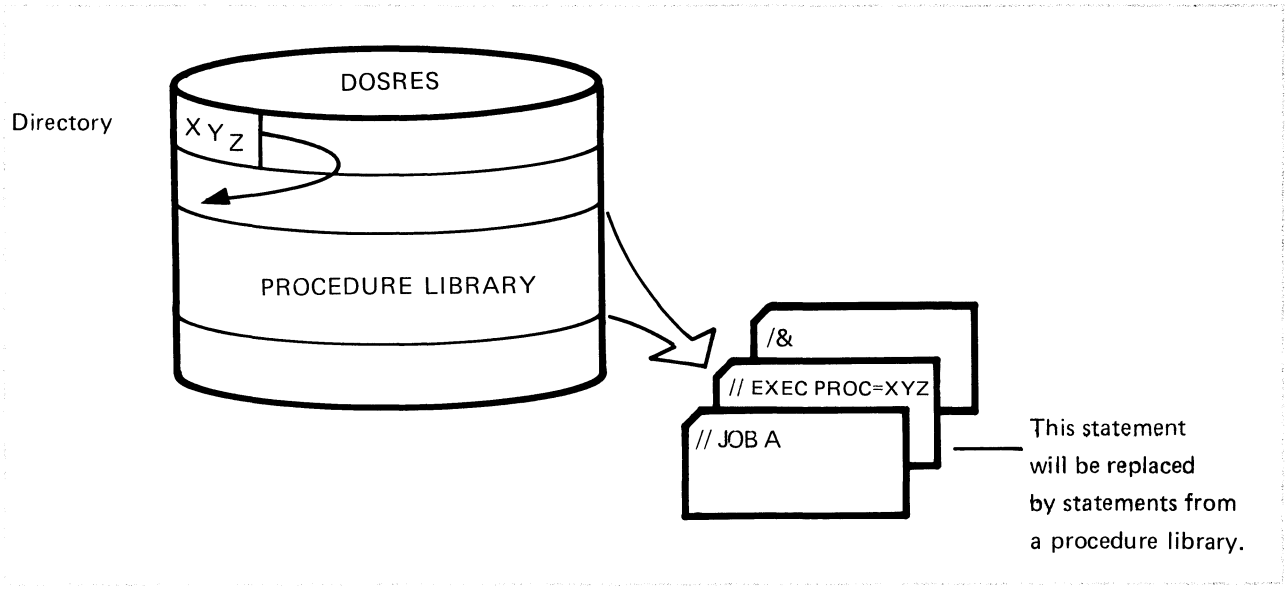


Figure 2.7 - Coding to Invoke a Procedure

The single statement EXEC PROC=XYZ causes the four job steps in the procedure named XYZ to be inserted in the job stream. When the EXEC statement is processed by the job control program, the PROC= parameter causes two things to happen:

1. The EXEC statement containing the PROC= parameter is eliminated from the job stream internally when job control constructs the effective job stream.
2. Substituted in its place in the job stream are the statements contained in the procedure named XYZ.

The effective job stream now includes the four job steps. This job stream is processed by the job control program. When the four steps are encountered they are processed just as though they had been manually submitted.

Assignment 2 - Introducing the ASSGN

Using Symbolic Device Names

Whenever a program needs access to a file on a storage device, VSE must be informed of the physical address of the device involved. Your program need not specify this physical address, but only a symbolic name which refers to a logical rather than physical unit. Before your program is executed, the symbolic name must be associated with an actual device. This is done with information which is either pre-set in the system or entered by the programmer or the operator by using the ASSGN job control statement or the ASSGN job control command.

The ability to reference an I/O device by a symbolic device name instead of a physical address permits a program to be written that is dependent only on device types and not on actual device addresses. The programmer selects a symbolic name from a fixed set of logical names. At execution time this name is associated with an actual physical device through tables stored in the VSE supervisor. Symbolic device names are also referred to as symbolic names, logical units, and logical unit names. These terms are used interchangeably.

Input/Output Configurations

Before we get into the details of handling device assignments, you should be familiar with the way in which the System/370 and 4300 Processors organize the addresses of input-output equipment. Take a look at the configuration chart in Figure 2.8.

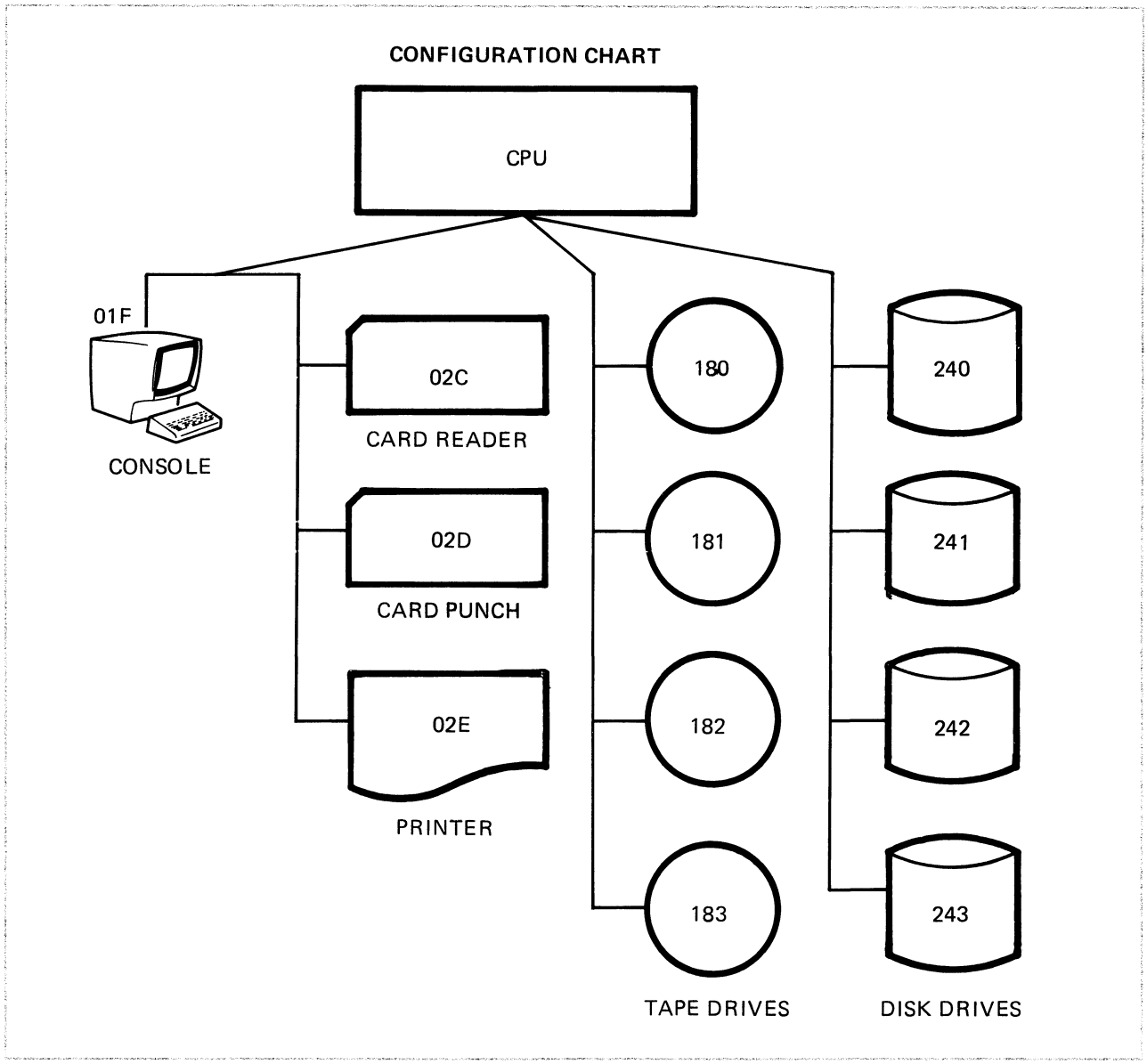


Figure 2.8 - Configuration Chart

This chart is not meant to represent a particular installation, but is merely designed to illustrate a typical set of physical device addresses.

Notice that each device is associated with a three digit hexadecimal number. These physical addresses are the means by which devices are known to the VSE supervisor. Remember, it is the supervisor that handles program requests for I/O transmissions. The supervisor must have a method for uniquely identifying every piece of I/O equipment attached to the CPU. The device addressing method that has been designed into the processor architecture is one that represents the address of any device as a three digit hexadecimal number. The digit positions represent channel, control unit, and device. See Figure 2.9.

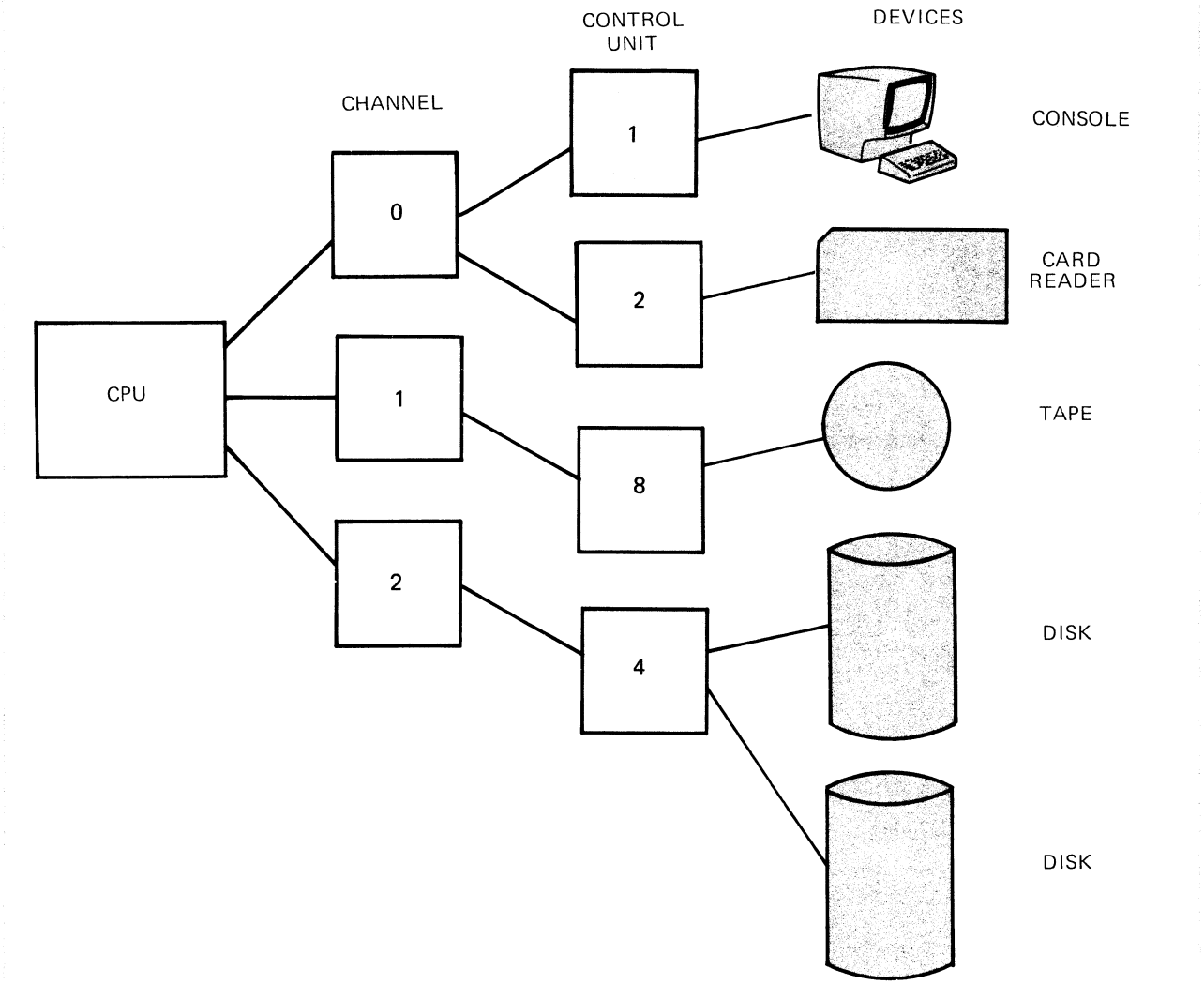


Figure 2.9 - Device Addresses

This threefold addressing system allows the user to control a great number of devices in a simple fashion. Each three digit address represents a single device and a data path to that device. For example, address 02C corresponds in this case to a card reader, and specifies a path of data flow through channel 0 to control unit 2 to device C. The disk devices 240 and 241 are both on the same channel and control unit, but have different device addresses.

The supervisor must know the three digit hexadecimal address of the unit you wish to access so it can construct the proper sequences of channel commands to handle the I/O operation. See Figure 2.10.

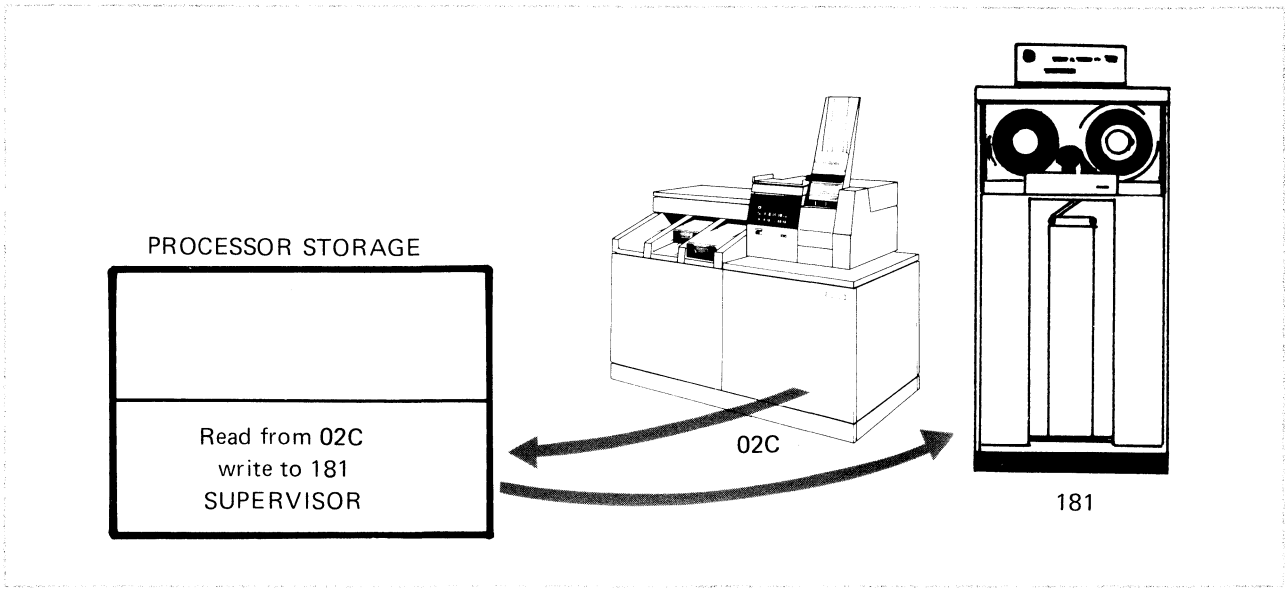


Figure 2.10 - Data Transmission

Then why not just code the device addresses right in your program? In other words, if you need to access the disk on disk drive 240, why not simply code the hexadecimal address value of 240 in your program? If you're not sure of the answer to this question, take a few moments to think about it before going on.

Device Independence

The point is that you want to access a particular disk, and don't care what disk drive the disk is on. This is the concept of device independence, a basic architectural consideration of modern processors. If you compile your program with the specific hardware address of 240 embedded in it, what happens when execution is attempted when device 240 is not available? You either recompile your program to point to the address of a device that is available, or else wait until 240 comes back up.

Neither of these is a satisfactory choice, so VSE allows you to defer choosing devices the program is to use until just before that program executes. The means by which this is accomplished is through symbolic (logical) unit names.

I/O Assignment

The ASSGN job control statement or command is used to make associations between symbolic unit names in your programs and actual physical device addresses. ASSGN is the principle means by which you relate a program to the location of the data files it is to process.

Permanent Assignments

Permanent assignments are usually made at IPL time. A permanent assignment permits a particular symbolic name to be connected to a given device when such an association is to be commonly used on a system. For example, SYSRDR, the device on which the job control program looks for its input, is generally assigned to a card reader, and that assignment is rarely changed. Permanent assignments are in effect at all times unless overridden by ASSGN job control statements or commands.

Types of Logical Units

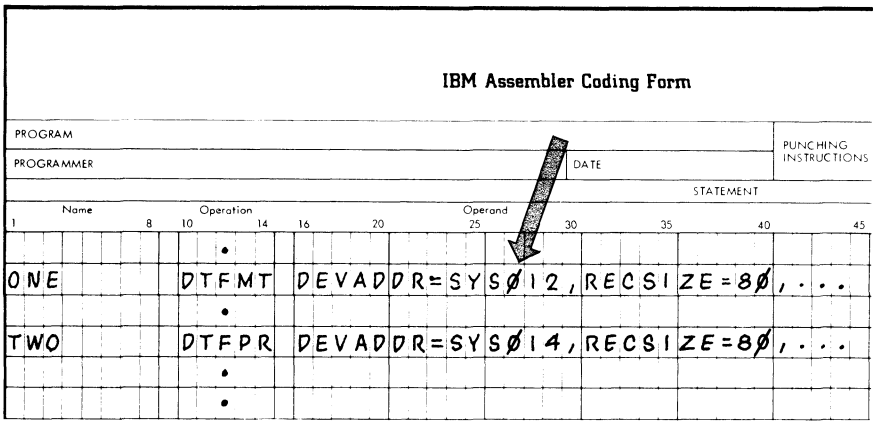
System logical units have names that consist of six alphabetic characters, such as SYSRDR, SYSPCH or SYSLST. They are used by the component programs of VSE and are listed for you along with their uses in the *Introduction to the VSE System* manual in the Illustration "Logical unit names recognized by VSE/Advanced Functions."

Programmer logical Units are for the programmer's use and have names of the form SYSnnn, where nnn ranges in value from 000 to 254. Be careful in choosing these names, as certain of them are also used by component programs of VSE. These include SYS000 through SYS004 and sometimes SYS005. The linkage editor uses SYS001 and the Assembler uses SYS001, SYS002, and SYS003. Some IBM language translators and utilities also use SYS004 and SYS005.

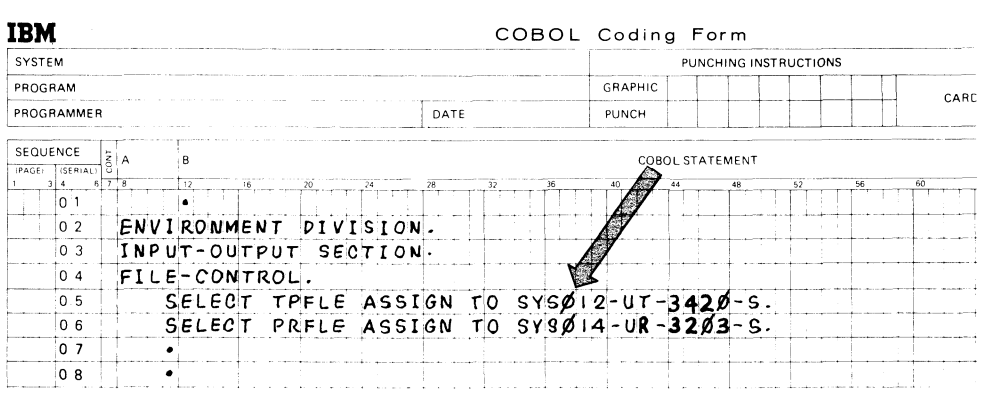
Using the ASSGN

In order to use the ASSGN you must first select programmer logical units in your program. Figure 2.11 shows how these units are designated in a variety of programming languages.

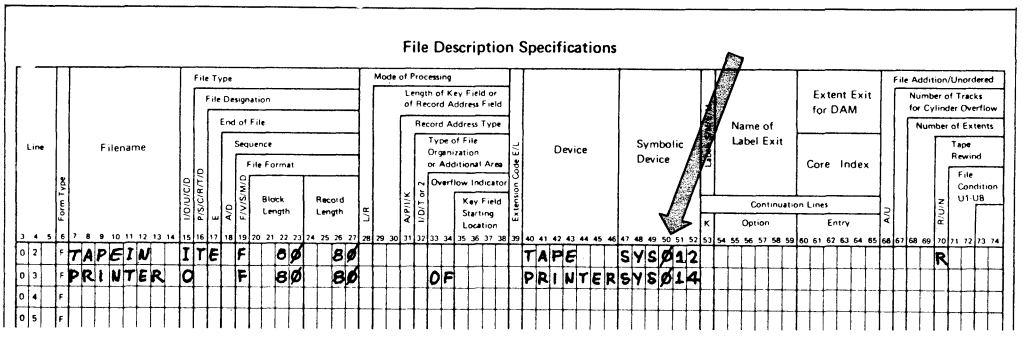
ASSEMBLER LANGUAGE



COBOL



RPG



PL/I

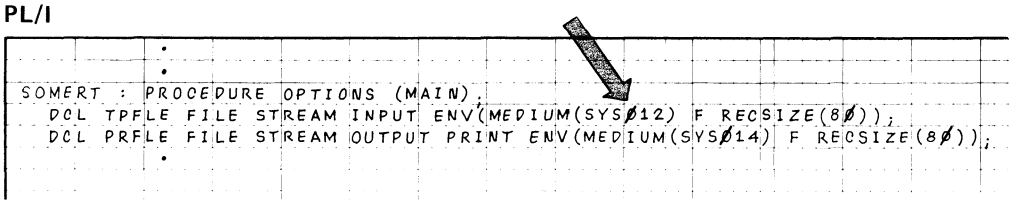


Figure 2.11 - Specifying I/O Devices

When a job is submitted for execution, connections between symbolic unit names used in the program and actual devices (supplied by ASSGN's) are completed. Note that the ASSGN format is

```
// ASSGN  SYSnnn,cuu
```

where the 'cuu' positions represent the channel, control unit, and device portions of the three digit hexadecimal address. See Figure 2.12.

```
// JOB    ANYNAME
// ASSGN SYS012,181
// ASSGN SYS014,02E
// EXEC   MYPROG
/&
```

Figure 2.12 - The Final Links

How it Works

The job control program processes ASSGN statements by making connections between tables in the supervisor known as the LUB and PUB tables.

The LUB Tables

LUB stands for Logical Unit Block, and the supervisor contains one of these tables for each partition in your system. The LUB Table for a partition contains an entry for each of the symbolic names that can be used in programs that run in that partition. These names include the system logical units (SYSRDR, SYSPCH, SYSLST, etc.), and some number of programmer logical units in the range SYS000 through SYSmax. The range SYS000 to SYSmax is continuous and has no gaps. The application programmer is usually not concerned with how the value of SYSmax is determined. Your operations department will be able to supply you with the proper value.

The PUB Table

PUB stands for Physical Unit Block and unlike the LUB Tables, there is only one PUB Table to a system. It contains an entry for each physical device attached to the system. As ASSGN statements are processed, the job control program causes the appropriate entries in the LUB Table associated with the partition in which job control is running to point to the designated PUB entry. See Figure 2.13.

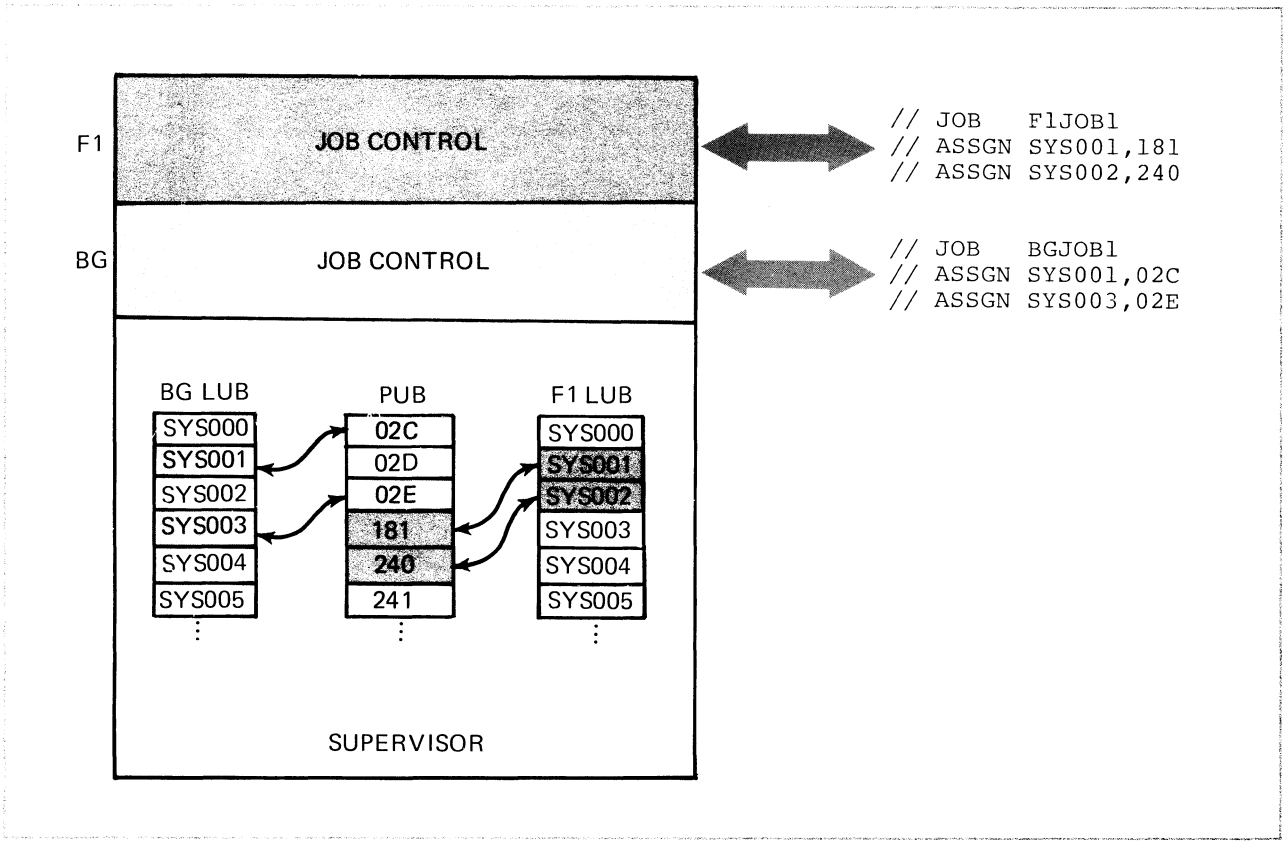


Figure 2.13 - ASSGN Processing

In this two partition example, job control is active in both BG and F1 at the same time. This is possible because the supervisor has loaded a copy of the job control program from the Core Image Library into both partitions.

Notice that since each partition has its own LUB table in the supervisor, it is no problem for both BG and Fl to be referencing any of the system or logical program names simultaneously.

Here, BG and F1 are both using SYS001, but this is a different SYS001 for each partition. There is, however, only one each of the physical devices in existence on the system and most of these are not shareable by more than one partition at a time. See Figure 2.14.

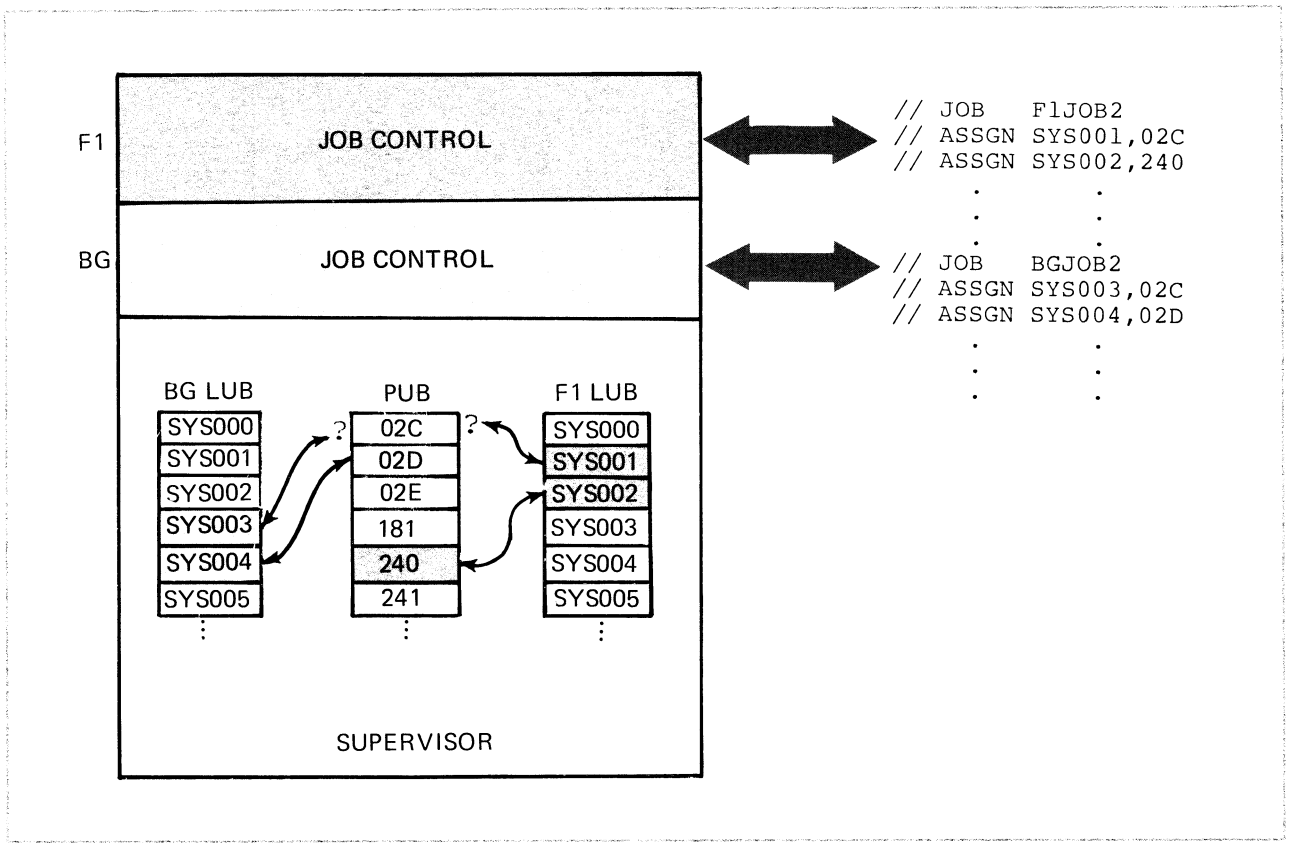


Figure 2.14 - Device Contention

In this case, both F1 and BG are contending for use of device 02C at the same time. The device can only service one partition, so one job will be held up while the other is allowed to execute. Only direct access storage devices (DASD) are shareable among your system's partitions.

The LISTIO Statement

Information from the LUB and PUB Tables can be obtained by using the LISTIO job control statement. The PROG parameter requests a listing of the physical units assigned to *all* programmer logical units of the partition in which the LISTPROG job is run. Figure 2.15 shows a portion of the output generated by LISTIO. The numbered headings are explained below the figure.

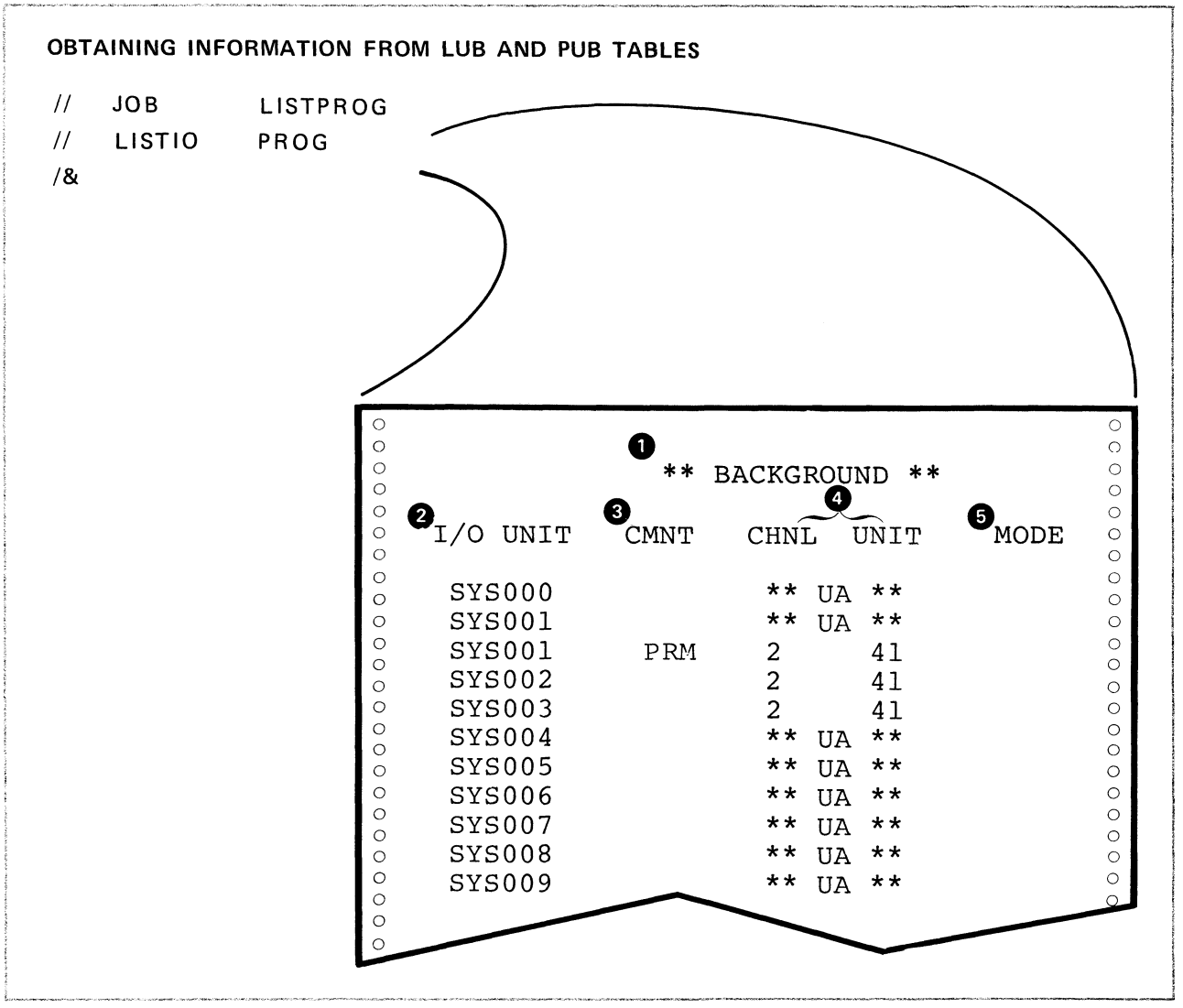


Figure 2.15 - LISTIO Output

1. Refers to the partition for which this information applies. Here you see the LUB's belonging to the BG partition and the PUB entries they are assigned to, if any.

2. Logical Unit Names. These can be system or programmer logical units.

3. Comments. PRM means that SYS001 was permanently assigned as 241 but is temporarily superseded by the assignment shown one line higher, that is, SYS001 is temporarily unassigned.

4. The three digit hexadecimal address of a physical device. UA means unassigned: there is no physical device assigned to the LUB. Any attempt by a program to reference a device which has been assigned as UA will cause an abnormal termination of that program.

   The UA entries can be eliminated from the listing by coding LISTIO with the ASSGN parameter. This will cause only the currently assigned devices in the partition to be printed.

5. The mode column indicates conversion and translation features for tape drives.

The *VSE/Advanced Functions System Control Statements* manual contains complete specifications on coding the LISTIO statement.

Figure 2.16 shows the program development process and illustrates how symbolic device names are embedded in a program from initial coding to final execution.

When the program is to be executed, these logical names are associated with real physical devices by means of permanent assignments generated with the VSE system or through ASSGN cards submitted with the job. The VSE supervisor can carry out all the program's I/O requests correctly by using the links established in the LUB Table for the partition in which the program is executing.

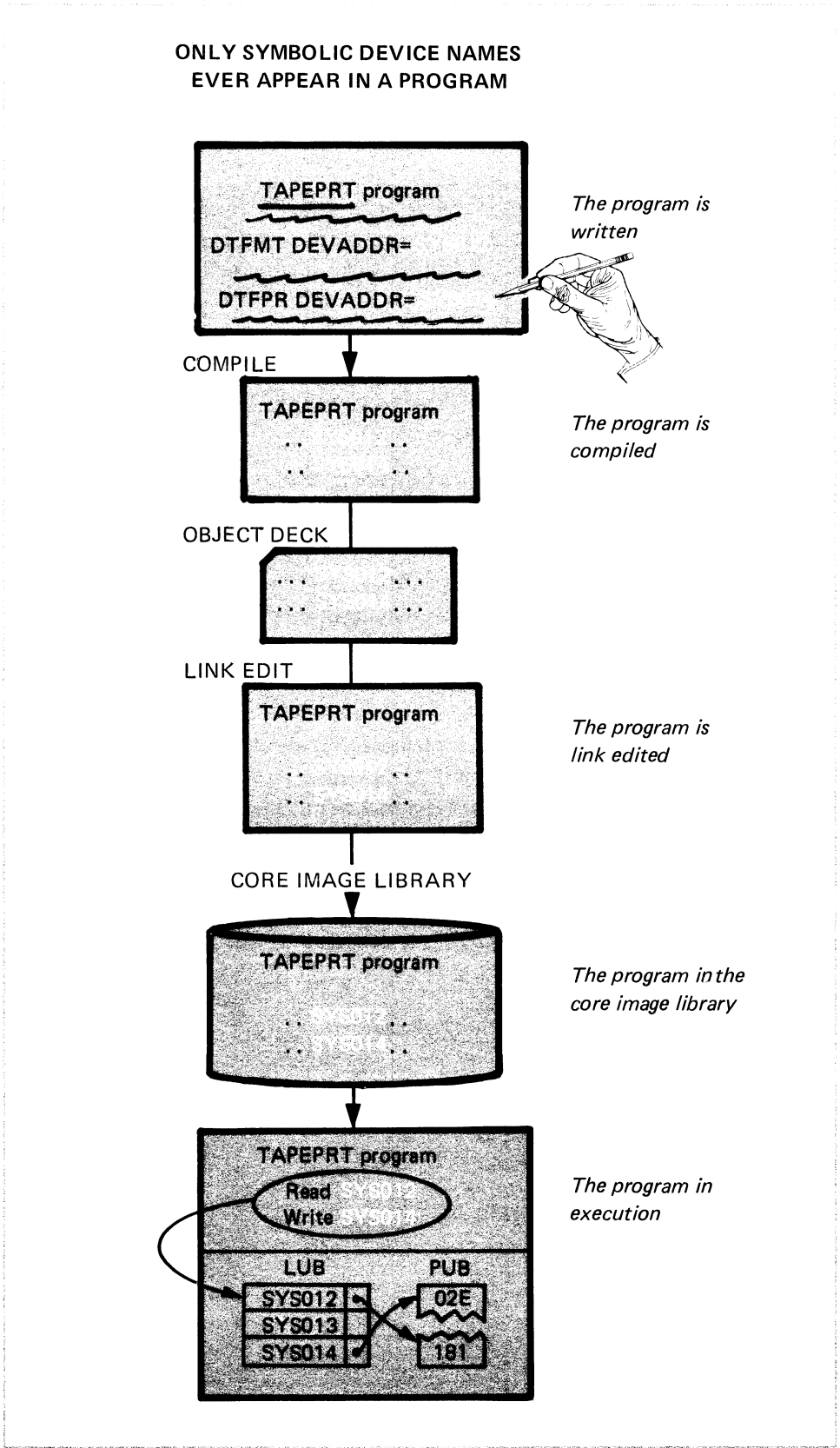


Figure 2.16 - Using Symbolic Device Names

Unit Summary

The System Control Statements manual is your basic guide to VSE job control. Never "guess" at how a job control statement or command is coded, but rather look it up in the manual to be sure you have the correct format.

Whether a job stream should consist of multiple jobs or a single job with a multiple number of steps is a function of the degree of dependence between the programs involved. The sequence of compile, link-edit, and execute is a common example of a multiple step job. Each step in this sequence (except the first) is dependent on the successful completion of the previous step or steps.

During the execution of your job stream, the VSE operator is kept informed of your program's requirements through console messages. The structure of a VSE message includes a partition identifier, a reply-ID, a message code, and the message text: these elements allow the operator to respond properly to system requests.

A Procedure Library holds commonly used sets of job control statements, and the EXEC statement is used to invoke a cataloged procedure. Later, we will cover more advanced uses of a Procedure Library, such as how to temporarily modify the contents of an invoked procedure.

The concept of device independence is an important one in any modern operating system. It allows programmers to reference logical names within their programs that only become associated with actual physical devices at execution time. The user is not "locked in" to any particular fixed addresses, and is usually able to run whatever programs must be run regardless of the physical devices available at any given time.

Device independence is implemented under VSE by means of tables for the logical and physical representation of I/O devices. The LUB tables (one per partition) are used to associate system and logical unit names to the actual physical devices attached to your VSE system. The physical devices are listed in the PUB table (one per VSE system) and the LUB/PUB association is made via the ASSGN statement or command. ASSGN will be discussed in more detail in the next Unit.

Finally, the LISTIO statement is available for investigating the status of device assignments at your request.

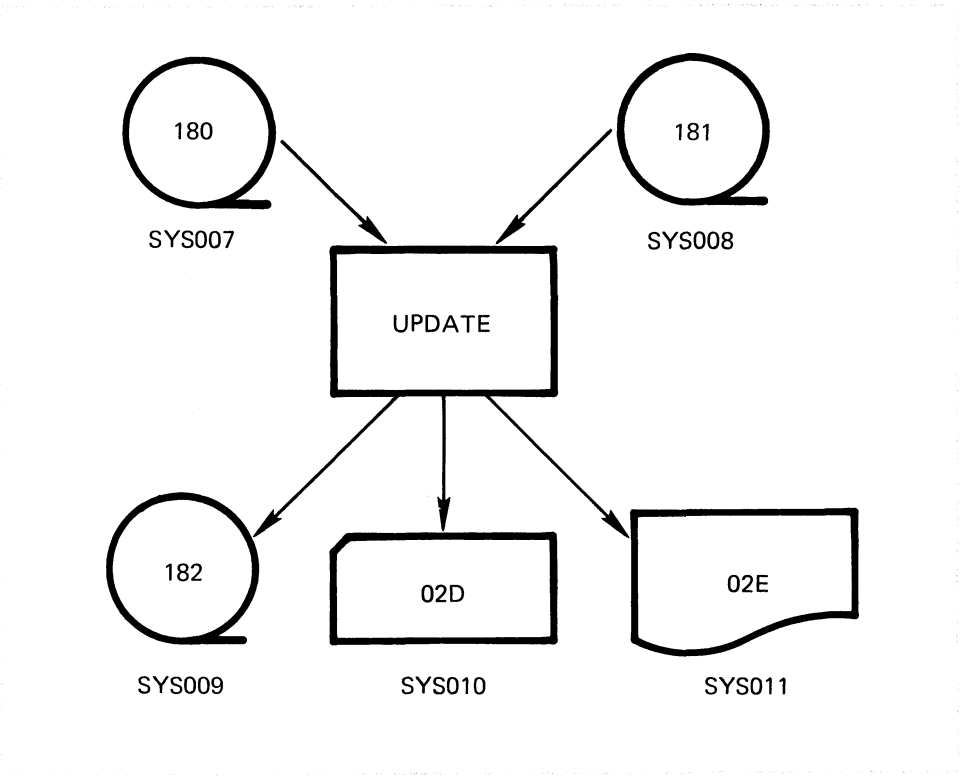
Take the Mastery Test that follows.

Mastery Test

1. The sequence compile, link-edit, and execute is a common example of a _____.
   a. multiple step job
   b. multiple job step
   c. independent job
   d. partition dependent procedure
2. Jobs may be cancelled because of errors in _____.
   a. your job control statements
   b. your program's logic
   c. your instructions to the operator
   d. all of these
3. The statement EXEC PROC=XYZ is invoking _____.
   a. a member of a Procedure Library named XYZ
   b. a member of a Core Image Library named XYZ
   c. a member of a Procedure Library named PROC
   d. a member of a Core Image Library named PROC
4. Cataloged procedures are useful because they allow you to _____.
   a. reduce operator card handling
   b. store frequently used job control on disk
   c. save file cabinet space
   d. all of these
5. Every piece of I/O equipment attached to a System/370 or 4300 has a three part address consisting of _____, _____, _____.
   a. channel, control unit, device
   b. control unit, channel, device
   c. control unit, device, channel
   d. device, channel, control unit
6. Device independence allows the user to _____.
   a. ignore program device requirements
   b. defer choosing program devices until execution
   c. embed actual device addresses within programs
   d. recompile programs at will

7. A device assignment is one that is _____.
   a. made with a job control statement
   b. made by the operator
   c. made to be temporary or permanent
   d. all of the above
8. There is a LUB table for each _____.
   a. device
   b. channel
   c. partition
   d. system
9. There is one PUB entry for each _____.
   a. device
   b. channel
   c. partition
   d. system
10. The device on which the job control program looks for its data is known as _____.
   a. SYSIPT
   b. SYSRDR
   c. SYS000
   d. SYSPCH

11. Code the JCL required to execute the program named UPDATE. Choose your own jobname and do not be concerned with label information.



Questions 12 through 15 are based on the following reading assignment:

In the document:

VSE/Advanced Functions System Management Guide

Under the heading:

"Relating Files to Your Program"

Read:

Through and including the sub-heading "Types of Device Assignments." (Don't be concerned about references to label or extent information. This material will be covered in later Units.)

12. The name SYSIN can be used to combine the functions of _____ and _____.
    a. SYSRDR and SYSPCH
    b. SYSRDR and SYSOUT
    c. SYSRDR and SYSIPT
    d. SYSIPT and SYSRES
13. Which one of the following assignments is invalid? Why?
    a. // ASSGN SYSRDR,140
    b. ASSGN SYSPCH,140
    c. // ASSGN SYSRES,140
    d. ASSGN SYS000,140
14. A temporary device assignment can be reset by a _____.
    a. /& statement
    b. // JOB statement
    c. RESET statement or command
    d. any of these
15. A permanent device assignment can be changed by a _____.
    a. /& statement
    b. // JOB statement
    c. RESET statement or command
    d. none of these

Solution

1. a
2. d
3. a
4. d
5. a
6. b
7. d
8. c
9. a
10. b
11. Two solutions are provided:

```
//  JOB     NAMEX
//  ASSGN   SYS007,180
//  ASSGN   SYS008,181
//  ASSGN   SYS009,182
//  ASSGN   SYS010,02D
//  ASSGN   SYS011,02E
//  EXEC    UPDATE
/&
```

```
//  JOB     NAMEY
//  ASSGN   SYS011,02E
//  ASSGN   SYS010,02D
//  ASSGN   SYS009,182
//  ASSGN   SYS008,181
//  ASSGN   SYS007,180
//  EXEC    UPDATE
/&
```

The point is that the sequence of the ASSGN statements makes no difference.

12. c
13. c is invalid because SYSRES is "assigned" by the IPL procedure and cannot be referenced as Operand 1 of an ASSGN statement.
14. d
15. d

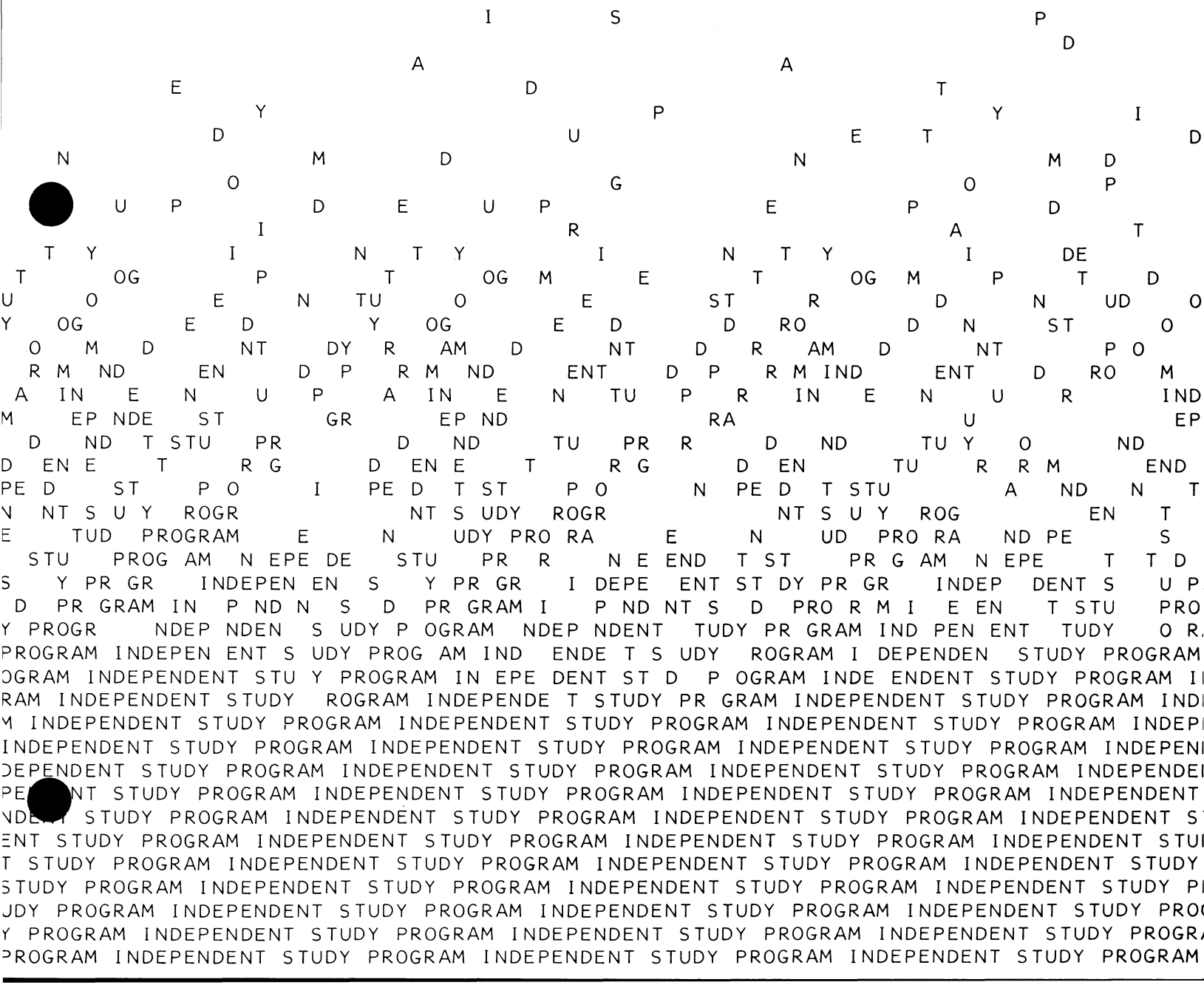
You should have completed this test with no more than five incorrect answers. If you had more than five wrong answers, it is suggested that you repeat your study of this Unit.

Computer Exercise

If you successfully completed the Mastery Test, do Computer Exercise 1 in Appendix A. When you have submitted this job for execution, continue with Unit 3.

Unit

3



Unit 3:

Controlling Program Execution

Introduction

Up to this point, much of what you have learned has been conceptual in nature, that is, you know a lot about how VSE works but not too much about how to make it work for you. In this Unit you will be taught a variety of the basic skills you need to work profitably within the VSE environment.

Objective

Upon completing this Unit, you should be able to:

Assignment 1

- Use the OPTION job control statement to control language translator operations.
- Prepare the job control statements necessary to assemble or compile a source program.

Assignment 2

- Code both temporary and permanent ASSGN statements for a variety of file conditions.
- Use generic device assignments.
- Prepare a basic set of job control statements to compile (or assemble), link-edit, and execute a program.

Assignment 3

- Use the *Messages* manual to respond to system messages.

Materials Required

Study Guide (SR20-7300)

The following VSE reference material:

VSE/Advanced Functions System Control Statements (SC33-6095)

VSE/Advanced Functions Messages (SC33-6098)

VSE/Advanced Functions System Management Guide (SC33-6094)

Assignment 1 - Using the Language Translators

The compilers, or language translators, are programs that transform source code into machine language object modules. Language translators include Assembler, COBOL, PL/I, RPGII, and FORTRAN. This Assignment covers what information the language translators need to do their work, how to supply that information, and how to control their operations.

Language Translator Requirements

Language translators require certain information about the facilities they are using, and it is up to you to provide that information. Figure 3.1 shows the input, output, and work files required for a compilation.

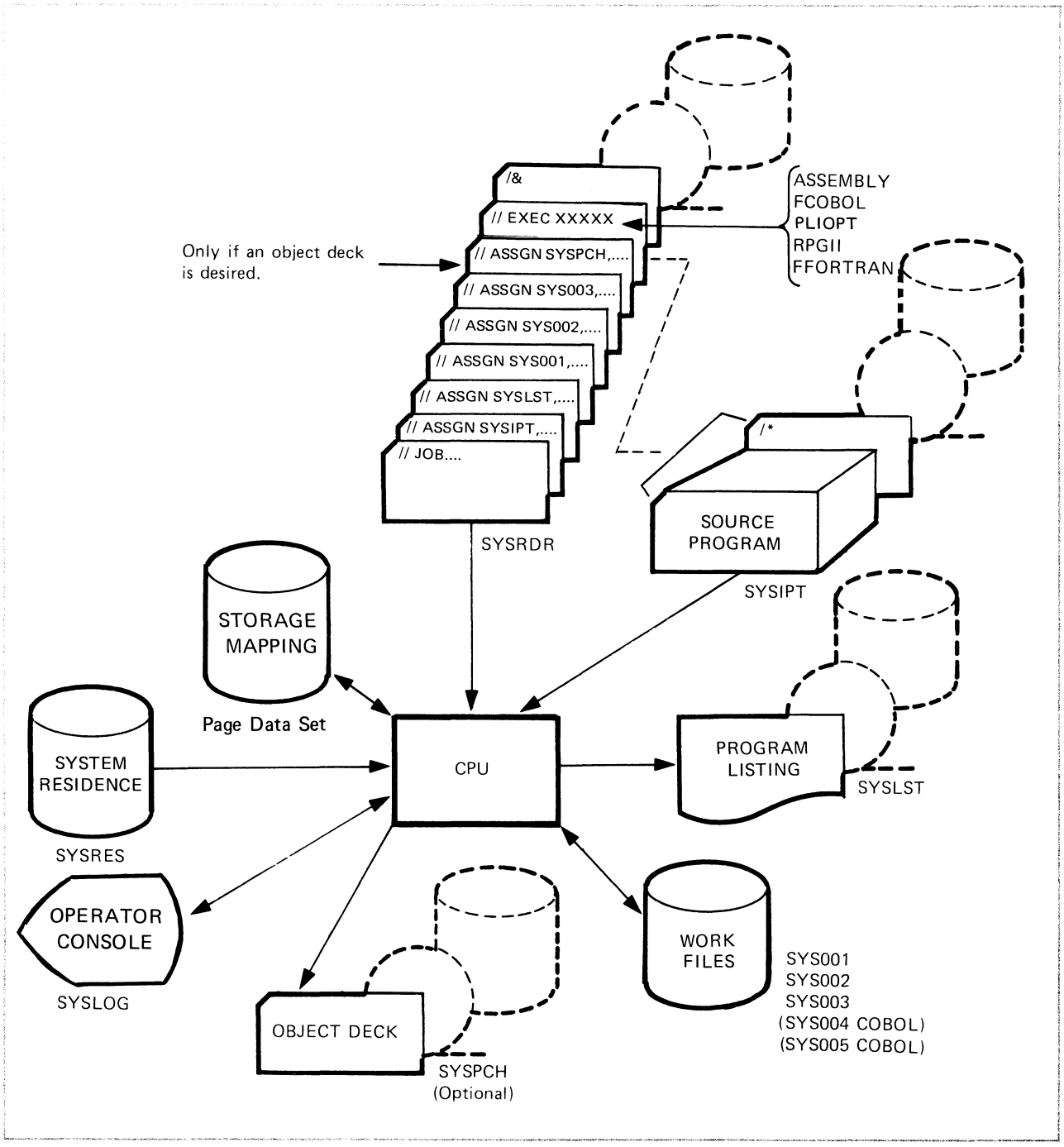


Figure 3.1 - Device Assignments Required for a Compilation

There are several important points to notice about this illustration.

- Many of the required device assignments are the same for Assembler, COBOL, PL/I, RPG, and FORTRAN. These common assignments perform the same function regardless of which compiler is active, e.g., SYSIPT is *always* the unit on which source language statements are entered, SYSLST *always* receives the printed output, and so on.
- SYSIPT and SYSRDR can have the same physical device assigned to them or they can be assigned to separate devices.

- SYS001 through SYS003 (and through SYS005 in the case of COBOL) are associated with work files.
- The SYSPCH assignment is optional. It is needed only if you desire an object deck.
- The ASSGN cards shown in the figure would not be needed if all the device assignments had been made permanent. Remember, permanent assignments are retained by VSE across jobs and across IPLs.

Most installations make these assignments permanent, so the number of job control statements that must be submitted are kept to a minimum.

Figure 3.2 illustrates the amount of JCL you would have to submit for a COBOL compile operation were it not for the ability of VSE to make required information standard in the system.

```
* COMPILE A SOURCE PROGRAM  PRODUCE A
* LISTING AND AN OBJECT MODULE
*
* ASSUMPTIONS: ONLY SYSRES, SYSRDR, and
*              SYSLOG ARE STANDARD ASSIGNMENTS.
*              NO LABEL INFORMATION IS STORED.
*
// JOB    COMPILE
// OPTION LIST,DECK
// ASSGN  SYSIPT,SYSRDR   Assign SYSIPT
// ASSGN  SYSLST,02E      Assign SYSLST
// ASSGN  SYS001,292      Assign SYS001 Workfile
// ASSGN  SYS002,292      Assign SYS002 Workfile
// ASSGN  SYS003,292      Assign SYS003 Workfile
// ASSGN  SYS004,292      Assign SYS004 Workfile
// ASSGN  SYSPCH,02D      Assign SYSPCH

// DLBL   ....          label information for:
// EXTENT SYS001,...      the SYS001 Workfile
// DLBL   ....
// EXTENT SYS002,...      the SYS002 Workfile
// DLBL   ....
// EXTENT SYS003,...      the SYS003 Workfile
// DLBL   ....
// EXTENT SYS004,...      the SYS004 Workfile
// EXEC   FCOBOL,SIZE=64K
   ...COBOL source deck...
/*
/&
```

1 2

Figure 3.2 - Full JCL Requirement

1 Assign required devices that are not permanent assignments for the installation.

2 Provide label checking information required for the work files. The DLBL and EXTENT job control statements are not completely coded here. These statements will be described in later Units.

Other Information Requirements

In addition to device assignments, certain other information is required for the execution of a language translator:

- How much processor storage will be available to the language translator? You communicate this to the system via the SIZE= parameter on the EXEC statement.
- Information for checking file labels is required. The translators use labeled work files, and these labels must be checked by the system.
- What kind of output do you want? Do you want a program listing? A relocatable module? A cross reference list? The OPTION statement is used to communicate your needs to the compiler.

JCL to Compile a Source Program

Figure 3.3 shows that just a few JCL statements are required when permanent assignments are used and when label information required to check labels on the compiler's work files has been stored within the system. Figure 3.3 also shows coding for the SIZE= parameter of the EXEC statement and for the OPTION statement.

```
* COMPILE A SOURCE PROGRAM, PRODUCE A
* LISTING AND AN OBJECT MODULE
*
// JOB    COMPILE
1 // OPTION LIST,DECK
2 // EXEC   FCOBOL,SIZE=64K
   ...COBOL source deck goes here...
/*
/&
```

Figure 3.3 - JCL to Compile a Program

1 The OPTION job control statement is used to specify output requirements within the job. LIST and DECK request a listing of the compiled program on the system printer (SYSLST), and a punched object module on the system punch (SYSPCH).

2 SIZE=64K limits the COBOL compiler to the use of only 64K of the partition in which it executes.

In the figure, the first three statements are comment statements and are not required except for descriptive purposes. That leaves just five statements (JOB, OPTION, EXEC, /*, /&) as necessary to this compilation.

The SIZE Parameter

The SIZE parameter limits the amount of partition space available to the program named in the EXEC statement (FCOBOL in Figure 3.3). Language translators are "storage gobblers," that is, they dynamically allocate processor storage to hold tables and work areas used during the translation process. Coding the SIZE parameter constrains the translator's use of the processor resource and forces the tables and work areas to disk.

Your operations department should be able to tell you what the proper value of the SIZE parameter is at your installation.

The OPTION Statement

When the job control program processes the OPTION card, it sets on bits in the partition's communication region. When the translator executes, it checks these bits to see what the requirements are.

In the example in Figure 3.3, the LIST option tells the translator to produce a listing of the source language statements, while the DECK option requests an object deck.

Obviously, most people would want to get a source language listing, so is it really necessary to code LIST on an OPTION card every time you run a translation job? The answer is generally no, because any of the OPTION parameters can be established as *system standard options* when you IPL your VSE system. When the system standard options agree with what you want from a translation, there is no need to provide an OPTION card. LIST would be a standard at most installations.

Figure 3.4 is a partial list of the parameters you can code on an OPTION statement. Notice that the options are listed in pairs:

DECK AND NODECK, LIST and NOLIST, etc.

Your installation will have one or the other member of each of these pairs as standard on your system. A complete list of all the options can be found in the *VSE/Advanced Functions System Control Statements* manual under the OPTION statement.

KEYWORD	ACTION CAUSED
DECK	Language translators produce object module, on SYSPCH.
NODECK	Suppresses the DECK option.
ALIGN	The assembler aligns constants and data areas on proper boundries and checks the alignment of addresses used in machine instructions.
NOALIGN	Suppresses the ALIGN option.
LIST	Language translators write the source module listing on SYSLST. The assembler also writes the hexadecimal object module listing and the assembler and FORTRAN write a summary of all errors in the source program. All are written on SYSLST.
NOLIST	Suppresses the LIST option. In addition, this option overrides the printing of the external symbol dictionary, relocation list dictionary, and cross-reference list (see the XREF option).
LISTX	The ANS and DOS/VS COBOL compilers write a PROCEDURE DIVISION MAP on SYSLST. The PL/I and FORTRAN compilers write the object modules on SYSLST.
NOLISTX	Suppresses the LISTX option.
SYM	The American National Standard and DOS/VS COBOL compilers write a DATA DIVISION MAP on SYSLST; the PL/I compiler writes the symbol table on SYSLST.
NO SYM	Suppresses the SYM option.
XREF	The assembler writes the symbolic cross-reference list on SYSLST.
NOXREF	Suppresses the XREF option.
ERRS	The FORTRAN, ANS and DOS/VS COBOL, and PL/I compilers summarize all errors in the source program on SYSLST.
NOERRS	Suppresses the ERRS option.
48C	Specifies the 48-character set on SYSIPT (for PL/I).
60C	Specifies the 60-character set on SYSIPT (for PL/I).

Figure 3.4 - Some of the Available Options

How They Work

Let's look at an example. Assume first that the LIST option is the system standard. This means that a program listing is produced for every translation. But suppose you don't want a listing for a particular translation--you want to suppress the LIST function. You would submit the following card with your job:

```
// OPTION NOLIST
```

NOLIST suppresses the listing for one job. At the job's conclusion, options set via the OPTION statement are reset automatically, and the system standard options are back in effect. LIST will be in effect for all translations that follow yours (assuming no one else has also suppressed the LIST function with an OPTION NOLIST statement).

When to Use OPTION

Code the OPTION statement either when you want to override a system standard option or when you are not sure what the system standard is and want to insure the proper output from your run. It does not hurt anything to code

```
// OPTION LIST
```

when the system standard is already LIST. If you are in doubt as to what the standard is, include the OPTION card with the parameters you want. The system standard will be taken for any parameters you leave out.

Exercise 3.1

Do this review Exercise, then do Computer Exercise 2 in Appendix A. *Do not go on* in this Unit until you have completed the review Exercise and prepared the Computer Exercise for its first run. You may continue your work in this text while awaiting the results of the Computer Exercise.

1. Match each item in the left hand column with its related element in the right hand column.

a.	SYSIPT	1.	Program listing
b.	SYSRDR	2.	Object deck
c.	SYSPCH	3.	Source deck
d.	SYSLST	4.	Work file
e.	SYS001	5.	JCL statements

2. If VSE did not have the ability to maintain permanent assignments, what other technique could be used to reduce the number of JCL cards you would have to handle for a translation?

3. Find the errors in the following sequences of JCL.

a.
```
// JOB ANYONE
// EXEC FCOBOL,SIZE=64K
// OPTION LIST,DECK
  . . . COBOL Source . . .
/*
/&
```

b.
```
// JOB ANYTWO
   OPTION NOLIST,NODECK
// EXEC FCOBOL,SIZE=64K
  . . . COBOL Source . . .
/*
/&
```

Use the *System Control Statements* manual to help you answer the following questions. Don't be concerned about any options other than the ones being asked about in each case.

4. Code the OPTION card for a run of the VSE Assembler that will *suppress* the relocation list dictionary and *cancel* the job if an ASSGN fails.

5. Code the OPTION card to cause PL/I to:

   a. Use the 48-character set

   b. Produce a symbol table on SYSLST

   c. Suppress the object deck

6. Code the OPTION card to cause *any* translator to pass its output to SYSLNK in preparation for later link-editing.

Solutions

1. a. 3
   b. 5
   c. 2
   d. 1
   e. 4
2. A Procedure Library could be used to hold the JCL required for a translation.
3. a. OPTION must precede the EXEC statement that invokes the translator, which is COBOL in this case.
   b. The OPTION statement must have slashes in columns 1 and 2.
4. // OPTION NORLD,ACANCEL
5. // OPTION 48C,SYM,NODECK
6. // OPTION LINK

Computer Exercise

Do Computer Exercise 2 in Appendix A. When you have submitted it for execution, go on to the next Assignment in this Unit.

Assignment 2 - The ASSGN Statement

You learned the basic format of the ASSGN in Unit 2, and used it to associate physical devices with symbolic unit names. In this Assignment you will learn the ASSGN statement's full capabilities. You will also learn how to code the JCL required for a simple compile, link-edit, and execute operation.

Becoming proficient in the use of the ASSGN statement simply requires practice. For this reason a number of exercises that follow require you to code ASSGN statements in a variety of ways. It is suggested you do not skip over any of these exercises, as practice with them will greatly reinforce your learning.

Look at the illustration of the general format of the ASSGN in the *VSE/Advanced Functions System Control Statements* manual, but do not read any of the text in the manual at this time. For now, read only the material presented below.

Temporary vs. Permanent Assignments

The ASSGN may be coded in either statement (slashes) or command (no slashes) format. Remember that JCS format is normally used by programmers while JCC format is normally used by the operator. Regardless of who uses it or how it is submitted (via an actual card or typed in at the system console), the JCS format denotes a *temporary* device assignment while the JCC format denotes a *permanent* device assignment.

A temporary assignment is in effect only during the job in which the ASSGN appears. Permanent assignments are made at IPL time and are always in effect unless specifically overriden by a temporary ASSGN or another permanent ASSGN. Normally, at IPL time, permanent ASSGN's are made for all the standard device assignments at the installation.

Another way to specify whether an assignment is to be temporary or permanent is with the TEMP and PERM optional operands. If either of these operands are present on an ASSGN card, it overrides the presence or absence of slashes in columns 1 and 2.

Other Operands

The first required operand is always a symbolic unit name of the form SYSxxx, where xxx corresponds to one of the valid system or programmer logical unit designations. Considerations for and restrictions on this operand are described in detail in the *System Control Statements* manual.

OPERAND 2: This field specifies the device to which the symbolic unit name SYSxxx will be assigned. Any supported SYSxxx can be assigned as shown in Figure 3.5.

		OPERAND 2
	• to NO device	UA, IGN
	• to a specific device	CUU
GENERIC	• to a selected device from a list of devices	(address-list)
GENERIC	• to a device already assigned to another symbolic unit	SYSyyy
GENERIC	• to a device of a particular "type"	3203, 3505, 3310, 3340, ...
GENERIC	• to a device of a particular "class"	READER, PUNCH PRINTER, TAPE DISK, DISKETTE CKD, FBA

Figure 3.5 - Types of Assignments

Generic assignments permit great flexibility in the way an association is made between a symbolic unit and a physical device. In a generic assignment, a specific physical device is not specified, rather a list of addresses, another symbolic unit name, or a device type or class is specified. The purpose of doing this is to let the VSE system select the particular device for you.

OPTIONAL OPERANDS: This field contains a variety of operands that generally have meaning only for particular device types. Exceptions to this are the TEMP/PERM operands, which may be applied to any device assignment.

Figure 3.6 illustrates a variety of generic assignments, some of which contain one or more optional operands.

```
1. // ASSGN SYS007,SYSRDR
2. // ASSGN SYS012,TAPE
3. // ASSGN SYS011,DISK,SHR
4. // ASSGN SYS013,3340,VOL=123456,SHR
5. // ASSGN SYS014,TAPE,C0
6. // ASSGN SYS016,(183,180,182)
7. // ASSGN SYS017,FBA,VOL=111111,SHR
```

Figure 3.6 - Generic Assignments

1. This specifies that SYS007 is to be assigned to whatever device is currently assigned to SYSRDR. If the SYSRDR assignment is to device 02C then SYS007 will also use 02C.
2. TAPE specifies that SYS012 is to be assigned to an available tape drive of any type. The job control program will search the PUB table for tape drives and will associate the first unassigned drive it finds with SYS012.

3. DISK specifies that SYS011 is to be assigned to any available disk. The SHR optional parameter says that the selected drive may already be assigned to another partition. If SHR were not specified, SYS011 could only be assigned to a disk not in use by any partition, that is, not ASSGNed in any partition. Only direct access storage devices (DASD) can be shared by different partitions. The method of storing data on direct access devices allows for different programs to concurrently access data. This is not possible for tapes, printers, or card devices, nor is it allowed for diskettes.

4. 3340 specifies that SYS013 is to be assigned to a 3340 disk drive that has a volume mounted with the serial number 123456. The VOL= specification cannot exceed six alphameric characters. The disk is to be shareable, that is, it may have other assignments associated with it.

5. This specifies that SYS014 is to be assigned to an available tape drive. Once the assignment is made, the C0 parameter is processed to set the recording made in the selected tape's PUB table entry. READ/WRITE operations will proceed at 1600 BPI in this case.

6. This specifies that SYS016 is to be assigned to the first one of these addresses (left-to-right search order) that is not in use by another program.

7. FBA specifies a 3310 or 3370 Fixed Block Architecture DASD device. The FBA concept is discussed in Unit 5. Non-FBA DASDs, such as 33xx devices, can be generically assigned by using the CKD (Count-Key-Data) parameter in the ASSGN. If you simply code DISK, you may get either an FBA or a CKD device.

NOTE: The generic assignment (// ASSGN SYS012, TAPE) is usually preferred over specific device assignments (// ASSGN SYS012, 180). Generic assignments allow you to run jobs on systems whose device addresses are different or to modify your own system's address configuration without creating new ASSGN statements.

Exercise 3.2

Before attempting this Exercise, read the section on the ASSGN statement in the *System Control Statements* manual.

For the purposes of this Exercise, assume a system with the following PUB Table:

PUB Table

Physical Unit	Device Type	Device Class
02C	3505	READER
02D	3525P	PUNCH
02E	3203	PRINTER
180	3420T9	TAPE
181	3420T9	"
182	3420T9	"
183	3420T7	"
240	3340	DISK
241	3340	"

1. Identify the errors (if any) on the following ASSGN's:

   a. `// ASSGN SYS001,02D,SHR`
   b. `// ASSGN SYS001,02E,VOL=123456`
   c. `// ASSGN SYS001,(02C,02E)`
   d. `// ASSGN SYSRDR,SYSIPT`
   e. `// ASSGN SYSRDR,IGN`
   f. `// ASSGN SYS001,3340,VOL=7654321,SHR`
   g. `// ASSGN SYS001,SYSRES`

2. Generic DASD assignments are shareable across partitions at the user's option, but explicit DASD assignments are *always* shareable. For the following ASSGN's, identify those that are shareable and those not shareable.

   a. `// ASSGN SYS001,240,PERM`

   b. `// ASSGN SYS001,241,SHR`

   c. `// ASSGN SYS001,DISK`

   d. `// ASSGN SYS001,3340`

   e. `// ASSGN SYS001,DISK,SHR`

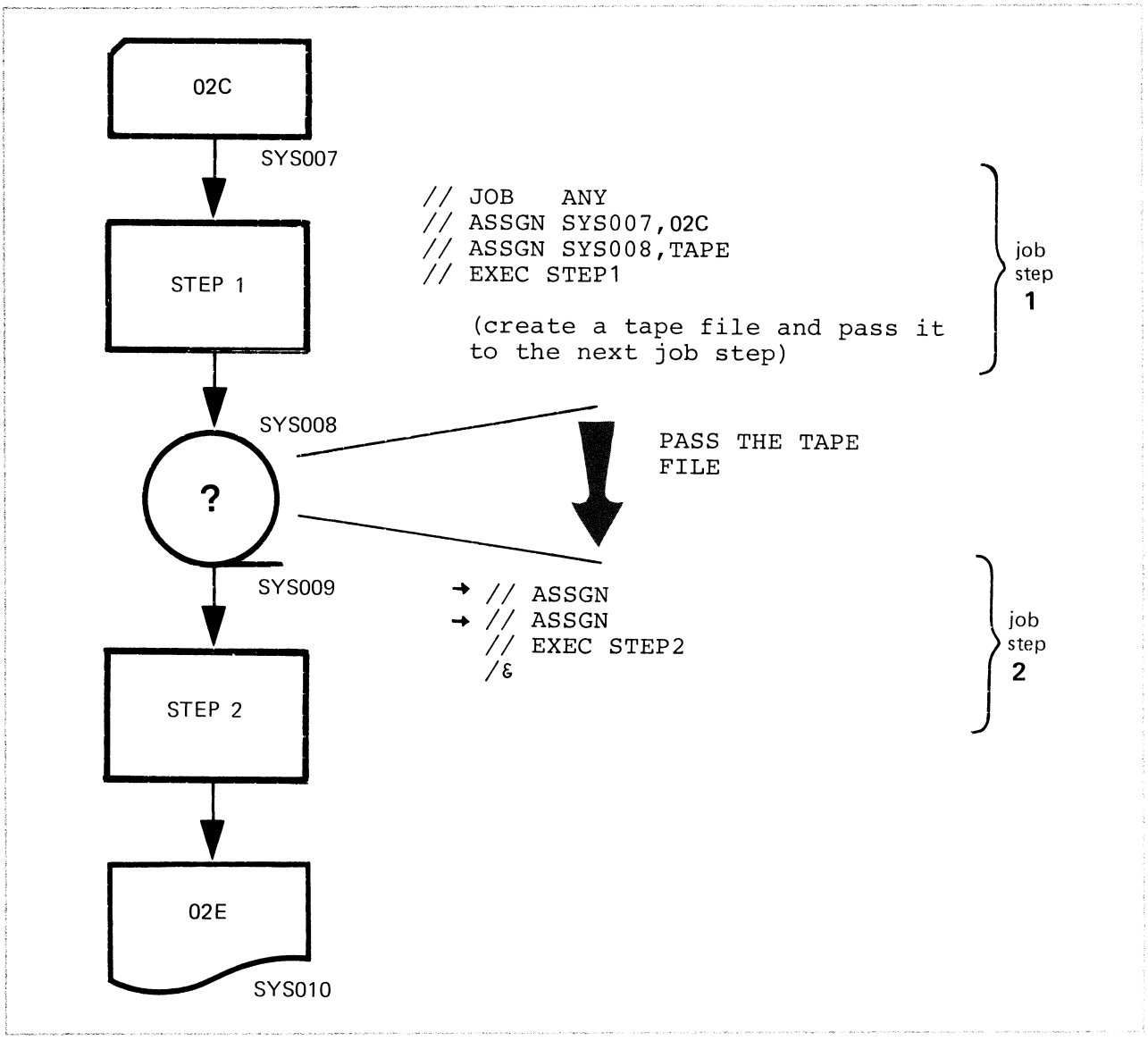
3. Using the PUB table at the beginning of this exercise, code the ASSGN cards needed in the following situations:

   a. To make the 9-Track tapes available to SYS001, but not the 7-Track tape.

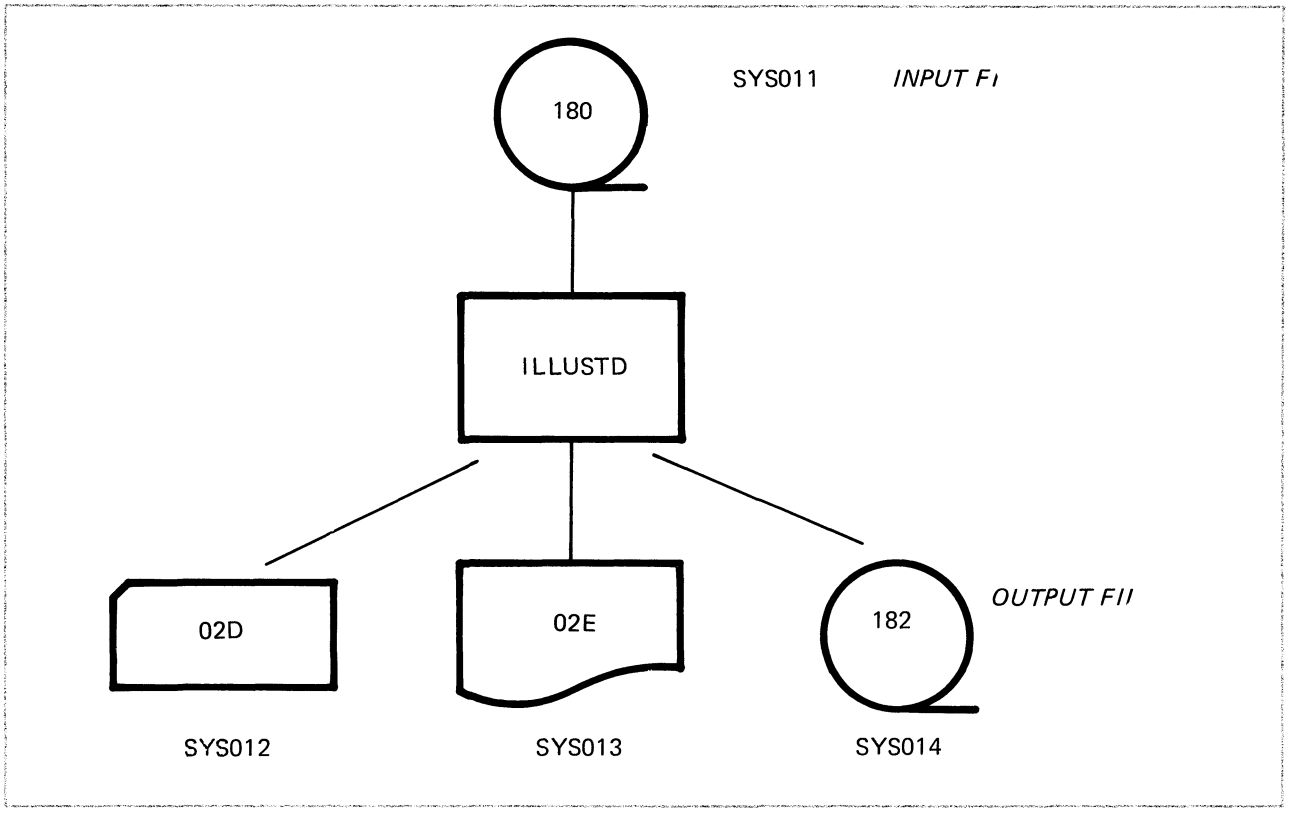
   b. To allow SYS001 to share either disk with another user.

   c. To remove any physical device assignment from SYS001.

4. Generic assignments are commonly used for specifying temporary storage files. In the following job, a tape file built in the first step is to be referenced in the second step. Since the step 1 ASSGN is generic, any tape can be chosen by DOS/VSE to hold the data, but this same tape must be accessed in step 2. Code the required ASSGN's to make this tape available to step 2 and to assign SYS010.


02C
SYS007
STEP 1
SYS008
?
SYS009
STEP 2
02E
SYS010
// JOB ANY
// ASSGN SYS007,02C
// ASSGN SYS008,TAPE
// EXEC STEP1
(create a tape file and pass it to the next job step)
job step 1
PASS THE TAPE FILE
// ASSGN
// ASSGN
// EXEC STEP2
/&
job step 2

5. Permanent assignments are usuallly established in the VSE supervisor at IPL time. They remain in effect at all times unless specifically overridden by other assignments. Program ILLUSTD has the device requirements shown. SYS011 and SYS014 are standardly assigned to tape drives 180 and 182, while the SYS012 and SYS013 assignments must be made by you.



Code the JCL required to execute ILLUSTD, from JOB to end-of-job.

6. Discuss what is wrong with the following pair of ASSGN specifications. Remember, you're still referencing the PUB Table given at the start of this Exercise.

```
// ASSGN SYS004,182
// ASSGN SYS004,183,ALT
```

Solution

1. a. SHR not allowed with a 3525P device.

   b. VOL not permitted with devices other than tape or DASD.

   c. It makes no sense to construct an address-list containing different device types as this one does.

   d. No error.

   e. IGN not valid for SYSRDR.

   f. VOL parameter cannot exceed six characters.

   g. No error. SYS001 is assigned to the same device as SYSRES.

2. a. Shareable

   b. Shareable. The explicit assignment (to a specific device address) is always shareable, so the SHR parameter is redundant here.

   c. Not shareable

   d. Not shareable

   e. Shareable

3. a.
```
// ASSGN  SYS001,(180,181,182)
```
   b.
```
// ASSGN  SYS001,DISK,SHR     or
// ASSGN  SYS001,3340,SHR
```
   c.
```
// ASSGN  SYS001,UA
```

4.
```
// ASSGN  SYS010,02E
// ASSGN  SYS009,SYS008
```

5.
```
// JOB    ANYNAME
// ASSGN  SYS012,02D
// ASSGN  SYS013,02E
// EXEC   ILLUSTD
/&
```

   Because SYS011 and SYS014 are standardly assigned, there is no need to reference them in your JCL.

   Using generic assignments, you could substitute the following cards for the unit record ASSGN's:

```
// ASSGN  SYS012,PUNCH    or  // ASSGN  SYS012,3525P
// ASSGN  SYS013,PRINTER or  // ASSGN  SYS013,3203
```

6. Device 182 is a 9-Track tape, while 183 is a 7-Track tape. It is doubtful that any file would consist of alternating 7- and 9-Track tapes.

Specifying Card Files

Disk and tape file processing requires information so far not covered--information about file labels. Card files, however, have no label information, so you can see the way JCL is specified to handle this type of data.

The situation that follows is one in which only a single card reader is available. This is the case at many installations, and means that the application program has to somehow gain access to the same device that job control uses for SYSRDR.

Let us see how the programmer logical units are specified (in an Assembler program), what the ASSGNs look like, and where the card data goes in the job stream. See Figure 3.7.

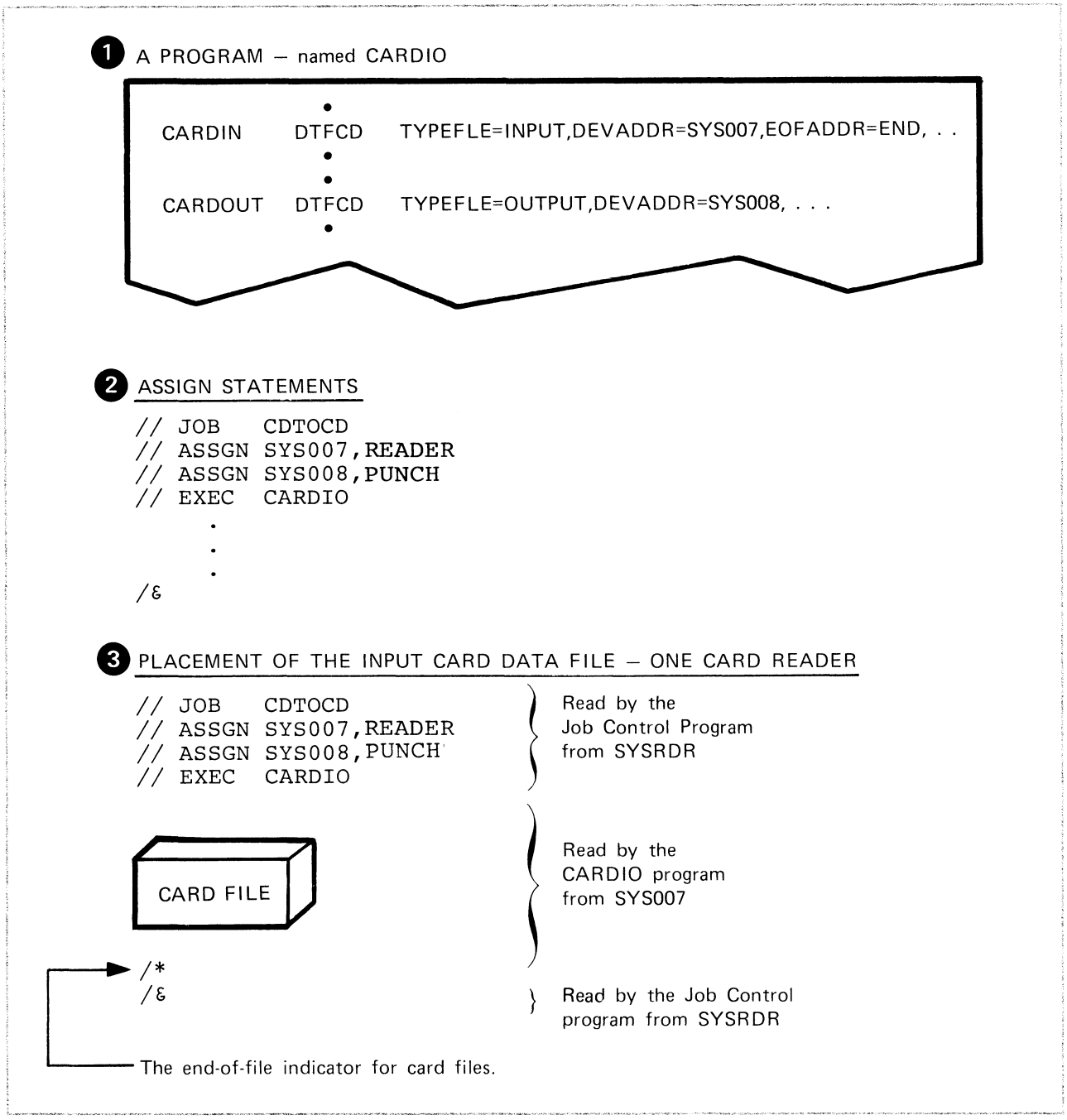


Figure 3.7 - Specifying a Card File

1. This is the portion of the CARDIO program that specifies the two card files. CARDIN is the name of the input card file, and CARDOUT is the name of the output card file. Note that DTFCD (Define The File for CarDs) is an Assembler language statement. Card files in COBOL and other programming languages are specified differently, but that is not important for the points illustrated here.

   The important thing to see is that SYS007 is specified as the *input* card file (TYPEFLE=INPUT), while SYS008 is associated with the *output* card file (TYPEFLE=OUTPUT).

❷ These are the required ASSGN statements. There is nothing unusual about this construction.

❸ Look at the sequence of JCL and data as presented to the single card reader. The JCL is first, and is read by job control from SYSRDR. When job control processes the ASSGN for SYS007, an association is made between SYS007 and the card reader. When the EXEC CARDIO statement is processed, the CARDIO program is loaded into the partition and begins to execute. CARDIO accesses the card data file by referencing the card reader through programmer logical unit SYS007. The card reader is used sequentially, not simultaneously, first by job control as SYSRDR, and then by the CARDIO Program as SYS007.

The /* card causes end of file to be posted to the CARDIO program. If there are no more data files to be read, CARDIO will do whatever final processing it does, then return control to VSE (by an EOJ macro, or a STOP in COBOL, etc.)

Job control will return to read the /& which signals it to perform end of job processing.

Rules of Order

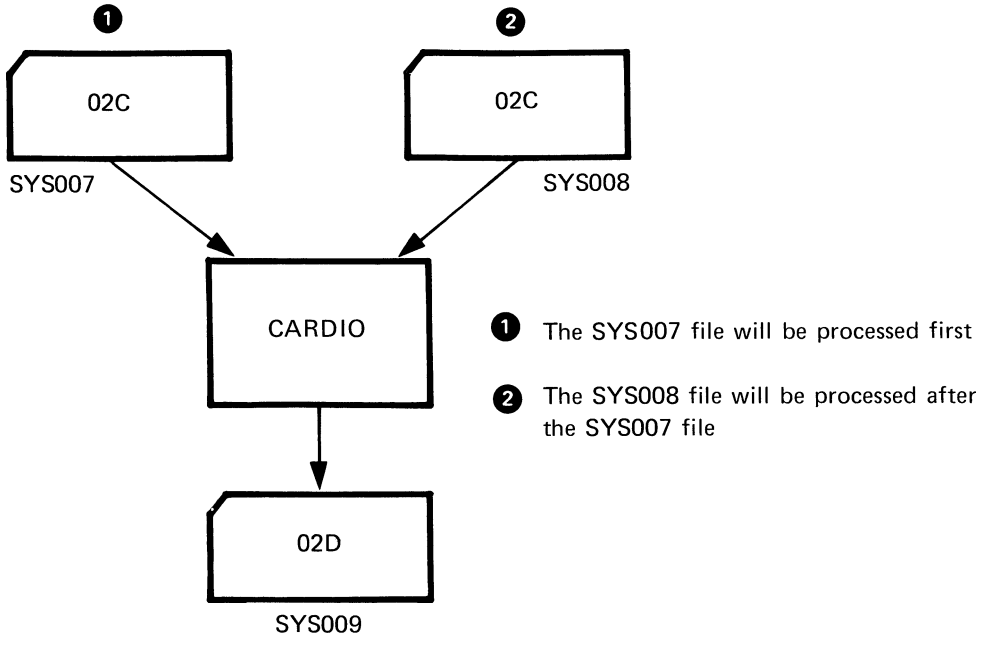
The two rules for specifying input card data files are:

1. When job control statements are interspersed with input data files on one card reader, the input data files must immediately follow the execute statement for the program that is to process them.
2. Each input card data file must be followed by the end of file indicator, the /* statement.

This concludes the material on the ASSGN statement. Do the Review Exercise that follows, then prepare Computer Exercise 3. When you have completed these tasks and have submitted the Computer Exercise for a run, continue with the next Assignment in this Unit.

Exercise 3.3

1. Code a job to execute the CARDIO program.



- Indicate placement of the input card files in your solution with the words "SYS007 DATA CARDS" and "SYS008 DATA CARDS."
- All required system logical units were permanently assigned at IPL time.

Solution

1.
```
// JOB     YOURNAME
// ASSGN   SYS007,READER
// ASSGN   SYS008,SYS007
// ASSGN   SYS009,PUNCH
// EXEC    CARDIO
   (SYS007 data cards)    first input card file
/*
   (SYS008 cata cards)    second input card file
/*
/&
```

The card data files must immediately follow the EXEC statement.

Both input card files are terminated by the /* statement. The /* is required as an end of file indicator for card input files.

The sequence of the card files is critical. Because of the way its logic flow is organized, the CARDIO program expects to process the SYS007 data cards first. These cards must therefore follow right behind the EXEC CARDIO statement in the job stream.

Computer Exercise 3

This Exercise will require you to use some functions you have not yet been explicitly taught. The hints below should help you get the job running, but if you find yourself having excessive difficulty, wait until you have studied the first Assignment of Unit 4 before attempting the Exercise again. Good Luck

- The linkage editor is a program that prepares compiler-produced object modules for execution. It resides in the system Core Image Library, and you can assume all its device requirements are standardly assigned.

  Its program name is LNKEDT.

- In order to invoke a program that has been "temporarily cataloged" in the CIL, use the EXEC card with a blank operand field.

Unit 4, The Linkage Editor, will expand upon these points.

The *VSE Advanced Functions Messages* manual is your guide to messages produced by component programs of VSE.

Using this manual is very much like using a dictionary. A five-character code is associated with each message. This code is like a "word" whose meaning you look up. You will find that many messages have multiple causes and alternative responses. It will be up to you to determine the cause and response applicable in any given situation.

While it is true that the operator receives and responds to most of the messages generated by VSE, it will often be your responsibility to determine what happened during your program's execution by examining the messages in your output.

Figure 3.8 illustrates the format of the message code and shows what each of its five characters represents. The two-character partition identifier is not really a part of the message code, but is there to indicate the partition from which the message was sent.

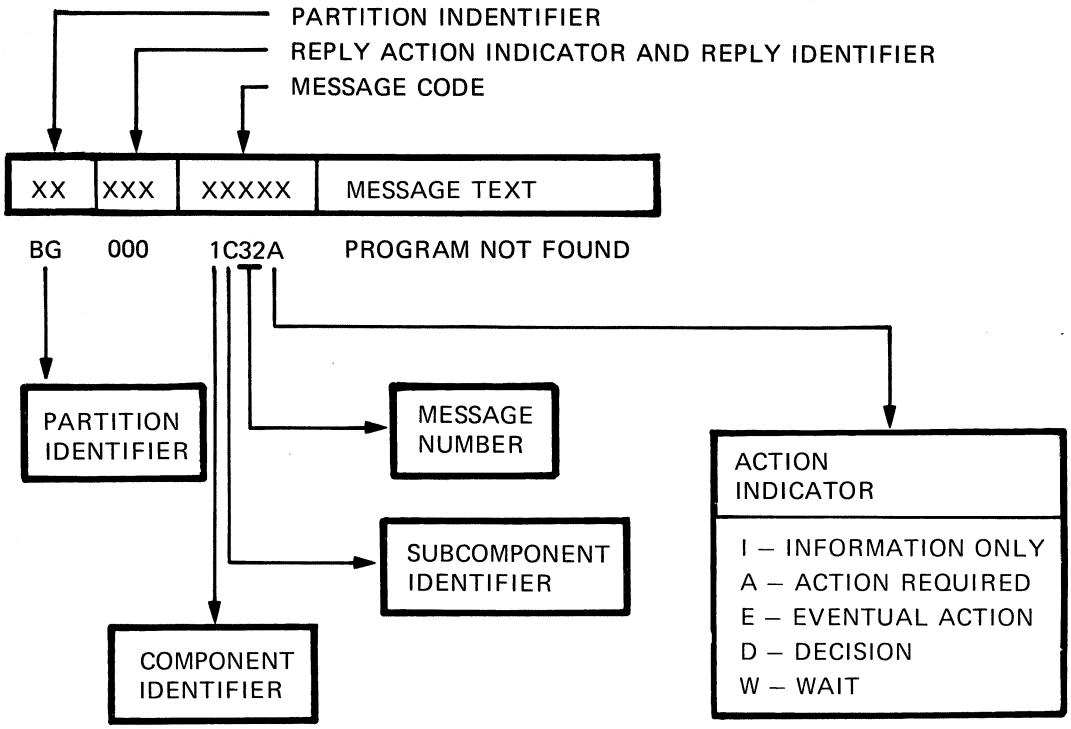


Figure 3.8 - VSE System Message Format

The component identifier tells which program in the VSE System is communicating with you. By knowing which component sent a message, you have some knowledge of the type of error that has occurred. For instance, a message from one of the access methods would probably pertain to some aspect of data management. A message from job control, on the other hand, probably would deal with an error in a job control statement.

Most of the major VSE components have some number of subcomponents. A subcomponent may be a unique program, or it may be a major routine. For example, job control is a major component. Within job control, the ASSGN routine, the Buffer Load program, and the Job Initiation and Termination routine are subcomponents.

The message shown in Figure 3.8 has "1C32A" as a message code. "1C" messages are from the Job Initiation and Termination Routine (C) of job control (1).

Action Indicators

The last character in the message code ("A" in this case) tells the operator what choices there are in responding to the message. These action indicators have the following meanings:

I *Information:* Informational only. No response is necessary.

A *Action:* The operator must do something to respond to a system request. For example, mount a particular volume needed by an active program.

E *Eventual Action:* A future action will be required.

D *Decision:* The operator must select one of several alternate responses.

W *System Wait:* The operator must respond to a hardware failure.

Using the Manual

Use the **Messages** manual to locate message 1C3nA. The information provided for the message is presented under four headings:

Cause

System Action

Programmer Action

Operator Action

The cause of this message is that the program specified for execution is not in the Core Image Library. The manual then proceeds to describe what the system will do in such a situation, and what the programmer and/or operator should do to remedy it.

The "A" action indicator means that the system will wait until the operator takes some action before processing will be allowed to continue. If the error was due to a misspelled phase name, the operator may key in the correct name and continue with the job. Otherwise the job is cancelled.

The "n" indicates the number of fields processed in the EXEC statement when the error was detected. This information helps you analyze the statement that caused the error. Figure 3.9 shows how the number "n" relates to the fields of any job control statement or command. Note that the "//" is considered as the first field. If the message code were "1C31A", then the field "//" would be in error.

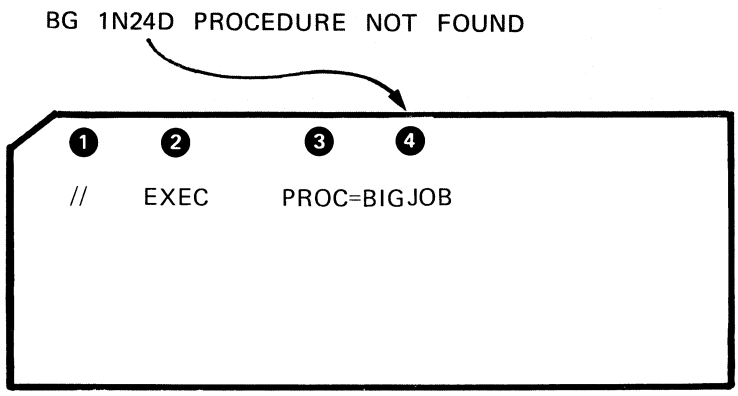


Figure 3.9 - Statement Parameter Fields are Numbered

In the messages just examined, there was only one possible cause for the error. This is not always the case. For example, look up message 1A0nD, and note that there are more than a dozen possible causes for this error message. What programmer action is called for in response to this message? Absolutely none. But the operator has several alternative actions, depending on what is determined to be the cause of the message.

In the manual *VSE/Advanced Functions Messages* (SC33-6098) read "The Message Code" and "When You Get a Message".

In the manual *VSE/Advanced Functions System Management Guide* (SC33-6094) under "Using the System", read "Executing a Program".

Unit Summary

Knowledge of how to use the language translators is essential in order to get work done at most data processing installations. Under VSE, the compilers are handled in a straight-forward fashion, with their output controlled primarily by the OPTION statement. System standard options save you the trouble of repeatedly requesting the most common type of compiler output, and the OPTION itself can be used for your specific requirements.

The program you compile references its external files by means of symbolic, or logical, unit names. It is the ASSGN that associates these logical names with actual devices at execution time. ASSGN card operands can tend to look a little complicated, but remember that they must merely be specific enough to properly identify the physical devices your program needs. The System Control Statements manual is your most important reference book for ASSGN formats.

Generic assignments allow you greater flexibility than explicit device assignments, as you can avoid limiting your JCL to specific devices. By requesting a device of a particular type or class, you can avoid having to change ASSGN statements from one device to another depending on what is available at any given time. In addition, generic assignments allow you to assign one logical unit name to another. This is a handy feature in passing temporary work files from job step to job step.

VSE communicates to its users by means of messages to the system console (SYSLOG) and to the system printer (SYSLST). The Messages manual is your means of interpreting the variety of information that accompanies a system message. Many times a mysterious situation can be quickly resolved by a careful examination of the messages generated by the job during its execution.

Take the Mastery Test that follows and check your answers against the solutions provided. You may use your System Control Statements manual for reference.

1. Find the errors in the following statements:

   a. `// ASSGN SYS001,3330,C0`

   b. `/  ASSGN SYS001,180,VOL=123456`

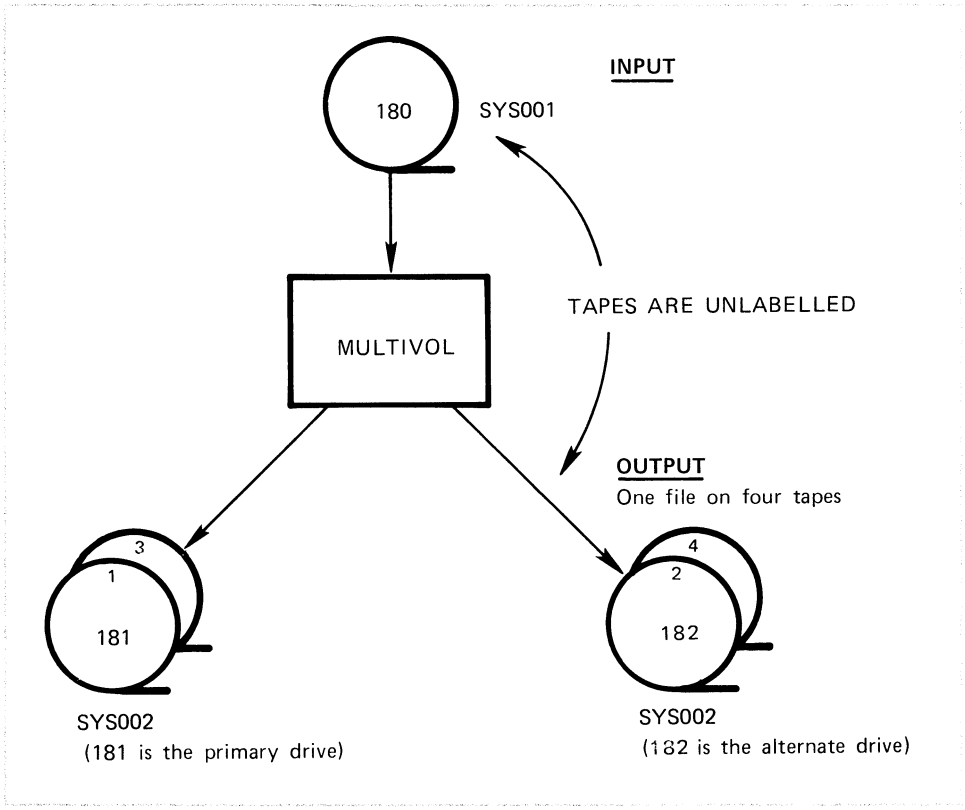
   c. `// ASSGN SYS001,VOL=123456`

   d. `ASSGN SYS001,SYSRES,TEMP`

   e. `ASSGN SYS001,READER,SHR`

2. Multivolume Tape Files

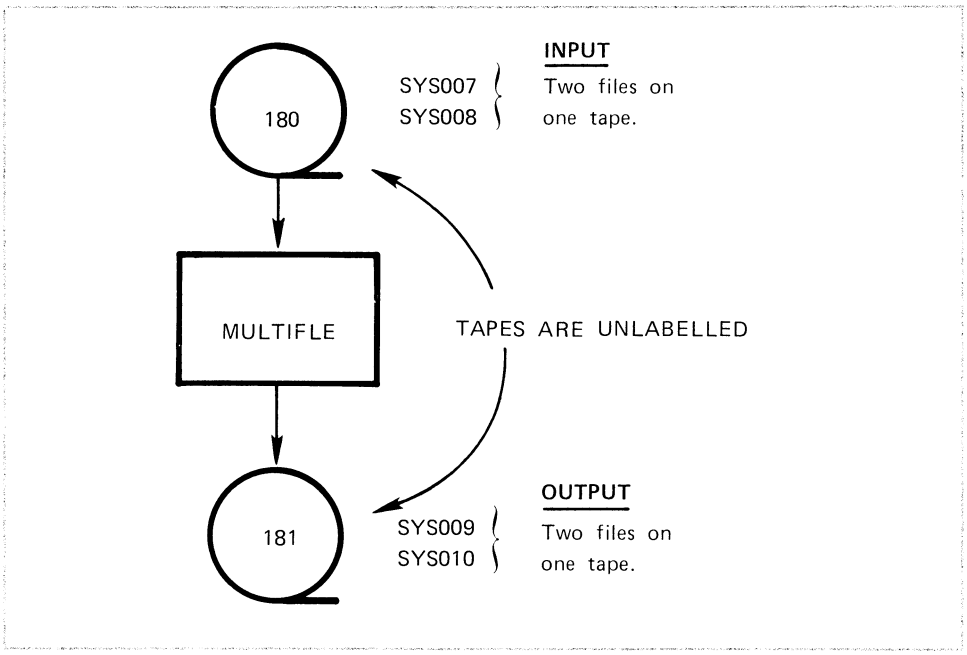
   Code a job to execute the MULTIVOL program.



   - SYS001 and SYS002 are not standard assignments.
   - All required system logical units (SYSRDR, etc.) have been permanently assigned.

3. Multifile Volumes

   Code a job to execute the MULTIFLE program.



- The SYS007 and SYS008 input files are on the same tape volume. SYS007 is at the beginning of the tape; SYS008 is the next file on the tape.
- The SYS009 and SYS010 output files are to be written on the same tape volume. SYS009 is to be written at the beginning of the tape; SYS010 is to be written following SYS009 on the tape.
- SYS007, SYS008, SYS009 and SYS010 have not been permanently assigned.
- All system logical units required have been permanently assigned. (SYSRDR, etc.)

4. Code statements for JOBA, JOBB, and JOBC.

   Note that SYS008 is used in each job, and must be assigned to 02C for JOBA, to 181 for JOBB, and again to 02C for JOBC.

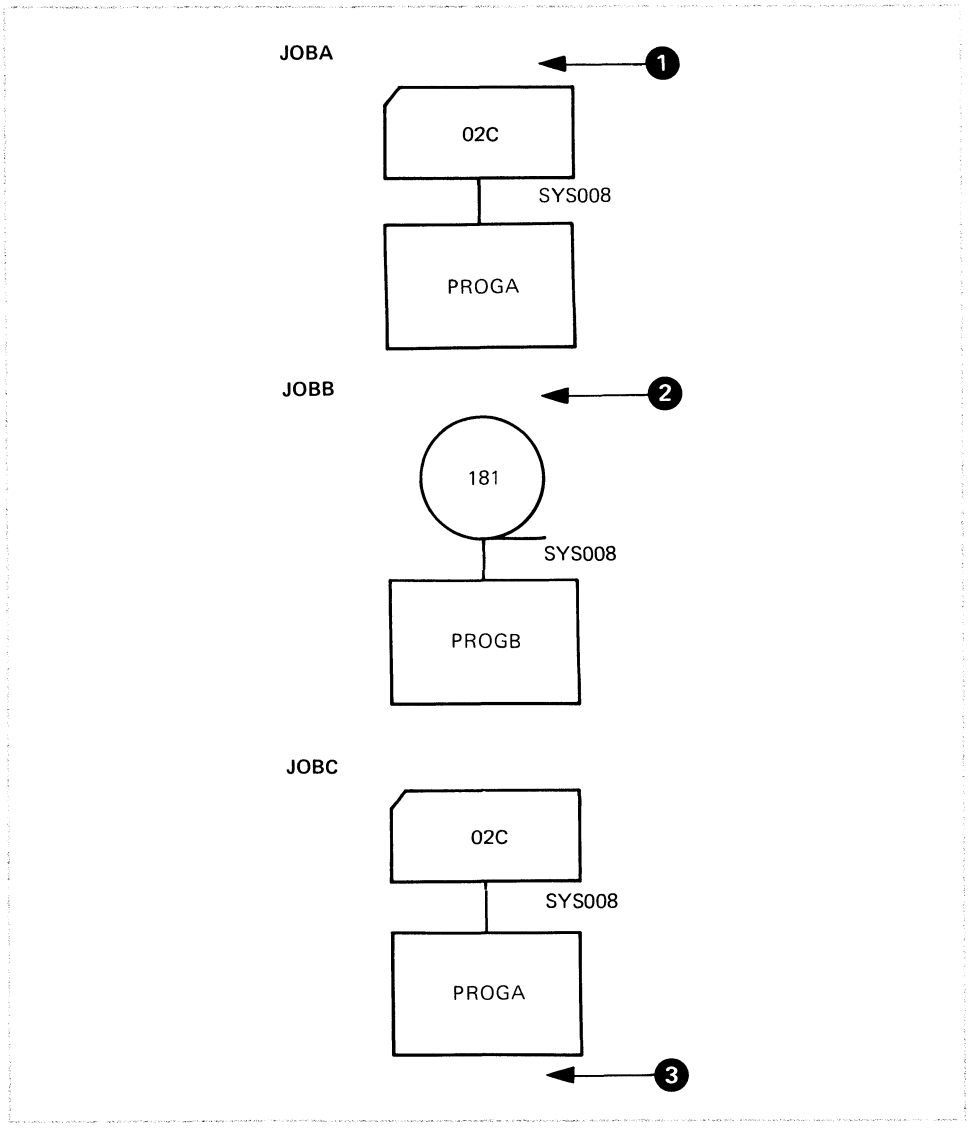
   A permanent assignment exists for SYS008, device 241. It will have to be changed to run these jobs. After the jobs are run, restore SYS008 to its prior assignment.

   Required in this exercise (see corresponding numbers in the diagram below):

   ❶ Assign SYS008 to 02C using a permanent assignment.

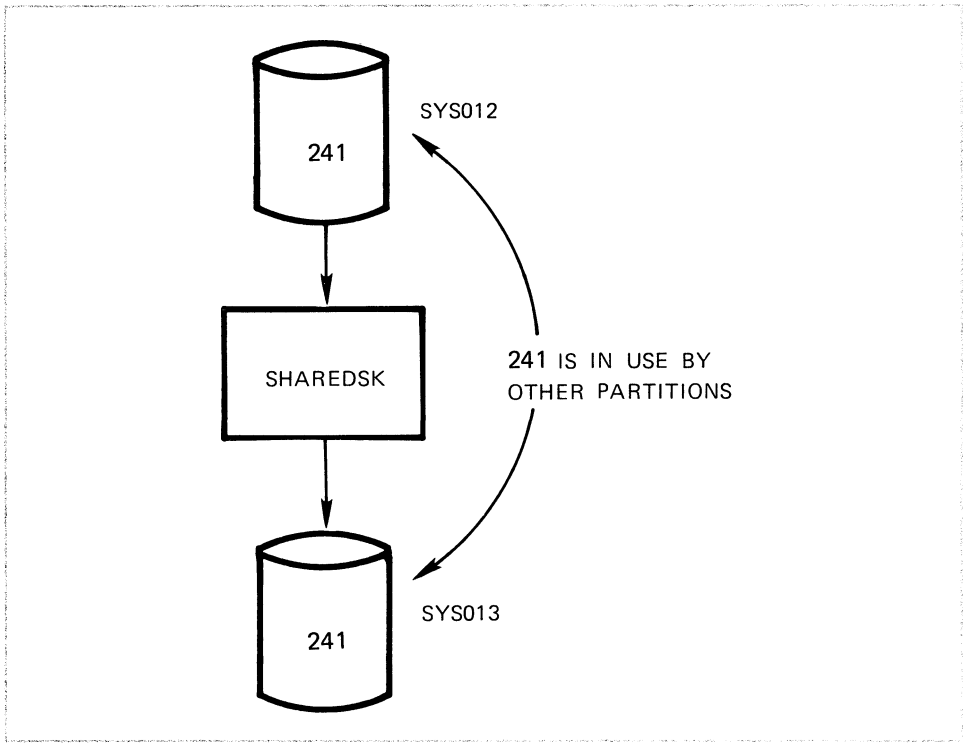
   ❷ Assign SYS008 to 181 using a temporary assignment.

   ❸ Restore SYS008 to its original assignment.



5. Sharing direct access devices with other partitions.

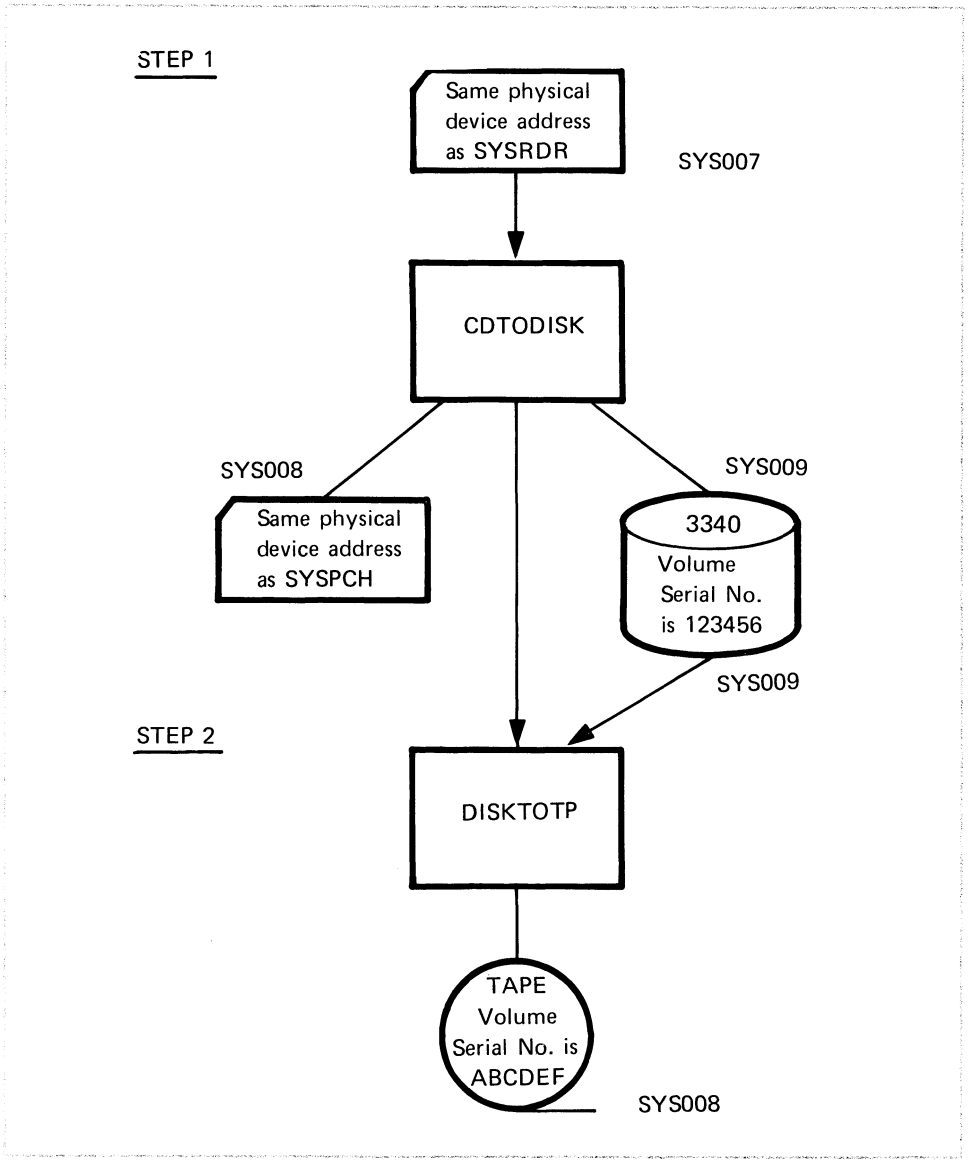
   Code a job to execute the program SHAREDSK.



- Other partitions will also be accessing the disk pack.
- Ignore label information requirements.
- SYS012 and SYS013 have not been permanently assigned.
- All system logical units required have been permanently assigned. (SYSRDR, etc.)

6. Shareable generic DASD assignments.

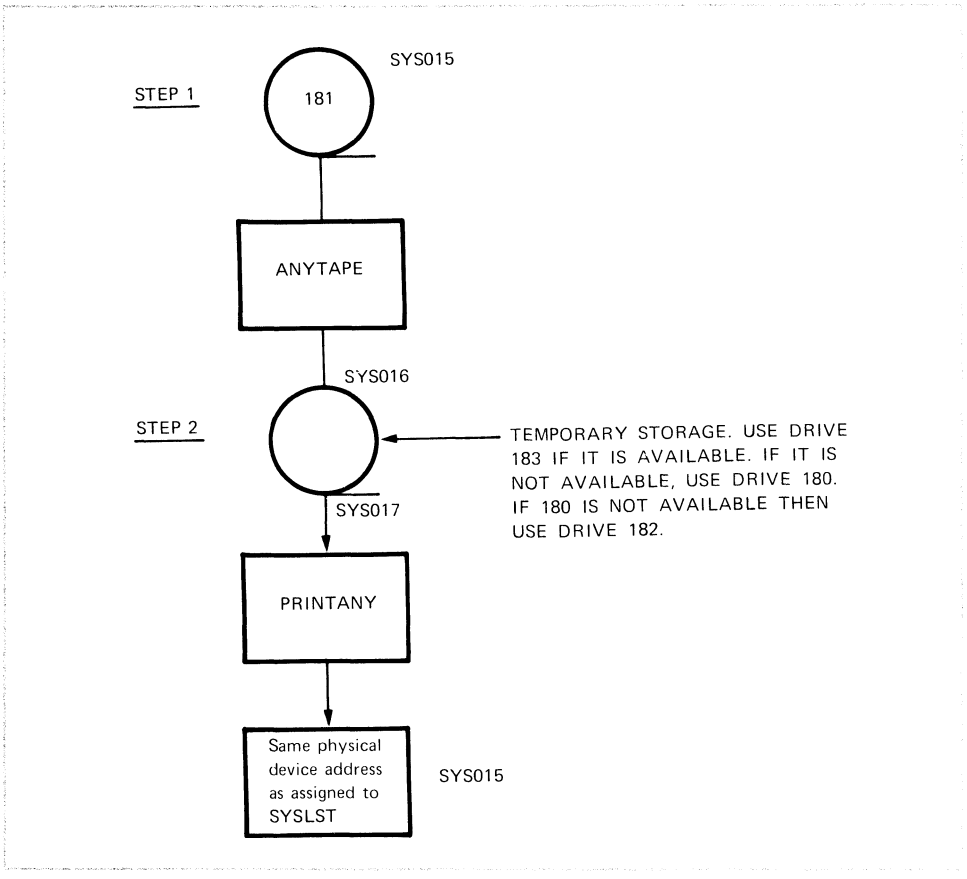
   Code a job to execute the CDTODISK and DISKTOTP programs.



   Indicate placement of the input card file with the words "SYS007 Data Cards."

   All required system logical units are permanently assigned.

7. The ANYTAPE and PRINTANY programs require a scratch tape. Any one of three drives could be used but there is no way to determine which of them will be available when the programs are run.



All required system logical units are permanent assignments.

Code the JCL necessary to make the appropriate assignments and execute these two programs.

Solution

1. a. Mode setting (the C0) is allowed only with tape devices. The 3330 is a disk.

   b. Slash missing in column 2.

   c. Device type is missing.

   d. No error.

   e. A card reader is not a shareable device.

2.
```
// JOB     TWOVOL
// ASSGN   SYS001,180          INPUT
// ASSGN   SYS002,181          OUTPUT
// ASSGN   SYS002,182,ALT      OUTPUT
// EXEC    MULTIVOL
/&
```

   ALT is not coded in the ASSGN for SYS002 to 181 because 181 is the primary drive.

3.
```
// JOB     MANYFILE
// ASSGN   SYS007,180        INPUT
// ASSGN   SYS008,180        INPUT
// ASSGN   SYS009,181        OUTPUT
// ASSGN   SYS010,181        OUTPUT
// EXEC    MULTIFLE
/&
```

4.
```
    // JOB     JOBA
a.  ASSGN   SYS008,02C
    // EXEC    PROGA
       (card input)
    /&
    // JOB     JOBB
b.  // ASSGN   SYS008,181
    // EXEC    PROGB
    /&
    // JOB     JOBC
c.  // EXEC    PROGA
       (card input)
    /&
d.  ASSGN   SYS008,241
```

   a. This ASSGN command permanently assigns SYS008 to 02C. It could also have been coded with the PERM operand.

   b. This statement temporarily assigns SYS008 to 181. It overrides, for this one job, the permanent assignment just made of SYS008 to 02C.

   c. No ASSGN appears in this job. None is needed. Execution of PROGA requires that SYS008 be assigned to 02C. This assignment is in effect already, because of the permanent assignment made in JOBA. Remember, the override in the previous job does not carry into subsequent jobs.

   d. Finally, this ASSGN command permanently overrides the assignment made in JOBA. Neither the RESET command or statement would work for this, as they will only reset temporary assignments.

A Suggestion

Discuss any intended changes to permanent assignments with your system programmers or operators before making them. They should be used with discretion, and temporary assignments used where possible.

5.
```
// JOB     FOUR
// ASSGN   SYS012,241
// ASSGN   SYS013,241
// EXEC    SHAREDSK
/&
```

Since these assignments were made using explicit addresses, the SHR parameter was not required. If generic assignments had been used, SHR would be required, as shown below:

```
// JOB     FOURMORE
// ASSGN   SYS012,DISK,SHR
// ASSGN   SYS013,SYS012,SHR
// EXEC    SHAREDSK
/&
```

Notice here that SYS013 has been forced to use the same disk as SYS012, even though no VOL information is available.

6.
```
// JOB     NAMEANY
// ASSGN   SYS007,SYSRDR
// ASSGN   SYS008,SYSPCH
// ASSGN   SYS009,3340,VOL=123456,SHR
// EXEC    CDTODISK
   (SYS007 data cards)
/*
// ASSGN   SYS008,TAPE,VOL=ABCDEF
// EXEC    DISKTOTP
/&
```

Volume 123456 and SYS009 are used in both steps, however only one ASSGN for SYS009 is needed. It will be in effect for the entire job since there are no other ASSGN's for SYS009.

Because 3340 is a generic assignment, the SHR parameter is required to make the device shareable.

7.
```
// JOB     SOMENAME
// ASSGN   SYS015,181
// ASSGN   SYS016,(183,180,182)
// EXEC    ANYTAPE
// ASSGN   SYS017,SYS016
// ASSGN   SYS015,SYSLST
// EXEC    PRINTANY
/&
```

SYS016 will be assigned to device 183 only if 183 is not already assigned in another partition or is otherwise unavailable. If that is the case, job control will attempt to assign SYS016 to 180, the next device in the list. Failing that, 182 will be tried. If all three of the devices are unavailable, the job will be cancelled.

Remedial

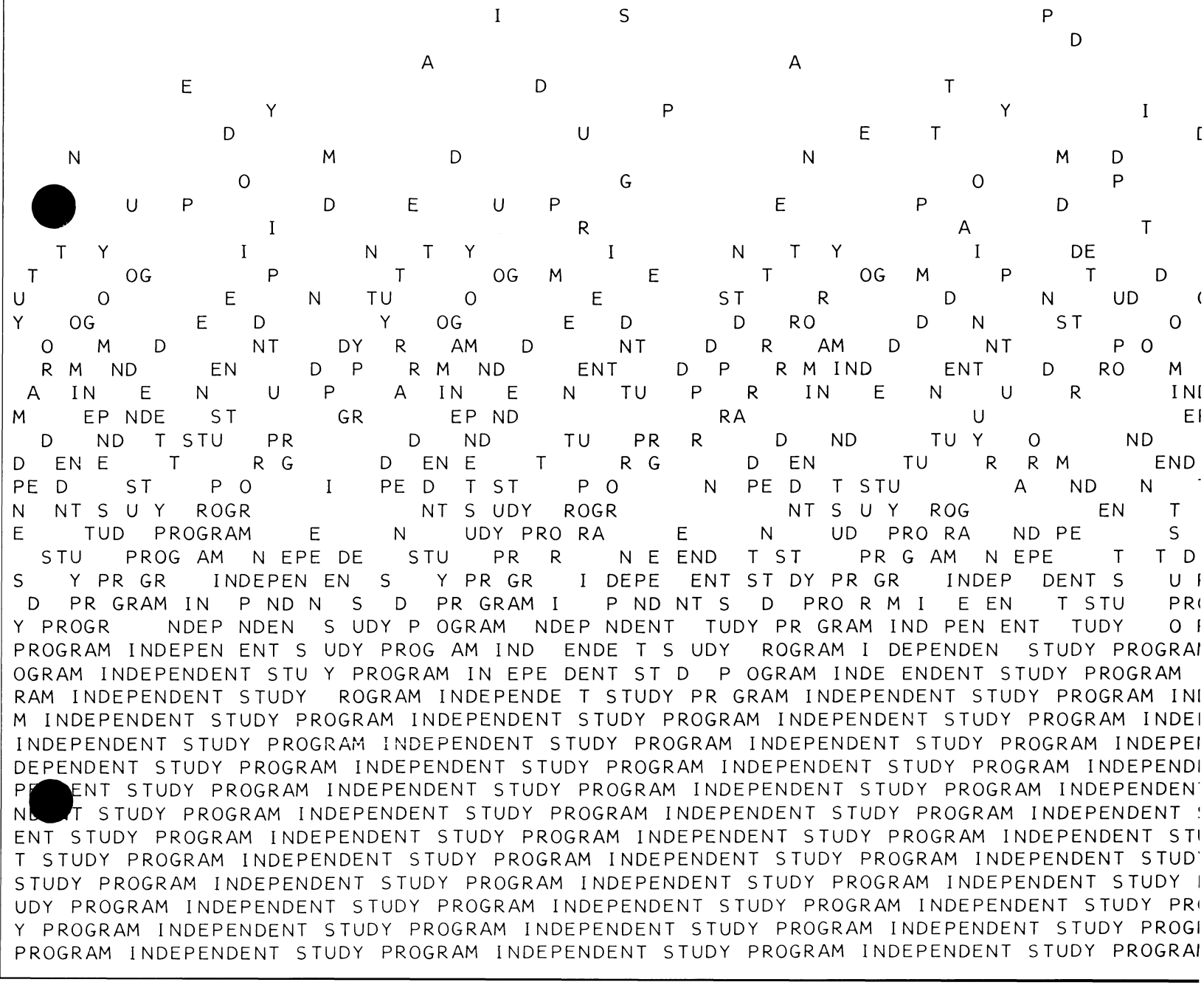
If you experienced difficulty with the questions on this test (three or more of the problems 2 - 7 incorrectly coded), it is due to either a lack of attention to details or a lack of understanding.

You should not proceed to the next Unit until you are confident you comprehend the material you have studied so far. If you coded three or more of problems 2 - 7 incorrectly, do the following:

- Reread the section on the ASSGN card in the System Control Statements manual.
- Review the problems you missed and try to see where you went wrong.
- If you are still confused, discuss your points of confusion with a knowledgeable person at your installation.

Unit

4



Unit 4:

The Linkage Editor

Introduction

Prior to execution in storage, all programs must be placed in the Core Image Library. The output of the linkage editor is an executable *phase*.

The name "linkage editor" appropriately reflects the editing and the linking operations that this program performs. The linkage editor prepares a program for execution by *editing* the output of a language translator into Core Image Library phases. The linkage editor also combines separately assembled or compiled program sections or subprograms (called "object modules") into phases. This process is referred to as *linking*.

A program can be link-edited and

- cataloged permanently
- cataloged permanently and executed immediately, or
- cataloged temporarily and executed immediately.

When a phase is cataloged permanently into the Core Image Library, the supervisor can load it directly from the library in response to an EXEC job control statement. If the phase is cataloged temporarily and executed immediately, link-editing is required the next time the program is to be run. Phases are stored either temporarily or permanently, depending on the option specified in the OPTION job control statement:

```
//  OPTION LINK
```

If the LINK option is specified, the phase is stored temporarily for immediate execution in the same job. This phase will be overwritten in the Core Image Library by the next program that is link-edited.

```
//  OPTION CATAL
```

If the CATAL option is specified, the phase is stored permanently and can be executed any time after the catalog job.

Objective

Upon completing this Unit, you should be able to:

Assignment 1:

- Use the OPTION card to direct the linkage editor's cataloging function.
- Prepare a basic set of job control statements to compile, link-edit, and execute a program (if you have not yet achieved this Objective from Unit 3).

Assignment 2:

- Use linkage editor control cards to structure linkage editor input and output, and control linkage editor operations.

Assignment 3:

- Use a Relocatable Library as an input source for the construction of program phases.
- Use private Core Image and Relocatable libraries when they are required for link-edit operations.
- Invoke or suppress the autolink feature to control linkage editor inclusion operations.
- Code UPSI and DATE job control statements to communicate information to executing program phases.

Materials Required

Study Guide (SR20-7300)

The following VSE reference material:

Introduction to the VSE System (GC33-6108)

VSE/Advanced Functions System Management Guide (SC33-6094)

VSE/Advanced Functions System Control Statements (SC33-6095)

Before getting into link-edit operations, we will first review the activities of the program development cycle. Figure 4.1 traces this flow and shows the relation of various VSE components. The dotted lines from the Source Statement and Relocatable libraries indicate that they are optional sources of input to the language translators or linkage editor respectively, although the Relocatable Library is used as an input source to the linkage editor more often than not.

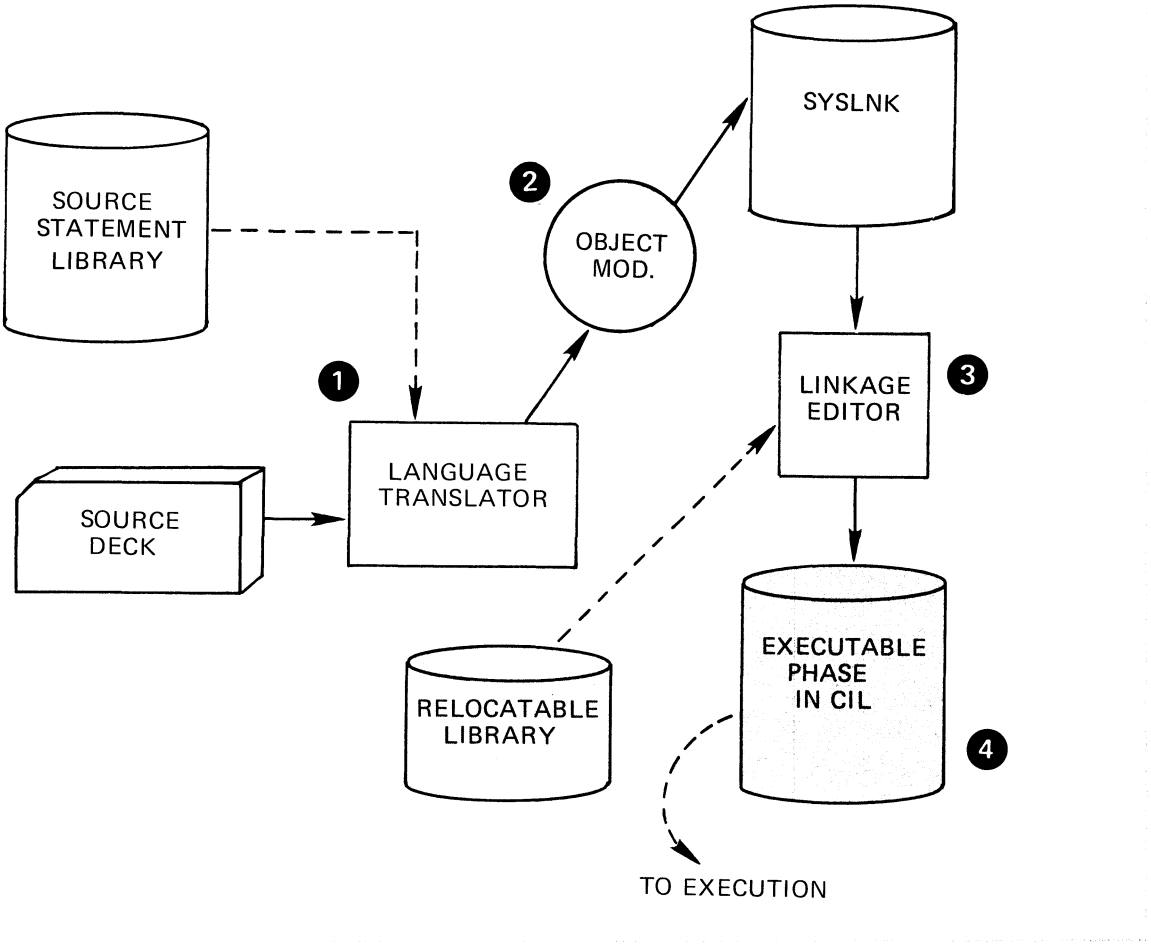


Figure 4.1 - Evolution of a Program

1. Source code can be submitted directly to the translator or it can be cataloged to a sublibrary of the Source Statement Library for later processing by the translator. The language translator produces an object module from your source program statements.

2. The object module is either written directly to SYSLNK by the compiler or is produced in deck format. If it is written to SYSLNK, it is immediately available to the linkage editor. If it is in deck format, it requires further processing.

3. A Relocatable Library is used for the resolution of external references made by your program. The linkage editor develops an executable program (called a *phase*) from one or more object modules. This phase is placed in a Core Image Library.

❹ Your phase in the CIL may be invoked and executed with the EXEC job control statement.

Why isn't the object code in step 2 of this sequence capable of being directly executed under VSE? Figure 4.2 shows what happens when a translator processes your source code into object module format.

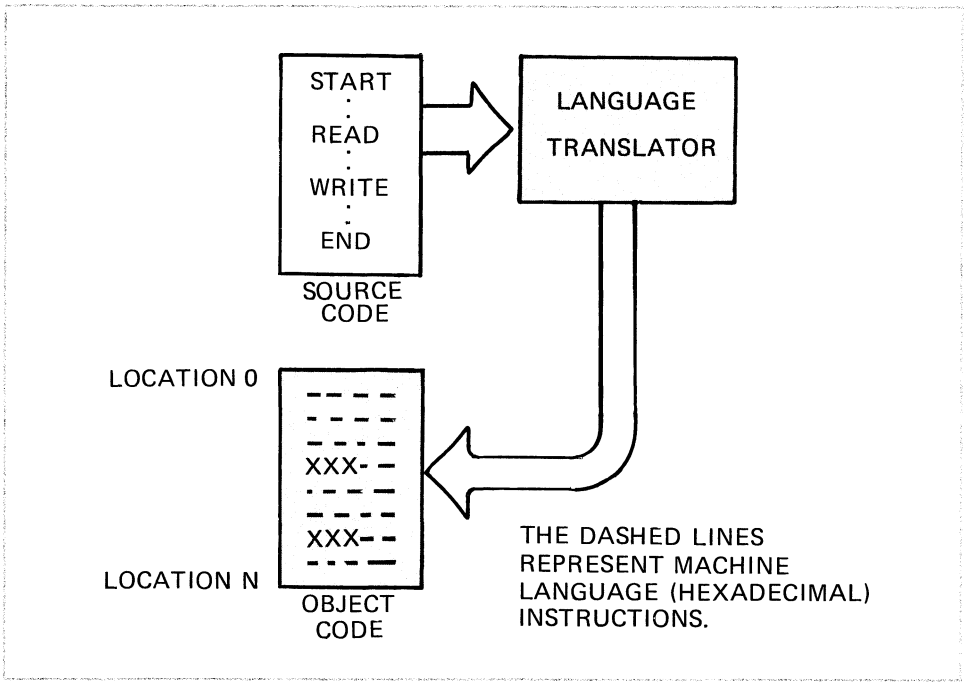


Figure 4.2 - Source to Object

Object code has two characteristics that make it incapable of being executed *as is* under VSE:

1. Its internal address references are relative to 0 or some other fixed value; and
2. It has pieces missing.

While it may not be immediately apparent why the first of these should be a problem, it certainly seems clear that the second could get in the way of successful program execution. Let's examine this first.

What is missing from the object code in Figure 4.2? Notice that the source contains READ and WRITE statements (these could just as well be GETs and PUTs). These I/O directing commands cause two things to be generated in the object code:

- In-line machine code to perform some basic functions; and
- An *external reference* to an I/O module or modules.

In other words, only a small part of the code required to do the I/O operation is included in-line. Most of the code resides in an I/O module that exists in another part of the VSE system (in a Relocatable Library, more of which will be discussed later).

This externally referenced code must somehow be associated with your object program before execution is possible.

Internal Address References

The language translators also develop addresses inside your object module in order for your code to reference itself. These references are usually based on 0 as a starting point.

The rows of X's in Figure 4.2 represent external references to I/O modules. Assume for a moment that the X's made no difference, that your program does not need them resolved. Why is your program still incapable of being executed under VSE? See Figure 4.3.

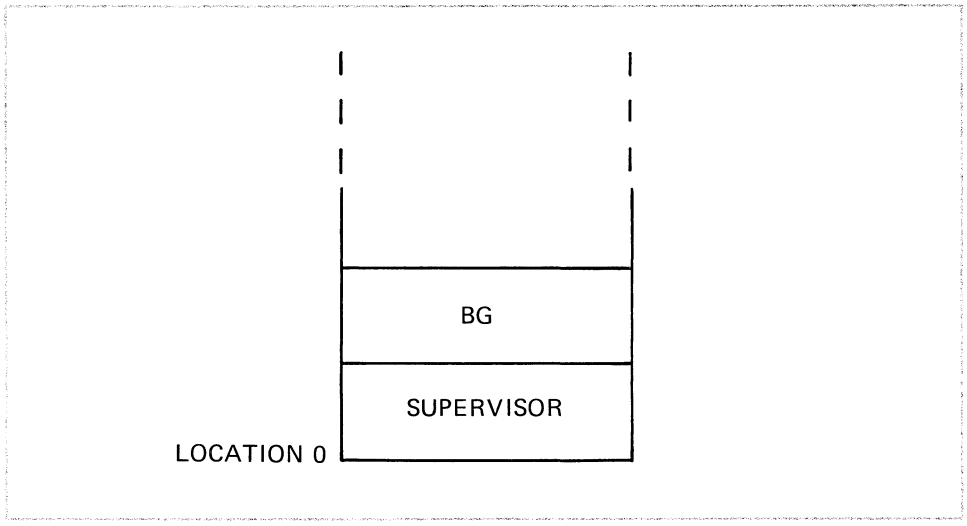


Figure 4.3 - The Supervisor Uses Lower Storage Addresses

The VSE supervisor uses storage starting at location 0. Your program can never overlay any part of the supervisor, so it cannot be loaded to run at location 0. The program must be *relocated,* that is, have all its internal address references modified to allow it to run in one of the partitions BG, F1, F2, etc. This is part of the function of the linkage editor.

Using the Linkage Editor

The linkage editor resides in the system Core Image Library where it is permanently cataloged under the name LNKEDT. It takes one or more object modules (or parts of modules) as input and produces executable program phases in a Core Image Library as output.

Symbolic Units

The linkage editor requires the symbolic units listed below.

SYSIPT	for module input (primary input source)
SYSLST	messages and listings for the programmer
SYSLOG	operator messages
SYSRDR	job control input (control statements)
SYSLNK	input to the linkage editor
SYS001	linkage editor work file

All of these units will normally be permanently assigned on your system. Your primary concern with these units is to be aware of the purpose and function of each of them.

Additionally, a Core Image Library must be available to receive the Linkage Editor output. Also at least one Relocatable Library may be needed for input to the Linkage Editor. How to provide availability to these libraries will be discussed in Unit 7.

Methods of Cataloging

Programs are stored either temporarily or permanently in a CIL by the linkage editor depending on what is specified in the OPTION control statement:

- If LINK is specified, your program is stored temporarily in a Core Image Library for immediate execution in the same job. The program will not be accessible after the job completes (after the /& statement is processed by job control). The program will be written over by the next program link-edited.

  This option is usually appropriate for newly written programs in the testing stages.

- If CATAL is specified, your program is stored permanently in a Core Image Library. It can be executed immediately or in later job steps or jobs. It can be deleted only by the library maintenance program or by the cataloging of another program with the same name.

LINK will be discussed right now, while CATAL will be covered in the next Assignment.

Temporary Cataloging

Compile, Link-Edit, and Execute

Using OPTION LINK, you can set up a single multistep job to compile a program, then link-edit and execute it without creating a permanent entry in a CIL. This job is also referred to as compile and test, compile and go, or compile and execute. The functions are the same regardless of what it is called.

Figure 4.4 shows the steps performed in a job to compile, link-edit, and execute a COBOL source program. The picture would be identical for the Assembler, PL/I, FORTRAN, and RPG language translators except that the EXEC FCOBOL statement would be replaced by an EXEC statement naming the Assembler, PL/I, FORTRAN, or RPG compiler programs.

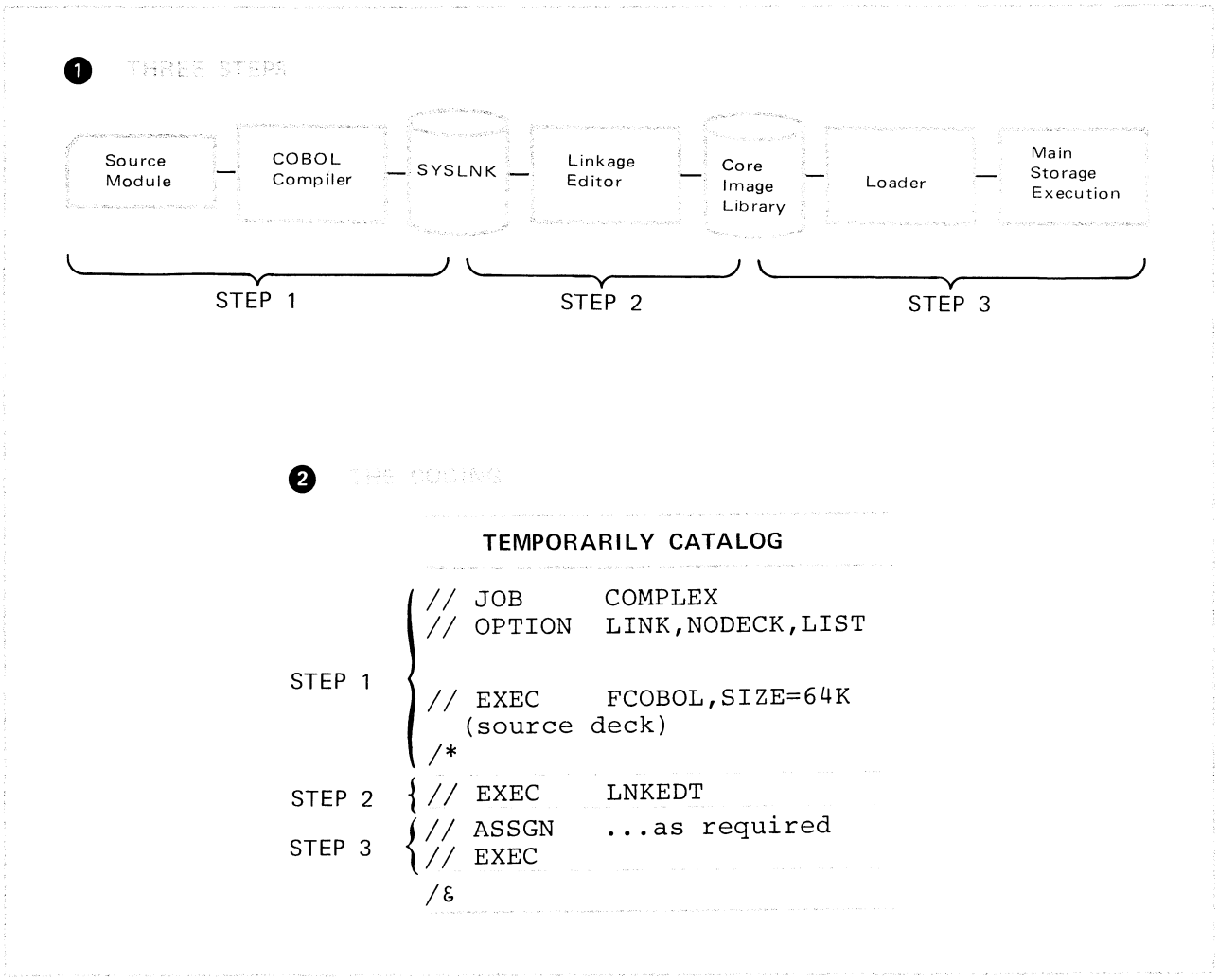


Figure 4.4 - Compile, Link-edit, and Execute

1. The three steps are the operations of compile, link-edit, and execute. Notice the presence of SYSLNK between steps 1 and 2. It is on this file that the language translators write their output for use as input by the linkage editor program.

   The loader (step 3) is a part of the supervisor that retrieves phases from the CIL and loads them into the CPU for execution.

2. This illustrates the job control statements required to perform the processing for the three steps. The most important things to see here are the functions of the LINK option and the way in which your link-edited program phase is requested for execution.

When this card is processed by job control, it causes a number of things to happen:

- A bit in the partition called the link bit is set on, and
- the SYSLNK file is opened. Job control does this in order to
  - write linkage editor control statements onto SYSLNK.

- Allow the compilers to write their object modules to SYSLNK. All the information on SYSLNK is used by the linkage editor to produce the final executable phase.

The fact that the link bit is on directs the compilers to use SYSLNK for their output and it is the setting of this bit that tells the linkage editor to catalog the phase on a *temporary* basis in a CIL.

Executing Your Program

When an object program is temporarily cataloged (as in Figure 4.4), it is specified for execution by using an EXEC statement with a blank operand. The blank operand tells the supervisor to load the last program link-edited into the Core Image Library. There is never more than one of these temporary phases in any CIL at any one time, so there is never a problem finding the right one.

If you are doing this operation you must have a CIL available to you. It is into that CIL that the linkage editor will place its executable output.

When your program completes execution (reaches end of job), the link bit is turned off and your program phase is no longer available for execution. In order to execute your program again, you must link-edit it again.

Simplified Link-Edit and Execute

This VSE/Advanced Function feature allows the user to invoke the compile, link-edit, and execute sequence with a simplified JCL set-up, as follows:

```
// JOB   IMPLICIT
// EXEC  FCOBOL,GO
   (source deck)
/*
   (card data)
/*
/&
```

The GO parameter on the EXEC statement causes the EXEC LNKEDT and EXEC of your program to be done automatically. There are several points to note about this job stream:

- The OPTION LINK statement is not needed, as GO sets the link bit on and opens SYSLNK. An OPTION statement may be required to specify other options you may want in this run, such as LISTX, SYM, or PARTDUMP.

  In addition, if you have INCLUDEs or other link-edit control statements in your job stream prior to the EXEC that invokes the compiler (FCOBOL in the example shown here), or if you are doing multiple compiles in one job, you must have an OPTION LINK in the job stream.

- If your program has its own JCL requirements, these statements must appear in the job stream *before* the EXEC of the compiler.

- No link-edit listing is produced if there are no errors in the run.

- If compiler or linkage editor errors do occur, the job is flushed to the final /& statement.

- Programs cannot be made to run in REAL mode with this set-up.

Obtaining A Storage Dump

In the event your program is canceled during execution, you are faced with determining the reason for cancellation. Often this is nearly impossible unless a dump of storage at the time of cancellation has been obtained.

A storage dump is requested (in the event of cancellation) by including in your job stream the statement:

```
// OPTION PARTDUMP
     . . or . .
// OPTION DUMP
```

The PARTDUMP parameter causes storage directly associated with the canceled program and the contents of the registers to be dumped on the system printer (SYSLST).

The DUMP parameter requests everything that PARTDUMP does and in addition causes all of the supervisor area to be dumped.

Normal debugging requires only a dump of the program area. So, generally, PARTDUMP is specified for program testing.

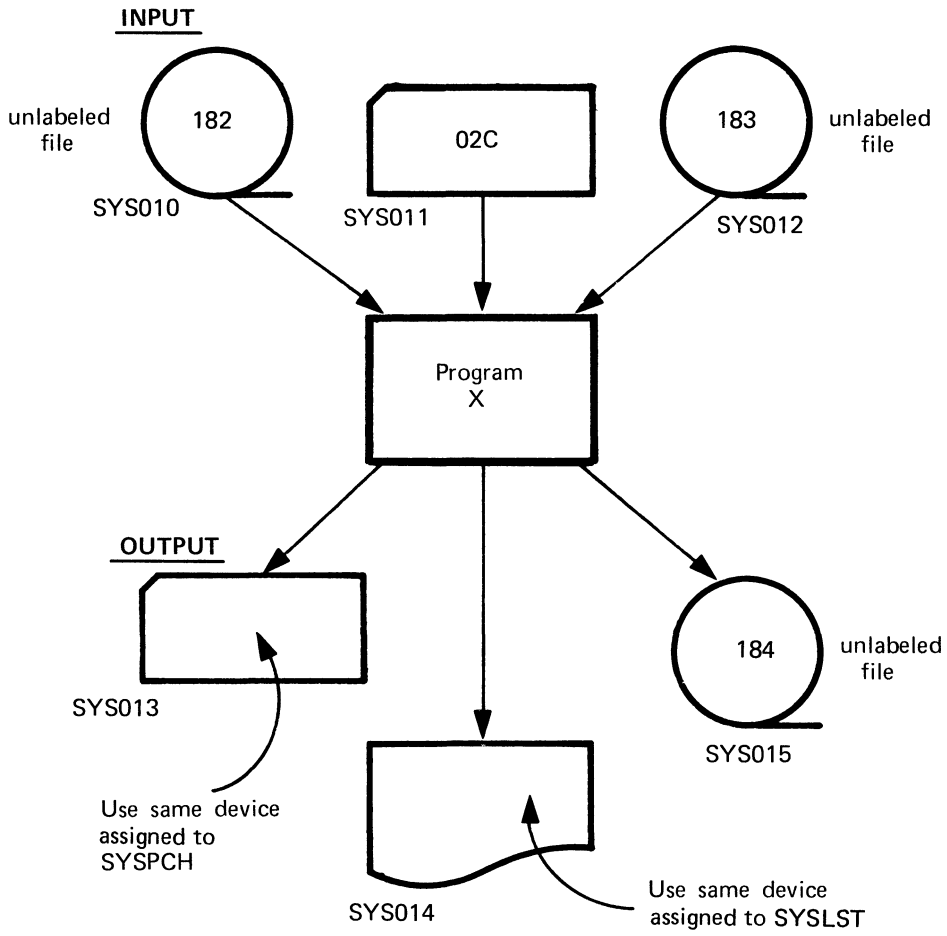
Assume SYS001, SYSLNK and other required system logical units are permanently assigned.

Prepare a single three step job to compile, link-edit, and test program X (written in COBOL). The program is to be deleted from the Core Image Library after it is tested. No object deck is desired from compilation. However, a listing of the source program should be obtained.

In the event the program is abnormally terminated, a dump of storage areas used by the program is to be printed on SYSLST.

Use the COBOL compiler named FCOBOL for the compilation, and limit it to the use of 64K of the partition. Indicate placement of source program and input data cards in your solution.

The input/output requirements of the program are as shown below.



Solution

```
// JOB      ANY
// OPTION   LINK,NODECK,LIST,PARTDUMP
// EXEC     FCOBOL,SIZE=64K
   (Source Program)
/*
// EXEC     LNKEDT
// ASSGN    SYS010,182
// ASSGN    SYS011,02C
// ASSGN    SYS012,183
// ASSGN    SYS013,SYSPCH
// ASSGN    SYS014,SYSLST
// ASSGN    SYS015,184
// EXEC
   (Input Data Cards)
/*
/&
```

Alternate Solution Using the GO Operand

```
// JOB     ANY
// OPTION    NODECK,LIST,PARTDUMP
// ASSGN     SYS010,182
// ASSGN     SYS011,02C
// ASSGN     SYS012,183
// ASSGN     SYS013,SYSPCH
// ASSGN     SYS014,SYSLST
// ASSGN     SYS015,184
// EXEC      FCOBOL,GO
   (Source Program)
/*
   (Input Data Cards)
/*
/&
```

Computer Exercise

Do Computer Exercise 3 if you did not do it when it was assigned in Unit 3. When you have submitted it for a run, go on to the next Assignment. If you have already completed this Exercise, you may go on to Assignment 2 immediately.

Assignment 2 - Building Program Phases - I

Permanent Cataloging

Figure 4.5 shows the JCL required to cause the linkage editor to catalog a member *permanently* into a CIL and then to execute that member.

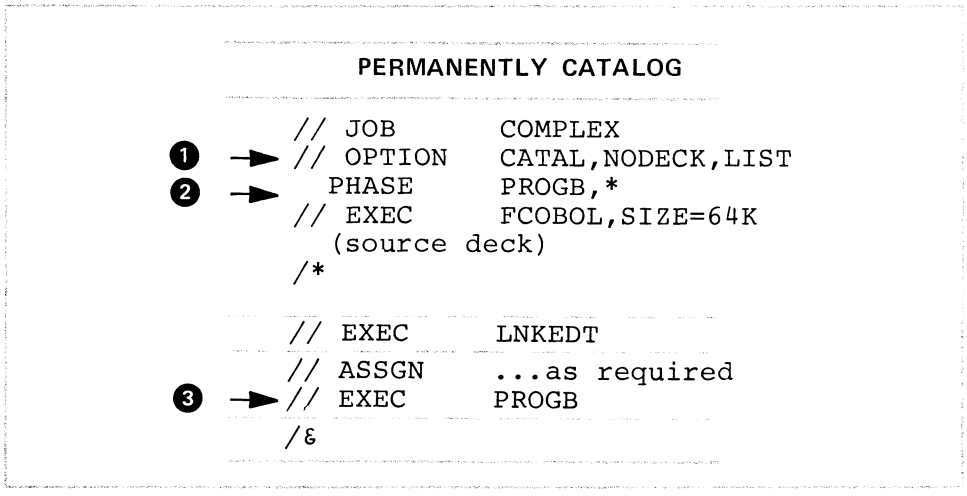


Figure 4.5 - Permanent Cataloging

The logical flow of operations is the same as in Figure 4.4. The difference is that in this case the program is named and made a permanent part of a Core Image Library. There are three differences in the job control statements to catalog a program phase permanently (Figure 4.5) and temporarily (Figure 4.4).

1. The CATAL option. This does the same four things as the LINK option, and additionally:
   - causes job control to turn on a CATAL bit in the partition. This bit is the signal to the link-edit program to perform a permanent catalog operation and to
   - create a permanent directory entry in a Core Image Library.
2. The PHASE card is a linkage editor control statement that names a program being cataloged into a Core Image Library, and supplies information about the program's loading characteristics.
3. The EXEC statement can now be used to invoke a program by the name that it has been cataloged under--PROGB in this case. When end of job is reached, the CATAL and LINK bits are turned off. Because your program phase is permanently cataloged, it can be retrieved for execution under its name (PROGB in this example) whenever it is needed.

The CIL Directory

The linkage editor creates a permanent entry for a program in a Core Image Library directory. This directory entry is a pointer to the named program in the library itself. Part of the directory information is supplied by the user, specifically the shaded areas of Figure 4.6.

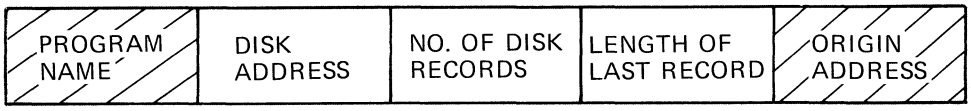


Figure 4.6 - A Directory Entry

The meaning of each field is as follows:

Program Name: The 1-8 character alphameric name supplied on the PHASE card.

Disk Address: The starting location on disk of the named program.

Number of Disk Records: The number of disk blocks required to contain the named program.

Length of Last Record: The number of bytes in the last disk block occupied by the named program.

Origin Address: Supplied by the PHASE card, this is the first available address in the partition where the phase will be loaded. This address is subject to modification by the relocating loader when the program is loaded for execution.

As long as this entry exists in a Core Image Library directory you can access the program with an EXEC card containing its name. Information about the contents of directory entries can be secured with certain librarian programs, to be covered in a later Unit.

Link-edit Control Statements

There are four linkage editor control statements available to you for ordering link-edit input and supplying required information. They are listed in Figure 4.7.

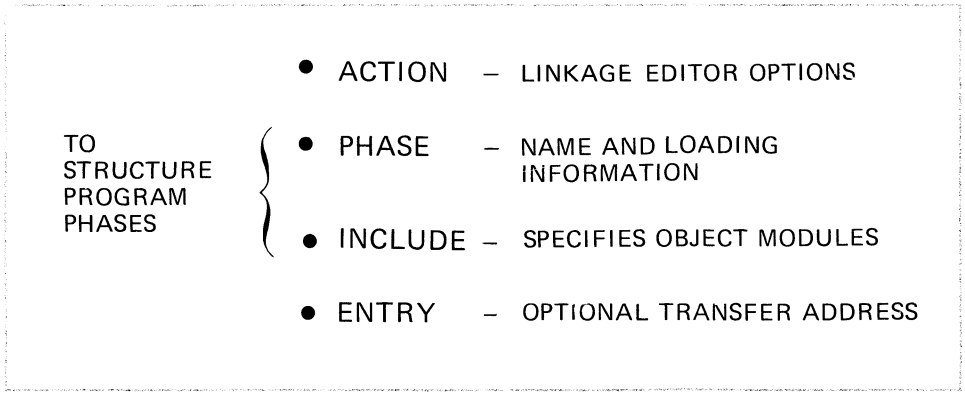


Figure 4.7 - The Four Control Statements

ACTION:

If used, this optional statement must be the first control statement presented to the linkage editor. It specifies a variety of options to control certain link-edit operations.

PHASE:

This statement always indicates the beginning of a program phase. The name in the PHASE card identifies the program in the CIL. The second operand specifies the load address where in processor storage it is to be loaded at execution time.

INCLUDE:

This control statement identifies either an object deck or a relocatable object module that is to be made part of a program phase.

ENTRY:

This control statement specifies a symbolic address where your program phase should begin execution when control is passed to it by the VSE supervisor. ENTRY is an optional statement and, if not used, control is given to the program represented by the first module (in INCLUDE statement sequence) encountered by the linkage editor in this run.

PHASE and INCLUDE

PHASE and INCLUDE are by far the most important of these four statements, for it is with them that you structure program phases for inclusion in a Core Image Library. The PHASE statement names your executable program and gives it a load address, while the INCLUDE specifies object modules that are to be made part of your program phase.

PHASE Statement Parameters

name

A one to eight character alphameric name for the program phase as it will be cataloged in a CIL. This name may not be ALL, S, or ROOT as these designators have other specific meanings to the linkage editor.

origin

The origin, or load address, specifies where in processor storage the phase will be loaded for execution. Remember, the linkage editor is processing your PHASE statement while running in one of the system's partitions. By coding the origin parameter in its most common form, as * or S, you cause the linkage editor to assign a load address to your phase *relative to the beginning of that partition.*

If you are processing multiple PHASE statements in the same run, code the origin of the first phase as * or S, and code subsequent ones as *. The * tells the linkage editor to load phases other than the first at locations following the previous phases.

Other specifications for origin may be found in the *System Control Statements* manual under the description of the PHASE statement.

The next three parameters are optional.

noauto

Suppresses the Automatic Library Lookup (Autolink) feature for this phase only. Autolink is discussed in the next Assignment.

sva

The presence of this parameter indicates that the phase is eligible for inclusion in the SVA.

pbdy

This parameter tells the linkage editor to ensure that the phase origins at the beginning of a page (on a page boundary).

modulename

This parameter may be left blank or may contain the one to eight alphameric character name of a Relocatable Library module. If modulename is left blank, the object module to be INCLUDEd is assumed (by job control when it processes the INCLUDE statement) to be read from the SYSIPT device.

(namelist)

This optional parameter allows the user to specify a series of *control section* (CSECT) names from which the phase will be constructed. A control section is a part of a program specified by the programmer to be a relocatable unit. The *System Control Statements* manual has full details on the use of namelist.

Figure 4.8 is an illustration of how the INCLUDE is used to process object modules that have been output by the previous action of a compiler.

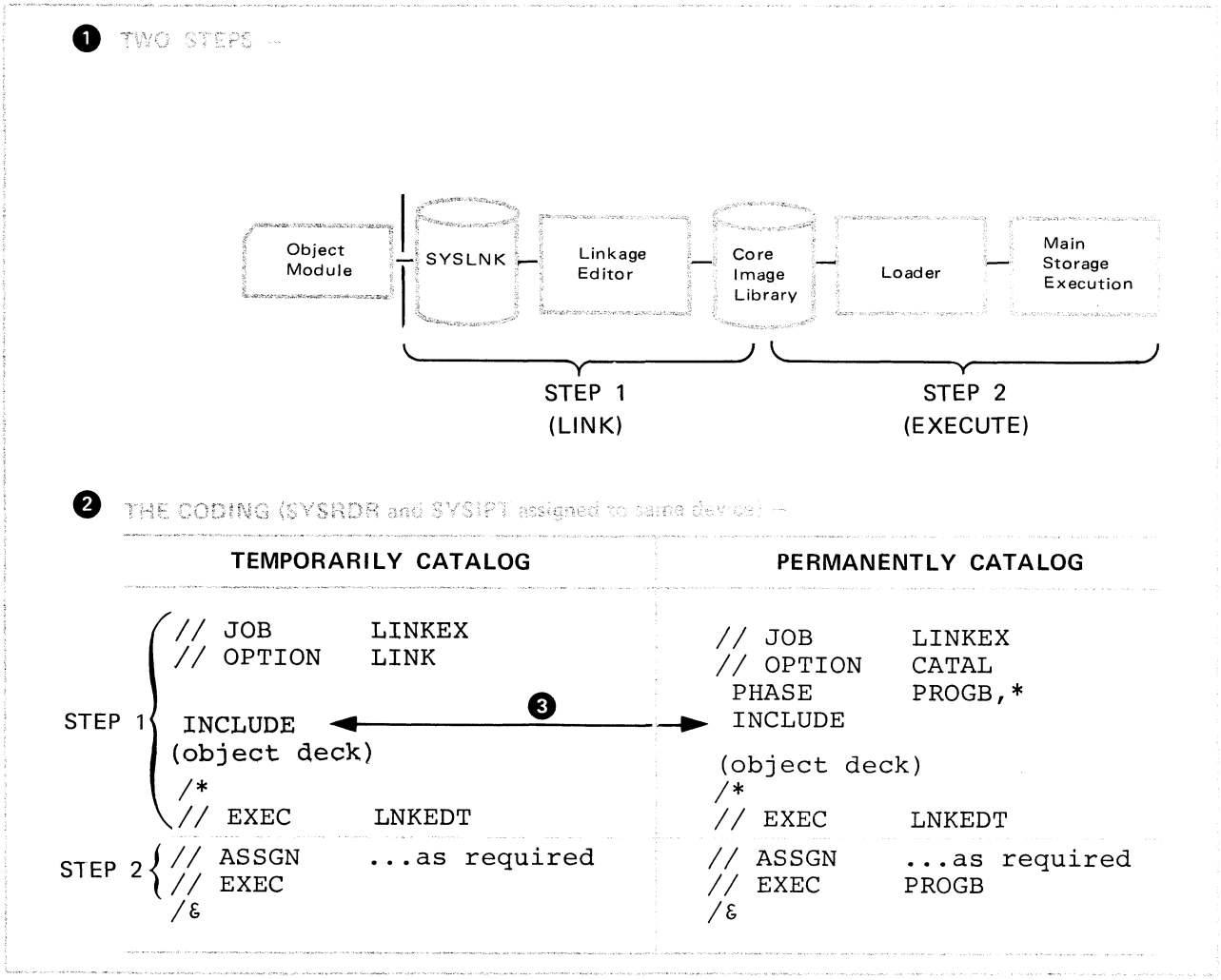


Figure 4.8 - Link-Edit and Execute an Object Deck

❶ The operation is considered to be in two steps because an object deck already exists from a prior compilation.

Notice the function of SYSLNK. It is the primary input file to the linkage editor. Job control *copies* the object module to SYSLNK from where it is processed by the linkage editor to become a phase in a CIL.

❷ The presence of the INCLUDE card with a blank operand causes job control to copy the object deck from SYSIPT to SYSLNK. Since SYSRDR and SYSIPT are assigned to the same device in this example, there is no problem making the object deck part of the job stream. If SYSIPT were assigned to another device, the object deck would have to be placed there in order for job control to find it.

The /* following the object deck signals end of input to job control, and tells job control to stop reading from SYSIPT and to resume reading from SYSRDR.

❸ The INCLUDE card must *precede* the object deck. Notice its relation to the PHASE card in the OPTION CATAL job. The PHASE card names the program card being cataloged in a CIL, and precedes the INCLUDE card. INCLUDE itself identifies the object deck as the next thing in the job stream.

In the example just discussed, the input object module was in card deck form, but it could easily have been residing in a Relocatable Library. A RL is used for disk storage of object modules, and is the *secondary input source* to the linkage editor. If a RL is available to your partition, it will also be used as an input source to the linkage editor.

A module is specified as residing in a RL by coding its name as an operand on the INCLUDE statement. See Figure 4.9. Here, MOD1 and MOD2 have been previously cataloged into a RL and they are now being made part of the input to the linkage editor.

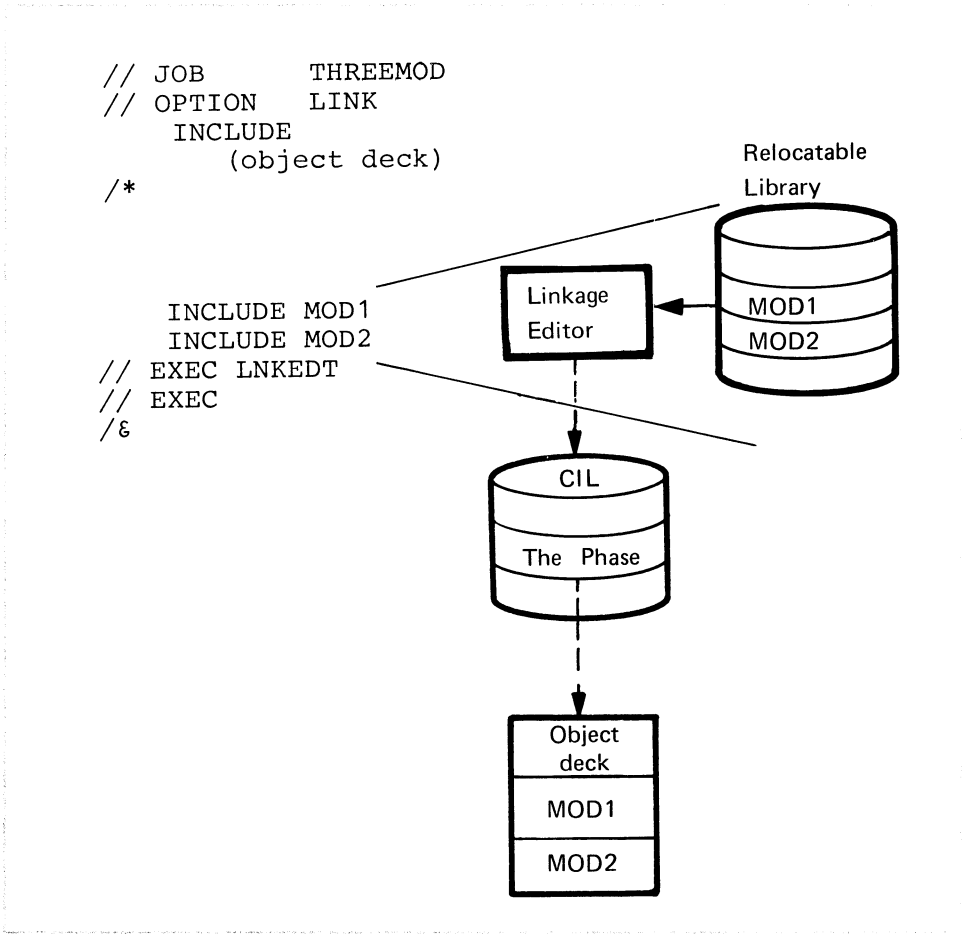


Figure 4.9 - Using the Relocatable Library

When job control reads the statements INCLUDE MOD1 and INCLUDE MOD2, it copies them to SYSLNK. When the linkage editor is later processing the SYSLNK file, it retrieves the actual code for MOD1 and MOD2 from a RL.

Another thing to notice about this illustration is that there is only *one* phase in the CIL, made up of *three* object modules. This is the way phases in the CIL are normally constructed.

If OPTION CATAL had been used in Figure 4.9, a PHASE statement would be required prior to the INCLUDEs and a permanent entry would be made into the CIL.

Types of Program Phases

Under prior DOS systems the linkage editor was very limited in the way it constructed phases, and the operating system was more restricted in how it loaded phases for execution. Although most of the program phases you will normally deal with are of the same type, relocatable, it is important for you to be familiar with the other possible types. This is especially true if you are involved in maintenance operations with programs written for earlier versions of DOS.

In addition, you should be aware of the existence of *self-relocating* programs. A self-relocating program is one that establishes its own starting address at execution time, and does not need the relocating loader to adjust its internal address references. A commonly used example of this type of program is the Sort/Merge program product.

In the document:

Introduction to the VSE System

Under the heading:

"Using the Libraries"

Read:

The section "Link-Editing a Program for Execution"

In the document:

VSE/Advanced Functions System Management Guide

Under the heading:

"Structure of a Program"

Read:

The section "Program Phases"

Before going on to the next Assignment, do the review Exercise and Computer Exercise 4 that follow. Do not proceed to Assignment 3 until you have completed the review Exercise and submitted the Computer Exercise for a run.

Exercise 4.2

1. Complete the following sentences:

   a. The job control program reads JOB, OPTION, EXEC, and /& statements from _____.

   b. Job control reads PHASE and INCLUDE linkage editor control statements from _____.

   c. Job control reads object decks and /* statements from _____.

   d. Job control writes PHASE statements and object decks to _____, which is an input file to the _____ program.

   e. The linkage editor program writes to and reads from a work file on _____.

   f. Phases produced by the linkage editor are written to a_____.

   g. A relocatable library can be used by the linkage editor, but only as an (input, output) file.

   h. OPTION CATAL or OPTION LINK both cause job control to _____ the SYSLNK file.

2. Identify the errors, if any, in the following sets of JCL.

   a.
```
// JOB TEMPCAT
// OPTION LINK
   INCLUDE
  (OBJECT DECK)
/*
// EXEC LNKEDT
// EXEC
/&
```

   b.
```
// JOB TEMPCAT
   INCLUDE
  (OBJECT DECK)
/*
// OPTION LINK
// EXEC LNKEDT
// EXEC
/&
```

   c.
```
// JOB TEMPCAT
// OPTION LINK
// EXEC  LNKEDT
   INCLUDE
  (OBJECT DECK)
 /*
 // EXEC
 /&
```

   d.
```
// JOB TEMPCAT
// OPTION LINK
   INCLUDE  MOD1
   INCLUDE  MOD2
   INCLUDE  MOD3
// EXEC LNKEDT
// EXEC
/&
```

   e.
```
// JOB PERMCAT
// OPTION CATAL
   INCLUDE
  (OBJECT DECK)
/*
// EXEC LNKEDT
// EXEC MYPROG
/&
```

   f.
```
// JOB TEMPCAT
// OPTION LINK
// INCLUDE
  (OBJECT DECK)
 /*
 // EXEC LNKEDT
 // EXEC
 /&
```

To help you answer Questions 3 - 5, do the following reading.

In the document:

VSE/Advanced Functions System Control Statements

Under the heading:

"Linkage Editor Control Statements"

Read:

The ACTION card, and the ENTRY card.

3. The following job stream, partially on SYSRDR and partially on SYSIPT, is presented as input to a LNKEDT run. Show what the contents of the SYSLNK file will be when the LNKEDT program begins execution:

SYSRDR

```
// JOB ANY
// OPTION CATAL
   ACTION CANCEL
   PHASE XYZ,*
   INCLUDE
   INCLUDE MOD1
// EXEC LNKEDT
/&
```

SYSIPT

```
(OBJECT DECK)
 /*
```

SYSLNK

1. _____
2. _____
3. _____
4. _____
5. _____

4. Sometimes jobs run correctly even though something is "not quite right" with the job stream. Try to discover what is "wrong" with the following two sets of JCL, and explain why they will work anyway.

A.
```
// JOB ONE
// OPTION LINK
   PHASE ABC,*
   INCLUDE MODX
// EXEC LNKEDT
   (PROPER ASSGNs
    FOR ABC)
// EXEC
/&
```

B.
```
// JOB TWO
// OPTION CATAL
   PHASE ABC,*
   INCLUDE MODX
// EXEC LNKEDT
   (PROPER ASSGNs
    FOR ABC)
// EXEC
/&
```

5. Identify the errors in the following linkage editor control statements.

   a. ACTION AUTO

   b. PHASE ROOT,*

   c. PHASE ABC,*

   d. PHASE XYZ, S

   e. INCLUDE MYMODULES

Solution

1. a. SYSRDR
   b. SYSRDR
   c. SYSIPT
   d. SYSLNK, linkage editor
   e. SYS001
   f. CIL
   g. input
   h. open

2. a. Correct.
   b. The OPTION card is out of place. It must precede the INCLUDE.
   c. The EXEC LNKEDT statement is in the wrong place. It must follow the /* card that follows the object deck.
   d. Correct. MOD1, MOD2, and MOD3 are modules in the RL.
   e. A PHASE card must be present for an OPTION CATAL run to name the program phase.
   f. The INCLUDE card never has slashes in columns 1 and 2 (or anywhere).

3. 1. ACTION CANCEL
   2. PHASE XYZ,*
   3. (object deck)
   4. INCLUDE MOD1
   5. ENTRY

   Job control writes an ENTRY onto SYSLNK even if you have not supplied one with your link-edit control statements. The ENTRY serves as a required end-of-file marker for the linkage editor (so this program knows when to stop looking for input data).

   The first program to get control at execution time is the physically first module in the job stream, which in this case is the object deck.

4. a. The PHASE card is not required in an OPTION LINK run. It will be edited by job control, but will not affect the outcome of the link-edit operation.
   b. You should invoke the program with an EXEC ABC card (using the program name) since it has just been permanently cataloged. The EXEC with a blank operand will work as shown.

5. a. Invalid parameter.
   b. ROOT illegal as a phase name.
   c. No error.
   d. No blanks allowed in the operand field.
   e. Operand is too long.

Do Computer Exercise 4 at this time. It is not necessary to get the results of the Computer Exercise before continuing as long as you have submitted the job.

This Assignment uses a short case study to acquaint you with two somewhat more complex, but nevertheless common, constructions of program phases using the Relocatable Library. In addition, you will see when and how private Core Image libraries are used, how the autolink feature works, and what type of diagnostic information you can expect from the linkage editor.

The Assignment concludes with a discussion of two job control statements that permit you to pass data to your executing program: the UPSI and the DATE statements.

You have been assigned to finish the testing of a complex billing program. It consists of four modules that currently reside in the RL, as shown in Figure 4.10. Some preliminary testing has been done, and the modules ran all right when tested individually. You have to combine them into a single executable phase, and test the whole program.

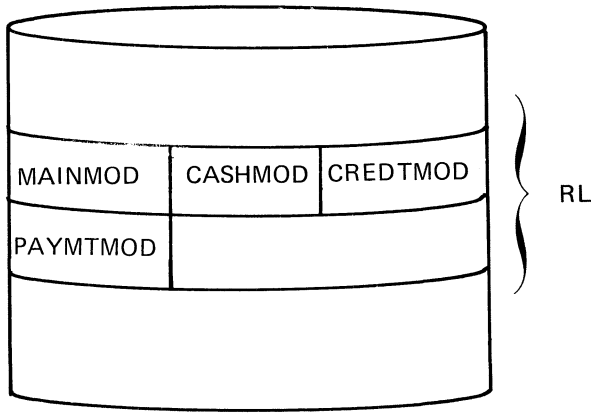


Figure 4.10 - The Object Code

A variety of transaction types have to be tested, so the executable phase you construct has to be able to be invoked again and again from a Core Image Library, without needing the link-edit operation repeatedly performed. In addition, you are required to insure that the composite phase will begin its execution with the instruction at MAININ, a CSECT name within MAINMOD, regardless of the sequence of the INCLUDE cards in your job stream.

Part 1

Code the JCL required to carry out the link-edit and execute run. Catalog the program phase you are constructing under the name 456TESTA. Assume that the necessary devices for the linkage editor and for 456TESTA are standardly assigned. When you have completed the coding, turn to the solution that follows.

Part 1 - Solution

```
        // JOB ONE
1       // OPTION CATAL
2          PHASE 456TESTA,*
           INCLUDE MAINMOD
           INCLUDE CASHMOD
3          INCLUDE CREDTMOD
           INCLUDE PAYMTMOD
4          ENTRY MAININ
5       // EXEC LNKEDT
        // EXEC 456TESTA
        /&
```

1. The CATAL option is required to permanently catalog the phase into a CIL. This way it can be retrieved again and again with the name 456TESTA.
2. Required to name the phase and assign an origin address.
3. Any order is permissable for the INCLUDES.
4. This creates the proper entry to the executable code. ENTRY must follow the last INCLUDE.
5. All assignments for LNKEDT and 456TESTA are assumed permanent, so no ASSGN cards are needed in this JCL.

Part 2

Because of the testing done, bugs were found in the CASHMOD module. You turn that problem over to the CASHMOD developer, who fixes the errors and hands you a new CASHMOD object deck.

Construct the job stream to recatalog 456TESTA using the new CASHMOD module. When you have finished, check your answer with the solution that follows.

Part 2 - Solution

```
                    // JOB TWO
                    // OPTION CATAL
                       PHASE 456TESTA,*
                       INCLUDE MAINMOD
                       INCLUDE
1                        (CASHMOD object deck)
                    /*
                       INCLUDE CREDTMOD
                       INCLUDE PAYMTMOD
                       ENTRY   MAININ
                    // EXEC LNKEDT
                    // EXEC 456TESTA
                    /&
```

1. This can go anywhere between PHASE and ENTRY. Remember to have a /* behind the deck to signal job control that there is no more SYSIPT data.

   You now have an obsolete copy of the CASHMOD module residing in the RL. You will want to recatalog the new CASHMOD object deck in the RL to replace the old version. This function is accomplished with one of the librarian programs that will be discussed in Unit 7.

Conclusions

This time testing finds no errors, and you have a good copy of 456TESTA in the Core Image Library. But what happened to the first version of 456TESTA? Take a look Figure 4.11.

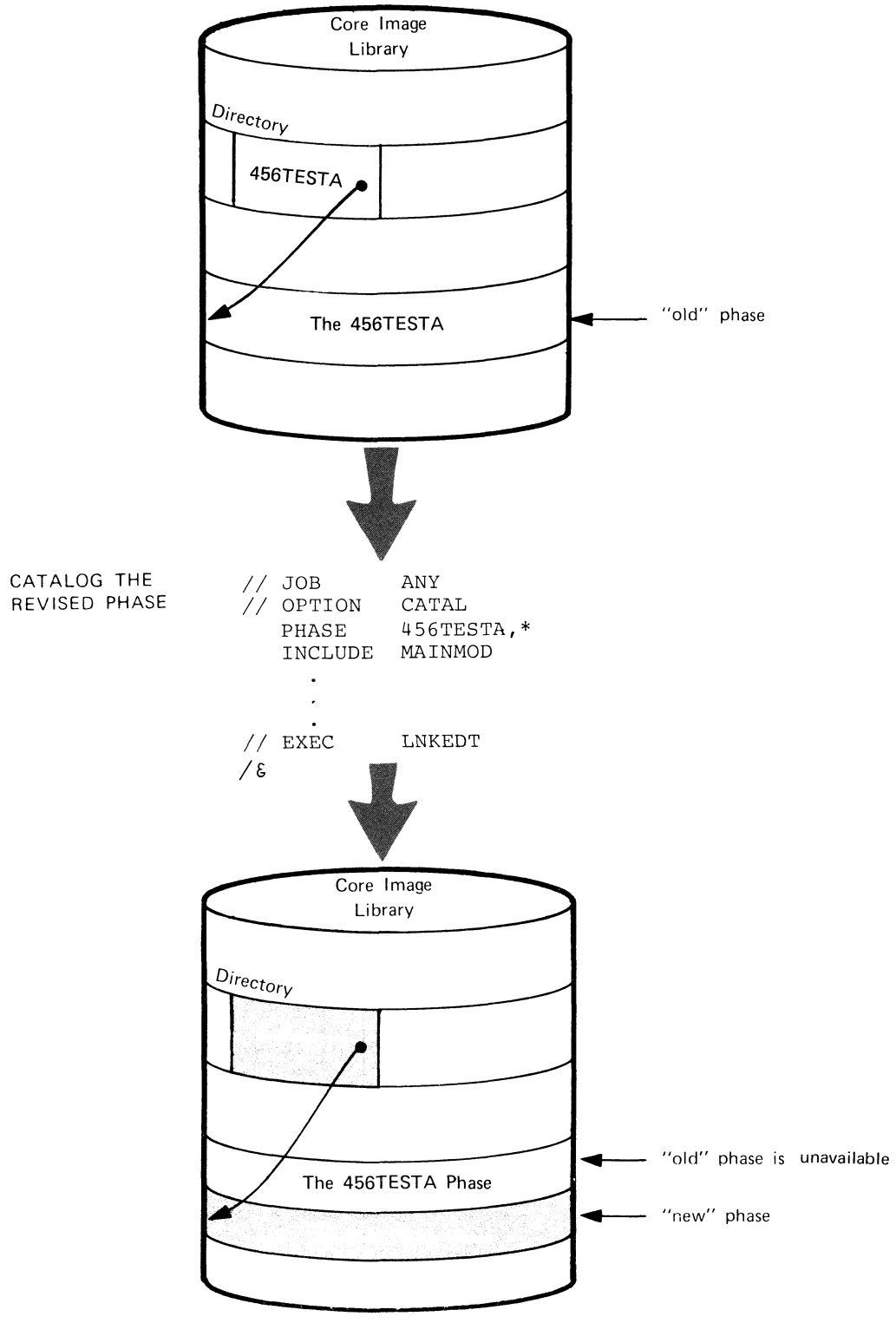


Figure 4.11 - Replacing a Phase in the CIL

Notice what happens when the revised phase is cataloged. The directory entry for 456TESTA is updated to point to where the new version resides in the CIL. The old version is unavailable,

since no directory entry points to it any longer. The space it occupies is also unavailable, and remains so until a library maintenance program (discussed in a later Unit) is run to reclaim the space.

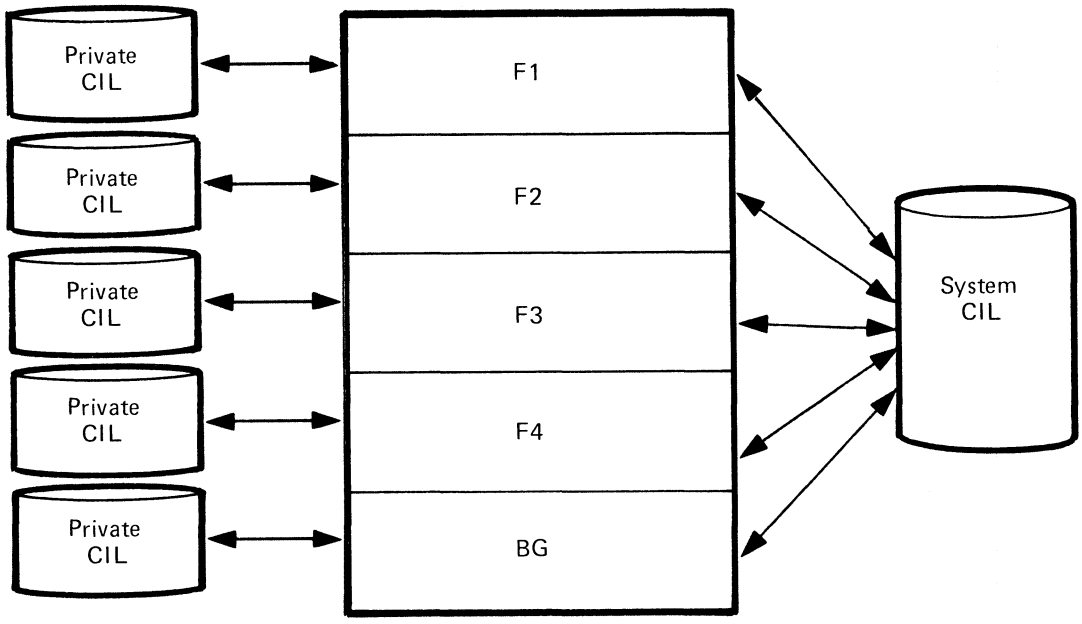


Figure 4.12 - Private Core Image Libraries

Figure 4.12 illustrates cataloging (link-editing) into and executing phases from both system and private Core Image libraries.

Any partition can link-edit into the system CIL or into any private CIL assigned to it. When the linkage editor is writing a phase into a CIL from a given partition, all other partitions are prohibited from updating that CIL. This prevents interference from another partition until the processing is done. When the first partition has completed its processing, the CIL is unlocked and cataloging may proceed from another partition.

Note that the system CIL is still available to each of the partitions even though they have private CILs assigned to them. A program may be loaded for execution from either a private or the system CIL. When you supply a program name in an EXEC statement, the supervisor searches in a CIL for that program in a fixed sequence that depends on the program's name.

Phase Search Order

Phase names may or may not be prefixed with the dollar-sign character. The presence or absence of this character will determine the order in which the VSE Supervisor searches the available libraries. Figure 4.13 shows the search order for non-$ phase names, while Figure 4.14 shows it for $ phase names.

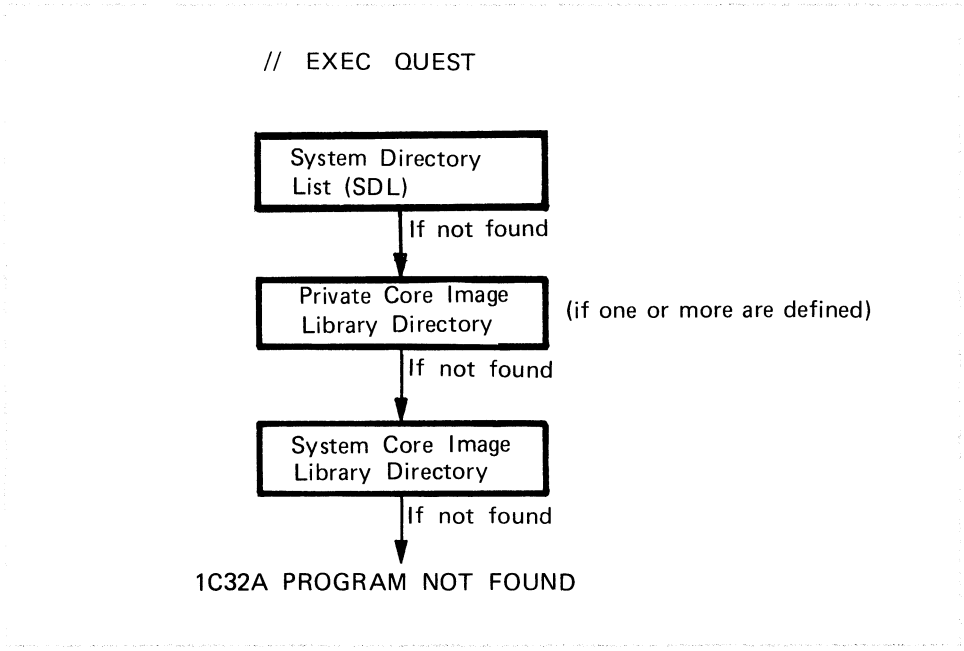


Figure 4.13 - Search Order, Non-$ Phases

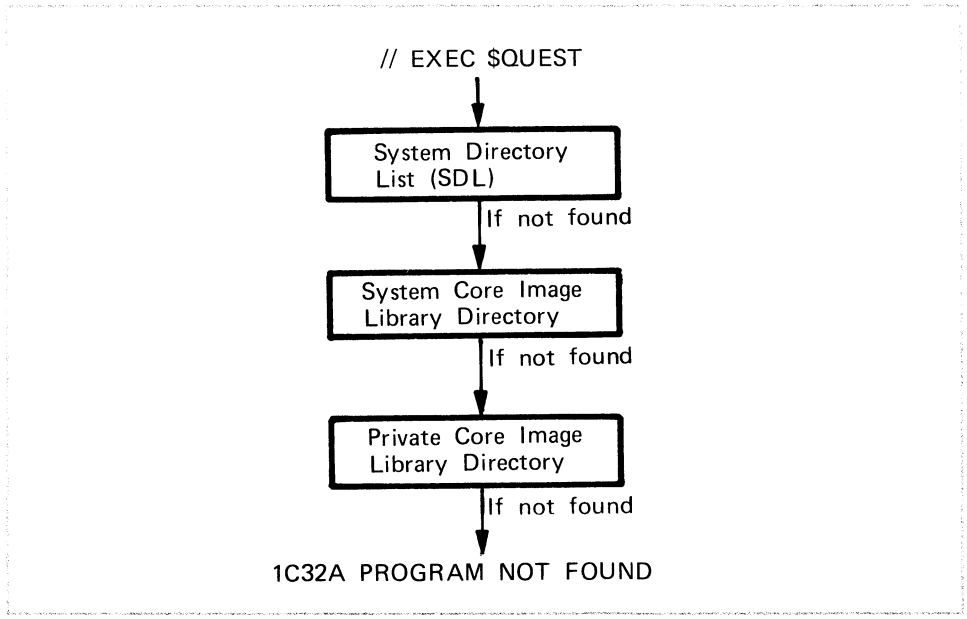


Figure 4.14 - Search Order, $ Phases

In either case, as soon as the phase is found it is loaded for execution. This means, for example, that if you had a phase name QUEST existing in both an available private CIL and in the system CIL, the version from the private CIL would be the one executed. If the phase name was $QUEST in both cases, the version in the system CIL would be retrieved.

The point of this convention is to reduce the time it takes to load a program. Many of the IBM-supplied system programs have phase names that begin with a dollar-sign. These names will generally be included in the system directory list in the SVA. The system loader will search here first and then in the CIL directory for dollar-sign phase names.

If a phase you are developing will normally reside in a private library, then no dollar-sign should prefix its name. On the other hand, IBM-supplied modules frequently have a dollar-sign as the first character of their names.

Assigning Private CILs

To access a private CIL from a given partition, you may assign the specific symbolic unit name SYSCLB to the device(s) containing the library or supply a // LIBDEF statement for it. The LIBDEF statement will be explained in Unit 7. Once job control makes the connection between the private CIL and your partition you can execute the linkage editor to store programs in that private CIL. Remember, as long as the private CIL is assigned to your partition, the linkage editor will use it for storing phases.

Autolink

Autolink is a feature of the linkage editor that causes referenced object modules to be included in a phase. It is an automatic feature that never needs to be invoked. It is always active unless you specifically inhibit its functioning.

I/O modules are the data management routines which process program requests for the input and output of records. Since most programs perform some I/O functions, these I/O modules are usually needed. Autolink searches the system Relocatable Library (and a private RL, if one is assigned) to resolve references to these and other modules.

Figure 4.15 shows part of the output from an Assembler run.

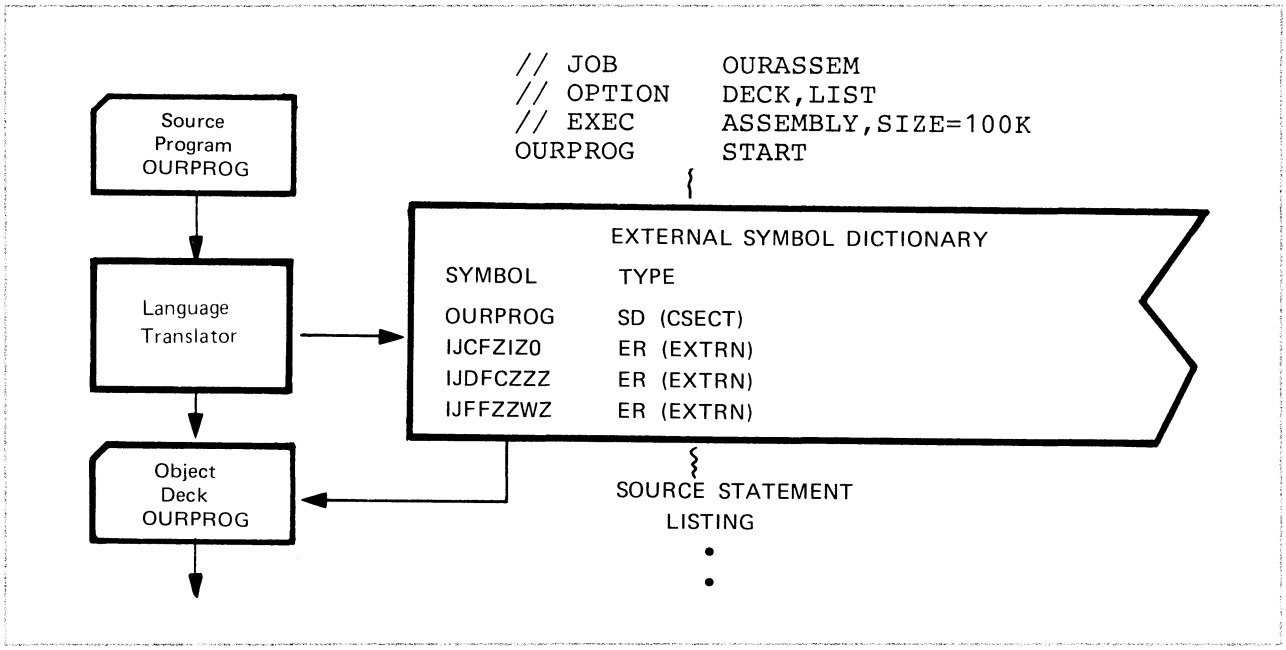


Figure 4.15 - Prior to Link-edit

ESD Contents

The external symbol dictionary (ESD) shown was produced by the Assembler, but the other compilers also produce ESDs. The ESD contains control section definitions, called CSECTs, and intermodule references (references between modules) called external references.

OURPROG is an SD type entry. SD stands for section definition. OURPROG is the CSECT name.

The next entries are the names of the I/O modules required by the program. ER stands for external reference. They correspond to the rows of X's in Figure 4.2 at the beginning of this Unit. These names were generated by the Assembler from DTF macros that defined the program's I/O requirements. For example, look at the name IJCFZIZ0. The Assembler composed this name as well as the others using information from the DTF macros coded in the program.

The arrows indicate that the ESD is included as part of the object deck. Its references will be resolved in a linkage editor run.

When the linkage editor is invoked, it will find that it cannot resolve the three external references within this module. The modules which the references name are not part of the SYSLNK input. Autolink will cause link-edit to automatically search a Relocatable Library for the three missing modules. See Figure 4.16.

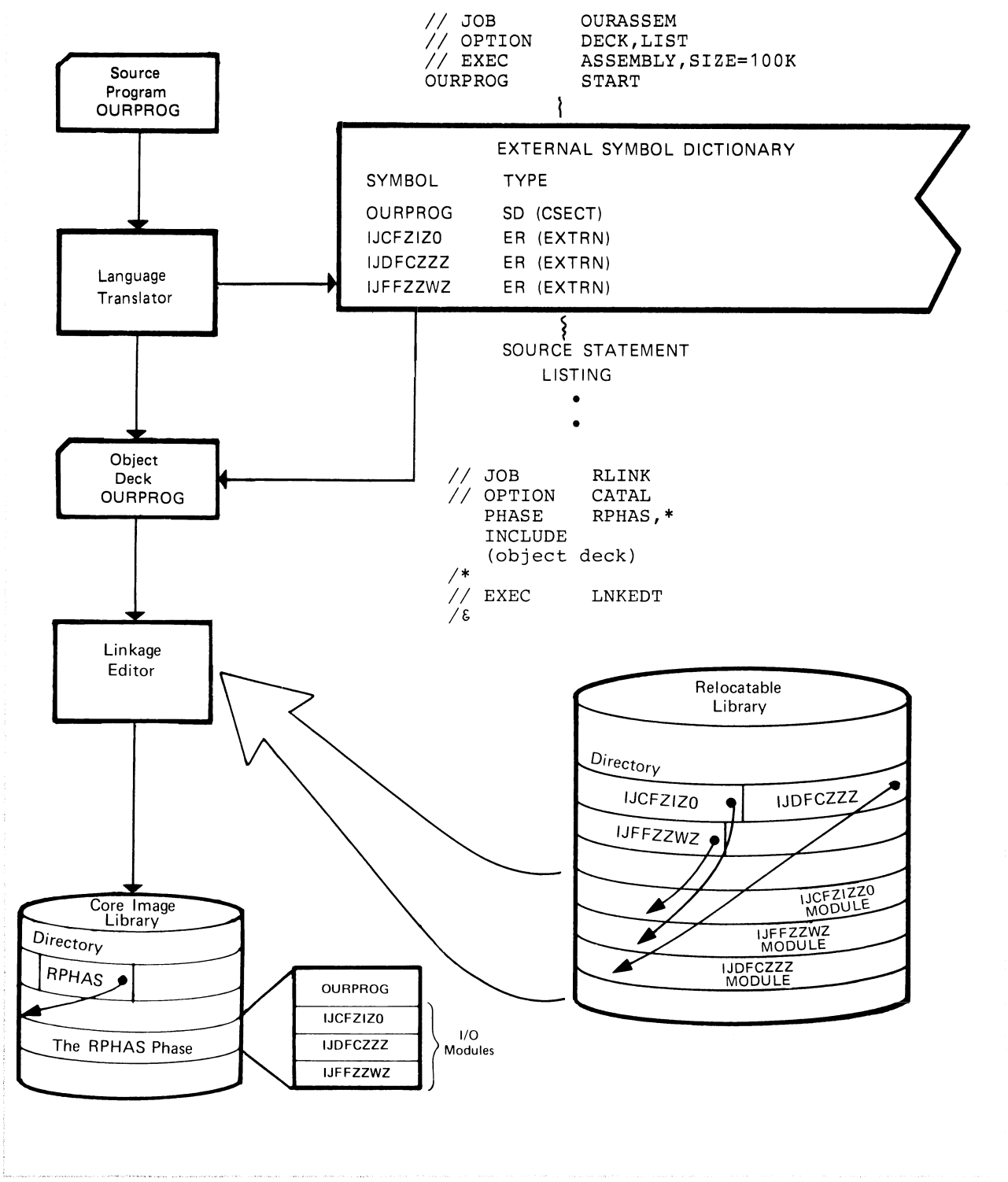


Figure 4.16 - Link-edit With Autolink

If any of the modules cannot be found in the RL, references to them will be left as "unresolved" and the linkage editor will generate an error message on SYSLST.

You can suppress autolink for the entire link-edit run by coding the NOAUTO parameter on the ACTION card, or you can suppress autolink for a single phase by coding NOAUTO on the PHASE card.

You would suppress autolink in order to prevent the automatic inclusion of modules from taking place. One case in which you might do this is when you are testing revised versions of modules that are already part of your system. You are constructing an executable phase from test modules (supplied in object deck format) and do not want to accidentally include any of the unrevised modules from the Relocatable Library.

The linkage editor produces a map of its activity on SYSLST. Figure 4.17 shows a sample of the input diagnostics while Figure 4.18 shows the output diagnostics and map. If SYSLST is not assigned, a map is not printed and any error messages are sent to SYSLOG.

```
JOB  EX4    07/29/80    5746-XE8 REL 1.2    LINKAGE EDITOR DIAGNOSTIC OF INPUT

ACTION TAKEN  MAP (1)

FOLLOWING LIBRARIES ARE ACTIVE FOR THIS RUN
LIBR.TYPE    SEQ.NO  FILENAME    VOLID (2)
 TARGET CIL      0     USRCL2   DOSRES
 SEARCH RLB      1     USRRL1   DOSRES
 SEARCH RLB      2     USRRL2   DOSRES
 SEARCH RLB      3     USRRL3   SYSWK4
 SEARCH RLB      4     PRDRLA   DOSRES
 SEARCH RLB      5     PRDRLB   SYSWK4
 SEARCH RLB      6     PRDRLC   DOSRES
 SEARCH RLB      7     PRDRLD   SYSWK4

LIST     PHASE ISPEXP1,* (3)
** MODULE IJCFZIZO V.35  M.1     AUTOLNKD FROM LIB.NO. 4
** MODULE IJDFCZZZ V.35  M.0     AUTOLNKD FROM LIB.NO. 2  (4)
** MODULE IJFFZZWZ V.35  M.1     AUTOLNKD FROM LIB.NO. 4
LIST   ENTRY
```

Figure 4.17 - Linkage Editor Map--Input

(1) Option MAP was specified in the ACTION statement for the linkage editor run to indicate that a map of virtual storage is to be generated.

(2) List and sequence of target library and search chain of (up to 30) libraries. This search chain of libraries will be discussed in Unit 7.

(3) A listing of control statements as submitted to linkage editor.

(4) Version and modification levels of included module (when cataloged), with cross-reference to library in list (2) above.

Input Diagnostics

The LIST entries indicate actions that took place, such as the PHASE card being processed or modules from the RL being autolinked. The ENTRY statement was either provided by job control or by the user. Since it has no operand, it is likely it was provided by job control.

Errors in the input are identified by messages prefixed with the digit 2, indicating they were generated by the linkage editor.

Output Diagnostics and Map

A variety of information is provided by the linkage editor in its output diagnostics and map. Figure 4.18 shows an error-free run where phase ISPEXP1 has been permanently cataloged. Note that the name 'PHASE***' would have been used if this were an OPTION LINK run.

```
07/29/80   PHASE    XFR-AD   LOCORE   HICORE   DSK-AD      LABEL      LOADED    REL-FR   OFFSET   INPUT
                                                                                                    RELOCATABLE
           ISPEXP1  0322FC   032078   0330CF   00112964
              (1)      (2)      (3)      (4)      (5)                      (6)
                                                              ISPEXP1    032078    032078   000000   SYSLNK
                                                              IJCFZIZ0   032CA0    032CA8   000C30   IJCFZIZ0    (7)
                                                              IJDFCZZZ   032D18    032D18   000CA0   IJDFCZZZ
                                                             *IJDFZZZZ   032D18
                                                                         032D18    032D18   000CA0   IJDFCZZZ
                                                              IJFFZZWZ   032D68    032D68   000CF0   IJFFZZWZ
                                                             *IJFFZZZZ   032D68

CONTROL SECTIONS OF ZERO LENGTH IN INPUT
```

(1)	ISPEXP1	This is the name under which the phase was permanently cataloged. If the phase had been temporarily cataloged, the phase name "PHASE***" would be printed here.
(2)	XFR-AD	The transfer address (entry point) of the phase.
(3)	LOCORE	The phase origin point.
(4)	HICORE	Highest storage address location of the phase.
(5)	DSK-AD	Disk address of the phase in the core image library.
(6)	LABEL LOADED REL–FR OFFSET INPUT	The labels of all CSECTs of a phase in ascending order, their load addresses, relocation factors, offsets relative to load address, and input points. NOTE: This is the default sequence (ascending load address). They may be printed in CSECT name sequence (alphabetically) by specifying "ACTION SMAP".
(7)	RELOCATABLE	Whether a phase is self-relocating or not relocating.

Figure 4.18 - Linkage Editor Map--Output

The points of information described in the Figure should be sufficient for your understanding of most link-edit maps. If you need to know more about linkage editor functions, see the *VSE/Advanced Functions Serviceability Aids and Debugging Procedures* manual (SC33-6099).

Communicating with Programs via JCL

The supervisor contains an area which can be used to communicate with an executing Assembler Language program. This area is called the communications region, and the supervisor maintains a communication region for each partition of your system.

There are two job control statements for changing portions of the communications region. The UPSI (User Program Switch Indicator) statement allows you to modify a byte in the communications region called the UPSI byte, while the DATE statement allows you to alter the calendar date for the duration of your job. By including either or both of the statements in your job stream, you change the UPSI and DATE information available to your program.

The UPSI Statement

The eight UPSI bit-switches are set by the job control program based on information you supply in the UPSI statement. They can then be tested by your program. The specific meaning attached to each bit-switch depends on how you have designed your program. It is your program logic that determines the significance of these switches.

Suppose a program is written so that it produces a weekly report if bit position 7 of the UPSI byte is a "1". It produces a monthly report if bit position 7 of the UPSI byte is a "0". The code below illustrates the JCL required to obtain a weekly and then a monthly report.

```
// JOB    ANY
// ASSGN  ...as needed
// UPSI   00000001
// EXEC   RPTPROG     weekly report format
// UPSI   00000000
// EXEC   RPTPROG     monthly report format
```

In order to use the UPSI properly, there are two things you must know:

1. How is a program written to test the UPSI byte?
2. How do you code the UPSI job control statement to set the UPSI byte?

The first of these questions is a matter of programming technique, and is not covered in this course. Most programming languages provide facilities to access and test the UPSI byte.

The UPSI job control statement is coded in a very straight-forward fashion, as shown in Figure 4.19, while some examples of its use are given in Figure 4.20.

Job Control Statement Format

```
//  UPSI nnnnnnnn
```

The operand

```
(nnnnnnnn)
```

consists of one to eight characters of 0,1, or X.

- **positions containing 0 are set to 0.**
- **positions containing 1 are set to 1.**
- **positions containing X are unchanged from their current setting in the Communications Region. Unspecified rightmost positions default to X.**

Figure 4.19 - The UPSI Statement

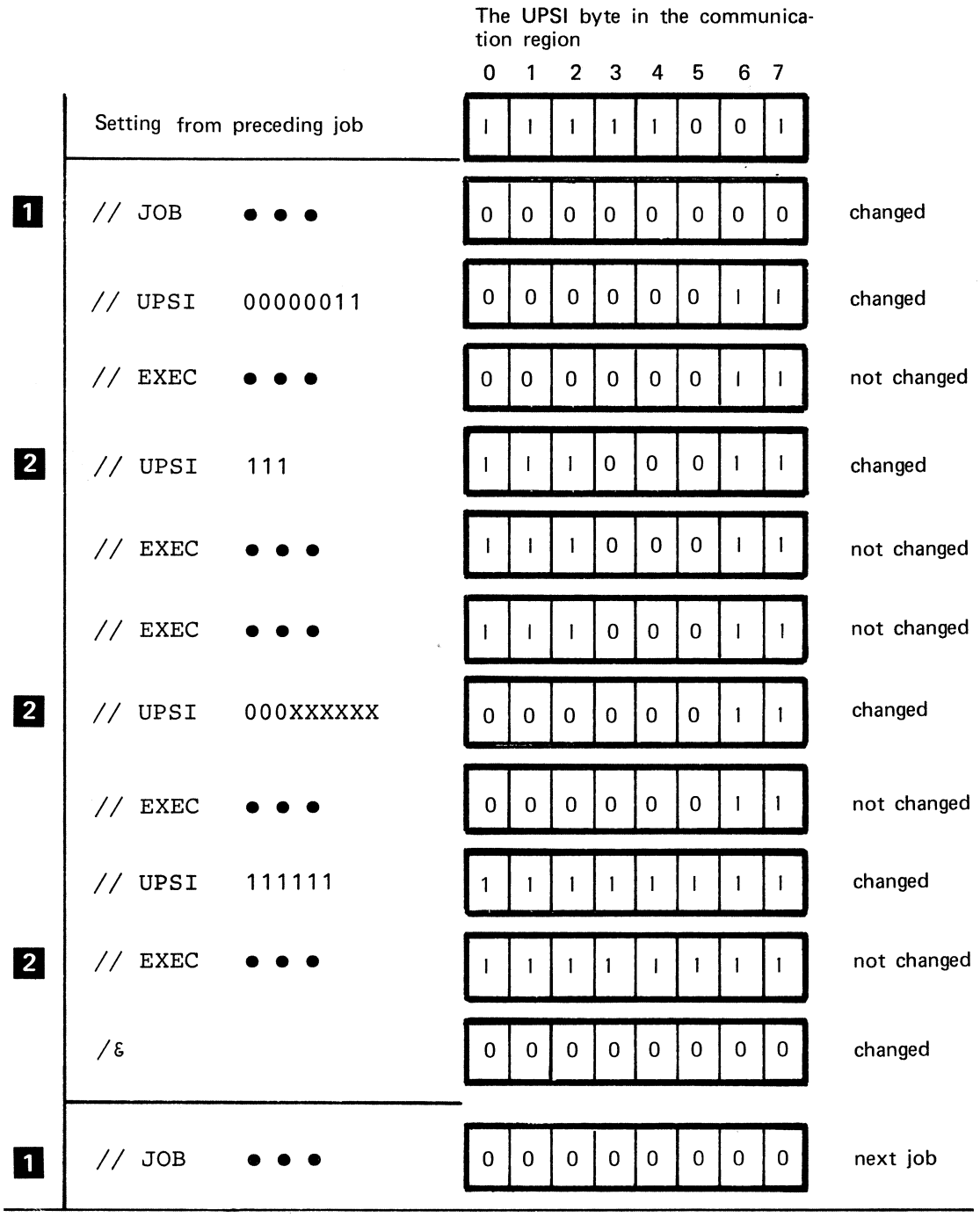


Figure 4.20 - Setting the UPSI Switches

1 Job control automatically clears the UPSI byte to zeros (at end of job time) before reading the control statements for each job.

2 The operand of the UPSI statement causes job control to modify the UPSI byte.

The DATE Statement

The supervisor maintains a communication region for each partition in the VSE system. Each of these communication regions contains both a SYSTEM DATE and a DATE field. The SYSTEM DATE is initialized when the system is IPLed and holds the date entered during IPL. The DATE field normally contains the same value as SYSTEM DATE. When your program requests date information, it gets the contents of the DATE field associated with the partition in which your program is running. The DATE job control statement allows you to temporarily respecify the DATE field of your partition's communication region.

There are two formats of the DATE statement:

```
// DATE mm/dd/yy      or
// DATE yy/mm/dd
```

where mm - month (01 to 12), dd = day (01 to 31), and yy = year (00 to 99). The format used is determined by your systems programmer. The job control program edits the statement only to see that the date is eight characters in length. You must know how the field will be used in your program before deciding to use other than the format standard for your installation.

The DATE statement may be submitted anywhere in your job stream prior to the EXEC statement. The date will be in effect only for the duration of the job. The SYSTEM DATE will replace your DATE in the partition's communication region when the /& that terminates your job is processed.

An Example

The following JCL shows the DATE card in use:

```
// JOB    FALSIFY
// ASSGN  SYS005,SYSLST
// ASSGN  SYS007,TAPE
// DATE   06/15/80
// EXEC   REPORT
/&
```

Note that the DATE card could be anywhere between the JOB card and the EXEC card. At end of job, the date will be reset to the value contained in the SYSTEM DATE field.

Reading Assignment

In the manual *VSE/Advanced Functions System Management Guide* read the section "Linking Programs".

In the manual *VSE/Advanced Functions System Control Statements* read the section "Linkage Editor".

Although you may have read portions of this material before, this reading will serve as a review.

Unit Summary

The linkage editor must process the object module output of any of the language translators to make that output ready for execution. This output of the linkage editor is a phase in a CIL.

Output from the linkage editor can be stored temporarily or permanently in either the system CIL or a private CIL. The OPTION statement is used to specify the type of cataloging function, and the availability CIL will determine where the resulting phase will go.

In order to properly construct your input to the linkage editor, four control statements are provided: ACTION, PHASE, INCLUDE, and ENTRY. Of these, PHASE and INCLUDE are the most important, as they control the structure of the executable program.

Private libraries are handled by the linkage editor much like the system libraries. A private CIL can be used as an extention or supplement to the system CIL, or merely as a place for programs to reside during their testing. A private RL can be used as an alternate source of object modules to a link-edit run.

The autolink feature causes the linkage editor to bring in any modules referenced by your program. Autolink is always active unless specifically suppressed for a particular phase or for the entire link-edit run.

Diagnostic information and a record of the linkage editor's activity during a run are provided on SYSLST and SYSLOG.

Finally, there are two job control statements you can use to communicate with your phase once it has been link-edited successfully and subsequently retrieved for execution: UPSI and DATE. Of these, the UPSI is the more powerful as you can use it to direct your program's execution time activities.

Mastery Test

1. If you were to look at a map of processor storage, you would find _____ starting at location 0.
   a. user programs
   b. I/O routines
   c. the supervisor
   d. library call modules
2. Your GET and PUT statements in a program generate _____.
   a. in-line machine code
   b. external references
   c. both of these
   d. neither of these
3. The logical unit _____ is used as a linkage editor workfile.
   a. SYSLNK
   b. SYS001
   c. SYSWRK
   d. SYS000
4. When a compiler finds the link bit on, it _____.
   a. ignores it
   b. writes its output to SYSLNK
   c. reads from the Relocatable Library
   d. turns on the CATAL bit
5. The PHASE control card is used to _____.
   a. create a CIL directory entry
   b. turn off the link bit
   c. identify modules in the RL
   d. direct compiler operations
6. The INCLUDE card always _____.
   a. names a module in the RL
   b. has a blank operand field
   c. signals the presence of an object module
   d. none of the above

7. Use of the relocating loader provides you _____.

   a. space in the RL

   b. with the facility to execute.a phase from a CIL in any partition with any load address

   c. less debugging time

   d. all of the above

8. The ENTRY statement acts to _____.

   a. specify program names in the CIL

   b. delimit the SYSLNK file

   c. control end of job processing

   d. perform optional editing functions

9. A private CIL can be used to _____.

   a. link-edit in a foreground partition

   b. link-edit in the background partition

   c. supplement the system CIL

   d. perform all of the above

10. The autolink feature is normally _____.

    a. suppressed

    b. optional

    c. active

    d. not used

11. The ESD (external symbol dictionary) _____.

    a. contains control section definitions

    b. contains external references

    c. is part of the object deck

    d. all of the above

12. The name _____ identifies a phase link-edited in an OPTION LINK run.

    a. PHASE***

    b. ***PHASE

    c. ISPEXP1

    d. NULL***

13. The job control statement

```
// UPSI   XXX
```

would change _____ bits in the UPSI byte.

a. 0

b. 3

c. 5

d. 8

14. The UPSI byte could (could not) be used to pass information from one job to another. (Circle your choice).

15. The SYSTEM DATE field is _____ by the DATE job control statement.

a. reset

b. replaced

c. unaffected

d. updated

1. c
2. c
3. b
4. b
5. a
6. c
7. b
8. b
9. d
10. c
11. d
12. a
13. a
14. Could not. The UPSI byte is reset between jobs.
15. c

If you had more than five of these questions wrong, it is suggested that you do the following review reading before going on to the next activity. If you passed this quiz, proceed to the Computer Exercise below.

In the document:

VSE/Advance Functions System Management Guide

Under the heading:

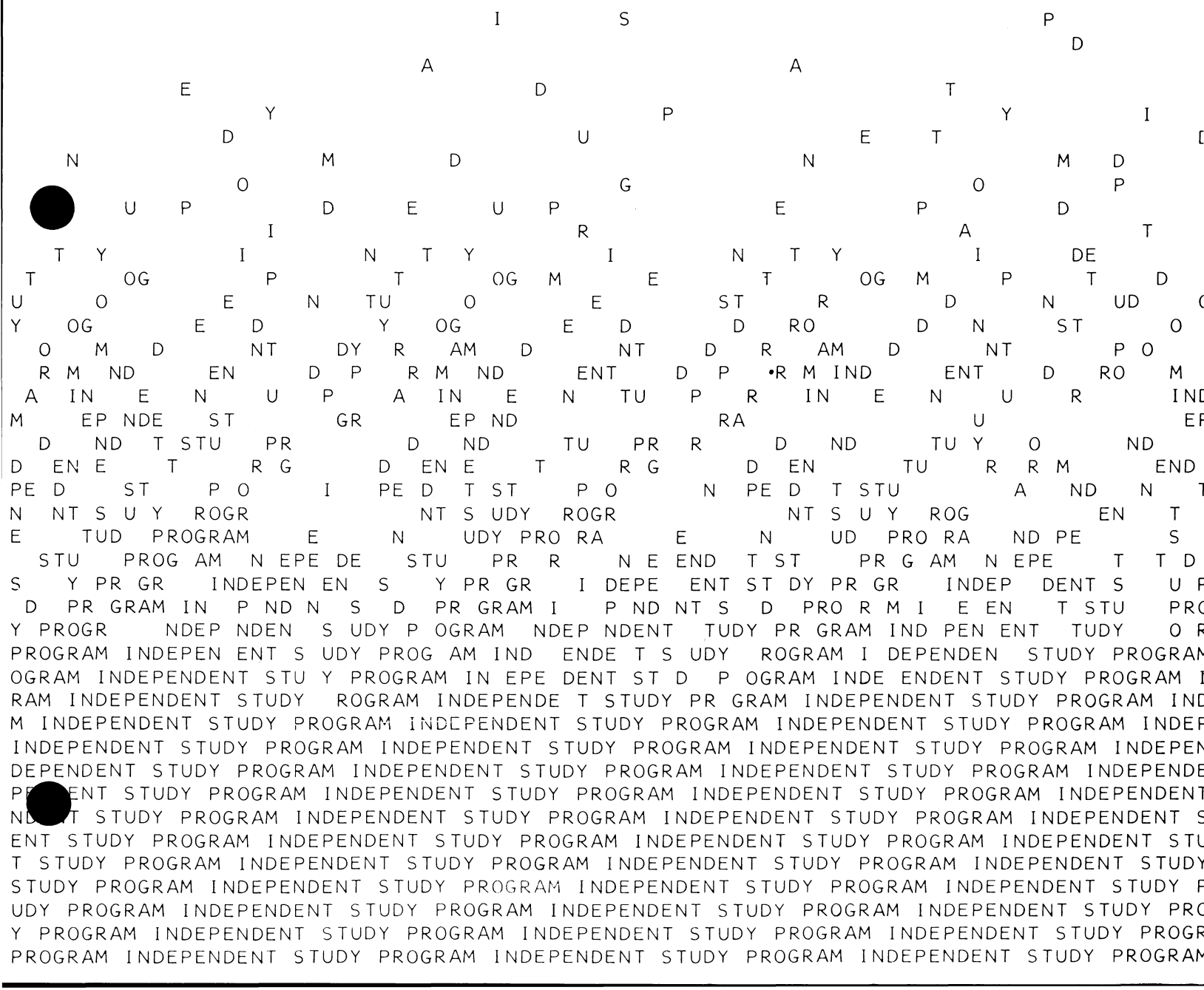
"The Three Basic Applications of the Linkage Editor"

The material up to but not including "Link-editing for Execution at any Address." In addition to this reading, you should review the material you had difficulty with in this Unit.

Prepare Computer Exercise 5 and submit it for a run before going on to the next Unit.

Unit

5



Data management is the control, storage, and retrieval of data to be processed by a computer. Under VSE, data management functions are accomplished by a combination of system routines, which interface with programs, and job control statements that relate programs to the data files they are to process.

This Unit introduces the basic concepts of data management and presents the job control statements required for DASD file label processing.

Upon completing this Unit, you should be able to:

Assignment 1:

- Define the terms field, record, file, volume, label, and access method.
- Describe the purpose and function of the Logical and Physical Input/Output Control routines (LIOCS and PIOCS).
- Describe the function and use of the label information area.
- Know when to use OPTION USRLABEL, OPTION STDLABEL, and OPTION PARSTD for storing label information.

Assignment 2:

- Describe the relation between DASD file labels and the Volume Table of Contents (VTOC).
- Describe the sequence of events involved in the creation and checking of disk file labels.
- Code the DLBL and EXTENT statements for creating and checking disk file labels.

Study Guide (SR20-7300)

The following VSE reference material:

VSE System Data Management Concepts (GC24-5209)

Introduction to the VSE System (GC33-6108)

Before investigating how data management works to give your programs access to the files they need, let's define some basic terms. See Figure 5.1 for an illustration of the defined relationships.

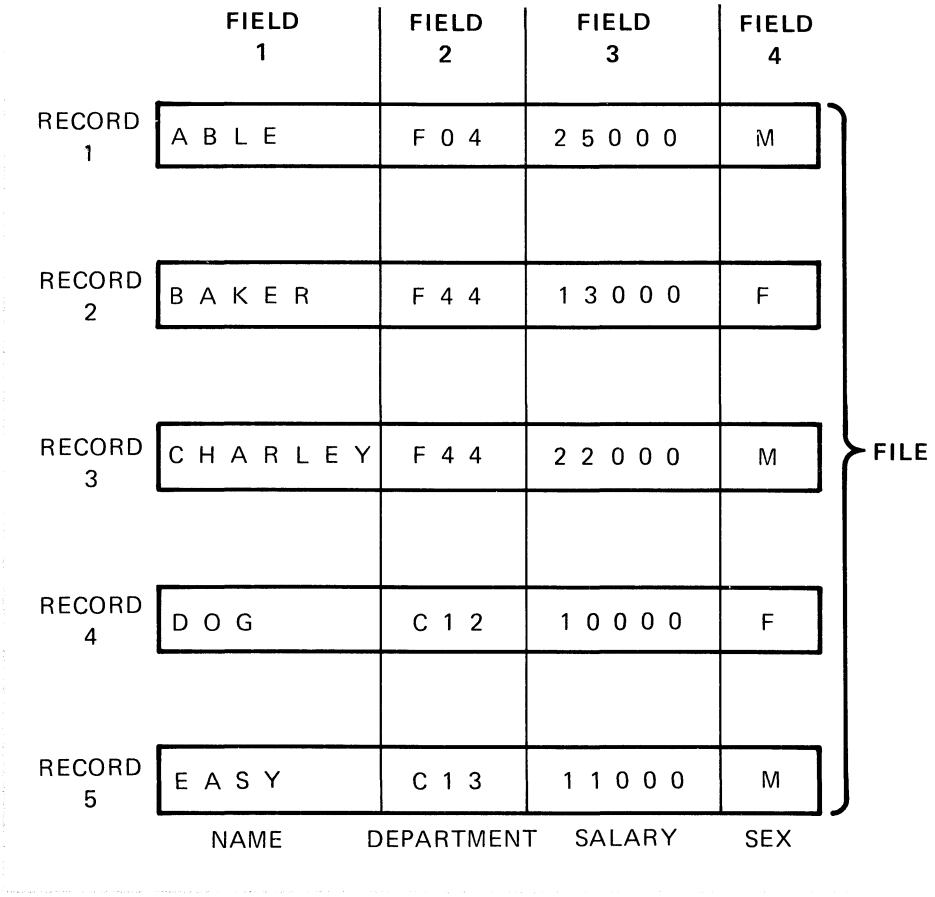


Figure 5.1 - Fields, Records, and Files

Fields

Information is defined as facts about people, places, or things. Fields are the smallest units of information. The data within a field must always be considered within the context of that field's definition.

In Figure 5.1, field 4 is a single byte of data that can be interpreted as information since it tells us the sex of an individual. Any single byte of data from the Name, Department, or Salary fields, however, is incomplete by itself and could not be considered meaningful information.

Records

There are two types of records, logical and physical.

Logical Records

These are a collection of fields that relate to the same entity. In Figure 5.1 there are five logical records each of which consists of four fields of information relating to specific employees.

The programs that you write will usually operate on logical records, one at a time.

Physical Records

These correspond to the way logical records are stored on the external media. The unit of transmission between your program and external storage is usually the physical block, which may contain any number of logical records or even a portion of a single logical record.

File

A file is a collection of related logical records. In Figure 5.1, the file is the collection of all employee records.

Volume

A volume is one uniquely identifiable unit of storage regardless of medium. It could be a single reel of tape or a single disk pack or diskette. The amount of information that can be stored on a volume is limited by the physical capacity of that volume.

The Function of Data Management

Data management serves as an interface between your application program and the data it processes. Your job as programmer is to provide the routines that perform the manipulation of data in main storage. The job of data management is to provide your program with access to data stored on external storage devices.

Data management will determine what to do in response to your program's input/output requests. The action may be to move the next record of a block into or out of your work area, or it may be to initiate an actual read or write operation. Whatever the action taken by data management, your program has only to issue a READ, WRITE, GET, or PUT statement to request data management services.

Information Links

You must supply data management with information concerning the files that you want to access, where they are located, and how they can be identified. The answers to these questions will determine which files will be processed by your program.

There is a complete chain from your program's input/output request statements (READs, WRITEs, GETs, PUTs) to the devices on which your files are located. Figure 5.2 shows an illustration of these links for a COBOL program.

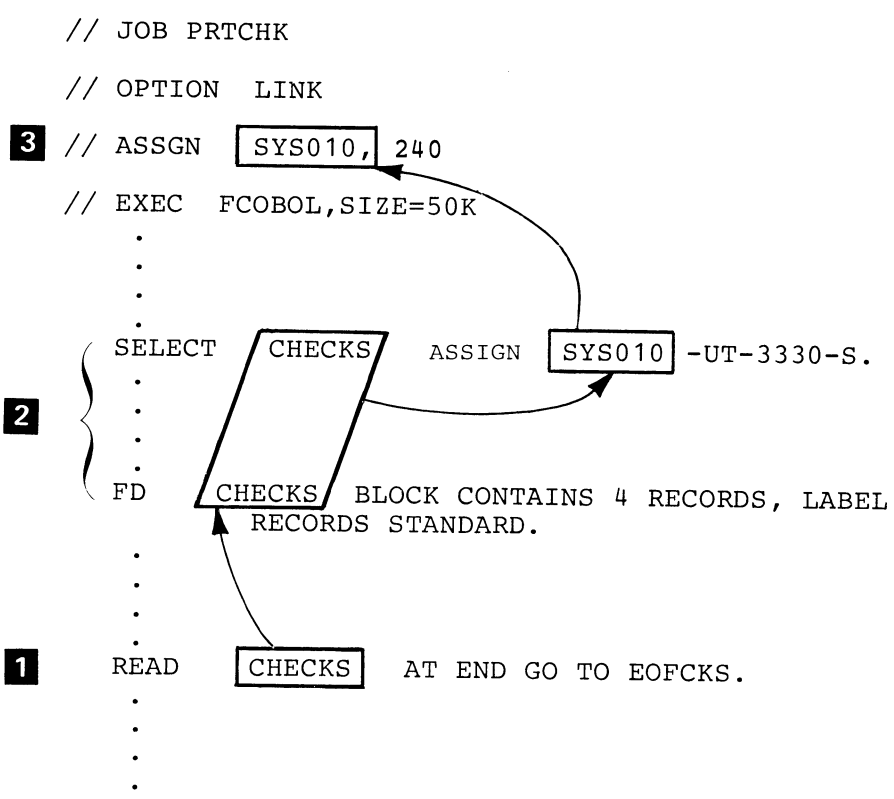


Figure 5.2 - The Information Chain

1 The input/output statement that requests a record indicates the name by which the file is called *within* the program (CHECKS in this case).

2 The logical unit name for this file (SYS010) is specified in the statements that define the file. In this way the logical name (SYS010) is associated with the file name (CHECKS).

3 An ASSGN statement in your JCL for this job associates the logical unit name with an actual physical device.

The Input/Output Control System is a vital part of data management. Included in this VSE component are three interacting elements that supply records to the processing program. These are LIOCS, PIOCS, and the access methods.

- Logical IOCS (LIOCS) is responsible for the blocking and deblocking of records after they are in main storage. LIOCS also determines when an actual transfer of data is required to or from external storage, and requests a physical write or read operation.
- Physical IOCS (PIOCS) is responsible for accomplishing the physical input or output operation. When a physical transfer of data is required, PIOCS causes the hardware device to transmit a record to or from the CPU.
- Access methods are systems for organizing and processing data on the variety of storage media available.

The PIOCS routines are part of the supervisor, while the LIOCS routines reside in a Relocatable Library and are made part of your program by the autolink function of the linkage editor at link-edit time.

Figure 5.3 illustrates the interactions of LIOCS and PIOCS for an input file.

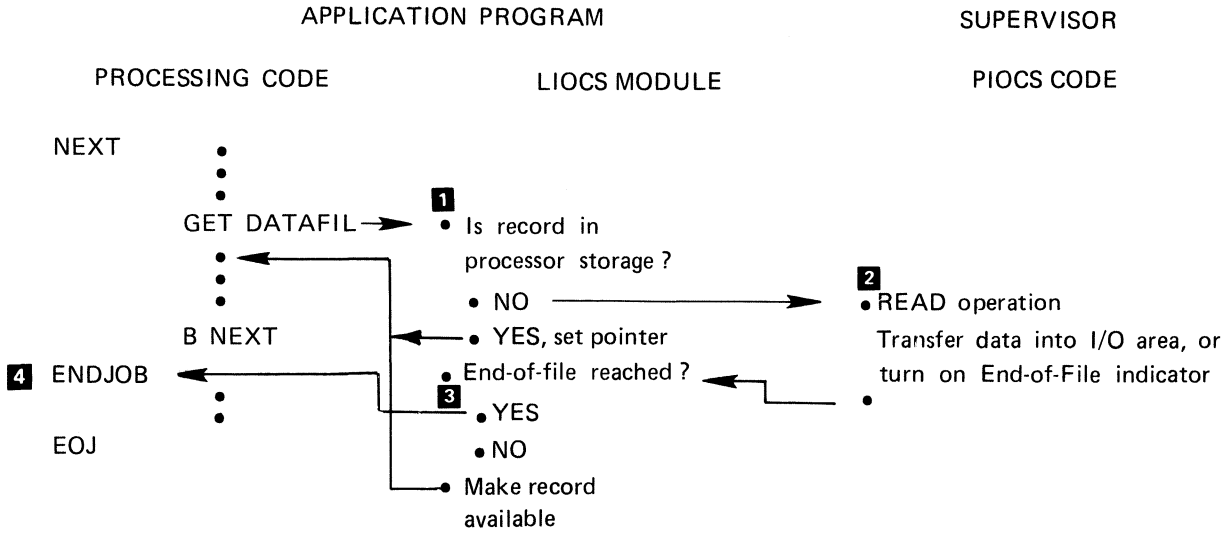


Figure 5.3 - LIOCS/PIOCS Relationship

Note the following:

1 LIOCS routines are part of the application program. At the statement GET DATAFIL, control goes to LIOCS to determine if a specific record is in processor storage. If this is the case (from a previous read operation), a pointer in your program is updated to point to the record, and control is passed to your processing routines.

2 If the record is not in processor storage, a physical data transfer is required and control goes to the PIOCS module where the record is read from external storage into an I/O area within your program. If there are no more records to be read, an end-of-file indicator is turned on.

3 If end-of-file has been reached (no more records to process), control is passed to step 4. If end-of-file has not been reached, the record retrieved by the physical I/O operation is made available to your program, and control is passed to your processing routines.

4 ENDJOB is the label of a routine specified in your program as the end-of-file routine: the part of your program to get control when there are no more records to process in this particular file.

There are a variety of ways in which records and files can be organized on external storage. For each method of organization there is an access method for the creation, retrieval, and update of records. The VSE access methods are:

- Virtual Storage Access Method (VSE/VSAM), which provides both sequential and random processing capabilities on DASD. VSE/VSAM has many facilities and efficiencies not available with any of the other access methods.
- Sequential Access Method (SAM) which supports sequential processing on DASD, tape, unit record and other devices where the method of data organization is sequential in nature.
- Direct Access Method (DAM), which supports random processing on certain DASD devices.
- Indexed-Sequential Access Method (ISAM), which supports both sequential and random processing on certain DASD.

The access methods are not covered in any detail in this course. A more detailed look at the access methods can be found in the following reading assignments:

In the *VSE System Data Management Concepts* manual read "Introducing Access Methods".

In the *Introduction to the VSE System* manual under "Data Management" read up to "Telecommunication Access Methods", and under "Additional Licensed and Nonlicensed Programs" read "VSE/Virtual Storage Access Method (VSE/VSAM)".

Identification of Volumes and Files

Look back at Figure 5.2. It appears that the program will access whatever volume is mounted on the unit specified as SYS010. How do you know that the volume containing the file you need will in fact be mounted on the physical device associated with SYS010? What distinguishes the files this program uses from those used by any number of other programs that reference SYS010 during their execution?

Under VSE it is possible to perform label checking on magnetic tape and DASD volumes and files. Label checking is a LIOCS function that assures that your program accesses only those volumes and files it is supposed to process.

The term labels here means special records magnetically encoded on tape or disk that are processed by VSE, and not the external markings that allow you to locate volumes in a library by visually scanning them.

Labels identify specific tape or disk volumes. Since it is common to have more than one file on a given volume, it is necessary to be able to check file as well as volume labels. For this reason, there are volume labels that identify the disk volume or reel of tape, and there are file labels to identify specific data files on a volume. Both of these are recorded in clearly defined ways as machine readable records with standard formats and are located in specific places on tape or disk. They will be discussed in detail in the sections on DASD and tape file labels.

Label Control Statements

IOCS gets part of the information it needs for label processing from the job stream. There are job control statements that are designed specifically to provide information for processing labels on tape or disk files. These are the DLBL and EXTENT for DASD label processing and the TLBL for tape label processing. DLBL, EXTENT, and TLBL are part of the control statements you submit with your job.

When job control reads the label statements DLBL, EXTENT, and TLBL, it stores them in the label information area for later use by your program. When you OPEN your files to make them available for processing, the LIOCS routines invoked will read the label data from the label information area and use it for label processing purposes. Figure 5.4 illustrates how the label information area is used.

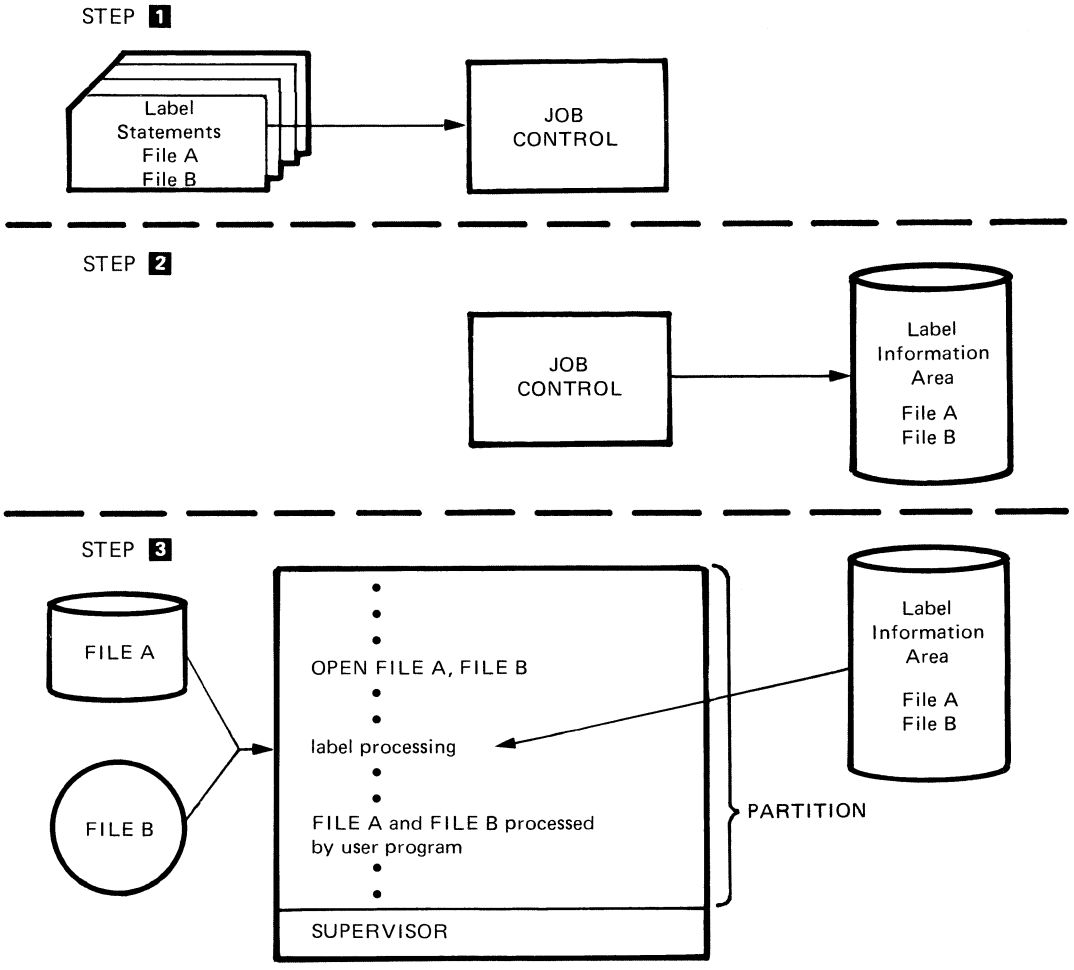


Figure 5.4 - The Label Information Area

1. DLBL, EXTENT, and TLBL are read by the job control program along with all the other statements (ASSGN's, UPSI, EXEC, etc.) you submit with your job.

2. DLBL, EXTENT, and TLBL are written to the appropriate portions of the label information area. The rest of your JCL is processed as previously discussed.

3. When the files are OPENed by your program, the data on the label information area is read by the LIOCS routines that are part of the OPEN function.

Temporary and Permanent Label Information

Labels are used to identify your program's files, and to differentiate these files from those used by other programs. Label information may be temporary - valid during one job or job step, or it may be permanent - valid until replaced.

Temporary label information is useful for describing those files that are specific to your program's needs, while permanent label information is useful for describing files associated with a large number of jobs. An example of the latter would be the work files used by any of the language translators or by the linkage editor. Instead of requiring each individual user to submit label statements for these work files with every compile or link-edit run, the label information is made standard, much as are the system standard options discussed earlier.

Three Types of Label Information

Three different portions of the label information area are used for storing different types of label information. Each partition in your system is allocated part of the area for temporary labels and part of it for permanent labels. A third portion of the area is used for system standard labels. The different label types are defined as follows:

- Partition Temporary Labels. These are submitted during a job or job step. They are available to that partition only during the job.
- Partition Standard Labels. These are available for a specific partition at all times. They remain available from job to job.
- System Standard Labels. These are the same as partition standard labels, but are available to all partitions, not just one.

Three Available OPTIONs

One of the three types of label information is created whenever you submit a job containing TLBL, DLBL, or EXTENT statements. Job control writes these statements to the label information area. By using the OPTION statement you can control the type of label information to be created. The OPTION statement can be coded as:

```
                USRLABEL
// OPTION       PARSTD
                STDLABEL
```

USRLABEL: Creates temporary labels for the partition in which the job is running. If no OPTION card is submitted for a job, USRLABEL is the default condition.

PARSTD: Creates permanent labels for the partition in which the job is running.

STDLABEL: Creates permanent labels available to all partitions in the system. *Any job containing an OPTION STDLABEL statement can be run only in the BG partition.*

Figure 5.5 shows each of these options in use. Except for the STDLABEL run, the label information goes to that part of the area appropriate to the partition in which the job control was read.

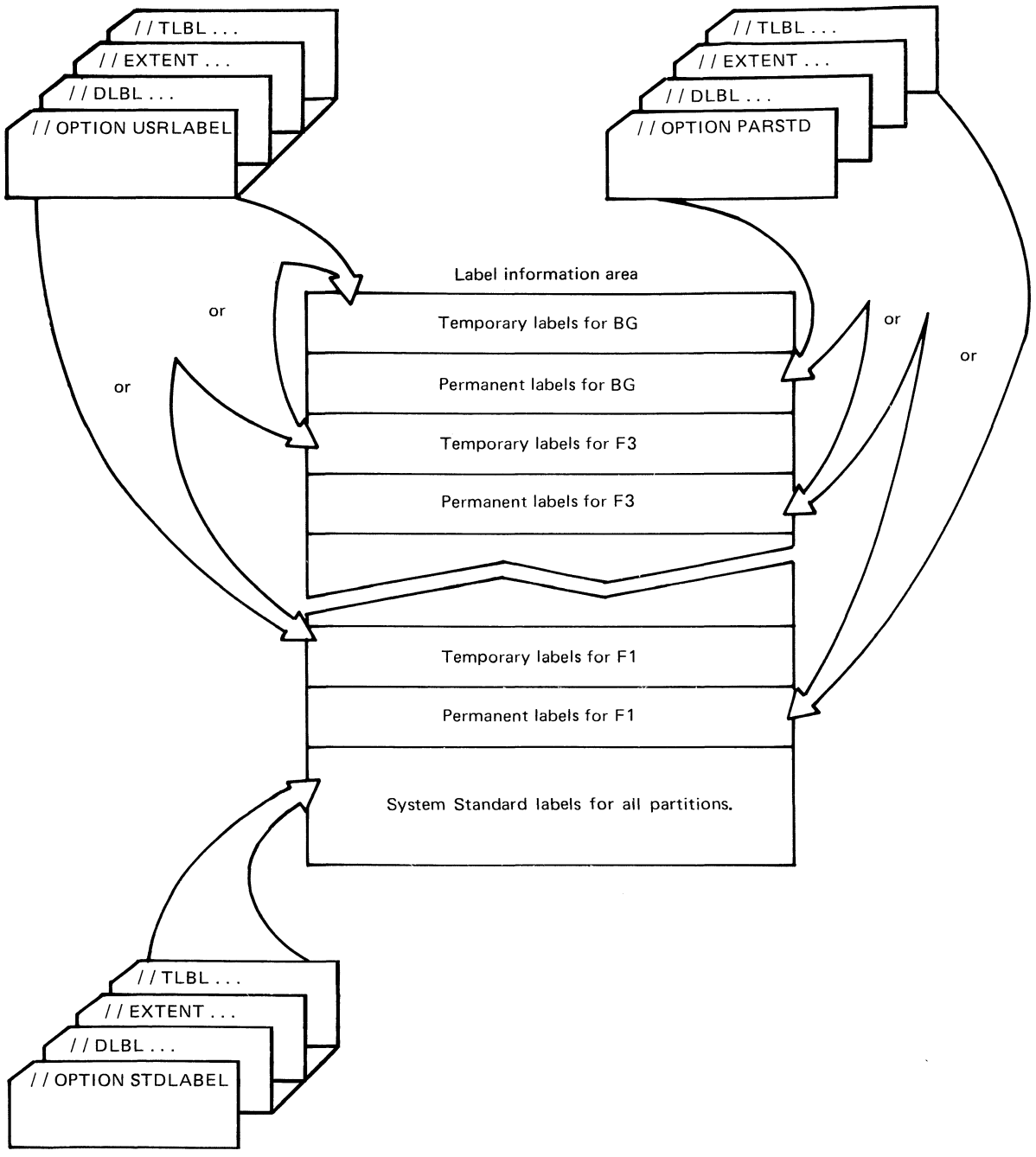


Figure 5.5 - Storing Label Information

When any type of label information is written to the label information area, it does not generally add to what is already there, but rather replaces it. This means that label information submitted with a later job step will wipe out label information submitted with earlier steps. Take a look at the following example.

This job consists of two steps, the first of which executes program A and the second of which executes program B. Program A requires FILEA while program B needs only FILEB. The JCL that follows is perfectly all right for this situation.

```
// JOB     AOK
// DLBL    FILEA
// EXTENT  FILEA
// EXEC    A
// DLBL    FILEB
// EXTENT  FILEB
// EXEC    B
/&
```

Now a third job step is added which executes program AC. This program requires *both* FILEA and FILEC.

```
// JOB     NG
// DLBL    FILEA
// EXTENT  FILEA
// EXEC    A
// DLBL    FILEB
// EXTENT  FILEB
// EXEC    B
// DLBL    FILEC
// EXTENT  FILEC
// EXEC    AC
/&
```

Will this job run correctly? The answer is no. Each time job control reads label statements from the job stream, the partition temporary label area is rewritten. When the EXEC statement for AC is encountered, the only label information available is that for FILEC. The proper way of handling this type of situation is to place all required label information together as follows:

```
// JOB     ALLISOK
// DLBL    FILEA
// EXTENT  FILEA
// DLBL    FILEB
// EXTENT  FILEB
// DLBL    FILEC
// EXTENT  FILEC
// EXEC    A
// EXEC    B
// EXEC    AC
/&
```

Since there are no label statements in the job stream once we get into program executions, the partition temporary label area is not rewritten and all label information is available to all three programs.

Label information submitted following a PARSTD or STDLABEL option, with no operand specified, is written at the beginning of the label information area, overlaying any previous contents.

The operands

STDLABEL=ADD or =DELETE
PARSTD=ADD or =DELETE

allow a label or groups of labels to be added to or deleted from those already present in the label area.

When IOCS is looking for label information, the search sequence is USRLABEL, PARSTD, and then STDLABEL.

The LSERV system utility program will display the contents of the label information area. All labels are shown and are identified by the partition to which they belong, and whether they are temporary or permanent.

LSERV examples and illustrations of the type of output it produces are in the manual *VSE Advanced Functions Serviceability Aids and Debugging Procedures* (SC33-6099).

When your program executes, the contents of the label statements must be checked against existing file labels (in the case of input files), or be used to create file labels (in the case of output files).

Two IOCS routines are responsible for label processing. These are the OPEN and CLOSE routines. OPEN is an initiating routine that is called upon to make a file available for processing by your program. CLOSE is a terminating routine, invoked to "put the file away" after your program has finished using the file.

Exercise 5-1

Take the short quiz that follows. If you get more than three of these questions wrong, review the Assignment you have just studied.

1. The smallest unit of information is the _____.
   a. field
   b. record
   c. file
   d. volume
2. Programs that you write will normally process _____ _____, one at a time.
   a. physical records
   b. logical records
   c. logical volumes
   d. record groups
3. A volume of data could be contained on a _____.
   a. reel of tape
   b. disk pack
   c. diskette
   d. any of these
4. One of the functions of LIOCS is to _____.
   a. transfer data to and from external storage
   b. handle end-of-file processing in your program
   c. block and deblock records
   d. request autolink include functions
5. Label processing and checking is a _____ responsibility.
   a. LIOCS
   b. PIOCS
   c. supervisor
   d. operator

6. Which of the following statements would never be found in the label information area?

   a. DLBL

   b. TLBL

   c. EXTENT

   d. OPTION

7. The job stream to create labels available to any partition would include the _____ statement.

   a. // OPTION PARSTD

   b. // OPTION STDLABEL

   c. // OPTION USRLABEL

   d. all of the above

1. a
2. b
3. d
4. c
5. a
6. d
7. b

Fixed Block Architecture offers a new way of storing data on DASD devices, such as the 3310 and 3370, that are available with the IBM 4331 and 4341 processors. The principle of FBA is that each storage device can be viewed as containing a string of equal length blocks. These blocks are numbered from 0 to n-1, where n depends on the capacity of the device in question. Blocks are addressed by Relative Block Number.

When using an FBA storage medium, the user need not be aware of the physical structure of the device and does not have to know track capacity, the number of tracks per cylinder, or even the number of cylinders available. The only thing that must be known is the number of available blocks.

FBA makes the application program completely independent of the physical storage device.

The fundamental unit of data transfer between the host system (the processor on which your program is running) and FBA I/O devices is a *fixed length block of data.* A *data record,* as viewed by the devices, consists of a *data block* plus a *block control field.*

The block control field contains the address of the block plus optional device-specific control information. The address of the data block is a binary number from 0 to n-1. The address range spans a physical unit of the storage medium, that is, an entire device.

The record format of a data record is device dependent. For example, the block control field and the data block may be recorded as separate fields on the device or they may be concatenated and recorded as a single field. Whatever scheme is used, the transfer unit between storage and the FBA I/O device in question is in multiples of fixed length data blocks. The control fields are *not transferred* to your program.

FBA is supported by two of the VSE access methods:

- SAM (Sequential Access Method)
- VSE/VSAM (Virtual Storage Access Method)

Because of the direct access nature of DASD devices, files may be written at any location on a volume. The beginning and ending addresses of a file delimit that file's extent, which is the area of the DASD volume it occupies. Because a file's extents may be placed anywhere on the volume, every DASD file must have a label to identify its extents. This applies even to temporary work files ("scratch" files).

Since files are scattered throughout DASD volumes, how does IOCS know where to locate label and file information? The answer is provided by a 2-level structure on each volume consisting of the DASD *volume label* and the *volume table of contents (VTOC).* See Figure 5.6.

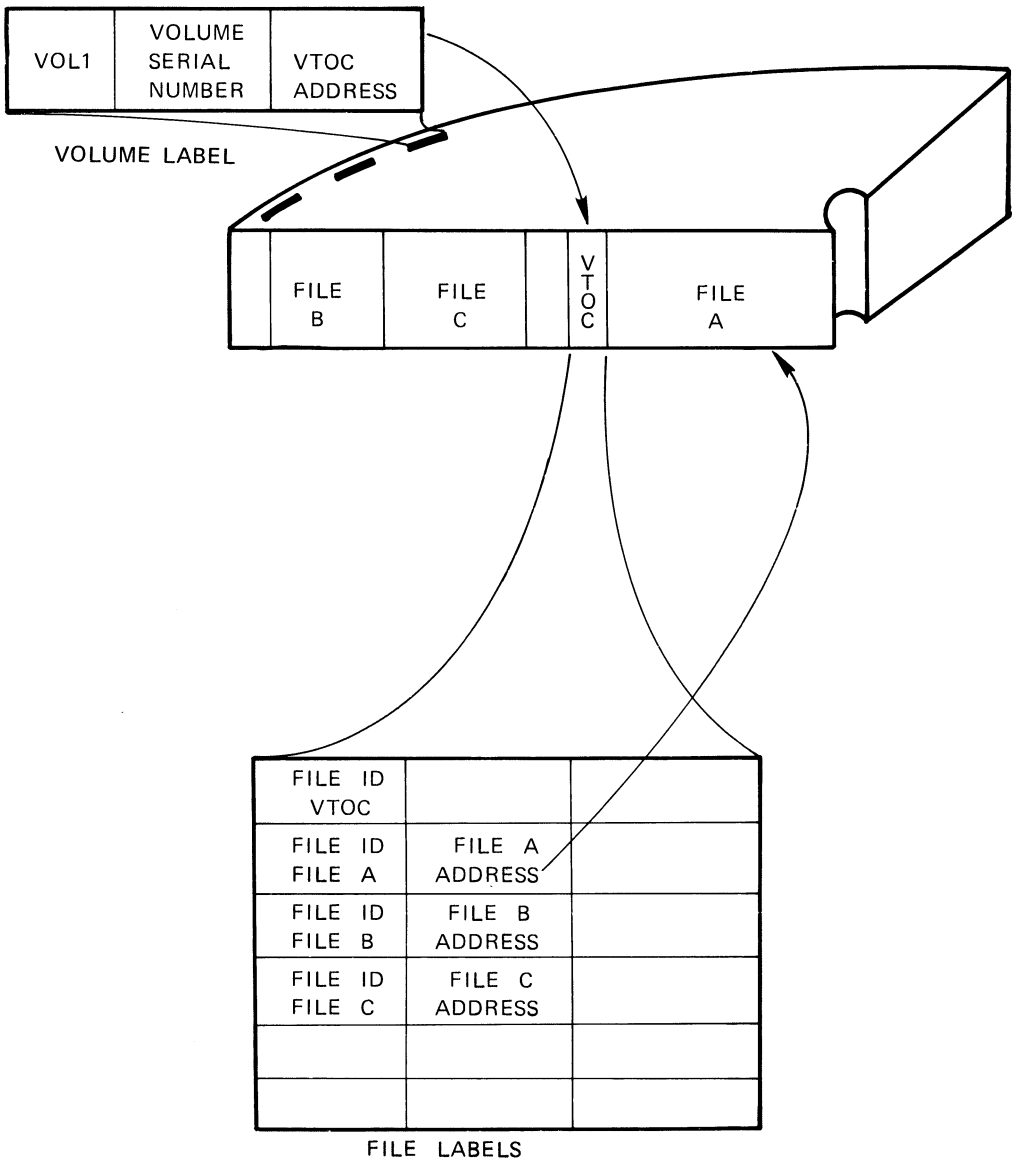


Figure 5.6 - Disk Volume Layout

The volume label is in a standard location for each device so IOCS is always able to find it. On a disk pack, the volume label is located on cylinder 0 track 0 record 3 (CKD) or block 1 (FBA). On a diskette volume, it is on cylinder 0 track 0 record 7.

The volume label points to the VTOC, which is a file of file labels. The file labels in the VTOC contain the starting addresses and other information pertinent to files on that volume. Every DASD volume has its own VTOC.

To find a file on a DASD volume, IOCS reads and checks the volume label, takes the VTOC address from it and then searches the VTOC until it locates the desired file label. Having

found the file label, IOCS can then proceed with its label checking function and, after verifying that this is indeed the correct file, make the file available to be accessed by your program.

When the volume label for a DASD volume other than diskette is created, one of the data fields put in it is the VTOC address. This initialization function is done by the system utility programs, Initialize Disk and Device Support Facilities.

The VTOC on a diskette, unlike other DASD, is in a fixed location: cylinder 0 track 0 records 8 through 26. This eliminates the first step in the search for a diskette file label.

The two job control statements that provide information for disk label creation and checking are the DLBL and EXTENT statements.

There must be one DLBL statement for each disk file to be accessed. The DLBL statement is used to provide the file name, identification, and expiration date of the file. The DLBL statement also specifies the access method used for the file.

The EXTENT statement provides the symbolic address, or logical unit, on which the file is mounted. The symbolic address in the EXTENT statement is related to the symbolic address in an ASSGN statement to determine the physical file address. Other information in the EXTENT statement identifies the disk pack volume serial number and specifies the physical location of the file within the disk pack. It also specifies the physical size of the file. Portions of a disk file may be located in different areas of a disk volume. The areas need not be adjacent, or contiguous, to each other. Also, portions of a disk file may be located on different volumes. Each area of disk that contains a portion of the disk file requires an EXTENT statement to define it. Thus one file may have multiple EXTENT statements in the job stream.

The job stream below contains DLBL and EXTENT statements that describe the disk file to be accessed by the application program. The ASSGN is used to specify the physical drive on which the file is mounted. Note that the DLBL, EXTENT, and ASSGN precede the EXEC statement for the program. Job control reads the label statements and stores them in the label information area. The label information is then available to IOCS at OPEN time during the execution of the application program.

```
// JOB    DISKPRT
// ASSGN  SYS003,DISK
// DLBL   MYFILE,...
// EXTENT SYS003,...
// EXEC   MYPROG
/&
```

Use Figure 5.7 for tracing the following sequence of events.

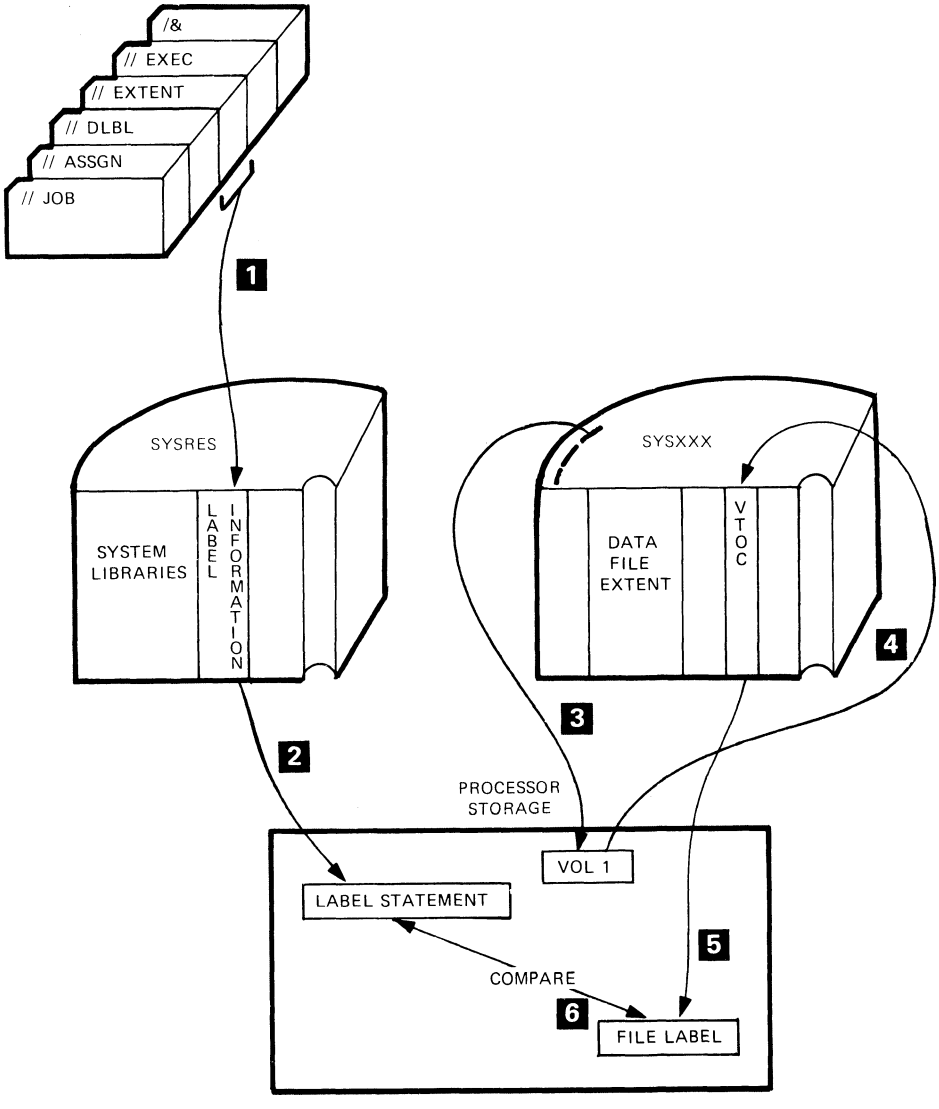


Figure 5.7 - Disk Label Checking

1. When your job stream is read, job control checks syntax and places the DLBLs and EXTENTs in that portion of the label information area appropriate to the partition in which your program is to be run.
2. At OPEN time, the input file name is used as a search key to locate the label statements in the label information area.
3. The volume label is read and checked. If the serial number of the mounted volume does not match your supplied label information, a message is sent to the operator and processing halts until the proper volume is mounted.
4. When the correct volume is mounted, the location of the VTOC is determined from the volume label. The VTOC is searched for a file-ID consistent with that on the DLBL statement read from the label information area. If a matching label is not found in the VTOC, a message is sent to the operator.
5. If a match is found, the file label is read into storage.
6. Its EXTENT information (defining the file's boundaries) is checked against that on the supplied EXTENT cards. When these checks are all positive, the file is made available to your program for processing.

Output Label Processing

Labels are created at OPEN time from the information you specify on your label job control statements. Before a label for a new file is created, however, a certain amount of checking must be done.

When an output file is OPENed, LIOCS searches the label information area using file name as the search argument. The starting disk address and number of tracks allocated to the file are taken from the EXTENT statement. LIOCS will then read the volume label of the pack on which the file is to be written, and will compare the volume serial number in the VOL1 label on the pack with the number specified in the EXTENT statement.

When it is certain the correct volume is mounted, all file labels in the VTOC are read by IOCS, and the extents for all unexpired files will be checked against the extents specified for the new file . IOCS also checks for an "equal file": an unexpired file whose file-ID matches the file-ID for the new file. If there are no active files with the same or overlapping extents, IOCS writes a standard file label in the VTOC for the new file. The application program can now use the file for processing. If matching or overlapping files were found, the operator will be notified so that corrective action may be taken.

When the application program executes a CLOSE, the file termination process is invoked, and updates the file label in the VTOC. The file is no longer currently accessible by the program, and any further processing of the file requires it to be OPENed again.

The DLBL Statement

You must provide information via the operands of the DLBL statement to identify the data file and the access method used to retrieve it. Most of the operands are optional, and default values will be assumed for them when omitted. However, caution should be observed when using defaults because defaults may not uniquely identify a given file.

File Name

File name is an arbitrary designation of your choosing that specifies the symbolic name of the file as used in your program. It is the only mandatory operand in the DLBL statement. File name may be from one to seven characters in length, and may be represented by alphameric characters, except for the first character, which must be alphabetic.

File name is not a part of the file label, but rather is the link between your program and the DLBL statement. When your program issues an OPEN using a particular file name, that file name is used to locate the DLBL statement in the label information area.

COBOL File Name

The designation of file names in COBOL programs is an exception to the above procedure. In COBOL, the FDNAME can be over seven characters in length. The DLBL statement, however, allows no more than seven characters for the file name parameter. Therefore, you must use the SYSnnn specified in the ASSIGN *clause* of the COBOL SELECT as the file name parameter in the DLBL. This always holds unless an EXTERNAL NAME is specified in the COBOL ASSIGN. In this case the SYSnnn value is overridden, and the EXTERNAL NAME must be used as the DLBL file name.

File-ID

The file identifier (file-ID) operand specifies the unique name given the file. The identifier is recorded in the file label. The file-ID can be one to 44 alphameric characters. If it is fewer than 44 characters, it is left-justified and padded with blanks. The file-ID specification must be contained within apostrophes. This operand is used as the search argument to search the VTOC for the file label.

If the file-ID is omitted, the file name parameter from the DLBL statement is used as a default value. For example, if the statement

```
// DLBL MYFILE
```

were processed for an output file, the value of MYFILE would be assumed for file-ID and would be left-justified, padded with blanks, and entered into the file label.

The file-ID is the link between the file name used in your program and the externally stored data. Since it is the file-ID and not the file name that is stored in the VTOC, a given data file can be accessed through what may be different file names in different programs. See Figure 5.8 for an illustration of this concept.

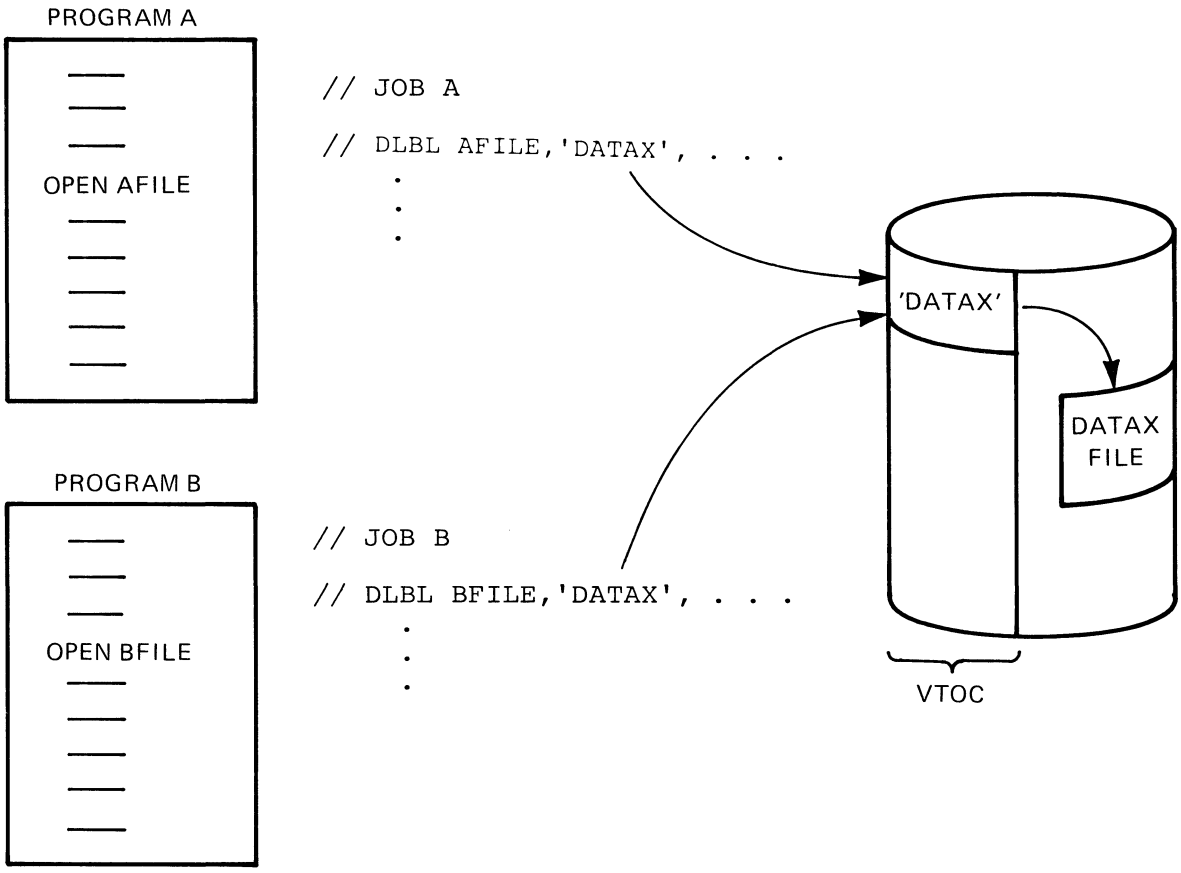


Figure 5.8 - Using the File-ID

Here, programs A and B reference a data file by the different file names AFILE and BFILE. The file-ID used in both DLBL statements identifies 'DATAX', which is in the VTOC of the appropriate volume and points to the file in question.

The COBOL programmer must once again realize that it would be the SYSnnn or EXTERNAL NAME from the COBOL ASSIGN that would be coded as the DLBL file name.

Generation and Version Numbers

You can make provision for generation and version numbers within the file-ID. These values allow you to identify specific versions of a file where multiple copies of the file exist. See the manual *VSE/Advanced Functions DASD Labels* (SC24-5213) for further information on this capability if it is something you need to use.

Output files have an expiration date written in their labels based on the date operand. The expiration date is the date after which the file is considered obsolete. At any time after that date, the file label may be removed from the VTOC. Removing a file label from the VTOC makes the disk space identified by that file's extent available for a new output file to use.

There are two allowable date formats for an output file. You may specify a retention period, or number of days the file is to be considered current, as one to four digits from 0 to 9999.

Optionally, you may specify an absolute expiration date in the form yy/ddd, in which yy indicates year and ddd indicates day of the year. The second format may also be used to indicate a retention period by coding 00 for year. For example, 00/014 specifies a retention period of 14 days, whereas 80/014 specifies an expiration date of January 14, 1980.

If the date operand for an output disk file is omitted, a 7-day retention period is assumed and the expiration date is calculated to be seven days after the creation date. Creation date is taken from the SYSTEM DATE field of your partition's communication region and occupies a place in the disk label separate from the expiration date.

The date operand in the DLBL statement is ignored for input files.

The code operand of the DLBL statement specifies the type of file with which this label is associated. Code is a two to four character field. Valid codes and their associated file types are listed below. (SD is the default.)

```
CODE        FILE TYPE

SD          Sequential Disk
DA          Direct Access
DU          3540 Diskette
ISC         Indexed Sequential-Create
ISE         Indexed Sequential-Other than create
VSAM        Virtual Storage Access Method
```

In many instances, it may be desirable to maintain very tight control over access to a file. Provision is made in VSE to cause a message to be sent to the operator each time a file is opened by an application program. The operator will then have the opportunity to allow or prohibit access to the file. Such a file is called data-secured.

Data security is initiated by including the DSF operand in the output file DLBL statement when the file is created. The presence of DSF causes a bit to be set in the file label, which identifies the file as one that is data-secured. Each time the file is accessed by the OPEN routines, the security bit in the file label will initiate a message to the operator. DSF does not apply to VSE/VSAM files as all VSE/VSAM files are data secured already.

An example of a DLBL statement using the DSF operand is shown below.

```
//  DLBL OUTPUT,'BRAND NEW FILE',30,,DSF
```

Note that the code operand was omitted, as indicated by successive commas. When code is omitted, SD (sequential disk) is assumed.

The BUFSP=, CAT=, DISP=, RECORDS=, and RECSIZE= operands pertain only to VSE/VSAM files and will not be covered here. They are explained in in the VSE/Advanced Functions System Control Statements Manual.

The BLKSIZE= operand is valid only for sequential disk files. It allows you to access DASD files originally created for *other than* these devices with a more efficient blocking factor, without program recompilation.

The value given here must be a number from 1 to 32,768, must be greater than the block size specified in the DTFSD macro within the program, and must be a multiple of RECSIZE if the file contains blocked fixed-length records.

CISIZE

A control interval (CI) is the unit of data transfer between processor storage and FBA devices. Each CI is mapped by LIOCS over an integral number of FBA blocks.

The CISIZE parameter permits specification of the Control Interval size for SAM files on FBA devices in order to improve space utilization on the devices. The specified size must be a multiple of the value specified in the BLKSIZE=n parameter, and must be also a multiple of 2K if it is greater than 8K. CISIZE is only valid for DLBL statements with the code SD.

For files being opened for output, the size of the Control Interval will be determined by the OPEN routines based on user-specified DTF parameters such as CISIZE, BLKSIZE, or RECSIZE, or on the user-specified CISIZE parameter on the DLBL statement. The CI must conform to VSE/VSAM CI size restrictions and to the FBA blocksize. The CI size will be stored in the FORMAT1 label on each volume when a new file is created, and retrieved from the first volume when a file is opened for input.

The Extent Statement

After providing IOCS with a name and identification for a file, you must indicate where the file is located and how big it is. The EXTENT statement is used to specify the disk pack, the starting location within the disk pack, and the size of your DASD files.

Disk files may be contained within single or multiple extents, and these multiple extents may range over different DASD volumes. Two cases are illustrated by Figure 5.9.

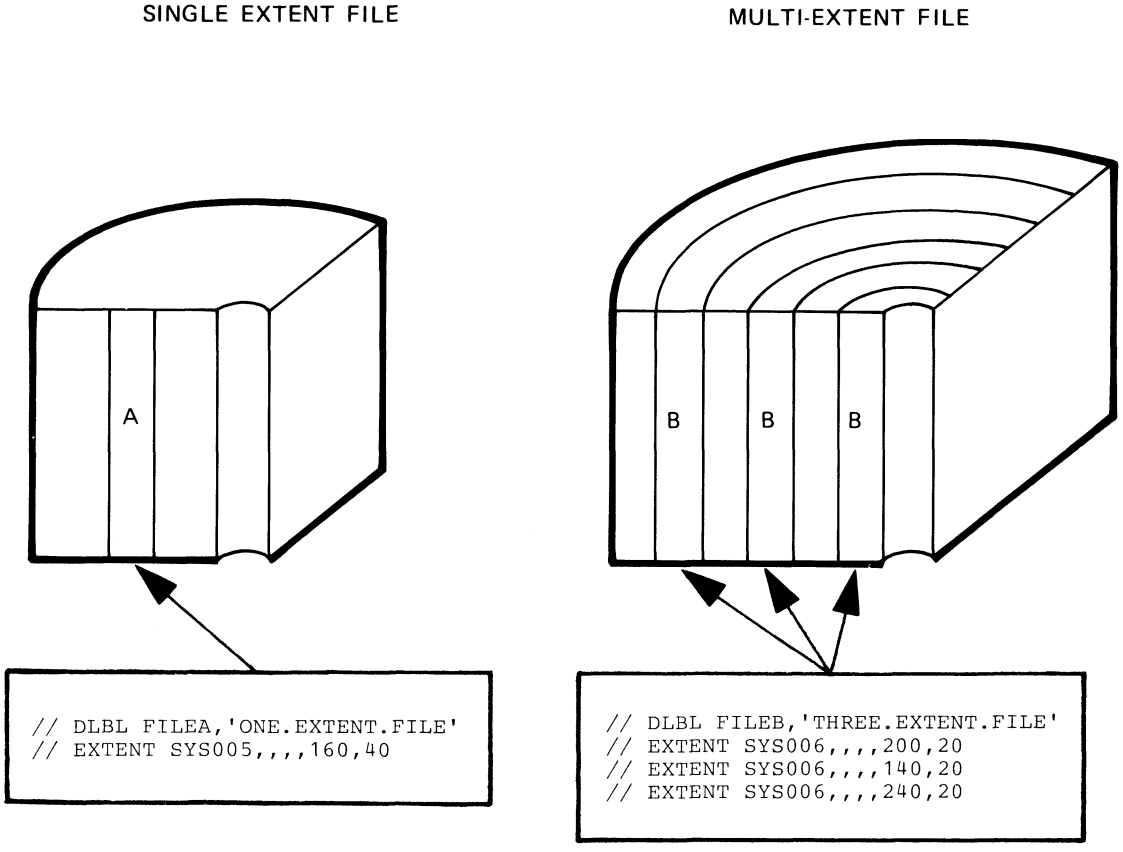


Figure 5.9 - Defining Extents

In the job stream, the EXTENT statement follows the DLBL statement for the file. If a file has multiple extents, all of its EXTENT statements are grouped together behind the DLBL.

The first operand in the EXTENT statement provides the logical unit address and corresponds with the logical unit in the ASSGN statement. Thus, if the EXTENT statement symbolic unit specifies SYS007 and if SYS007 has been assigned to physical drive 243, IOCS has no problem locating the disk drive. Figure 5.10 shows the relevant linkages.

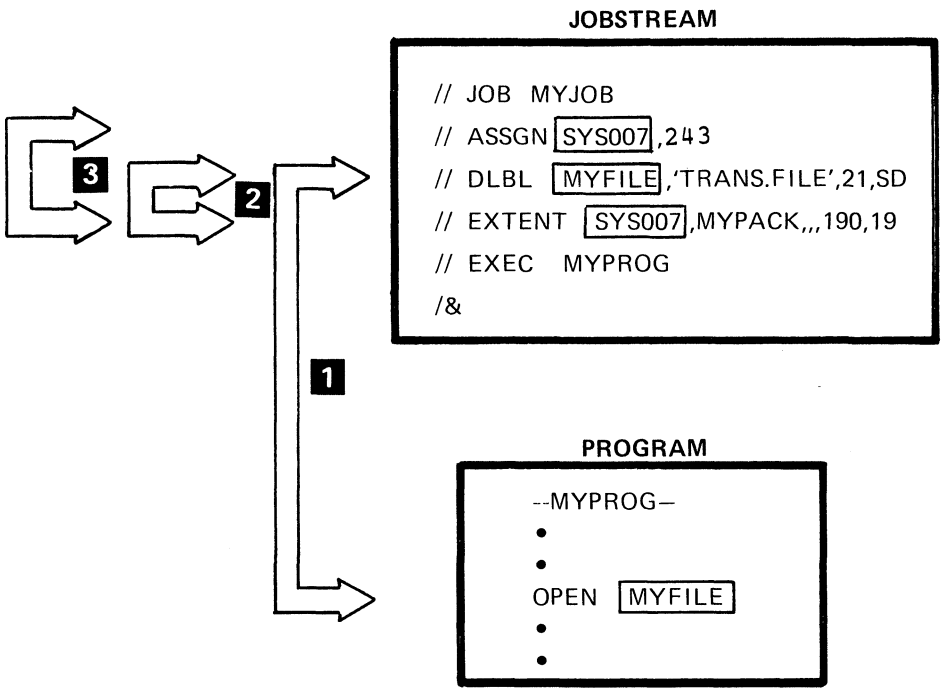


Figure 5.10 - Linkage From Program to Device

1 The file name in the OPEN statement relates to the file name on the DLBL.

2 Associated with the DLBL statement is an EXTENT that specifies the symbolic address as SYS007.

3 An ASSGN relates SYS007 to physical device 243.

Multi-extent disk files may reside on one or on multiple disk packs. All extents for a file on one pack should specify the same symbolic unit. The JCL below illustrates the label statements required for a disk file composed of three extents on the same pack. Note that symbolic unit (SYS006) is specified only in the first EXTENT statement. This is permissible, since the symbolic unit of the first EXTENT becomes the default value for following EXTENTS for the same file.

```
// JOB     MULTY
// DLBL    MYFILE,'TRANS.FILE'
// EXTENT SYS006,VSE004,,,190,38
// EXTENT ,,,,380,38
// EXTENT ,,,,855,190
// EXEC    MYPROG
/&
```

The leading commas on the last two EXTENTs are required as these parameters are *positional.* Job control identifies them by their position in the statement.

Specifying the Volume

You identify the specific volume, or pack, on which a file is located by using the serial number operand. When the file is OPENed, IOCS checks to be sure that the specified volume is mounted. If the volume is not correct a message will be sent to the operator so that the right pack can be mounted and processing continued. If this operand is omitted from the first or only EXTENT statement, no volume number checking is done. If omitted from subsequent EXTENT statements, the volume number from the preceding statement is used.

The following JCL is an illustration of the EXTENT statements for a multivolume disk file. With the EXTENT and ASSGN statement as shown, pack VSE004 should be on drive 242 and pack VSE015 should be on drive 243.

```
// JOB    MULTY
// ASSGN  SYS006,242
// ASSGN  SYS007,243
// DLBL   MYFILE,'TRANS.FILE'
// EXTENT SYS006,VSE004,,,380,76
// EXTENT SYS007,VSE015,,,19,190
// EXTENT SYS007,VSE015,,,399,95
// EXEC   MYPROG
/&
```

Specifying Extents

You should specify the type of extent you are referencing, its sequence number if it is part of a multi-extent file, and its starting location and size on the volume. There are operands on the EXTENT to provide each of these pieces of information.

Extent Types

There are four types of extents for a disk file. Three of these are data areas, and the fourth is an index area for ISAM files. The type operand indicates which is desired.

The three data areas are *normal, split cylinder,* and *independent overflow.* The most common type is normal, in which the file is allocated all tracks within the cylinders or blocks assigned to it (one continuous extent). A split cylinder file is one which is restricted to certain specified tracks within the cylinders assigned to it. Independent overflow is an extent used by ISAM for recording records which do not fit in the primary data area. Split cylinder and independent overflow data areas are not valid for FBA devices.

The different types are given below. Type 1 indicates that the extent is a normal data area, and is used for DAM, ISAM and non-split cylinder SAM files. Normally this type of extent will start on track 0 of a cylinder, but not always. A SAM or DAM file may begin and end on any track or block. In fact, a SAM or DAM extent can be as small as a single track or block. ISAM data areas must be allocated in full cylinders: they must start on track 0 of a cylinder and end on the last track of the last cylinder. The type 1 extent for an ISAM file is called the *prime data area.*

TYPE	DESCRIPTION
1	Normal data area
2	Independent overflow (ISAM only)
4	Index area (ISAM only)
8	Split cylinder extent (SD only)

An extent statement for a data area of 100 blocks beginning at block 2157 might be coded as either:

```
// EXTENT SYSnnn,,1,,2157,100
                  or
// EXTENT SYSnnn,,,,2157,100
```

If the type operand is omitted, type 1 is assumed.

Extent Sequence

This operand indicates the number of an extent within a multi-extent file. It is required for ISAM files and optional for SAM and DAM files. It is recommended that sequence number be coded so that EXTENT statements can be sorted easily into the proper sequence in case they are ever dropped.

Beginning Disk Address

File space is ordinarily allocated in terms of cylinders, but file location and size are stated in terms of tracks on the EXTENT. Conversion from cylinder to track reference is straight forward.

Assume, for example, that you have a file of 10 cylinders beginning at cylinder 131 on a 3330. To determine the starting track you multiply the beginning cylinder number by the number of tracks in a cylinder for the device in question. Since there are 19 tracks per cylinder on a 3330, the beginning track for this example is 2489 (131 x 19). The file is allocated 10 cylinders, so there are 190 tracks (10 cylinders x 19 tracks per cylinder).

The beginning disk address for a CKD device is specified in the *relative track* operand of the EXTENT statement. This is a one to five digit number, relative to zero, indicating the track on which the file begins.

For FBA, this parameter is called *block address,* and specifies the physical block address at which the extent is to start. It is a number in the range 2 to 2,147,483,645.

If a starting address of 0 is specified, it is treated as an error for both CKD and FBA devices. If a block address value of 1 is specified for FBA, this is not treated as an error by job control, as job control cannot distinguish between DLBL and EXTENT statements for CKD and FBA. However, the lowest block address should be 2, since block number 1 is occupied by the volume label.

Extent Size

The size of the disk extent is specified in the *number of tracks* operand of the EXTENT statement. This is a one to five digit number which simply states how many tracks there are in this extent. The number of tracks is determined by multiplying the number of cylinders for the file by the number of tracks per cylinder for the device in question. Thus, 20 cylinders of 3340 extent would require 20 times 12 tracks per cylinder, or 240 tracks. Of course, a file may not always start and end on a cylinder boundary. This would have to be taken into account in the calculation.

For FBA, this parameter is called *number of blocks,* and specifies the number of physical blocks in the extent. Its value lies in the range 1 to 2,147,483,645.

A type 8 extent indicates that the extent is for a split cylinder file. A split cylinder file is one that uses only a portion of the tracks within the cylinders allocated to it. Only CKD SAM files can be set up as split cylinder.

Split cylinder files can be used when significant amounts of data are to be passed back and forth between two or more SAM files being processed "in step" with each other. When such files are set up as split cylinder, access arm movement is reduced because the different files occupy portions of the same cylinders. See Figure 5.11. In order to take advantage of access arm positioning in this example, cylinder 20 of FILE2 would have to be processed at the same time as cylinder 20 of FILE1, and so on. A major application of split cylinder files in VSE is for system work files.

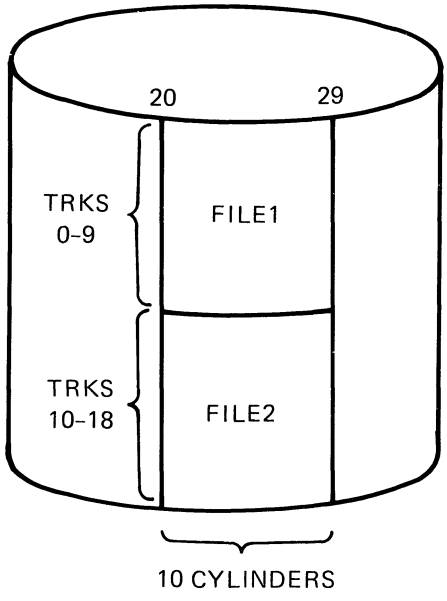


Figure 5.11 - Two Split Cylinder Files

Split Cylinder Track

As with non-split cylinder files, the relative track and number-of-track operands define the beginning and size of the file. One additional operand is required for split cylinder files. The split cylinder track operand specifies the highest numbered track in each cylinder for the file. For example, if a file is to occupy tracks 10-18 of the cylinders allocated, then the split cylinder track operand would be 18.

Suppose you are to set up two files as split cylinder files on a 3330. They are allocated 10 cylinders beginning on cylinder 20. The first file has 100 tracks and will be on tracks 0-9 of each cylinder. The second file has 90 tracks and is on tracks 10-18. Figure 5.12 is an example of the job stream to do this. Note that these are two separate files and not one file with two extents. The files are not required to have the same number of tracks.

Each file is identified with DLBL and EXTENT statements.

```
// JOB SPLITS
// ASSGN SYS008,3330,VOL=VSE011
// ASSGN SYS010,SYS008,VOL=VSE011,SHR
// DLBL FILE1,'FIRST.FILE',80/365,SD
// EXTENT SYS008,VSE011,8,0,380,100,9
// DLBL FILE2,'SECOND.FILE',80/365,SD
// EXTENT SYS010,VSE011,8,0,390,90,18
// EXEC MYPROG
/&
```

Figure 5.12 - Split Cylinder DLBLs and EXTENTs

Another factor is involved in determining the relative track number when you are dealing with split cylinder files. After multiplying cylinder number times number of tracks in a cylinder, you must add the starting track number within the cylinder. In the example in Figure 5.12, FILE1 has tracks 0-9 and FILE2 has tracks 10-18. The files start at cylinder 20. The calculation of relative track is as follows:

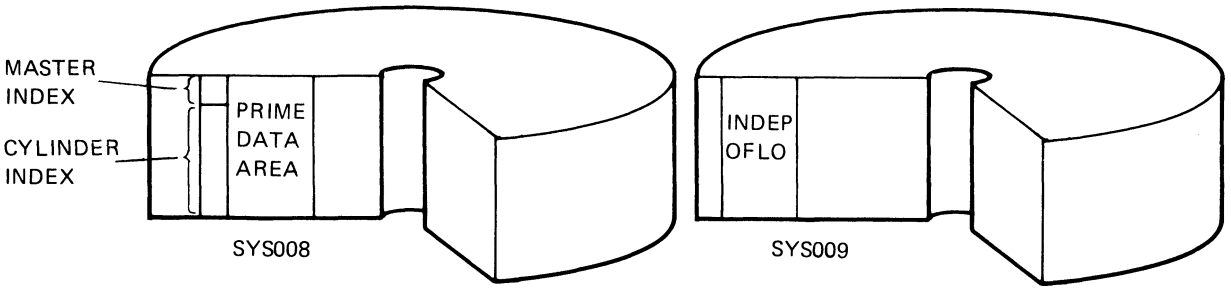
	STARTING CYLINDER		TRACKS PER CYLINDER			STARTING TRACK			RELATIVE TRACK
FILE1	20	X	19	=	380	+ 0	=		380
FILE2	20	X	19	=	380	+ 10	=		390

Label statements for ISAM files differ from those for other files in the codes operand of the DLBL, and in the type operand and sequencing of EXTENT statements.

The DLBL allows two codes for ISAM files, ISC (ISAM Create) and ISE (ISAM Extend). The choice of which to use is determined by what the processing program will do with the file. If the program is to create the file -- that is, open a non-existent file and load records into it -- then the code in the DLBL statement must be ISC. All operations dealing with existing ISAM files require ISE.

In the case of the EXTENT statement, three different type operands apply which must be submitted in a prescribed sequence.

Figure 5.13 illustrates a sequence of DLBL and EXTENTs for creating an ISAM file.




```
// DLBL   ISFILE,'ISAM.FILE',365,ISC
// EXTENT SYS008,VSE015,4,0,190,2      MASTER INDEX
// EXTENT SYS008,VSE015,4,1,192,17     CYLINDER INDEX
// EXTENT SYS008,VSE015,1,2,209,1900   PRIME DATA
// EXTENT SYS009,VSE021,2,3,19,95      INDEPENDENT OVERFLOW
```

Figure 5.13 - ISAM File Structure

Sequence of EXTENT Statements for ISAM

The type and sequence values for ISAM extents are as follows:

- Type 4, Sequence 0 -- Master index
- Type 4, Sequence 1 -- Cylinder index
- Type 1, Sequence 2...n -- Prime data area
- Type 2, Sequence n+1 -- Independent overflow

The statements must be submitted in this order. Note that the prime data area is the only type that allows multiple extents. The others must each be described by a single extent. Note also that there are two Type 4 extents. Both of them are index areas. The sequence code distinguishes between the two. The master index is optional and, if present in a file, must be the first extent submitted. If the master index is not present, the cylinder index extent would be first but its sequence code would still be 1.

The independent overflow area is always optional. When it is present, its extent is always the last to be submitted.

The minimum number of extents an ISAM file may have is two. There must be a cylinder index and a prime data area, and an EXTENT statement is required to define each of these. The EXTENT statement for the cylinder index must precede the EXTENT statement for the data area.

Unit Summary

As a programmer, you are concerned with processing data records associated with the various files kept by your installation. In order to relieve you of the burden of programming at the physical I/O level, VSE provides data management routines to interface between your program and the information it processes. These routines are collectively known as the Input/Output Control System (IOCS) and consist of:

- LIOCS modules that are usually included with your program at link-edit time. One of their functions is to request physical data transfer operations from the
- PIOCS modules. These reside in the supervisor and handle the physical transfer of data to and from I/O devices. LIOCS and PIOCS work under the umbrella of the
- access methods. The access methods allow the user a variety of ways in which to organize, store, and retrieve data.

In order to maintain integrity among the files at your installation, there must be a method for uniquely identifying each file so that programs access only their own data. VSE supports magnetically encoded machine readable file and volume labels for this purpose. Labels identify specific tape, disk, and diskette files and volumes. Labels are optional with magnetic tape, but are mandatory for all DASD and diskette files and volumes. .

The label information area is used to hold label statements read by job control until these statements are needed by your program. OPEN statements issued in your program cause this label information to be read, and cause new labels to be built for output files and old labels to be checked for input files. You have the option of creating any of three types of label information in the label information area: partition temporary labels, partition standard labels, and system standard labels. The OPTION statement is used to specify which of these you want in any given run.

The DLBL and EXTENT statements provide the full range of information required to uniquely identify a specific DASD file and volume. These statements and all their parameters are completely described in the System Control Statements manual. Even after you have had some experience coding DLBLs and EXTENTs, it is always wise to check the structure of your parameters against the specifications in the manual. Further information on disk labels and disk label processing can be found in the *VSE/Advanced Functions DASD Labels* manual.

Mastery Test

You may use the *VSE/Advanced Functions System Control Statements* manual as a reference for the coding problems in questions 6-10. If you make more than three or four errors in your solutions to the coding problems, it is suggested that you reread the material presented here on DLBL and EXTENT as well as reviewing the DLBL and EXTENT material in the *System Control Statements* manual.

1. The maximum length file-ID specification for a disk file is _____ characters.

   a. 17

   b. 24

   c. 44

   d. 48

2. The code operand in the DLBL statement to specify a new ISAM file is _____.

   a. IS

   b. ISC

   c. ISE

   d. ISX

3. _____ files may be defined as split cylinder.

   a. Any and all

   b. Only sequential disk

   c. Only ISAM

   d. Any except VSE/VSAM

4. An EXTENT statement coded with type 4 sequence 1 defines the disk area for a(n) _____.

   a. track index

   b. cylinder index

   c. master index

   d. overflow area

5. The type code for a split cylinder extent is _____.

   a. 8

   b. 4

   c. 2

   d. 1

6. Code a DLBL statement for the input disk file that has PAYMAST as its file name and 'PAYROLL MASTER RECORDS' as its file-ID.

7. Code a DLBL statement for a sequential disk file with OUTPUT as its file name. The file-ID is 'NEW FILE.NUM7', and it should have a retention period of 30 days.

8. Code the label statements required to create a sequential disk file named SDFILE. The identification for the file is 'VSE.TEST.FILE'. The file is on SYS009. It is to begin on track 8 of cylinder 14 on 3330 pack JBC129, and will be allocated the remainder of that cylinder plus the following 8 full cylinders.

9. Code the label statements for two split cylinder files. They have cylinders 100 through 105 on 3330 pack VSE123. Logical device address is SYS007. FILE1 has tracks 0-10, and FILE2 has tracks 11-18. Use the default file-ID.

10. Code the DLBL and EXTENT statements for the following split cylinder files on volume VSE009.

    - FILE1 is on SYS001. Its file-ID is 'SPLIT.FILE.SYS001'. FILE1 has 60 tracks of 3330 space, occupying tracks 0-5 starting on cylinder 10.
    - FILE2 is on SYS002, and its file-ID is 'SPLIT.FILE.SYS002'. This file occupies tracks 6-12 of the same cylinders as FILE1.
    - FILE3 is on SYS003, with a file-ID of 'SPLIT.FILE.SYS003'. FILE3 has tracks 13-18 of the same cylinders.

Solution

1. c
2. b
3. b
4. b
5. a
6.
```
// DLBL PAYMAST,'PAYROLL MASTER RECORDS'
```
7.
```
// DLBL OUTPUT,'NEW FILE.NUM7',30,SD     or
// DLBL OUTPUT,'NEW FILE.NUM7',30
```

   Either of these is right. If the code operand is omitted, as is the case in the second solution, SD is assumed.

8.
```
// DLBL   SDFILE,'VSE.TEST.FILE'
// EXTENT SYS009,JBC129,1,0,274,163
```
9.
```
// DLBL   FILE1
// EXTENT SYS007,VSE123,8,0,1900,66,10
// DLBL   FILE2
// EXTENT SYS007,VSE123,8,0,1911,48,18
```
10.
```
// DLBL   FILE1,'SPLIT.FILE.SYS001',,SD
// EXTENT SYS001,VSE009,8,,190,60,5
// DLBL   FILE2,'SPLIT.FILE.SYS002'
// EXTENT SYS002,VSE009,8,,196,70,12
// DLBL   FILE3,'SPLIT.FILE.SYS003'
// EXTENT SYS003,VSE009,8,,203,60,18
```

    There are several possible variations for this solution, so yours need not be exactly like the one shown here.

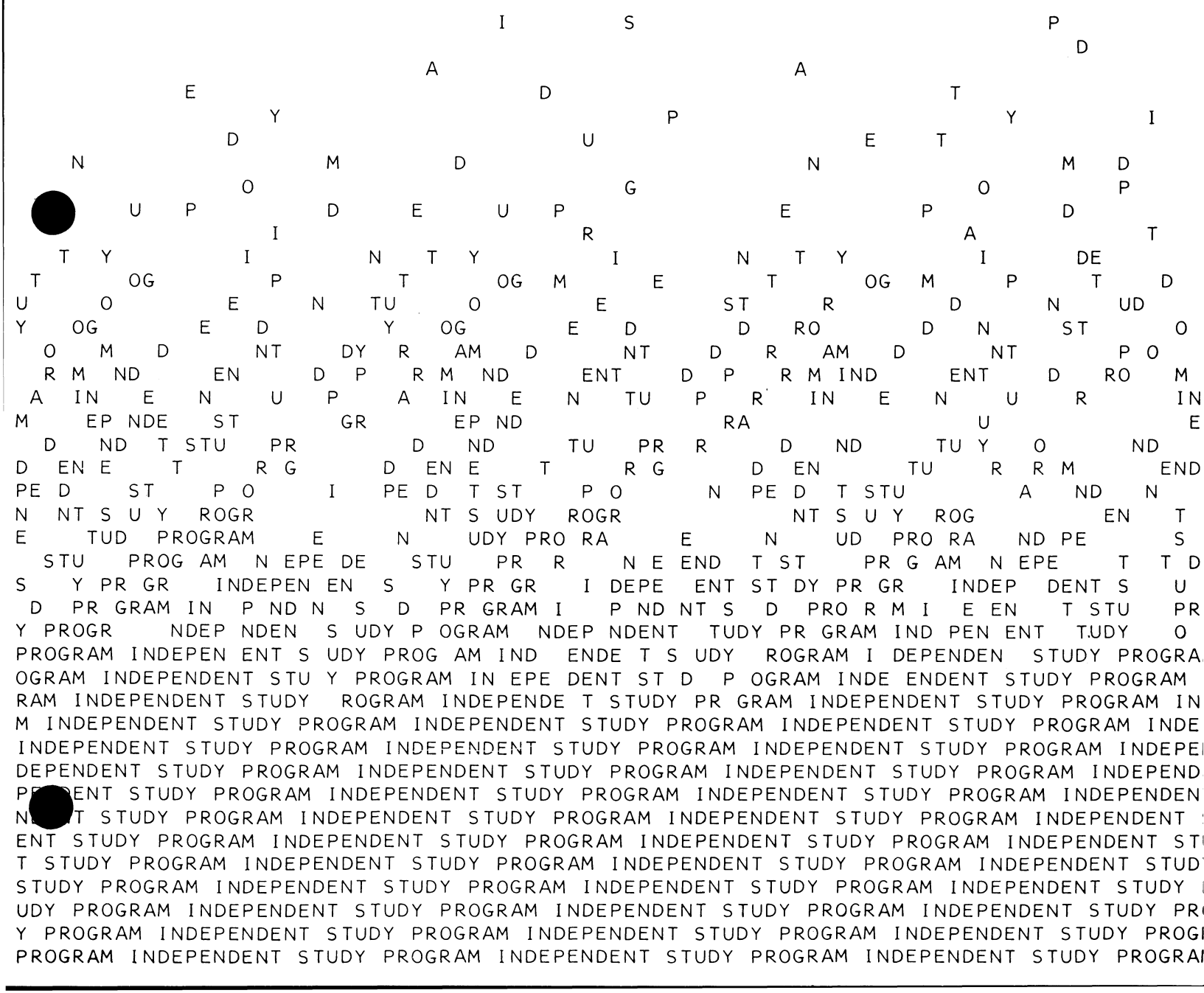
    In the DLBL statement, the code SD may or may not be coded since the default is SD. It is important that you remember, though, that split-cylinder files are sequential. In the EXTENT statement, the extent sequence number may be coded or omitted.

Computer Exercise

Do Computer Exercise 6 at this time. It is not necessary to wait for the results before going on to the next Unit.

Unit

6



Unit 6 (Optional):

Data Management: Tape Labels

Introduction

This Unit is optional so that those of you who do not have access to installations that support magnetic tape as a storage medium may bypass this material. If you will not be working with magnetic tape, or if your installation uses only unlabeled tapes for scratch use, proceed to Unit 7. If you will be using labeled magnetic tapes, continue here.

The concepts of tape label processing are similar to those of DASD label processing. Only one job control statement is used, the TLBL, which you will learn to code in this Unit.

Objective

Upon completing this Unit, you should be able to:

- Describe the layout of file and volume label information on magnetic tape.
- Recognize the different types of label identifiers and be able to describe their use.
- Describe the responsibilities involved in checking non-standard and user-standard labels.
- Code the TLBL statements to create tape labels on output files and check them on input files.

Materials Required

Study Guide (SR20-7300)

The following VSE reference material:

VSE/Advanced Functions System Control Statements (SC33-6095)

Tape Label Processing

The use of labels on magnetic tape storage volumes is completely optional. Unlike the three-dimensional structure of DASD storage, tape is flat and two-dimensional. It can only be processed sequentially. If a file exists on a reel of tape, it can be found by simply scanning the tape.

Files that are not to be retained, such as those used as "work files" and having no value after termination of the job in which they are created, do not have to be labeled. Those files that are to be retained for later retrieval and processing, however, should be considered for labeling. This is to ensure that the correct file is mounted when your program calls for it, and that unauthorized programs not have access to your file. The use of tape labels provides a degree of file security. Likewise, by ensuring that a correct file is mounted for updating, labels help to maintain data integrity.

Label Identifiers

In order for IOCS to be able to perform label checking, labels must be identifiable and must have a standard format. Standard tape labels are 80 bytes in length, and are located at the beginning and end of a data file.

Standard labels have label identifiers: codes that identify the different types of labels. The identifier code appears in the first four positions of the label. The code that identifies a volume label is VOLn. A file header label is identified by the code HDRn. A trailer label is written at the end of the file and may be specified either as EOVn (end of volume) or EOFn (end of file). In all of these identifiers, n is a single digit value from 1 to 8 specifying the number of the label. It is most often 1.

Label Volume Layouts

Single File Volumes

In most instances, a tape file will be contained on one tape volume (one reel of tape). In addition, a tape volume will usually contain a single file. The structure of a single file on a single volume is called a single file volume. Its format is illustrated by point Ⓐ in Figure 6.1. Notice that the labels are separated from the data records by tape marks, and that the EOF1 label is followed by two tape marks. Two successive tape marks indicate to IOCS that there are no more files on the volume.

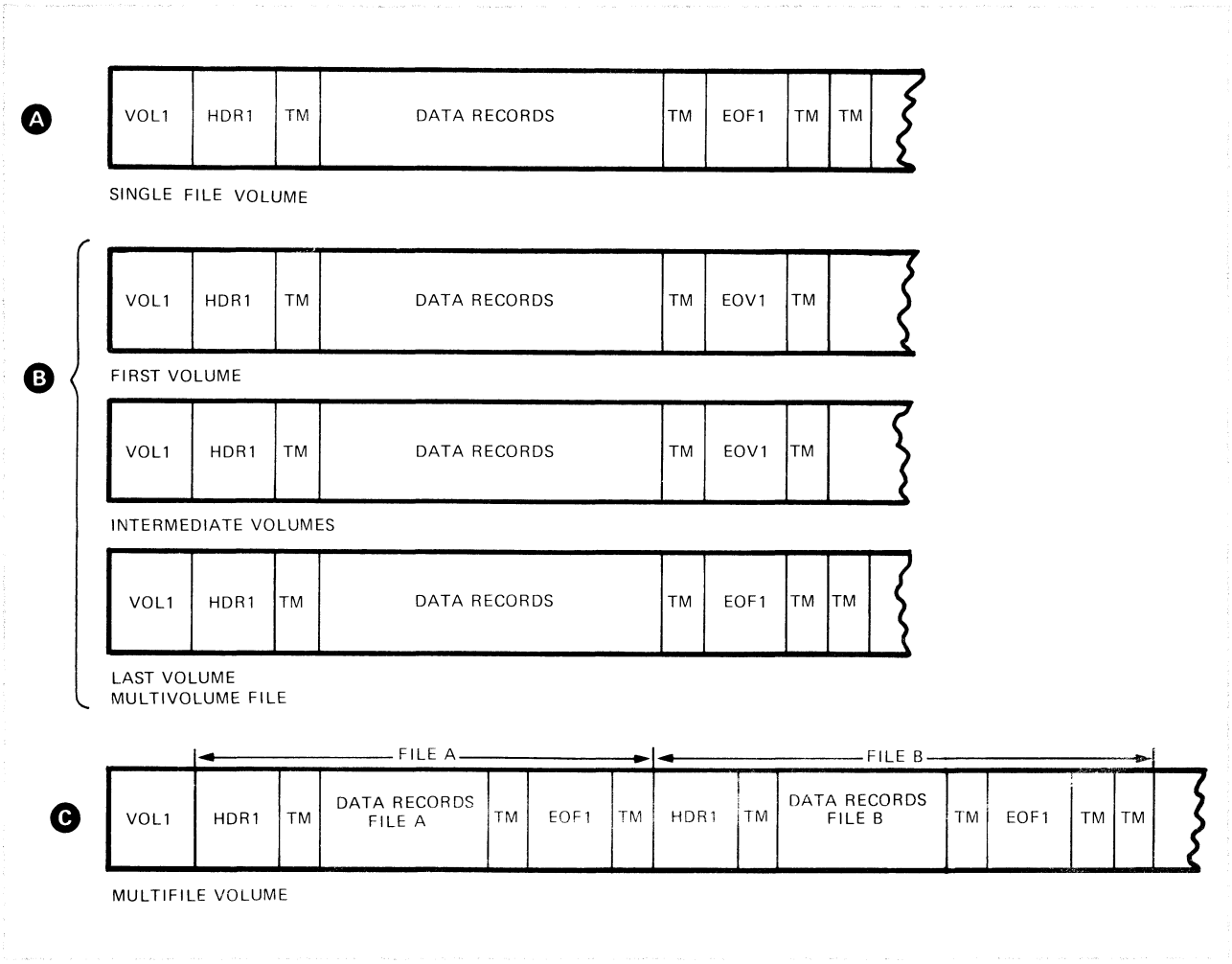


Figure 6.1 - Tape Volume Layouts

Multivolume Files

When a file extends beyond one volume, it becomes a multivolume file. The format of a multivolume file is shown by point Ⓑ in Figure 6.1. Note that the trailer labels for the first and intermediate volumes are identified by EOV1. Those trailers are end of volume labels. In a multivolume file only the last volume has an EOF1 trailer label, which signifies end of file. Also note that only the last volume has two tape marks following the trailer.

Multivolume files are only practical if they are labeled. Although you could write many volumes of unlabeled output, the only way to check multiple volumes of unlabeled input is for your program to provide the required processing logic in its end of file routine.

Multifile Volumes

A multifile volume is one that contains more than one data file. Point Ⓒ in Figure 6.1 illustrates the layout of a multifile volume. The number of files allowed is only limited by the physical capacity of the tape reel. Each data file includes its own header and trailer labels. The volume label appears only at the beginning of the tape, preceding the first file. Note that the trailer for the first file is followed by one tape mark, while the trailer for the last file is followed by two tape marks.

The activities involved in creating labels for tape output files and in checking labels for tape input files are similar to the same activities on DASD volumes.

Figure 6.2 illustrates the flow of information that takes place at OPEN time.

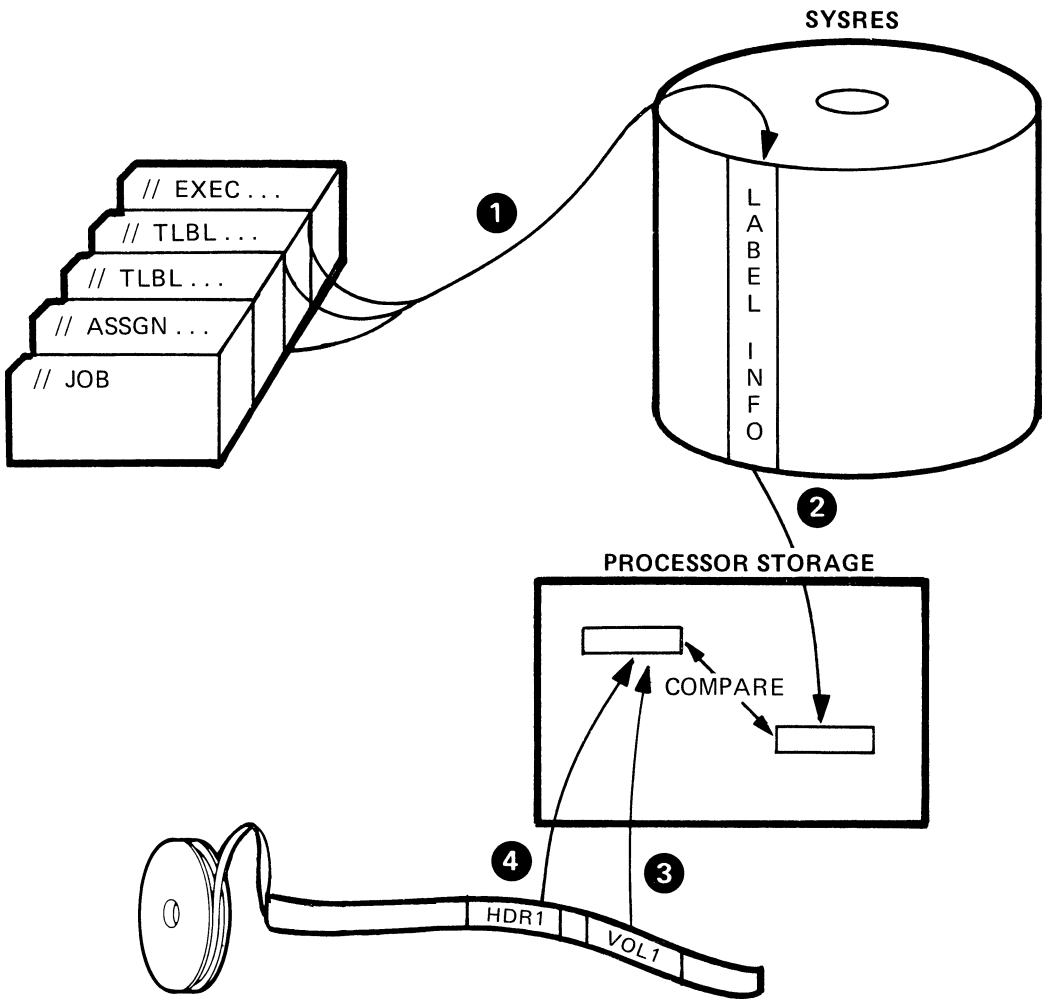


Figure 6.2 - Tape Label Processing

1. When your job stream is read, job control checks syntax and writes your TLBLs to the label information area for use at OPEN time.
2. The file name on the OPEN statement in your program is used as a search argument to find the appropriate TLBL in the label information area. The first TLBL found whose file name matches the name in your OPEN is brought into processor storage.
3. The volume label is read and checked. If the serial number does not match your request, a message is sent to the operator and processing halts until the proper reel is mounted.

4 If this is the proper volume, the file label is read and checked against your TLBL data. If the file-ID from the label matches the file-ID on your TLBL, your program will be allowed to process the file. Otherwise, a message is sent to the operator who must either locate and mount the proper tape or cancel the job.

Output Label Processing

The first two steps are the same as for input processing. The tape is then checked for an existing file label. If one is found, its expiration date is checked to see if the file is current. If the expiration date has not yet been reached, the tape cannot be used and a message is sent to the operator.

If the date has expired, or if the tape has no label, the tape will be used. The information from your TLBL statement is used to create a label for this tape.

A new reel of tape, as it comes from the supplier, does not contain any labels. The Initialize Tape utility program is available with VSE to write volume labels on tapes in preparation for their use.

The Volume Label

The volume label identifies the tape reel. It contains information that pertains to one specific reel of magnetic tape. The volume label itself is an 80 byte record differentiated from other records on the tape by the code VOL1 in its first four character positions.

Figure 6.3 shows the layout of a tape volume label. Up to eight volume labels may be specified on a reel, although VSE checks only the first of these and bypasses the rest. The additional labels allow for full compatibility with OS systems. The volume label format is the same as that used for disk.

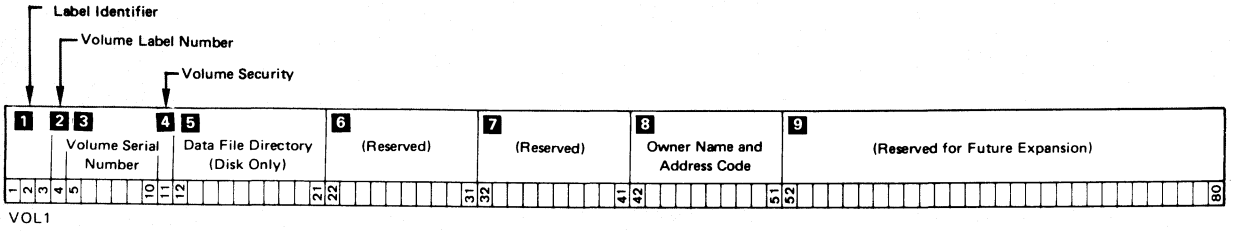


Figure 6.3 - Volume Label Format

1, 2 VOLn where VOL identifies this as a volume label and n is a single digit from 1 to 8 specifying the number of the volume label. This code is most often 1.

3 The volume serial number is a six-digit field containing the number assigned to the volume. This number should agree with the visible external label, if present. Thus the tape handler and the system read the same information but from different sources.

4 - 9 These fields are bypassed by the VSE label processing routines. Field 8 provides space for an owner identification. In the event that external tape labels are missing, the internal tape labels can be printed to identify the reel. It is the user's responsibility to set up this field.

The bulk of the space in the tape volume label is not used by DOS/VSE and is reserved for future expansion.

There are three types of file labels: *standard, user-standard,* and *non-standard.* They all have the same purpose, but label processing is different in each case.

Figure 6.4 shows the format of the standard file label. Fields in the standard label are generally taken from operands in the TLBL statement.

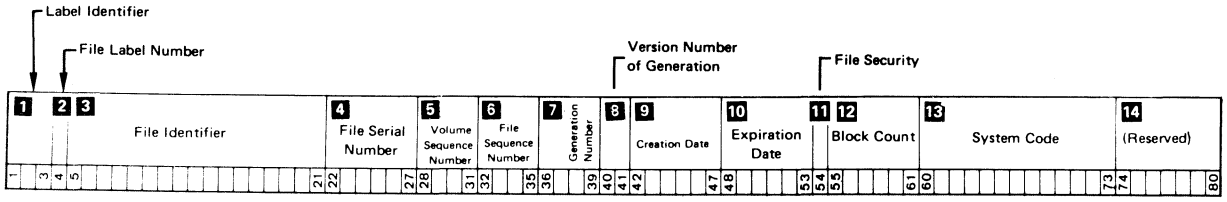


Figure 6.4 - Standard File Label Format

1,**2** HDRn where HDR identifies this as a file label, and n is a single digit from 1 to 8 specifying the number of the label.

3 The file identifier is a 1 to 17 character name taken from the file-ID or file name operands of the TLBL statement.

4 This field provides a numeric identification for the file, and should contain the volume serial number from the VOL label of the first or only volume of the file.

5,**6** Sequence number fields to identify the order of files and volumes in multifile, multivolume situations.

7,**8** Generation and version numbers to provide a detailed identification of multiple editions of a file, all of which may have the same file-ID and creation date.

9,**10** Date fields to identify the file and to indicate how long the file is to be considered current.

11-**14** These fields have no correspondence to TLBL operands. Information on their use can be found in the manual *VSE/Advanced Functions Tape Labels* (SC24-5212).

Like standard labels, user-standard labels are 80 characters long and have an identifier in positions 1 through 4. See Figure 6.5. You determine what information goes into the user-standard labels. IOCS will write the label for you after you have assembled it in your program.

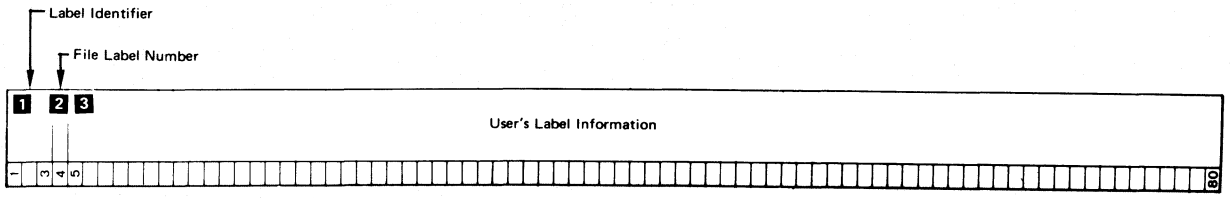


Figure 6.5 - User-Standard Label

No job control statement is associated with user-standard labels. The information contained in these labels must be either program generated or read from some input file, such as a card or diskette file.

On input, IOCS will read the user-standard label into processor storage, but will do no checking. This is because there is no standard information that IOCS can check. As it is the programmer's responsibility to create the label information on output, so it is also the programmer's responsibility to check the label on input.

There may be one to eight user-standard labels for a given file. They are recognized by the label identifiers UHL1 through UHL8 (headers) and UTL1 through UTL8 (trailers) in positions 1 through 4. The label records are read and passed to the program one at a time. The user's label checking routine examines each label, and either accepts or rejects the file based on that examination.

User-standard labels follow the standard header and trailer labels as shown in Figure 6.6. It is not possible to have user-standard labels without also having standard labels for a file.

Figure 6.6 - Volume Layout With User-Standard Labels

Non-Standard Labels

Non-standard label processing is usually done with tape files that were created under a system other than VSE, and that have totally different label formats.

Non-standard labels are free-form. They may be any length, they have no formatted data fields, and there is no specific identification code. Non-standard labels are strictly the programmer's responsibility. Figure 6.7 shows some possible formats of non-standard labels.

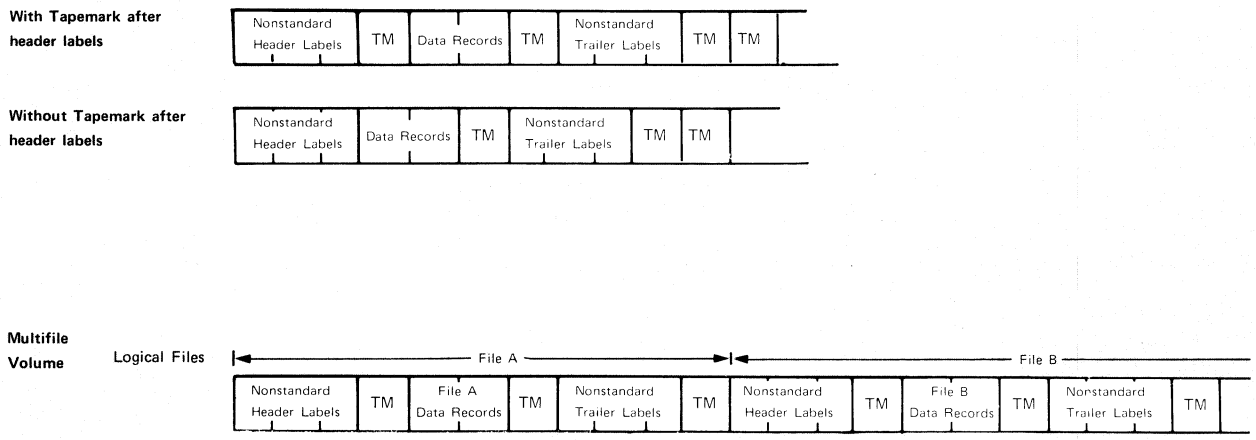


Figure 6.7 - Tape Files With Non-Standard Labels

Label Sets

A label set can be defined as all those labels with the same label identifier. The volume label set, for example, consists of all the VOLn labels on a particular tape. A summary of standard tape labeling is given below.

1. Standard volume label set consists of:

   VOL1 label (required)

   VOL2...VOL8 labels (optional)

2. Standard file header label set consists of:

   HDR1 label (required)

   HDR2...HDR8 (optional)

3. Standard file trailer label set consists of:

   EOV1, or EOF1 (required)

   EOV2...EOV8, or EOF2...EOF8 (optional)

4. Standard user header label set consists of:

   UHL1...UHL8 (optional)

5. Standard user trailer label set consists of:

   UTL1...UTL8 (optional)

The TLBL Statement

The TLBL is the means by which you specify tape labeling information to VSE. Its fields are described below.

File Name

As on the DLBL statement, file name is an arbitrary one to seven character designation that relates the TLBL to the OPEN and CLOSE statements in your program. The name in your OPEN and CLOSE statements must match the TLBL file name field in order for a particular TLBL statement to be associated with a given file.

The COBOL programmer must keep in mind that the value of SYSnnn or the EXTERNAL NAME specified in the COBOL ASSIGN clause has to be used for the file name on the TLBL statement.

File-ID

The file identifier is a 1 to 17 character descriptive name for the file. On input, the TLBL file-ID is compared with the file-ID in the label. If they agree, IOCS recognizes the file as being the one you want. On output, the TLBL file-ID is used to create the file-ID in the label.

If this operand is not coded on the TLBL statement, no checking is done on input tape files. On output, the file name parameter is used to create the file-ID field in the label.

File-ID must be enclosed in single quotes on the TLBL statement.

Date

The date operand in the TLBL statement has different meanings for input and output files. This is because there are two date fields in the file label itself: fields 9 and 10 in Figure 6.4. For an input file, the date in the TLBL statement refers to creation date, the date the file was written. For an output file the TLBL date operand refers to expiration date, the date on which the file is no longer current. See Figure 6.8.

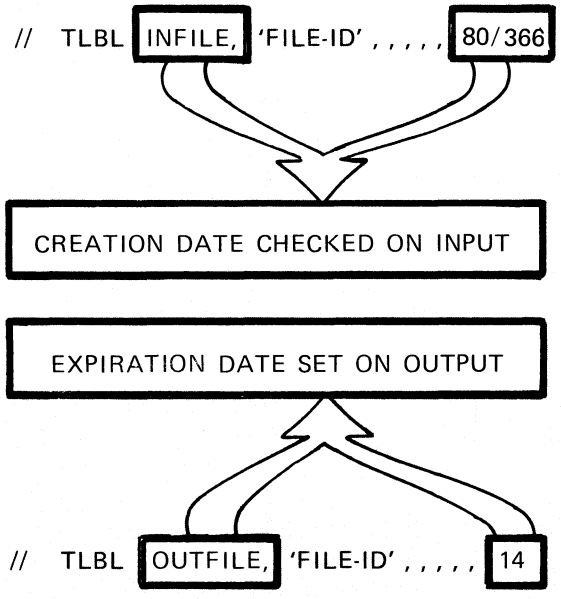


Figure 6.8 - The DATE Field

Input Processing

The TLBL date parameter, if coded, must be in the form yy/ddd (where yy=0-99, ddd=1-365). IOCS compares this TLBL date against the *creation date* in the file label. In order for the tape to be accepted for processing, these two dates must match. If the TLBL date operand is omitted, the creation date in the label is not checked.

The expiration date in the label is not checked on input except with older versions of DOS job control.

Output Processing

The TLBL date parameter, if coded, may be either in the form yy/ddd or may be written as a four digit number 0-9999 specifying a *retention period* in days. It is the *expiration date* field in the label that is affected by the TLBL date operand.

When a new file is created, the expiration date in the label is set directly from the TLBL specification if the operand is coded yy/ddd, or is set by *adding* the retention period to the current date in the supervisor to arrive at an expiration date. If the TLBL date is omitted, a zero retention period is assumed.

The creation date field in the label is taken not from the TLBL statement when an output file is created, but rather from the current date in the supervisor.

Figure 6.8 shows a TLBL statement coded to check for a creation date of December 31, 1980. Since 1980 is a leap year, December 31 is the 366th day. Figure 6.8 also shows a TLBL statement that specifies a retention period of 14 days. If the latter statement were used on January 10, 1981, the HDR1 label for the file would have a creation date 81010 (taken from the current date in the supervisor), and an expiration date of 81024.

Note that when the TLBL date operand is coded as yy/ddd, the "/" does not appear in the label.

File Serial Number

This one to six character alphanumeric field is used to maintain the relationship between file and volume. It contains the volume serial number taken from the first (or only) reel of a file.

If omitted for an input file, IOCS does no checking and will accept whatever file serial number is in the HDR label. If omitted on output, IOCS writes the volume serial number of the first (or only) volume into the HDR label.

Sequencing Parameters

The following two parameters are used for handling either multivolume files or multifile volumes. They have no significance for single file volumes, and you may bypass them if they do not apply to your installation.

Volume Sequence Number

The volume sequence number is used to insure that the volumes of a multivolume file are processed in the proper order. Normally, the first volume of the set has volume sequence number 0001. The second volume is 0002, the third is 0003, etc. Figure 6.9 illustrates a three-volume inventory file, showing the file-ID, file serial, and volume sequence number that would be in each HDR1 label. The file-ID and file serial remain constant.

Note that the *volume serial* (from the VOL label) is different from the *file serial* (from the HDR label) on all but the first reel. This is as it should be. It is the correspondence of file

serials on subsequent reels to the volume serial on the first reel that maintains the file-volume relationship.

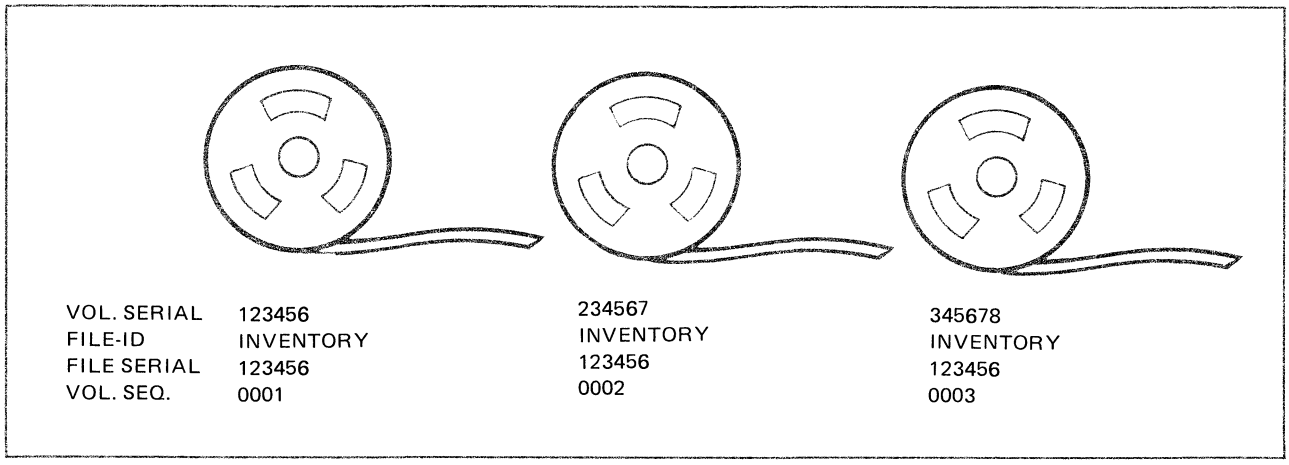


Figure 6.9 - A Multivolume File

If the volume sequence number is not specified in the TLBL statement for an output file, IOCS will assume a value of 0001 for the first volume and will automatically increase the number by one for each succeeding volume.

You must include the volume sequence number operand in a TLBL statement only if you choose to begin processing the file with a volume other than the first. If, for example, a utility program is to print only volumes two and three of the file shown in Figure 6.9, the number 2 must be coded in the volume sequence number operand of the TLBL statement. If that operand were omitted, IOCS would do no checking and process whatever volume were mounted.

The volume sequence number is called the file section number for ASCII file processing.

File Sequence Number

The file sequence number is used with multifile volumes, and is the number of the file on the volume. If there are four files on a volume, the file sequence number of the first will be 0001, and that of the last will be 0004.

File sequence number is used by IOCS along with serial number and volume sequence number to insure that the proper file has been located on a multifile volume.

Generation/Version Numbers

When processing tape, a new file is created in each update run. Usually, at least two prior versions of the file are kept for backup. These three generations of the file are often referred to as the grandfather, father, and son.

Since these several files have the same file-ID, it is necessary to check more than that field to be sure you are processing the correct generation of the file. To distinguish between tape files having the same file-ID and even the same date fields, the generation and version number fields are provided.

These operands are optional with default values of 0001 and 01 respectively. When used, they should be numbered in sequence. This is to maintain a logical order for the files. If omitted on input, these fields are not checked.

Generation number is a four digit field that can be used to identify an edition of a file. You could, for instance, use the generation number to indicate the month in which a file is created. In this instance the valid codes would be 0001 through 0012. Version number, a two digit field, can be used as a subdivision of generation. As in the instance stated for generation number, version number could be used to indicate the week in which a file is created. In that case, the codes used would be 01 through 52. The maximum value of code is 99. The specification for generation 12 version 7 of MYFILE is

```
// TLBL MYFILE,'MYFILEID',80/029,,,,12,7
```

In this example, four successive commas are required following the date to indicate that checking is not to be done for the file serial number, volume sequence number, and file sequence number fields.

Complete details on all aspects of tape labels and tape label processing are in the manual *VSE/Advanced Functions Tape Labels*.

Using the TLBL

The TLBL statement must precede the EXEC statement for the phase or procedure that will process the tape file. Job control stores TLBL statements in the label information area for the duration of the job. In the case of a job consisting of multiple jobsteps, the label statements read in one jobstep will be overlaid by the label statements of subsequent jobsteps. For this reason, place the TLBL statements (and DLBLs and EXTENTs as well) for the entire job in the first jobstep.

The following set of job control statements shows the proper placement of TLBLs in a job stream.

```
// JOB RUNTAPE
// ASSGN SYS006,TAPE
// ASSGN SYS007,TAPE
// TLBL TAPEIN,'MASTER.FILE',80/135
// TLBL TAPOUT,'MASTER.FILE',30
// OPTION LINK
   INCLUDE
* OBJECT DECK *
/*
// EXEC LNKEDT
// EXEC
* DATA CARDS *
/*
MTC REW,SYS006
MTC REW,SYS007
// EXEC FILEPRNT
/*
// ASSGN SYS006,SYS007
// EXEC MONTHLY
* DATA CARDS *
/*
/&
```

Reading Assignment

In *VSE/Advanced Functions System Control Statements* manual read the section that discusses the TLBL statement.

Unit Summary

The use of labels on volumes of magnetic tape is completely optional. If you are using tape for temporary storage of data within a job or between job steps, for example, it is probably not necessary for you to go to the trouble of labeling that tape. However, if you are maintaining permanent data files on magnetic tape (customer records, name and address files, transaction files, history files, etc.) it becomes important to label your tapes as a means of insuring data security and data integrity.

Tape files can be constructed in one of three ways: single file volumes, multifile volumes, and multiple volume files. There are configurations of standard tape labels to handle each of these file and volume constructions.

It is the LIOCS routines for tape label processing that perform the operations required for checking tape labels on input and for creating them on output. LIOCS does not handle anything but standard labels. It is the user's responsibility to take care of the processing requirements for user-standard or non-standard labels.

The TLBL job control statement is used to specify tape label information to VSE. Complete specifications for this statement are in the System Control Statements Manual, and it is there that you should look to resolve any questions as to the coding of TLBL parameters.

Mastery Test

You may use the *System Control Statements* manual as a reference for the coding problems that follow.

1. Your program is to process an input tape file with standard labels. The name given the file in the program is INVMAS. When the file was created, the file-ID assigned was 'INVEN MASTER FILE'. Code the appropriate TLBL statement for this file.
2. Code the label statement for the input tape file INVMAS whose file-ID is 'INVEN MASTER FILE' which was created on February 3, 1980.
3. Assume you have a program that is run several times each day and creates a tape file in each run. The file name is INVEN and the file-ID is 'UPDATE INVEN'. Generation number is used to indicate the month of creation and version number indicates the number of the daily run in which the file is created. You have a program that prints a report using one of these files as input. Prepare the TLBL statement required to process the third file created on February 10, 1980.
4. Code the JCL statement to do the job named EXER16, which is a link-and-execute run using an object deck. The program reads a labeled tape from SYS008 and prints it on SYSLST. The tape file is called INTAPE, and was created on April 21, 1980 as generation 21 of version 9. It was written on volume 234567. 'BATCH TRANS FILE' is the file-ID. Creation date, version and generation numbers, and volume serial should be checked. Assign INTAPE to any available tape drive, and check the volume serial number. Assume that SYSLST is already assigned.

Solution

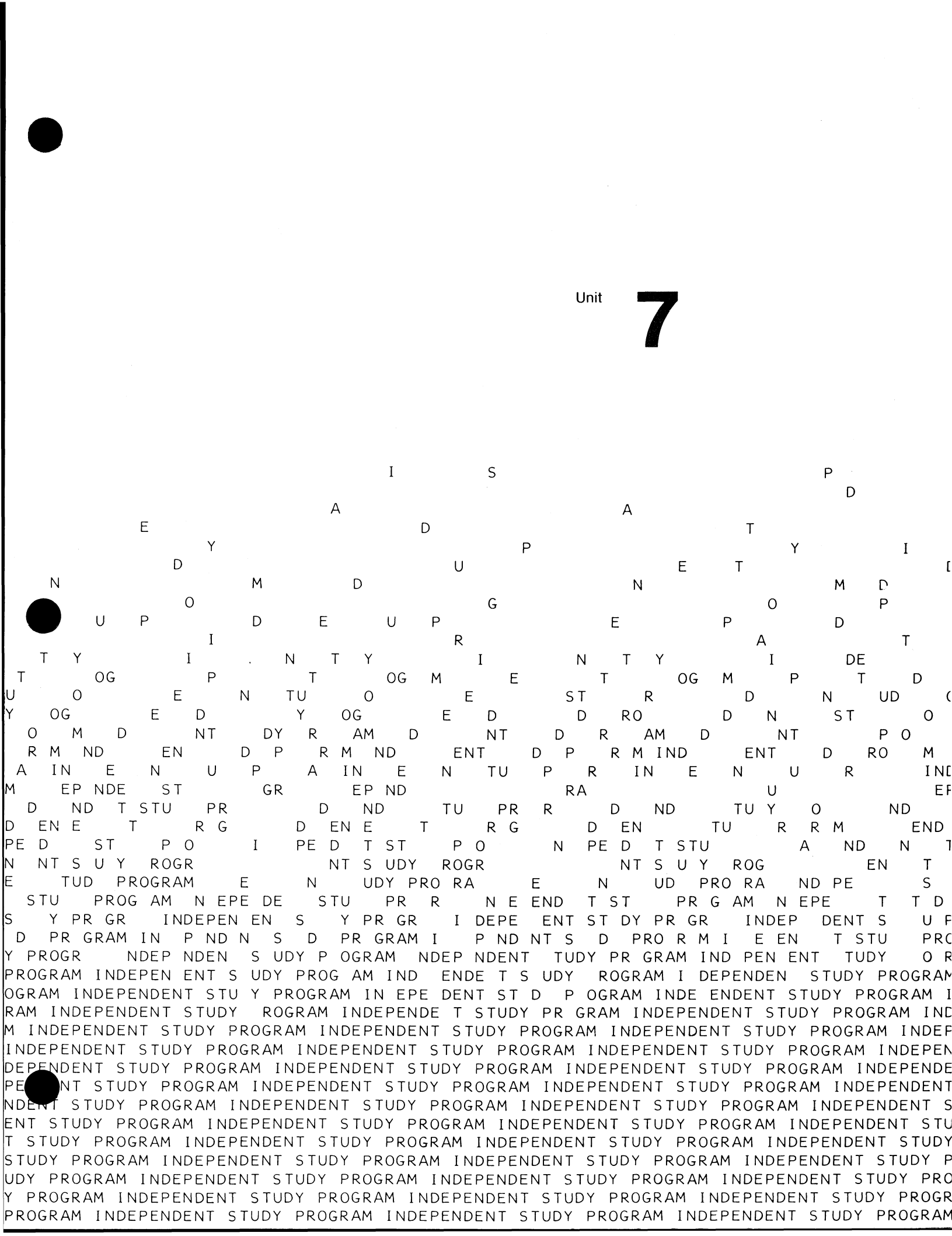
1. `// TLBL INVMAS,'INVEN MASTER FILE'`
2. `// TLBL INVMAS,'INVEN MASTER FILE',80/034`

   With a TLBL statement coded as shown above, IOCS checks not only for the correct file-ID but also for the specific creation date.

3. `// TLBL INVEN,'UPDATE INVEN',80/041,,,,2,3`
4.
```
// JOB EXER16
// OPTION LINK
// ASSGN SYS008,TAPE,VOL=234567
// TLBL INTAPE,'BATCH TRANS FILE',80/112,234567,,,21,9
 INCLUDE
*** OBJECT DECK ***
/*
// EXEC LNKEDT
// EXEC
/&
```

Unit

7



Unit 7:

Using the Libraries

Introduction

The VSE libraries exist in both system and private forms. System libraries are defined as those residing within the SYSRES extent, while private libraries are those residing outside that extent. Your installation may have any number of private libraries available.

The use and function of each library is summarized below.

- Core Image Library (CIL) - contains executable program phases produced by the linkage editor.
- Relocatable Library (RL) - contains compiler object module output to be used as input to the linkage editor.
- Source Statement Library (SSL) - contains source language statements, macros, and pre-edited macro definitions in the form of books which can be used as input to a compiler.
- Procedure Library (PL) - contains sets of commonly used job control and link-edit control statements. Inline SYSIPT data can also be included in the PL.

This Unit will cover the programs involved in maintaining and servicing the libraries. In addition, you'll learn how to invoke cataloged procedures using the overwrite function.

Objective

Upon completing this Unit, you should be able to:

Assignment 1

- List the functions available with the librarian programs.
- Code the statements required for cataloging, deleting, and renaming elements in the various libraries.
- Code the statements required for updating books in the Source Statement Library.
- Code the job control statements required to access private libraries.
- Code the control statements required to display library directories, and to display and punch library contents.

Assignment 2

- Code the control statements required to temporarily modify a procedure at execution time.
- Code the control statements to catalog and execute partition-related procedures.
- Describe the use of SYSIPT data within a cataloged procedure.

Materials Required

Study Guide (SR20-7300)

The following VSE reference material:

VSE/Advanced Functions System Management Guide (SC33-6094)

VSE/Advanced Functions System Control Statements (SC33-6095)

Assignment 1 - The Librarian Programs

The librarian programs give you the functions necessary to manage the VSE system and private libraries. There are three basic librarian functions:

Maintenance - adding, updating or deleting library entries.

Service - translating to some form of output.

Copy and Reorganization - restructuring all or parts of libraries.

Application programmers are primarily concerned with maintenance and service functions, while copy and reorganizing functions are generally in the domain of the system programmer. It is the CORGZ program that performs copy and reorganizing operations. CORGZ will not be covered here. The *VSE/Advanced Functions System Management Guide* has information on its use.

Maintenance Functions

The MAINT program provides you with the following five maintenance functions:

Catalog

To add members to libraries

Delete

To remove members from libraries

Condense

To reclaim space "lost" by certain delete operations and make it available for new library members

Rename

To alter the name of an existing library member

Update

To modify the contents of books in the SSL

Service Functions

These include capabilities to display library directories, and to display and punch library contents. The programs in this group and the libraries they service are illustrated in Figure 7.1.

FUNCTION	PROGRAM	LIBRARY AFFECTED
DISPLAY MEMBER	CSERV	CIL
PUNCH MEMBER	RSERV	RL
DISPLAY & PUNCH MEMBER	SSERV	SSL
	PSERV	PL
DISPLAY DIRECTORIES	DSERV	ALL
SERVICE EDITED MACROS	ESERV	SSL*

*ONLY THE E SUBLIBRARY OF THE SSL IS AFFECTED

Figure 7.1 - Service Program Functions

Notice in the figure that the first letter of each program identifies the library which that program services. The CSERV program handles the CIL (system or private), RSERV handles the RL (system or private), and so forth. This type of naming convention is common with the librarian programs and control statements.

Librarian Control Statements

The format of control statements used with the librarian programs is shown below. The only exceptions to this general structure are the statements used to update a book in the SSL.

- Column 1 = Blank
- At least one blank between operation and operand fields
- Column 71 = Last usable column
- No continuation statements allowed

Private Libraries

As was explained in Unit 1, individual users or user departments at an installation can maintain phases, modules, or books in separate private libraries that are independent of each other and of the system libraries. Private libraries can be accessed by a program executing in any partition, as long as the libraries are assigned to that partition. The VSE librarian programs are used to maintain and service private as well as system libraries.

Accessing Private Libraries

The librarian control statements for cataloging and retrieving data to or from private libraries are the same as those for the system libraries. Private libraries, like other disk files, must be made known to the various programs that access them. This is accomplished using DLBL and EXTENT statements in conjunction with the LIBDEF and/or ASSGN job control statements. The DLBL provides the name and a description of the library. The EXTENT describes the exact location of the library on the DASD. The ASSGN and/or LIBDEF statements make available to a given partition the libraries defined by the DLBL and EXTENT statements. The ASSGN statement can be used with any file (except a private Procedure Library) while the LIBDEF statement is used for **libraries** only. The LIBDEF statement, however, provides greater flexibilty with fewer restrictions on accessing libraries than does the ASSGN statement.

The ASSGN Statement

When using the ASSGN, DLBL and EXTENT statement to define private libraries you will need to provide the following information:

- the file name in the DLBL statement; and
- the system logical unit for the ASSGN and EXTENT.

From the material you covered on disk labels, you should recall that the DLBL file name operand must correspond to the name on the OPEN statement in the program accessing the file. The system logical unit in the ASSGN and EXTENT must also match whatever symbolic name is used in the program. Because the private libraries are accessed by librarian programs, linkage editor, and compilers, the names used in these programs are used in the ASSGN, DLBL, and EXTENT statements.

For private libraries, these are standard. The required file names in the DLBL statement are:

IJSYSSL - private Source Statement Library

IJSYSRL - private Relocatable Library

IJSYSCL - private Core Image Library

The system logical units used in the EXTENT and ASSGN statements are:

SYSSLB - private Source Statement Library

SYSRLB - private Relocatable Library

SYSCLB - private Core Image Library

NOTE: Private Procedure Libraries may not be defined with an ASSGN statement. They may be defined with the LIBDEF statement only.

The assignment for SYSCLB must be permanent. The other assignments may be temporary. Be sure the DLBL and EXTENT statements are before the ASSGN statement when a private Core Image Library (SYSCLB) is used. The control statements that follow illustrate these points.

```
// ASSGN   SYSSLB,cuu
// DLBL    IJSYSSL,...
// EXTENT  SYSSLB,...
// ASSGN   SYSRLB,cuu
// DLBL    IJSYSRL,...
// EXTENT  SYSRLB,...
// DLBL    IJSYSCL,...
// EXTENT  SYSCLB,...
   ASSGN   SYSCLB,cuu
```

The final ASSGN statement is placed as it is because job control performs the open processing on the Core Image Library and requires the information in the DLBL and EXTENT statements for this processing. The other statements may be placed as shown.

When using the ASSGN, DLBL and EXTENT statements, only one private Core Image Library, one private Relocatable Library, and one private Source Statement Library may be assigned at the same time in a given partition.

The LIBDEF Statement

Using the LIBDEF statement not only removes many of the ASSGN restrictions but also expands private library support. Over and above those functions supported by the ASSGN statement, the LIBDEF statement allows you to:

- Have more than one library of a given type defined within a single partition
- Define private Procedure Libraries
- Define private Core Image Libraries temporarily (for the duration of the current job only)
- Perform maintenance and service on system libraries while the corresponding private libraries are still available for other processing.
- Access a private library under a filename that differs from the one specified when the library was created (the file identification however, must always be the same).

Both LIBDEF and ASSGN statements always require DLBL and EXTENT information. The LIBDEF statement must have the DLBL and EXTENT information available when it is processed.

When using LIBDEF you need not adhere to the standard private library names (IJSYSSL, IJSYSRL, IJSYSCL) for the DLBL statement. In addition, you need not specify the system logical unit (SYSxxx) on the EXTENT statement. With LIBDEF, the operating system does not need the SYSxxx specification, instead, the VSE system is capable of determining the physical device address via the volume identification in the EXTENT statement.

The LIBDEF statement contains the following parameters:

Operation	Operands
[//] LIBDEF	{ CL[RL[SL[PL } ,SEARCH=(name,name,...)] [,FROM=name] [,TO=name] [,NEW=name] [,PERM TEMP]

The LIBDEF SEARCH Parameter

The SEARCH parameter in the LIBDEF statement allows you to establish a chain of libraries (library concatenation). Library chaining simply means that multiple libraries of a given type (CIL, RL, SSL,PL) may be made available to a single partition. The LIBDEF SEARCH chain contains a list of filenames that correspond to the filenames in the DLBL statements. The number of libraries which may be chained in the SEARCH parameter depends on a value specified in the VSE Supervisor at supervisor generation. The maximum value in any SEARCH chain is 15 libraries. The position within the SEARCH list determines the sequence in which the libraries will be searched.

Search chains may be defined as permanent (PERM) or temporary (TEMP). For a given private library type you may define both a temporary and a permanent chain. The search order for Relocatable, Source Statement, and Procedure Libraries is:

Temporary Chain
Permanent Chain
System Library

For Core Image Libraries the search order is:

SDL (System Directory List)
Temporary Chain
Permanent Chain
System Core Image Library

The system library is **always** assumed to be the last member of the chain and is searched last for all library types.

A sample LIBDEF statement which defines a library search chain is shown as follows:

```
// DLBL  YOURLIB,...
// EXTENT  ,123456,....
// DLBL MYLIB,....
// EXTENT  ,654321,...
// LIBDEF  CL,SEARCH=(YOURLIB,MYLIB)
```

Note that the search file names correspond to the file names in the DLBL. Each library type requires its own LIBDEF, with a corresponding identifier:

LIBDEF	CL-	(Core Image Library)
LIBDEF	RL-	(Relocatable Library)
LIBDEF	SL-	(Source Statement Library)
LIBDEF	PL-	(Procedure Library)

The SEARCH chain may be used in retrieving phases from a CIL, object modules from an RL, cataloged procedures from a PL, and source statements from an SSL by language translators.

LIBDEF FROM and TO Parameters

In addition to the SEARCH parameter the LIBDEF statement may contain the TO and FROM parameters used when performing librarian functions.

TO=name: Specifies the name of a (target) library used for output, update, delete, or condense activities by:

- MAINT
- Linkage Editor Output
- CORGZ MERGE

FROM=name: Specifies the name of an input (source) library to be used by:

- xSERV
- CORGZ MERGE

An example of a LIBDEF statement used to define a temporary private Core Image Library for linkage editor output is shown as follows:

```
// DLBL PRODCIL,'PRODUCTION/HISTORY CIL',...
// EXTENT ,VOLID2,...
// DLBL TESTCIL,'TEST CIL FOR APARS',...
// EXTENT ,VOLID1,...
// LIBDEF CL,SEARCH=(TESTCIL,PRODCIL),TO=TESTCIL,TEMP
```

It is recommended that you use LIBDEF for all librarian activities. ASSGN should only be used for compatibility with previous releases of DOS/VS or for certain system programming librarian functions which are currently only supported by the ASSGN statement (example: creating a new SYSRES). If you find it necessary to use ASSGN statements in conjunction with LIBDEF, you should be aware of the following restrictions. ASSGN and LIBDEF statements may be used within the same partition for different library types. You may not, however, use ASSGN and LIBDEF statements for the **same** library type within a job step.

LIBDROP and LIBLIST

Two additional control statements which are used in conjunction with LIBDEF are the LIBDROP and LIBLIST statements.

LIBDROP is used to drop (reset) some or all of the library definitions made by the LIBDEF command.

LIBLIST is used to display the currently active library definitions for a particular library type.

The LIBDEF, LIBDROP and LIBLIST statements are described in the "Job Control" section of the *VSE/Advanced Functions System Control Statements* manual. Read the discussion of these statements before continuing.

Also in the *VSE/Advanced Functions System Management Guide* under "Using the System", read "Job Control for Library Definitions".

Librarian Partition Dependencies

When using the CONDS function of MAINT there are some partition restrictions to be considered. Figure 7.2 shows these restrictions as well as the capabilities of shared libraries by specific function.

FUNCTION	SYSTEM LIBRARY	PRIVATE LIBRARY
MAINT CATAL	BG,FG	BG,FG
DELETE	BG,FG	BG,FG
RENAME	BG,FG	BG,FG
UPDATE	BG,FG	BG,FG
CONDL	BG,FG	BG,FG
CONDS	BG (1)	BG,FG (2)
xSERV	BG,FG	BG,FG
LNKEDT (CATAL LINK)	BG,FG	BG,FG

1. FG partitions must be inactive
2. Libraries to be condensed must be dedicated to the partition requesting CONDS

Figure 7.2 Librarian Sharing Capabilities

Invoking the MAINT Program

The maintenance program is invoked by the job control statement:

```
// EXEC MAINT
```

Its functions are initiated by various control statements.

Cataloging

The catalog function adds a module to a relocatable library, a book to a source statement library, or a procedure to the procedure library. You cannot use the catalog function of the librarian to add a phase to the core image library: this is done by the linkage editor.

When a member is cataloged to a library, an entry for the member is placed in the library directory.

The catalog control statements specify the name of the member to be cataloged and, optionally, a change level number.

A change level number is a number associated with the cataloged library member. It is used to indicate the current level of the item most recently cataloged. One purpose it serves is to differentiate that a change was in fact made in a given program. This kind of information is essential for program debugging.

A change level can only be assigned by using an option of the control statement when a member is cataloged.

The MAINT control statements are:

```
CATALR -  Relocatable Library
CATALS -  Source Statement Library
CATALP -  Procedure Library
```

In general, when a member is cataloged to a library under the same name as an existing member of that library, the original member can no longer be retrieved. In addition,

- Any subsequent reference to that name will be to the new member
- No warning message is issued

CATALR

The following job stream would be used to catalog two object decks to a Relocatable Library.

```
// JOB   CAT
// EXEC  MAINT
 CATALR  MOD1
(OBJECT DECK IN SYSIPT)
 CATALR  MOD2
(OBJECT DECK IN SYSIPT)
/*
/&
```

Notice that there is no /* statement following the first object deck. The object deck itself contains an END card that is recognized as an end of deck indicator by MAINT. When the /* is encountered, it signifies end of data to the MAINT program. MAINT looks for its input (CATALR statements and object decks) on the SYSIPT device.

The cataloged modules can be retrieved from the RL and made part of an executable phase in the CIL in two ways:

1. Through the use of INCLUDE statements in a link-edit run
2. By being autolinked during a link-edit run

CATALS

When CATALS is used to catalog into a Source Statement Library, provision must be made on the CATALS statement to identify the sublibrary involved.

The SSL is composed of several sublibraries, each of which is defined by a single alphameric character. Some of these sublibraries are used for particular purposes, as indicated below.

SOURCE STATEMENT LIBRARY SUBLIBRARIES

A - Z, 0 - 9, #, $, and @

RESERVED LIBRARIES:

A - I and Z

A: ALC BOOKS

C: COBOL BOOKS

E: Edited Assembler MACROS

D,F: TP Applications

When source statements are stored in the SSL, they are stored in compressed form, that is, blanks are removed. They are expanded to their original format when they are retrieved.

CATALS specifies the sublibrary and book name for the cataloged statements. The source statements to be cataloged must be enclosed between BKEND statements. The example below illustrates this:

```
// JOB    SOURCE
// EXEC   MAINT
   CATALS K.MYBOOK
   BKEND
     .
     .
   Source Statements
     .
     .
   BKEND
/*
/&
```

Here, the CATALS statement has a K.MYBOOK operand to indicate the source statements are to be cataloged in the K sublibrary. MYBOOK is the book name in the sublibrary.

The BKEND statement is required to precede and follow every book (except macro definitions) cataloged to the SSL. If you desire, the BKEND preceding your library member may contain a number of optional parameters for sequence checking, count control, and data compression functions. The *VSE/Advanced Functions System Control Statements* manual has a full explanation of these parameters.

Figure 7.3 is an example of Source Statement Library activity. It shows two sets of statements: one for cataloging, and one for retrieving a book.

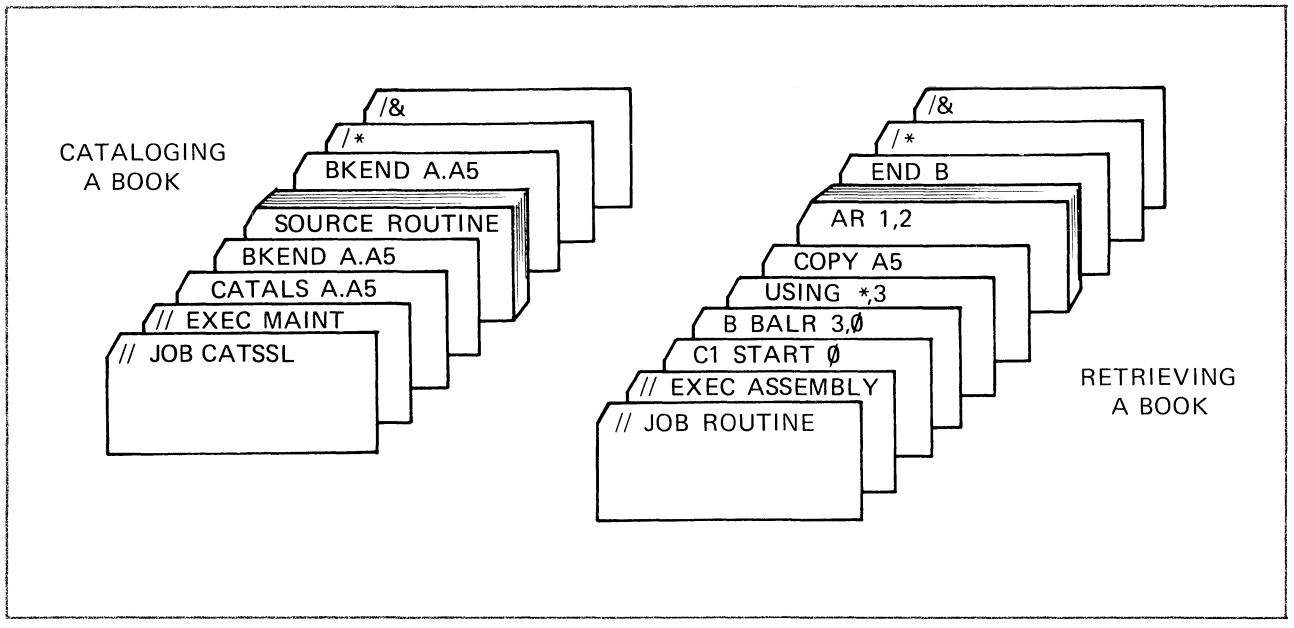


Figure 7.3 - Using the SSL

CATALP

The CATALP function of the MAINT program is used to catalog into a Procedure Library. Details of CATALP and Procedure Libraries will be discussed in Assignment 2 of this Unit.

Deleting

An unwanted element can be deleted from a library either by cataloging a new element with the same name or by means of the delete function of the librarian, using the following MAINT control statements:

DELETC - Core Image Library

DELETR - Relocatable Library

DELETS - Source Statement Library

DELETP - Procedure Library

To delete individual elements from the libraries, you must specify each element's name in full in the delete control statement.

The statements below delete the book named MSTRFLE from the N source statement sublibrary and the module IJSSKL from a Relocatable Library.

```
// JOB    DELETE
// EXEC   MAINT
   DELETS   N.MSTRFLE
   DELETR   IJSSKL
/*
/&
```

If a group of elements is to be deleted, you can simplify the specification of the control statement provided that recommended naming conventions were used when the elements were cataloged.

These naming conventions allow you to group "families" of related programs together under related names. In the CIL, a family of phases is identified by each phase name having the same first four characters; in the RL, a family of modules is identified by each module name having the same first three characters; and in the SSL, a family of books is identified by the single character sublibrary designator. Deletions can be made on individual library members, families of members, or entire libraries. Figure 7.4 illustrates various forms of the DELET control statement.

	LIBRARY	DELETE FUNCTION	
1	PROCEDURE	DELETP	PROC1,PROC2, . . .
		OR	
		DELETP	ALL
2	SOURCE	DELETS	A. WORK2
		DELETS	A. ALL
		DELETS	ALL
3	RELOCATABLE	DELETR	MOD1,MOD2, . . .
		DELETR	IJQ.ALL
		DELETR	ALL
4	CORE IMAGE	DELETC	PROGNAME
		DELETC	FCOB.ALL
		DELETC	ALL

Figure 7.4 - The DELET Statement

1. Individual members can be deleted by name or the entire library can be erased by using the ALL function. Unlike the other libraries, there is no "family" naming convention for the PL.
2. Individual sublibrary members, whole sublibraries, or the entire SSL can be deleted.
3. Individual modules, module "families" (IJQ identifies all modules related to the Assembler program), or the entire RL can be deleted.
4. Individual phases or families can be deleted (FCOB represents all COBOL phases) from the CIL. It is not possible to delete the entire system CIL with one library control statement, however, the ALL function is available for private CILs.

Condensing

Unless you have used an ALL delete, a deletion does not make available the space occupied by the deleted library member. To get the space back, you must use the condense function of the MAINT program.

In Figure 7.5, PROGB has been marked for deletion by a previous run, but still occupies space in the CIL. Any reference to the name PROGB will *not* retrieve the phase because there is no longer a valid directory entry for it.

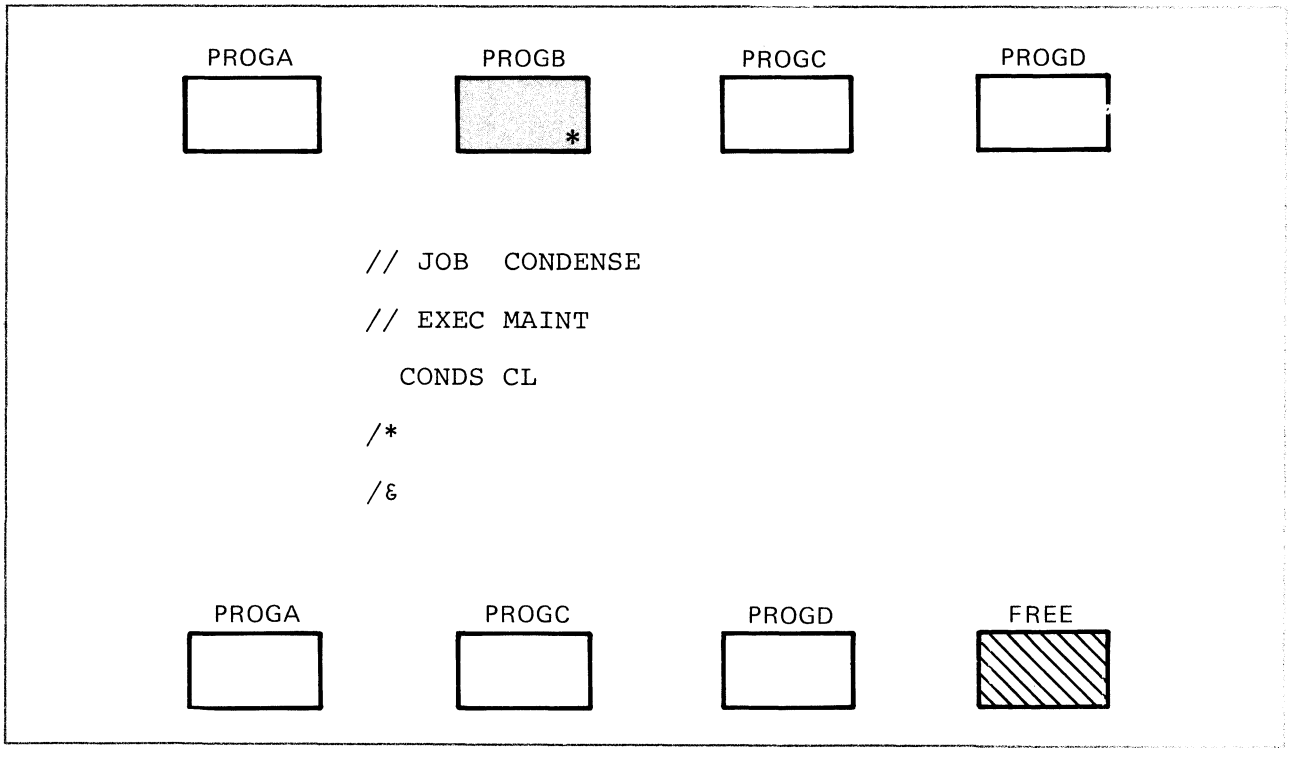


Figure 7.5 - Condensing the CIL

After the condense job is run, the space associated with PROGB is available for use by other phases. A job stream to delete members from the CIL, the RL, and the SSL is shown below, along with a condense function for each of these libraries.

```
// JOB      DELETE
// EXEC     MAINT
   DELETS   A.MSTRFLE
   DELETR   INV1
   DELETC   APL1.ALL
   CONDS    SL,RL,CL
/*
/&
```

Condense Limit

You can specify that the operator be notified by a message each time the number of available blocks in a library drops below a specified minimum. This minimum is referred to as the condense limit. The automatic condense function is requested by the CONDL control statement, which has the provision to indicate the library or libraries to be condensed and the condense limit(s).

The following example shows how this can be done:

```
// JOB     AUTOCOND
// EXEC    MAINT
   CONDL   CL=10
/*
/&
```

CL=10 indicates the Core Image Library condense request will be issued to the operator when the number of available blocks reach ten or less. The value of 10 is maintained in the CIL directory.

For FBA devices, the value coded specifies the number of physical blocks which should not be used up by a librarian operation. It can be a number of up to 9 digits in length.

For automatic condense of other libraries, use:

RL for Relocatable Library

SL for Source Statement Library

PL for Procedure Library

The available block entry may be as many as five digits.

Restrictions on CONDS

Because the libraries are effectively being reorganized during a condense operation, there are certain restrictions that must be observed.

- A condense operation must *never* be interrupted. If the condense job is cancelled, the library involved is lost.
- The system libraries can only be condensed when MAINT is running in BG and all foreground partitions are inactive.
- A private library can be condensed in any partition in which it is exclusively assigned.
- A job stream to condense the Procedure Library cannot be executed from a cataloged procedure.

Renaming

Normally, if a member having the same name as an existing member is cataloged, the existing member is lost. It can be saved by changing its name. This is done to maintain backup copies of program data.

The rename function changes the name of a cataloged phase, module, book, or procedure. Each library has a unique rename control statement.

RENAMC for Core Image Library

RENAMR for Relocatable Library

RENAMS for Source statement Library

RENAMP for Procedure Library

The name in all cases is changed in the appropriate directory entry. Figure 7.6 shows a job for renaming an existing module and then cataloging a new program under the old name.

```
// JOB . . .
// EXEC   MAINT
   RENAMR ROUTINE,OROUTINE
   CATALR ROUTINE
   (OBJECT DECK)
/*
/&
```

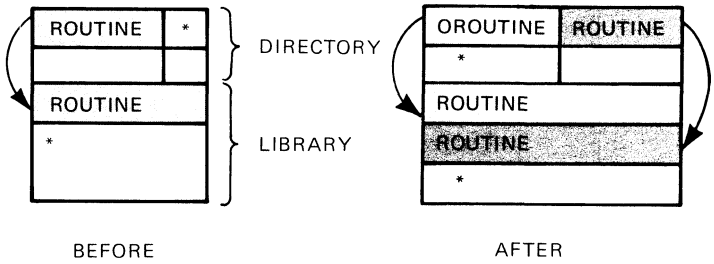


Figure 7.6 - Rename and Catalog

- Before this job is run, the existing library module named ROUTINE is pointed to by a directory entry for ROUTINE.
- After the job is run, there are two retrievable ROUTINE modules in this library: the "old" one now named OROUTINE, and the "new" one just cataloged and named ROUTINE.

The "old" module could be retrieved by means of an INCLUDE OROUTINE statement, while the "new" module is retrievable with the statement INCLUDE ROUTINE.

Updating

The update function applies only to a Source Statement Library. This function revises one or more source statements within a particular book, without having to catalog an entire new book.

Besides adding, deleting, or replacing a certain number of source statements within a book, the update function allows you to:

1. Resequence statements within a book.
2. Revise a change level (version and modification) of a book.
3. Add or remove the requirement for change level verification.
4. Copy an entire book and rename the non-updated version of the book (this is done for backup purposes).

The UPDATE control statement identifies the update function. This statement may be followed by one or more of these additional statements as required:

ADD: add source statements

DEL: delete source statements

REP: replace source statements

END: signifies end of update cards

Statement Format

Source book update statements are shown below. Notice the format requires a right parenthesis in the first position with the second position blank.

```
    UPDATE sublib.book,(s.book1),(v.m),(nn)
) ADD seq-no
) DEL first-seq-no[,last-seq-no]
) REP first-seq-no[,last-seq-no]
) END [v.m.[,C]]
```

A complete explanation of these statements is given in the Update portion of the section on Librarian functions in the *System Control Statements* manual. The following remarks are presented to familiarize you with the use of the various statements.

UPDATE

The UPDATE control statement allows the following parameters:

- sublib.book: the name of the sublibrary that contains the book to be updated and the updated book name respectively.
- s.book1: if present, causes the old book to be renamed to this specification. If this parameter is omitted, the old book is deleted.
- v.m: the change level of the book to be updated.
- nn: resequencing control.

The sequence number (seq-no) shown in some of the statements refers to the identification sequence found in source statement columns 73-80. Any decimal number from one to four characters in length in columns 77-80, or any decimal number from one to six characters in length (columns 73-78) may be used.

ADD

The ADD statement has one operand.

- seq-no: one or more statements following the ADD card will be inserted in the book following the statement identified by seq-no.

DEL

This statement has two parameters that are used as follows:

- first-seq-no: if only the first parameter is coded that one statement will be deleted from the book.
- last-seq-no: if both parameters are coded, all statements from first to last-seq-no inclusive will be deleted.

REP

The statement(s) following the REP card will replace the original statement(s) in the source book.

- first-seq-no: if only this parameter is specified, that single statement is replaced.
- last-seq-no: if both parameters are specified, all statements from first to last inclusive will be replaced.

END

This statement indicates the end of updates to a book. There are two parameters.

- v.m: provides a way of explicitly setting the change level.
- C: indicates that change level checking is required before subsequent updating.

If v.m is specified and C is not, verification of change level will not be required for subsequent updates.

Take a look at the example of an UPDATE job given in the Librarian section of the *VSE/Advanced Functions System Control Statements* manual, then do Exercise 7.1. Be sure to reference the UPDATE function for the SSL and *not* the update function of the ESERV program when you look at the manual.

Exercise 7.1

Code the appropriate control statements to update the book MYBOOK in the Q source statement sublibrary as follows:

a. Add two cards after card 150 - Cards ABC and ACC

b. Replace card 152 with card CXX

c. Delete card 153

d. Resequence cards in increments of 10.

The before and after of Q.MYBOOK is shown below.

Before

```
Q.MYBOOK          cc 73-80
BKEND
.
.
.
A _____           ..0150..
B _____           ..0151..
C _____           ..0152..
Z _____           ..0153..
.
.
.
BKEND
```

After

```
Q.MYBOOK
BKEND
.
.
.
A _____           ..0150..
ABC _____         ..0160..
ACC _____         ..0170..
B _____           ..0180..
CXX _____         ..0190..
.
.
.
BKEND
```

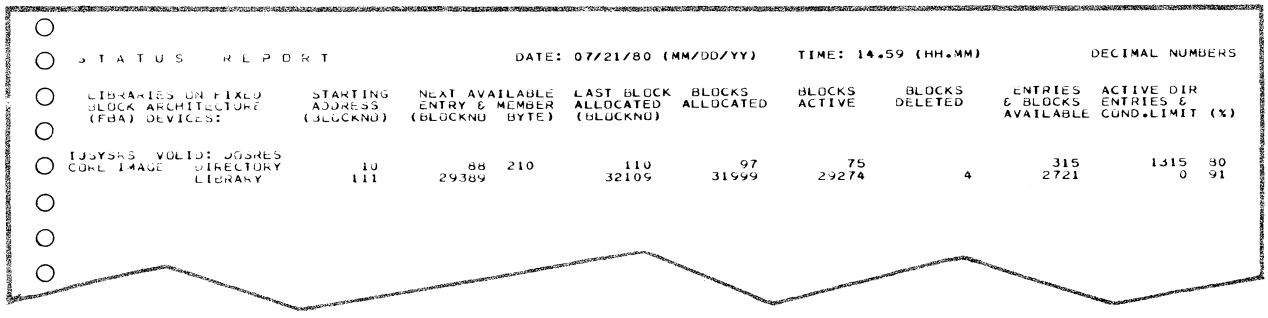
Solution

```
// JOB  Q6
// EXEC MAINT
   UPDATE Q.MYBOOK,,,10
) ADD 150
  ABC
  ACC
) REP 152
  CXX
) DEL 153
) END
/&
```

Status Reports

When linkage editor // OPTION CATAL functions or maintenance functions are performed, you will automatically get a status report of the library involved printed on SYSLST. A status report is a listing for a library that shows the library's starting address, the location in the library that is next available for cataloging a new member, the last entry in the library, the number of active members (not DELETEd), the status of library blocks, and the autocondense limit.

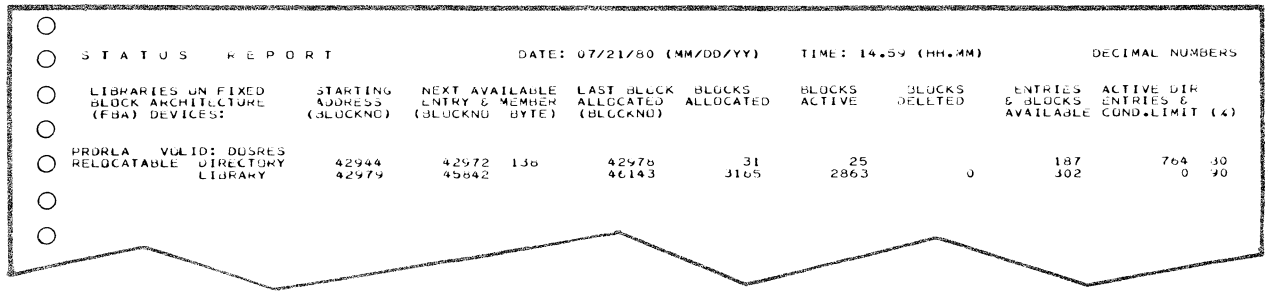
Figure 7.7 shows two status reports. At the top is a report following the update of the system Core Image Library (IJSYSRS). Below it is a report following a CATALR function to a private RL.



```
STATUS   REPORT                    DATE: 07/21/80 (MM/DD/YY)   TIME: 14.59 (HH.MM)        DECIMAL NUMBERS

LIBRARIES ON FIXED      STARTING   NEXT AVAILABLE   LAST BLOCK  BLOCKS     BLOCKS    BLOCKS    ENTRIES   ACTIVE DIR
BLOCK ARCHITECTURE      ADDRESS    ENTRY & MEMBER   ALLOCATED   ALLOCATED  ACTIVE    DELETED   & BLOCKS  ENTRIES &
(FBA) DEVICES:          (BLOCKNO)  (BLOCKNO  BYTE)  (BLOCKNO)                                  AVAILABLE COND.LIMIT (%)

IJSYSRS  VOLID: DOSRES
CORE IMAGE  DIRECTORY        10          88  210         110         97        75                315        1315  80
            LIBRARY         111       29389              32109      31999     29274         4    2721           0  91
```



```
STATUS   REPORT                    DATE: 07/21/80 (MM/DD/YY)   TIME: 14.59 (HH.MM)        DECIMAL NUMBERS

LIBRARIES ON FIXED      STARTING   NEXT AVAILABLE   LAST BLOCK  BLOCKS     BLOCKS    BLOCKS    ENTRIES   ACTIVE DIR
BLOCK ARCHITECTURE      ADDRESS    ENTRY & MEMBER   ALLOCATED   ALLOCATED  ACTIVE    DELETED   & BLOCKS  ENTRIES &
(FBA) DEVICES:          (BLOCKNO)  (BLOCKNO  BYTE)  (BLOCKNO)                                  AVAILABLE COND.LIMIT (%)

PRORLA   VOLID: DOSRES
RELOCATABLE DIRECTORY     42944      42972  136      42978         31        25                187         764  80
            LIBRARY       42979      45842              46143      3165      2863         0    302           0  90
```

Figure 7.7: Two Status Reports

Invoking the Service Programs

Librarian service functions are provided by a group of six programs:

- DSERV - To display the directories of each of the libraries
- CSERV - To display and/or punch phases from the Core Image Library
- RSERV - To display and/or punch modules from the Relocatable Library
- SSERV - To display and/or punch books from the Source Statement Library
- PSERV - To display and/or punch procedures from the Procedure Library
- ESERV - To de-edit, display and/or punch, verify and update edited assembler macros from the Source Statement Library

DSERV

The DSERV (Directory Service) program allows you to obtain a listing of all the library directories, the transient directory (the directory of those phases that use the transient area of the supervisor when they execute), and the system directory. Figure 7.8 illustrates the service functions for directory display.

```
// JOB DISPLAY
// EXEC DSERV
                 ┌ TD
                 │ CD
                 │ SDL
   DSPLY(S)      { RD
                 │ SD
                 │ PD
                 └ ALL
/*
/&
```

Figure 7.8 - Service Functions for Directory Display

Notice that the operation may be either:

- DSPLY - Displays the directory entries in the sequence in which they appear in the directory.
- DSPLYS - Displays the directory entries sorted alphamerically.

There is no need to use DSPLYS for the Core Image and transient directories as they are in alphabetic sequence already.

The operand indicates the directory and can be one of the following:

- TD (transient directory) - $-phases in the core image directory
- CD - The core image directory
- SDL - The system directory list
- RD - The relocatable directory
- SD - The source statement directory
- PD - The procedure directory
- ALL - All the above (TD, CD, SDL, RD, SD, and PD) are specified

If a blank operand follows DSPLY(S), or if no control statement is submitted, the status report is the only printed output.

Displaying and Punching Library Contents

The four programs CSERV, RSERV, SSERV, and PSERV provide service functions for the CIL, RL, SSL, and PL respectively. You can request these functions by means of the control statements:

- DSPLY - To print the elements of a library
- PUNCH - To punch the elements of a library
- DSPCH - To print and punch the elements of a library.

The job statements shown below list the contents of the book named LOOK from the J source statement sublibrary.

```
// JOB SHOW
// EXEC SSERV
   DSPLY J.LOOK
/*
/&
```

Each of the control statements can specify one or more individual members, one or more groups of members (using standard naming conventions), or ALL to cause a complete library to be printed or punched.

The statements that follow are printing, punching, and printing and punching particular phases from the Core Image Library.

```
// JOB   LOOKCIL
// EXEC  CSERV
   DSPLY phasename
   PUNCH phasename
   DSPCH phasename
/*
/&
```

The punched output (either on cards, tape, or disk) of any service program can be used as input for recataloging into the type of library from which it was extracted. Also, note the following:

1. Except for the CSERV punched output, the service programs automatically punch a CATALR, CATALS, or CATALP statement immediately preceding each element, and a /* statement immediately following the last element (/+ in case of the procedure library).
2. Punched output of the CSERV program is suitable for input to the linkage editor for recataloging to the core image library because a phase card is punched out for each group of cards.

ESERV

The E-sublibrary of the SSL is used to store assembler macro definitions. These assembler macros are preprocessed by the assembler and are said to be edited. A macro in an edited state cannot be directly updated. The ESERV program converts the edited macro back to source format so the macro may be updated.

This subject is relevant only to the assembler programmer. Further references to ESERV are presented in the *VSE/Advanced Functions System Control Statements* manual, and the *Guide to the DOS/VSE Assembler* (GC33-4024).

Take the following review Exercise before proceeding to the next Assignment.

Exercise 7.2

1. A function *not* available with the MAINT program is _____.
   a. cataloging phases to the CIL
   b. renaming members of the CIL, SSL, RL and PL
   c. condensing the SSL
   d. deleting phases from the CIL
2. The one library that contains sublibraries is the _____.
   a. CIL
   b. RL
   c. SSL
   d. PL

Use the following JCL for questions 3-5.

```
// JOB      LIBJOB
// DLBL     TSTRLA,...
// EXTENT   ,VOLID1,...
// DLBL     TSTRLB,...
// EXTENT   ,VOLID2,...
// DLBL     PRODCL,...
// EXTENT   ,VOLID3,...
LIBDEF      RL,SEARCH=(TSTRLA,TSTRLB),TO=TSTRLA
LIBDEF      CL,SEARCH=PRODCL
// EXEC     MAINT
   DELETR   ALL
   CONDS    RL,SL
/*
/&
```

NOTE: LIBDEF and // LIBDEF perform the same function.

3. The control statement shown will delete library members called _____.
   a. phases
   b. modules
   c. books
   d. procedures
4. There seems to be a discrepancy between the LIBDEF assignments and the librarian control statements. Which libraries will actually be condensed?
   a. system RL and system SSL
   b. private RL and private CL
   c. system RL and private CL
   d. private RL and system SSL

5. Match the programs or control statements in the first column with the functions in the second column.

a. DSERV	1. delimits source books
b. CONDL	2. macro servicing
c. BKEND	3. print library directories
d. ESERV	4. print library members
e.) END	5. sets condense limit
f. DSPLY	6. UPDATE control statement

Solution

1. a
2. c
3. b
4. d No private SSL is assigned.
5. a-3, b-5, c-1, d-2, e-6, f-4

Computer Exercise

Begin the Computer Exercises 7 and 8. These will give you the chance to work with the librarian programs. Exercise 8 is dependent on the results of Exercise 7.

When you have submitted Exercise 7 for a run, go on to the next Assignment.

Assignment 2 - The Procedure Library

Every job you submit requires a certain amount of associated JCL. Many frequently run jobs have JCL and link-edit control statement requirements that vary little from run to run. If you are doing a compile, link-edit, and execute operation, for example, your changes from run to run are in your program and not in the JCL needed by the compiler.

For this reason, commonly used sets of job control and link-edit control statements may be stored in card-image format in the Procedure Library to be invoked as needed. By using procedures, you reduce the amount of JCL you need to submit with your job.

In addition to SYSRDR data (job control and link-edit control statements are termed SYSRDR data as SYSRDR is the logical device from which they are read), SYSIPT data may also be put into a PL.

In this library each member is a procedure and consists of 80 character unblocked card images.

Cataloging to the Procedure Library

The CATALP function of the MAINT program is used to catalog into the PL. Figure 7.9 illustrates the job stream for cataloging a procedure named SAMPLE.

```
// JOB CATPROC
// EXEC MAINT
                                    COLUMNS
   CATALP SAMPLE                    73-79
                                    ~~~~~
// ASSGN SYS004,DISK,SHR            ASGN004
// ASSGN SYS009,DISK,SHR            ASGN009
// EXEC SAMP1
// ASSGN SYS014,TAPE                ASGN014
// TLBL TAPEOUT                     TLBLOUT
// EXEC SAMP2
/*
/+
/&
```

Figure 7.9 - Cataloging a Procedure

Note the following:

- Columns 73-79 contain symbolic identifiers that are used to locate statements for temporary modification at execution time.
- SAMPLE is the name of the procedure. SAMPLE includes all statements up to the /+. The /+ is the procedure delimiter and signifies end of procedure to the MAINT program. The /+ is cataloged along with the procedure.
- The procedure does not itself contain a JOB or a /& statement, although it does have a /*. Certain job control statements are not permitted in a procedure. They are listed in the *VSE/Advanced Functions System Control Statements* manual. JOB statements may be present, although they can give rise to difficulties that will be discussed shortly.

The EOP Parameter

Figure 7.10 shows the same job stream with one change: an EOP parameter has been coded on the CATAL card to change the delimiter characters. If the standard /+ is inconvenient for you, or if your installation has other standards, the two characters that delimit the procedure may be made anything except /*, /&, or // by the EOP parameter.

The /+ characters, however, are the ones that actually appear in the procedure on disk, regardless of the EOP values.

```
// JOB CATPROC
// EXEC MAINT                                  COLUMNS
   CATALP SAMPLE,EOP=@@                        73-79
// ASSGN SYS004,DISK,SHR                      ASGN004
// ASSGN SYS009,DISK,SHR                      ASGN009
// EXEC  SAMP1
// ASSGN SYS014,TAPE                          ASGN014
// TLBL TAPEOUT                               TLBLOUT
// EXEC SAMP2
/*
@@
/&
```

Figure 7.10 - Changing the Delimiter

The DATA Parameter

If you wish to include SYSIPT data (service program control statements or compiler input) within a procedure, you must code DATA=YES on the CATALP control statement. As /* cards may be part of any procedure, they would be used in this case to delimit your SYSIPT data.

It is not recommended that you overuse this capability. Since job control statements and data in the PL are kept in unblocked, uncompressed card image format, space can be rapidly used up by the inclusion of great volumes of SYSIPT data. A Procedure Library is primarily intended to hold SYSRDR data, that is, frequently referenced job control statements.

Restrictions

The *VSE/Advanced Functions System Management Guide,* under the heading "Cataloging to the Procedure Library" in the section "Maintaining the Libraries," discusses what may not be made part of a cataloged procedure. Most of the restrictions revolve around the way in which SYSRDR is affected when VSE encounters a cataloged procedure. See Figure 7.11.

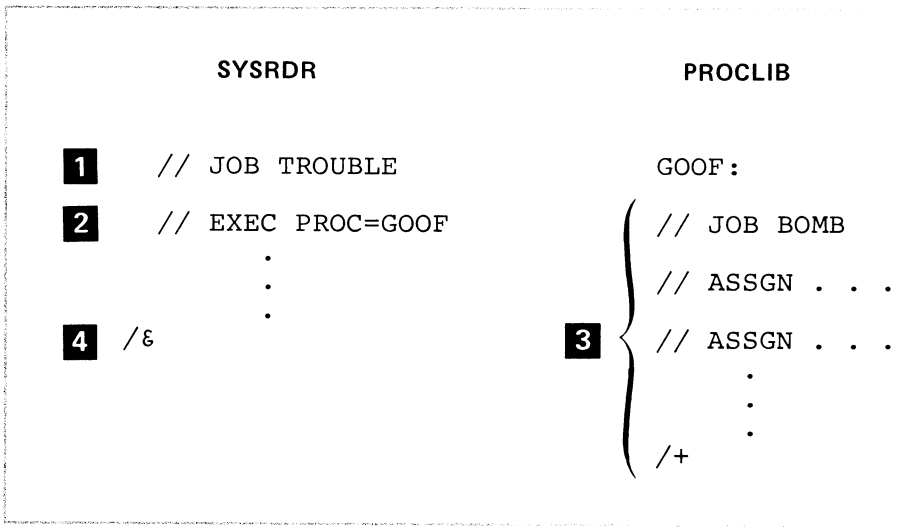


Figure 7.11 - Invoking a Procedure

1 Job control reads this statement from SYSRDR and assigns a name to the job.

2 This statement invokes the procedure GOOF. The processing of this EXEC statement causes the SYSRDR assignment to be changed to point to the member GOOF in a Procedure Library.

3 These cards are read by job control *as if* they were coming from SYSRDR. When the second job statement is encountered, this whole job will be cancelled. A procedure may contain a JOB statement only if the procedure is invoked by an EXEC not itself associated with a JOB statement. This can be done through the operator's console.

4 If step 3 above had not contained a JOB statement to cause cancellation, the /& would be read by job control from SYSRDR. It is the /+ in the procedure itself that switches the SYSRDR assignment back.

When a procedure has been cataloged with DATA=YES, the SYSIPT assignment is also switched to point to the procedure when the EXEC PROC= statement is processed.

The Overwrite Function

A cataloged procedure may have to be modified in order to run a specific program.

For example, a tape drive normally used for a job may be temporarily unavailable, and another tape drive must be assigned to run the job. Rather than catalog a new procedure to run the job or create a complete job stream to accommodate the new tape drive assignment, it is easier to make up a modification statement with the change in assignment. The modification will only

affect the invoked procedure and will leave the library member as is. When the original tape drive is put back in service, all that is necessary is to remove the modification from the input job stream.

The overwrite function provides the ability to effectively replace or delete statements in a cataloged procedure or to insert new statements. The procedure remains unchanged in the library, but the effective job stream for your execution will reflect your changes.

In order to use the overwrite capability, your cataloged procedure statements must have identifiers in columns 73-79. This allows you to identify the points in the procedure where you wish to make changes.

SYSIPT data in a procedure may not be modified.

Executing with Overwrites

To do overwrite(s), you must specify OV in the invoking EXEC statement. The overwrite statements follow the EXEC statement, and an OVEND statement follows the last overwrite statement.

```
// JOB WHATEVER
// EXEC PROC=A,OV
     .
     .
overwrite statements
     .
     .
// OVEND
/&
```

The modifying control statement is written in the usual manner, and a code in column 80 indicates the type of overwrite to be done. There are four codes for delete, insert, or replace.

- D - means delete
- A - means insert-after
- B - means insert-before
- any other code or blank means replace

Columns 73-79 of an overwrite statement must contain the same identifier as the procedure statement being overwritten. The overwrite and the procedure statements are matched on this id-field. An overwrite with blanks in columns 73-79 is ignored by job control.

Naturally, it is best not to have identical identifiers (cols. 73-79) in any two statements within a given cataloged procedure.

Overwrite statements must be in the same order as the statements in the procedure.

Procedure statements without identifiers are allowed, but they cannot be directly modified.

Deletions, Replacements, Insertions

Job control holds each overwrite statement until a match is found in the procedure. It does this one statement at a time. If end of procedure is reached and there are still overwrite statements remaining, a warning message is issued but your job continues.

The series of illustrations in Figures 7.12 - 7.15 show examples of the various overwrite functions.

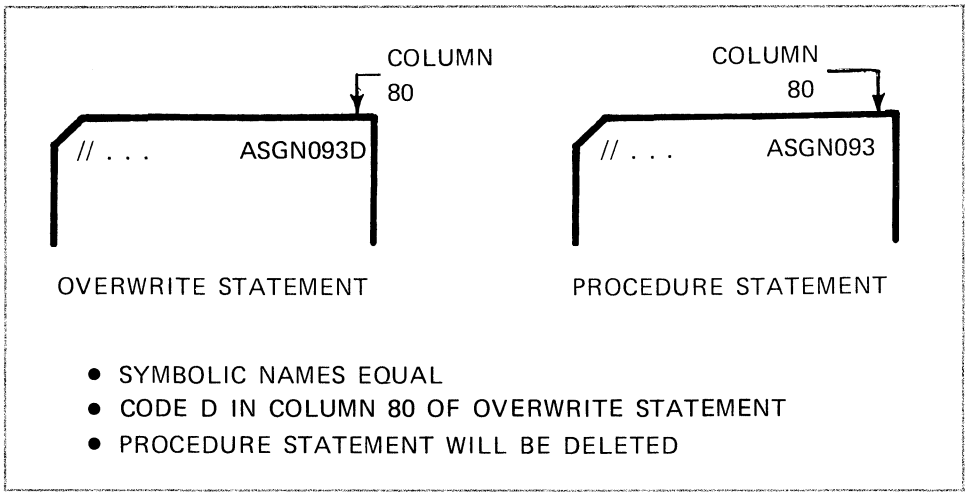


Figure 7.12 - Deleting

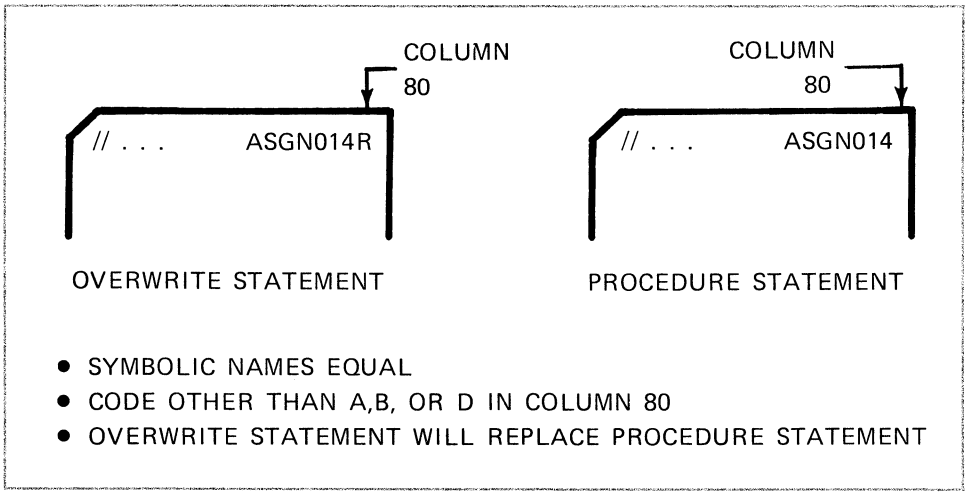


Figure 7.13 - Replacing

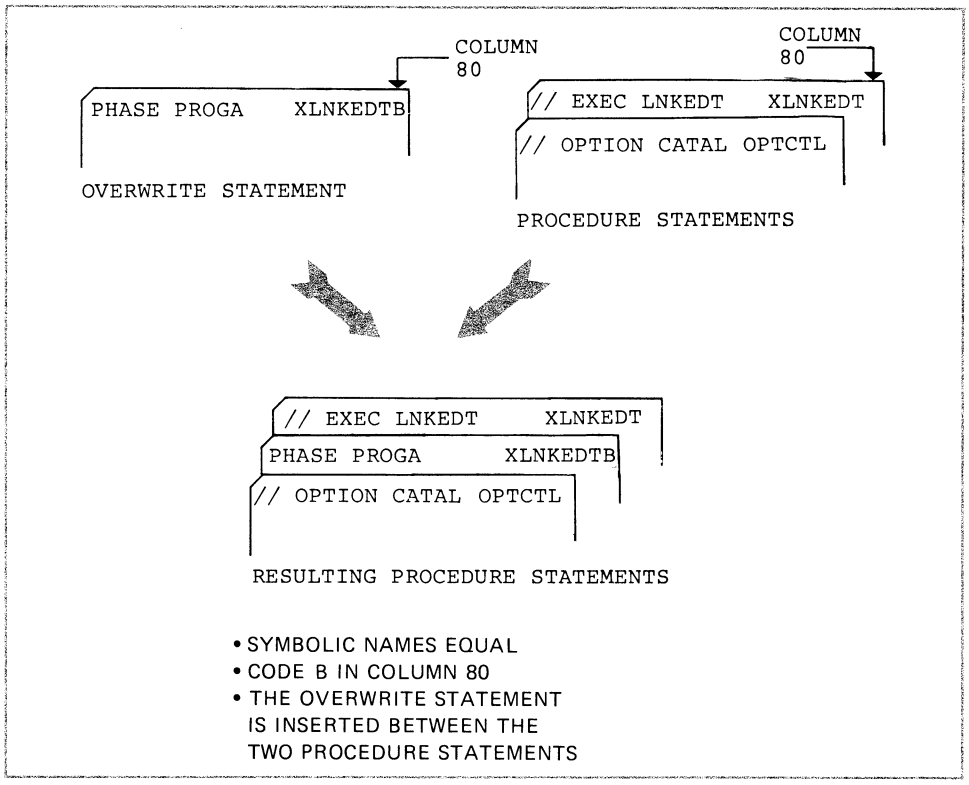


Figure 7.14 - Inserting Before

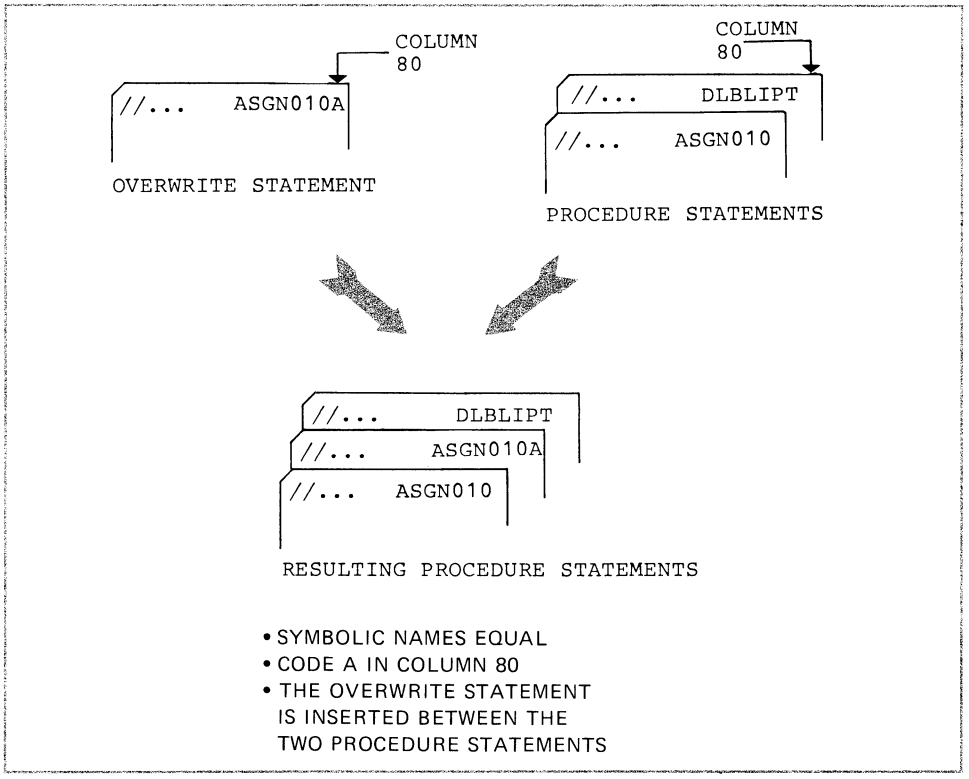


Figure 7.15 - Inserting After

An Example

Examine Figure 7.16. The SYSRDR input shown is to be applied against the cataloged procedure PAYR. Note that two statements in the procedure (the EXECs) have no identifiers in columns 73-79. This is permitted. It merely means that you will never be able to overwrite these statements. The EXECs are therefore secure from modification.

Take a few minutes to come up with the effective job stream on a piece of scratch paper. The solution is given in Figure 7.17.

SYSRDR INPUT

```
// JOB     PAYROLL
// EXEC   PROC=PAYR,OV
// DLBL    . . . .
// EXTENT . . . .
// ASSGN   SYS005,SYS004
// EXTENT  . . . .
*
// OVEND
/&
```

COLUMNS 73-79 80

```
INV1 A
INV2 B
INV3 R
INV5 A
INV6
```

PROCEDURE 'PAYR'

COLUMNS 73-79

```
// ASSGN   SYS004,152          INV1
// ASSGN   SYS009,240          INV2
// EXEC    INV
// ASSGN   SYS005,SYS009       INV3
// ASSGN   SYS010,151          INV4
// DLBL    . . . .             INV5
// TLBL    TAPEOUT             INV6
// EXEC    INVA
/+
```

Figure 7.16 - A Procedure Overwrite

```
  // JOB PAYROLL
  // ASSGN  SYS004,152
• // DLBL . . .
• // EXTENT . . .
  // ASSGN  SYS009,240
  // EXEC INV
• // ASSGN  SYS005,SYS004
  // ASSGN  SYS010,151
  // DLBL . . . .
• // EXTENT . . . .
• *
  // EXEC INVA
  /&
```

Figure 7.17 - The Effective Job Stream

The statements flagged with bullets indicate an overwrite has taken place. Note that the * prints. This statement is treated as a comment. It has been used to remove the TLBL card for this execution of the procedure.

Figure 7.18 shows how cataloging a procedure with DATA=YES can lead to trouble for the unknowing user.

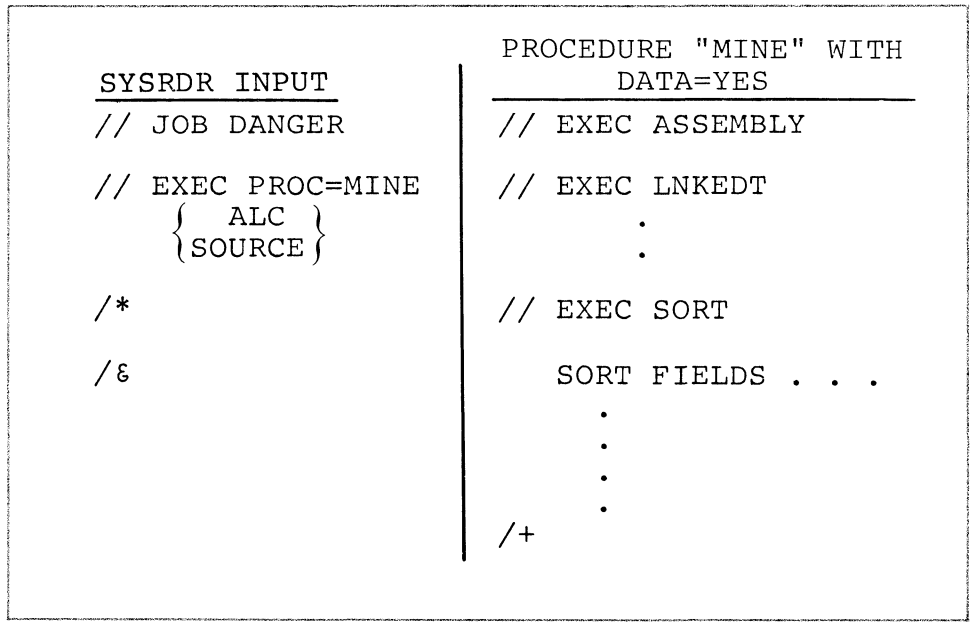


Figure 7.18 - Incorrect Use of SYSIPT Data

When MINE is invoked, both SYSRDR *and* SYSIPT are switched to the procedure library. When the Assembler program becomes active it looks for its data on the SYSIPT device, which is now the cataloged procedure. The rest of MINE would be read as Assembler input.

The point is to be very aware of how you catalog procedures, and not to use the DATA=YES option indiscriminately.

Partition Related Procedures

In some cases a cataloged procedure may need a specific set of job control statements to run successfully in a given partition.

This comes about because things such as permanent logical unit assignments and device addresses vary from partition to partition. Cataloged procedures to control this kind of job are said to be partition related.

The user must prepare a procedure for each of the partitions in which the job is expected to run so that it will satisfy partition dependencies. The name given to each of these procedures must follow prescribed naming conventions in order that only one EXEC PROC statement is needed no matter what partition is used.

Naming Conventions

The naming convention used for procedures to be placed in the library are:

First character of name:	$
Second character:	0 for BG partition, 1 for F1, 2 for F2..., A for FA, and B for FB
Third - eighth characters:	Any alphameric

An example of a procedure name that is to be run in the background partition is:

```
$0ANYNAM
```

The example below illustrates the coding for cataloging procedures for a job that is to be run in partition F3 and F4.

```
// EXEC MAINT
   CATALP $3PAYROL
   .
   .
control statements
   .
   .
/+
   CATALP $4PAYROL
   .
   .
control statements
   .
   .
/+
/&
```

The EXEC PROC= statement used for running in any partition must begin with $$ in positions one and two, as below.

```
// EXEC PROC=$$PAYROL
```

Reading Assignment

As a review, you may want to read in the *VSE/Advanced Functions System Management Guide* under "Using the System" the following sections:

- "Using Cataloged Procedures"
- "Temporarily Modifying Cataloged Procedures"
- "Using the Libraries"

Unit Summary

The system and private libraries available to you with VSE are maintained and serviced by a group of programs known as the librarian. The maintenance, service, and reorganization functions of the librarian provide everything you need to fully support your libraries at all times.

Your private libraries are maintained by the same programs that handle the VSE system libraries. Both LIBDEF and ASSGN statements may be used to define private libraries. An exception is a private Procedure Library which can only be defined via LIBDEF. It is recommended that LIBDEF be used rather than ASSGN when defining libraries as LIBDEF provides greater flexibility (library chaining,...) with fewer restrictions.

The various library service programs CSERV, RSERV, SSERV, PSERV, ESERV, and DSERV give you the capability of displaying and, for all except DSERV, of punching library members. This makes the contents of your libraries highly portable - they can be punched onto card or tape, for example, and transported to another location where they are required. Since the service programs are read-only, there are no partition restrictions on their operations.

The VSE Procedure Libraries are useful for holding commonly used sets of job control data. This relieves the user of much of the burden of submitting voluminous sets of JCL over and over again with repetitively run jobs. Cataloged procedures can be invoked as is, or with overwrites when a temporary variation from the fixed procedure is required. Since overwrites do not change the procedure in the library, each user can make whatever modifications are needed without affecting what anyone else is doing.

SYSIPT data (compiler input, service program control statements) may be made part of a cataloged procedure. Because of space limitations within the Procedure Libraries and the danger of operational errors, however, it is not recommended that this feature be used indiscriminately.

Partition related procedures are those that vary according to the partition in which they are run. If you have a situation that calls for certain differences to be maintained in a procedure for each of twelve partitions, then that procedure is stored under twelve different names in a PROCLIB. As long as you follow the specified naming conventions, the appropriate procedure is invoked at execute time.

Take the Mastery Test that follows then prepare Computer Exercise 9.

Mastery Test

You will need your *VSE/Advanced Functions System Control Statements* manual for some of the following questions.

1. Assuming all library assignments are made, code a job to print and punch the phases PHASTAX and PHASINV.

2. Assuming all library assignments are made, code a job to print all the elements of the Q-sublibrary.

3. Code the appropriate control statements to rename elements in the source statement library shown below.

```
L.sublibrary
  Old Name          New Name
    INV1              INV3
R.sublibrary
  Old Name          New Name
    INV4              INV6
```

4. Code a statement that will catalog an element to the relocatable library with the following definition:

```
Element name      FIX4
Version           3
Modification      5
```

5. Construct the job stream necessary to display a book from the source statement library. The book is called BOOK2323 and is held in the X.sublibrary of a private Source Statement Library (PRDSLA). Permanent label information exists for this library.

6. The MAINT program, when executing in a *foreground* partition, can access _____.

   a. any private library referenced in the TO=parameter of the LIBDEF statement

   b. any system library

   c. any private library assigned via an ASSGN statement

   d. all of the above

7. Which of the following cards could *not* be included in a cataloged procedure?

   a. `//`

   b. `/&`

   c. `/*`

   d. `// JOB ANY`

8. The overwrite function allows you to _____.

   a. temporarily modify a procedure

   b. permanently modify a procedure

   c. invoke an uncataloged procedure

   d. none of these

9. Regardless of what value is used in the EOP parameter, the procedure is cataloged with _____ as its delimiting characters.

   a. /$

   b. /*

   c. /+

   d. /-

10. Assume a procedure named PAYR has been cataloged. It includes a statement

```
                                Cols. 73-79
// ASSGN SYS014,182              ASGN014
```

and that assignment is to be changed to 184. The job also requires the insertion of

```
// ASSGN SYS020,DISK,SHR
```

after the ASGN014 statement. Code the job stream to invoke PAYR and make these changes.

Solution

1.
```
// JOB  Q1
// EXEC CSERV
   DSPCH  PHASTAX,PHASINV
/*
/&
```

2.
```
// JOB  Q2
// EXEC SSERV
   DSPLY Q.ALL
/*
/&
```

3.
```
// JOB  Q3
// EXEC MAINT
   RENAMS L.INV1,L.INV3,R.INV4,R.INV6
/*
/&
```

4.
```
CATALR  FIX4,3.5
```

5.
```
// JOB    Q5
// LIBDEF SL,FROM=PRDSLA
// EXEC   SSERV
   DSPLY  X.BOOK2323
/*
/&
```

6. d

7. b

8. a

9. c

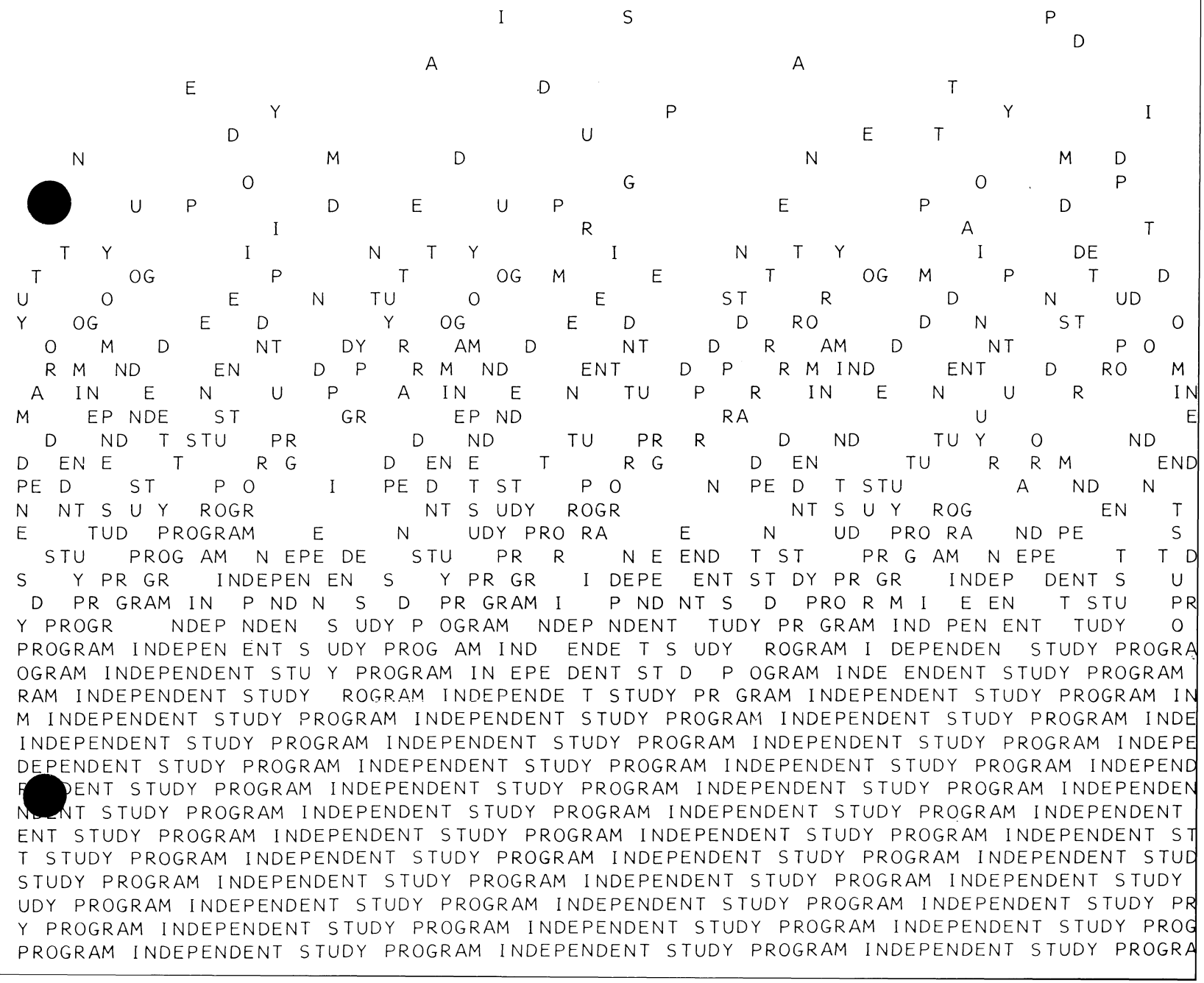
10.
```
// JOB   Q10
// EXEC  PROC=PAYR,OV
// ASSGN SYS014,184          ASGN014
// ASSGN SYS020,DISK,SHR     ASGN014A
// OVEND
/&
```

Remedial

If you had more than 3 of the coding problems (1 through 5 and problem 10) incorrect, it is suggested that you reread the material in this Unit. In addition, study the section entitled "Using the Libraries" in the *VSE/Advanced Functions System Management Guide*.

Unit

8



Unit 8:

Introduction to Interactive Computing

Introduction

This unit describes interactive computing and explains the advantages of interactive computing over batch computing. VSE/Interactive Computing and Control Facility (VSE/ICCF) is the facility that transforms your VSE system from a batch system to a system that can operate in both batch and interactive modes.

In the preceding units and exercises, the process of assembling, link-editing, and executing a program in a VSE partition was explained. The exercises, using the Computer Exercise Card Deck (SR20-7302), demonstrated how these tasks are run in a batch partition. To introduce you to interactive computing, in this unit these same tasks will be performed in a VSE/ICCF interactive partition.

This material is intended to serve only as an introduction to interactive computing and to VSE/ICCF. A more detailed explanation of the VSE/Interactive Computing and Control Facility is presented in the *VSE Installation and Maintenance Using System IPO/E Study Guide* (SR20-7377) and the *VSE/Interactive Computing and Control Facility for Programmers Student Text* (SR20-4676).

Objective

Upon completing this Unit, you should be able to:

- Describe interactive computing and list its advantages over batch computing.
- Explain the difference between a VSE batch partition and a VSE/ICCF interactive partition.
- Compare several VSE/ICCF job entry statements with their corresponding job control statements.

Materials Required

Study Guide (SR20-7300)

Interactive Computing

The process of an individual directly utilizing a computer via a terminal is defined as interactive computing. It allows the user to enter input into a computer from a terminal and to receive output, or responses to that input, back at the terminal. In effect, the terminal keyboard and display screen replace the card reader, card punch, and printer of a batch computing system. Instead of entering input on cards through a card reader, the user types the input into the keyboard of the terminal and, after processing by the computer, the output is displayed on the terminal screen instead of being printed on paper or punched into cards.

Although the concept of interactive computing has almost unlimited applications, it has proved most useful in the areas of program development and maintenance, individual problem solving, and education. Interactive computing allows a programmer, an engineer, a financial planner, a computer operator, and others to utilize the same central processing unit simultaneously.

To better understand the advantages of interactive computing, let us take a look at a programmer doing program development. This development may take the form of creating new programs or modifying code from programs that have been previously written.

Batch vs Interactive

Application programmers, developing new programs on a batch system, perform the functions diagrammed in Figure 8.1. They write their programs, have them keypunched, and submit them for compiling and execution. Usually there is a delay before the output is returned. Frequently a program is not successful the first time it is run; therefore, corrections must be made, keypunched, and the program resubmitted for compiling and execution. The loss of productivity in running from desk to keypunch to computer room and in waiting for computer output can be significant.

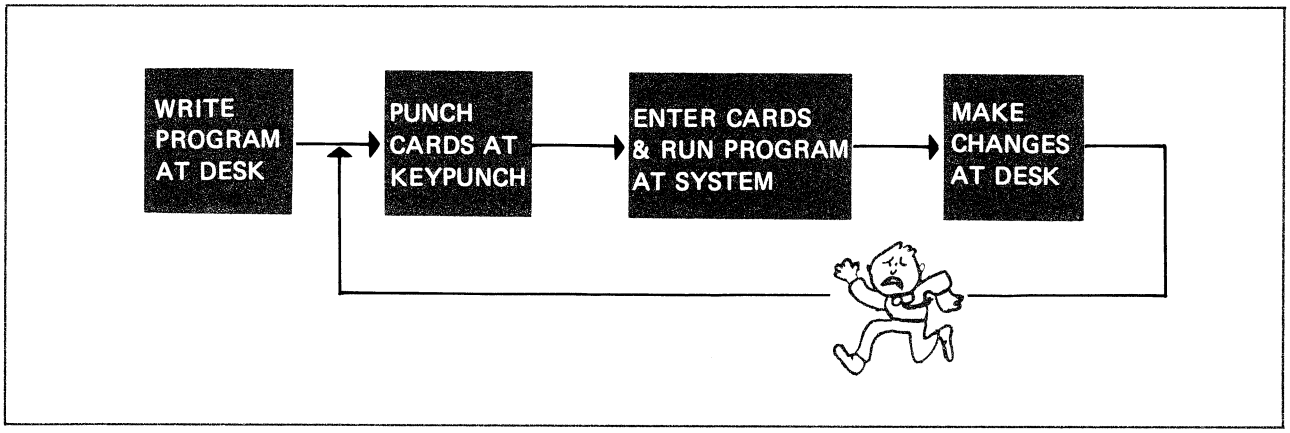


Figure 8.1 Programmer's Activities in a Typical Batch System

With an interactive system, the same tasks must still be performed, that is, writing the program, running the program, making corrections, running the program again. However, when using an interactive system, this work is done at a terminal connected directly to the computer. There is no keypunching of cards, no running from one place to another, no wait to enter cards into the system or receive output from the system. All the programming activities can be done using display terminals. VSE/ICCF is the interactive system for the VSE System. It facilitates the program development activities just described. VSE/ICCF comes as a component part of the VSE System IPO/E *base*. Some of the activities that can be performed on a VSE/ICCF terminal include:

- Entering a program
- Displaying that program at your terminal
- Making changes if necessary
- Compiling and executing the program
- Obtaining output information
- Making corrections
- Rerunning the program

VSE/Interactive Partitions

As we have seen in previous units, for a program to execute under the control of VSE, it must be loaded into a VSE partition. The loading is done by the job control program and the VSE Supervisor. VSE/ICCF is no exception. It is a VSE program which is loaded by job control and executes in one of the VSE partitions. This is shown in Figure 8.2.

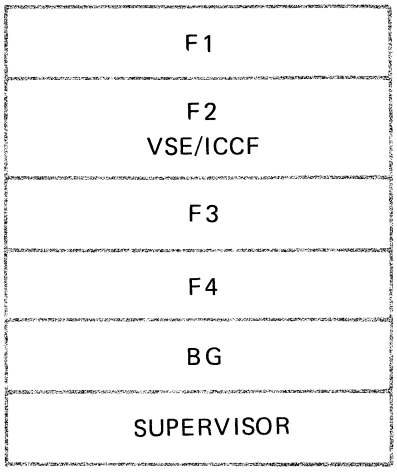


Figure 8.2 VSE/ICCF in a VSE Partition

Once VSE/ICCF is loaded into the partition (F2 in this example) the activity in this partition is controlled by VSE/ICCF not the job control program. VSE/ICCF commands replace job control statements.

When a terminal user enters a job for execution in this interactive environment, VSE/ICCF looks for a block of virtual storage within this VSE/ICCF partition where the job can be run. This block of virtual storage is called an interactive partition. There can be as many as 35 of these VSE/ICCF interactive partitions.

VSE/ICCF interactive partitions are similar to VSE partitions. Each interactive partition has the same size requirements as a standard VSE partition. Each requires a minimum of 128K (131,072 bytes) of virtual storage. Figure 8.3 shows the interactive partitions as a part of the VSE/ICCF partition.

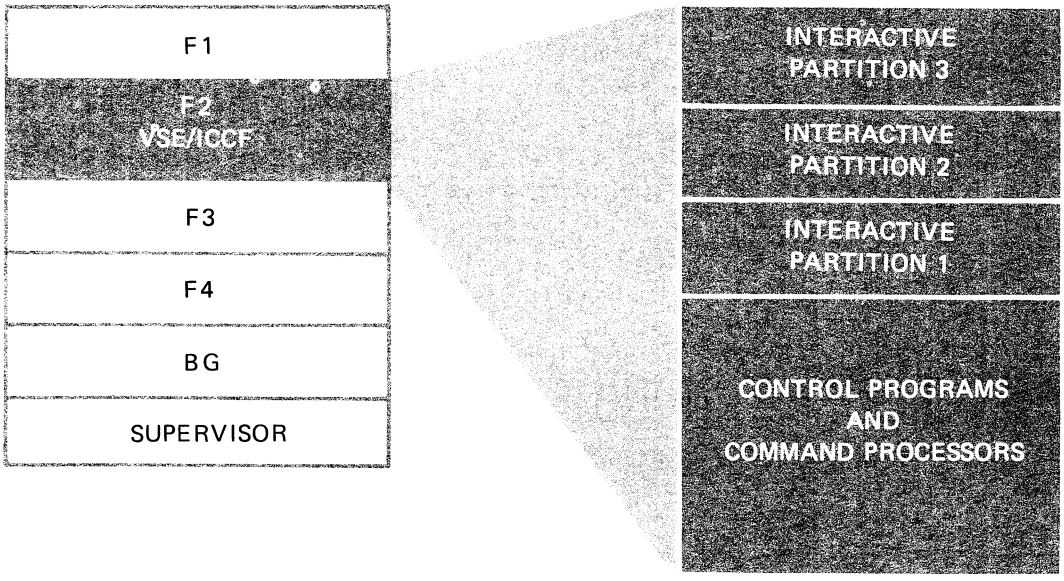


Figure 8.3 The VSE/ICCF Partition

The VSE/ICCF interactive partition differs from a VSE partition in that the job control program is never loaded into any interactive partition. This means that jobs run in interactive partitions do not use job control statements. Instead, work in the VSE/ICCF interactive partitions is controlled by VSE/ICCF statements called job entry statements.

VSE/ICCF Job Entry Statements

In an interactive partition, statements that replace JCL are called VSE/ICCF job entry statements. The *VSE/ICCF Terminal User's Guide* (SC33-6068) explains the various job entry statements as well as the other commands and statements involved in using VSE/ICCF.

In Computer Exercise 2 (Appendix A), you were instructed to code a job stream to assemble a program in a VSE partition and obtain an object deck, a source listing, and a cross reference listing. The solution was as follows:

```
// JOB EX2
// OPTION DECK,XREF,LIST
// EXEC ASSEMBLY,SIZE=64K
   ....source deck....
/*
/&
```

To accomplish the same job in a VSE/ICCF interactive partition, the job entry statements would be as follows:

```
/LOAD ASSEMBLY
/OPTION DECK,XREF,LIST,NOGO
 ....source program....
/RUN
```

The /LOAD statement tells VSE/ICCF which program to load and execute. The /OPTION replaces the job control statement // OPTION. The NOGO parameter is used to perform the assembly only. If this is omitted, the default is GO and the program will not only assemble but will also link-edit and execute. The /* is generated automatically by VSE/ICCF. The // JOB and end-of-job (/&) statements are replaced by the /RUN command.

The following table shows a comparison of some of the more common job control statements and their corresponding job entry statements:

VSE JOB CONTROL	VSE/ICCF JOB ENTRY
// EXEC	/LOAD
// OPTION	/OPTION
// ASSGN	/ASSGN
// DLBL	/FILE
// UPSI	/UPSI

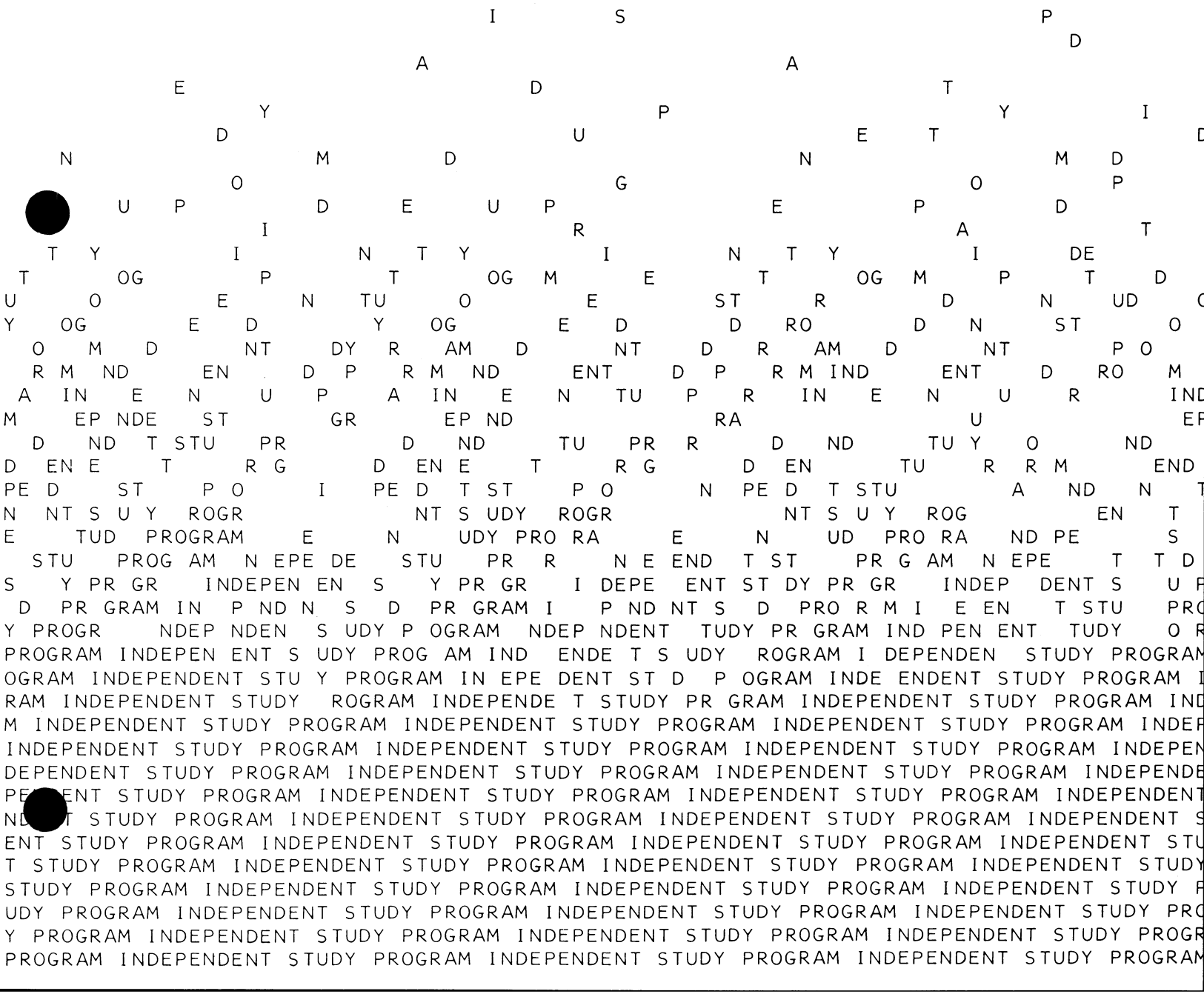
Unit Summary

In this unit, the subject of interactive computing has been introduced. Its advantages over batch computing are significant. It makes it possible for many users to access the computer concurrently. Productivity increases and costs decrease.

To adequately use VSE/ICCF more information is required. This information includes logging on to VSE/ICCF, modes of operation, and VSE/ICCF commands. The student text *VSE/ICCF for Programmers* (SR20-4676) is a good source for this information.

Appendix

A



Appendix A

Computer Exercises

This appendix contains instructions for doing the computer exercises. Do the exercises when instructed in the Text.

An experienced person at your installation should be designated as your advisor to provide help when debugging.

If not testing the exercises via computer, compare your solutions to those provided in *Computer Exercise Solutions* (SR20-7301).

The last page of each exercise provides information useful to an operator when running the exercise solution. This page is labeled "PROGRAM RUN DESCRIPTION". It should be removed and attached to your card deck when submitting an exercise for execution.

Note to Student's Advisor:

Provide information necessary to run these exercises to the student and the system operator by writing in the information on copies of *Figure A.1* which are on the next pages. Give a copy to the student and a copy to the system operator. Retain a copy for yourself.

COMPUTER EXERCISE REQUIREMENTS

REQUIREMENT	COMPUTER EXERCISE NUMBER								
	1	2	3	4	5	6	7	8	9
• Operator must provide any required JECL	√	√	√	√	√	√	√	√	√
• Student provides JECL									
• Assembly		√	√					√	
• L.E./temporarily catalog a phase			√					√	
• L.E./permanently catalog a phase				√					√
• Execution of student's phase:			√		√	√		√	√
– SYSRDR/SYSIPT assigned to a card reader			*A		*A	*A		*A	*A
– SYSLST assigned			*B		*B	*B		*B	*B
– Unlabeled tape volume to be assigned generically (to tape) by student					√				
– Disk volume to be assigned generically (to DISK) by student						*C			
• Source statement library							√	√	
• Private source statement library									
• Private relocatable library									
• Private core image library									
• Procedure library									√

*A Student assigns SYS007 to SYSIPT device.

*B Student assigns SYS010 to SYSLST device.

*C Mount on any DASD.

Figure A.1 Student Computer Exercise Requirements

COMPUTER EXERCISE REQUIREMENTS

REQUIREMENT	COMPUTER EXERCISE NUMBER								
	1	2	3	4	5	6	7	8	9
• Operator must provide any required JECL	√	√	√	√	√	√	√	√	√
• Student provides JECL									
• Assembly		√	√					√	
• L.E./temporarily catalog a phase			√					√	
• L.E./permanently catalog a phase				√					√
• Execution of student's phase:			√		√	√		√	√
– SYSRDR/SYSIPT assigned to a card reader			*A		*A	*A		*A	*A
– SYSLST assigned			*B		*B	*B		*B	*B
– Unlabeled tape volume to be assigned generically (to tape) by student					√				
– Disk volume to be assigned generically (to DISK) by student						*C			
• Source statement library							√	√	
• Private source statement library									
• Private relocatable library									
• Private core image library									
• Procedure library									√

*A Student assigns SYS007 to SYSIPT device.

*B Student assigns SYS010 to SYSLST device.

*C Mount on any DASD.

Figure A.1 Student Computer Exercise Requirements

COMPUTER EXERCISE REQUIREMENTS

REQUIREMENT	COMPUTER EXERCISE NUMBER								
	1	2	3	4	5	6	7	8	9
• Operator must provide any required JECL	√	√	√	√	√	√	√	√	√
• Student provides JECL									
• Assembly		√	√					√	
• L.E./temporarily catalog a phase			√					√	
• L.E./permanently catalog a phase				√					√
• Execution of student's phase:			√		√	√		√	√
– SYSRDR/SYSIPT assigned to a card reader			*A		*A	*A		*A	*A
– SYSLST assigned			*B		*B	*B		*B	*B
– Unlabeled tape volume to be assigned generically (to tape) by student					√				
– Disk volume to be assigned generically (to DISK) by student						*C			
• Source statement library							√	√	
• Private source statement library									
• Private relocatable library									
• Private core image library									
• Procedure library									√

*A Student assigns SYS007 to SYSIPT device.

*B Student assigns SYS010 to SYSLST device.

*C Mount on any DASD.

Figure A.1 Student Computer Exercise Requirements

List of Exercises

Computer Exercise No.	Assigned In Unit No.	Subject
1	2	Listing Device Assignments
2	3	Assembling A Program
3	3	Assemble, Link-Edit and Execute a Program
4	4	Permanently Cataloging a Phase
5	4	Executing a Cataloged Program Using UPSI and Generic Assignments
6	5	Creating and Accessing Sequential Disk Files
7	7	Catalog and Display a Source Program
8	7	Update Source Statement Library, Assemble and Execute a Program
9	7	Using Procedure Library

Job Accounting Information.

If the job accounting interface is included in your VSE system, accounting information must be included in your JOB statements. For example:

```
// JOB  JOB1 0123456789ABCDEF
```

The jobname "JOB1" is followed by a required *single* blank. "0123456789ABCDEF" is accounting information. The accounting information will have to be something that is valid for your system. Ask your system operator or your advisor for this information.

Computer Exercise 1

Listing Device Assignments

Objective

Your objective is to obtain a listing of device assignments for all partitions and for a selected partition.

Materials Required

VSE/Advanced Functions System Control Statements manual

Instructions

Code two jobs as follows and run them:

Job 1.

Obtain a listing to show the device assignments for all partitions on your system. Name the job JOB1.

Job 2.

Obtain a listing to show the device assignments for only the background partition only. Name the job JOB2.

Keypunch

your solution, remove the next page - labeled "PROGRAM RUN DESCRIPTION" - and attach it to your solution.

Submit your solution for execution if a computer is available. If not, compare it to the solution in *Computer Exercise Solutions* (SR20-7301).

Comment:

Your output may be different than the output provided in the *Computer Exercise Solutions* manual (SR20-7301) due to different Input/Output devices and other variations in the systems.

PROGRAM RUN DESCRIPTION

Computer Exercise 1

Listing Device Assignments

```
*********************************************
*   NOTE: This page is for the use of the   *
*         operations department to assist in *
*         running student jobstreams. It should *
*         be attached, by the student, to the *
*         deck submitted for execution.       *
*********************************************
```

Execution requirements are listed on *Figure A.1,* which you should have received from the student's advisor.

Exercise Description

- these two jobs are to obtain listings of device assignments for all partitions and for the BG partition.
- student does not provide VSE/POWER in the job. If VSE/POWER is being used, please provide any required JECL statements.

Computer Exercise 2

Assembling A Program

Objective

Your objective is to to code a jobstream to assemble a source deck. The job stream will include:

- an OPTION statement to cause the Assembly program to produce an object card deck, a source listing, and a cross reference listing.
- an EXEC statement that invokes the Assembly program.

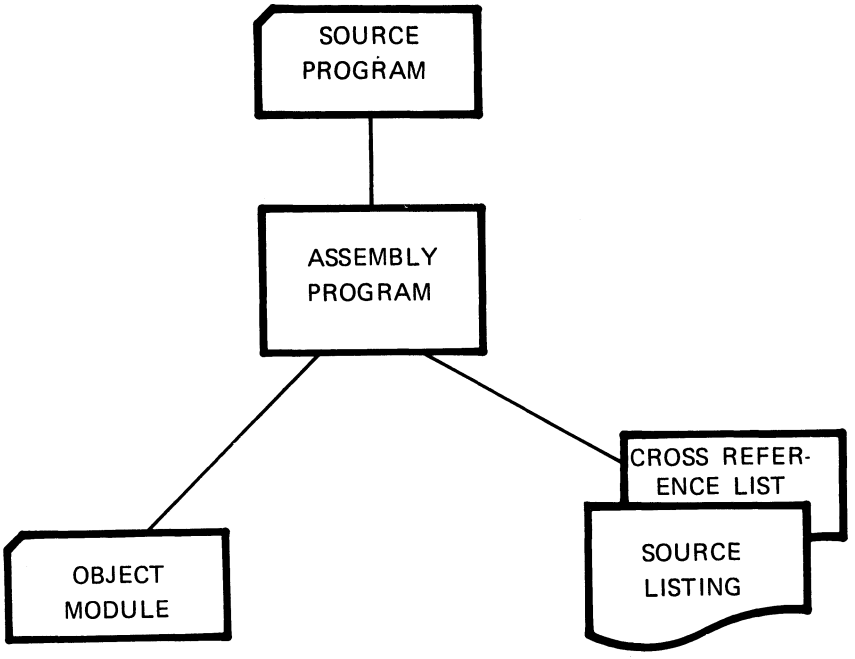
Materials Required

VSE/Advanced Functions System Control Statements manual

Computer Exercise Card Deck (SR20-7302) or equivalent

Examine the following schematic and proceed with the instructions on the next page.

ASSEMBLING A PROGRAM



Instructions

Assemble the source deck included in the course materials. Obtain from this assembly: an object card deck, a source listing, and a cross reference listing. Save the object deck for use in Computer Exercise 4 and in other computer exercises.

Keypunch

your solution, remove the next page - labeled "PROGRAM RUN DESCRIPTION" - and attach it to your solution.

Submit your solution for execution if a computer is available. If not, compare it to the solution in *Computer Exercise Solutions* (SR20-7301).

Note: Appendix B lists the DTFs used in the source deck provided. Have a knowledgeable person at your installation review these DTFs to make sure the macros for their assembly are in the Source Statement Library.

PROGRAM RUN DESCRIPTION

Computer Exercise 2

Assembling a Program

```
*************************************************
*   NOTE: This page is for the use of the       *
*         operations department to assist in    *
*         running student jobstreams. It should *
*         be attached, by the student, to the   *
*         deck submitted for execution.         *
*************************************************
```

Execution requirements are listed on *Figure A.1,* which you should have received from the student's advisor.

Exercise Description

- this one-step job executes the Assembly program to produce -
  - object deck
  - source listing
  - cross reference listing
- student assumes that:
  - required system logical unit and workfile assignments are already made.
  - required label information for Assembler workfiles is in the standard label area.
- student does not provide VSE/POWER JECL in the job. If VSE/POWER is being used, please provide any required JECL statements.

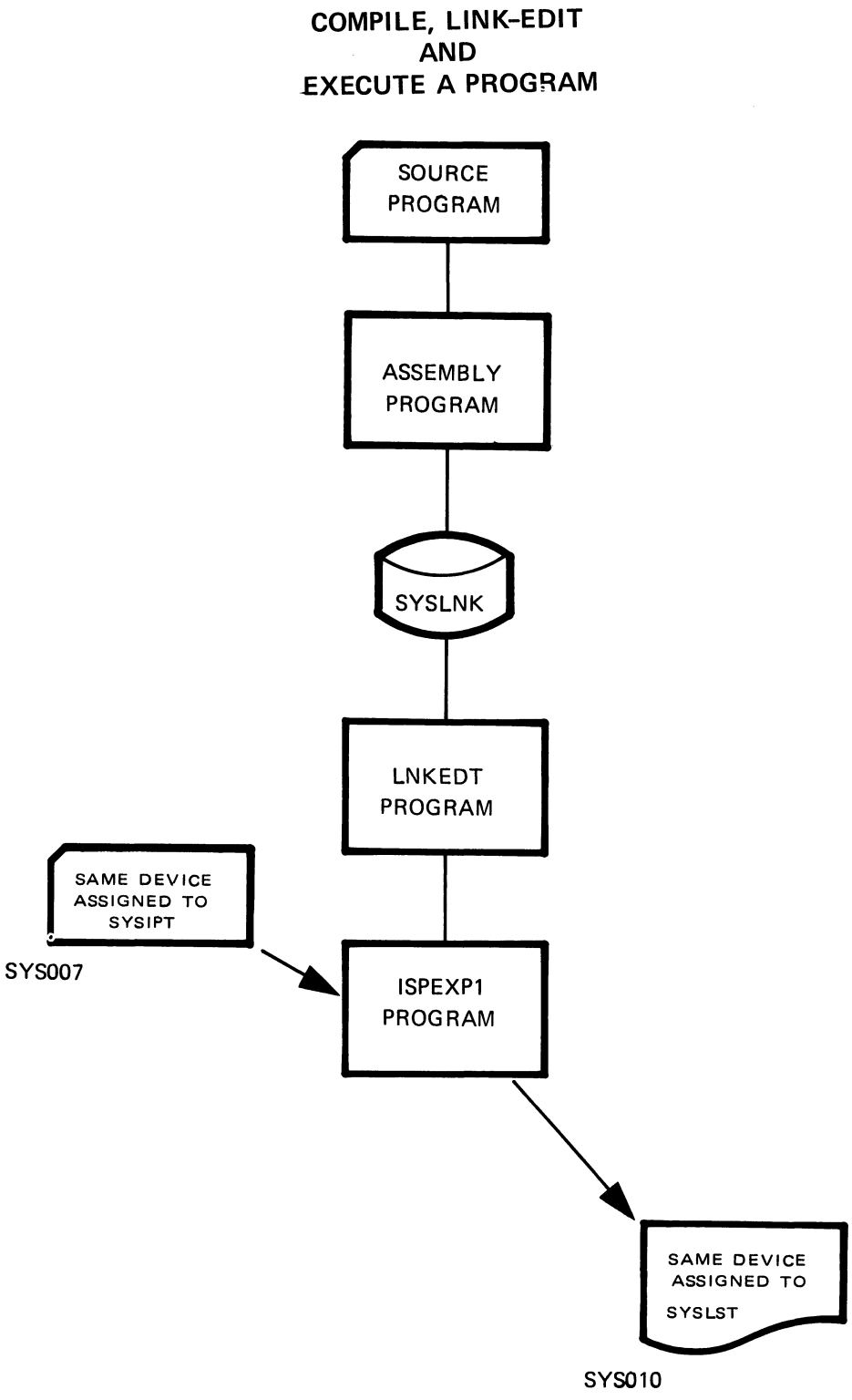
Your objective is to code a jobstream to assemble, temporarily catalog and execute a program. This will include coding to:

- specify that the output of an assembly is to be written to the system logical unit SYSLNK.
- specify that no object deck is to be punched by the Assembly program.
- invoke the LNKEDT program to temporarily catalog a phase in the Core Image Library.
- execute a temporarily cataloged phase.
- assign a programmer logical unit to the same physical device as a systems logical unit.
- specify a multiple-step job stream.
- include data in the job stream.

VSE/Advanced Functions System Control Statements manual

Computer Exercise Card Deck (SR20-7302) or equivalent

Examine the following schematic and proceed with the instructions on the next page.



Instructions

Code a three-step job:

Step 1.

Assemble the source deck included in the course materials. (This is the same source deck used in running Computer Exercise 2). Output from this step will be an object module. Cause the Assembly program to write it directly to the system logical unit SYSLNK.

Step 2.

Link-edit the object module produced in step 1. Temporarily catalog the resulting phase in the Core Image Library.

The VSE system requires a LIBDEF command to define into which core image library the linkage editor will catalog. The LIBDEF command and concatenated libraries are discussed in Unit 7. For this exercise, use the following command in Step 2:

```
//     LIBDEF     CL,TO=name
```

where "name" is supplied to you by your advisor.

Insert the LIBDEF statement in your job stream immediately following the // JOB statement.

Step 3.

Execute the phase temporarily cataloged in step 2. This phase reads a data card from SYS007. Assign SYS007 to the same device assigned to SYSIPT. The data card is to be punched:

008500001125070978

and included in the job. Card layout is:

Column 1-8:	principle amount (008500.00)
Column 9-12:	interest rate (11.25%)
Column 13-14:	no. of years for loan repayment (07)
Column 15-18:	month/year payments begin (09/78)

The program prints two pages of output on SYS010. Assign SYS010 to the same device assigned to SYSLST.

Keypunch

your solution, remove the next page - labeled "PROGRAM RUN DESCRIPTION" - and attach it to your solution.

Submit your solution for execution if a computer is available. If not, compare it to the solution in *Computer Exercise Solutions* (SR20-7301).

Note: The following modules must be present in an accessible Systems Relocatable Library in order for the LINK-EDIT portion of this job to run properly:

IJCFZIZ0
IJDFCZZZ
IJFFZZWZ

PROGRAM RUN DESCRIPTION

Computer Exercise 3

Assemble, Link-Edit And Execute

```
*********************************************
*   NOTE: This page is for the use of the   *
*         operations department to assist in *
*         running student jobstreams. It should *
*         be attached, by the student, to the *
*         deck submitted for execution.       *
*********************************************
```

Execution requirements are listed on *Figure A.1,* which you should have received from the student's advisor.

Exercise Description

- this three-step job is to assemble, temporarily catalog and execute a phase.
  - the phase reads a single data card from SYS007 which is assigned by the student
  - phase output is on SYS010 which is assigned by the student
- student assumes that:
  - required system logical unit and workfile assignments for assembly and link-edit are already made.
  - required label information for workfiles is in the label information area
  - required label information for the target Core Image Library is in the label information area, or has been supplied by the student's advisor
  - access to the necessary Relocatable Library (or Libraries) has been provided
- student does not provide VSE/POWER JECL in the job. If VSE/POWER is being used, please provide any required JECL statements.

Computer Exercise 4

Permanently Cataloging a Phase

Objective

Your objective is to link-edit an object deck and permanently catalog the resulting phase. This will include coding control statements to:

- specify that a phase is to be permanently cataloged.
- specify a name for cataloging a phase.
- specify object card deck input to the linkage editor.

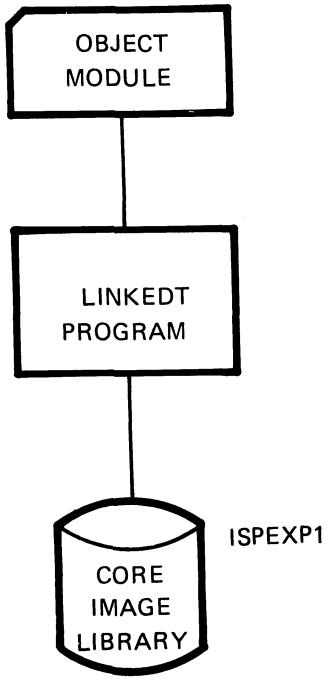
Materials Required

VSE/Advanced Functions System Control Statements manual

Object deck produced in Computer Exercise 2 or equivalent

Examine the following schematic and proceed with the instructions on the next page.

PERMANENTLY CATALOG AN EXECUTABLE PROGRAM



Instructions

Link-edit the object deck produced in Computer Exercise 2, and permanently catalog the resulting phase. Name the phase ISPEXP1.

The same LIBDEF statement that was used in Exercise 3 should be inserted after the // JOB statement in this job stream.

(ISPEXP1 will be executed in Computer Exercises 5 and 6, and should remain in the Core Image Library until you have completed these exercises.) The object deck you use should be saved for use in later exercises.

Keypunch

your solution, remove the next page - labeled "PROGRAM RUN DESCRIPTION" - and attach it to your solution.

Submit your solution for execution if a computer is available. If not, compare it to the solution in *Computer Exercise Solutions* (SR20-7301).

PROGRAM RUN DESCRIPTION

Computer Exercise 4

Permanently Cataloging A Phase

```
*********************************************
*   NOTE: This page is for the use of the   *
*         operations department to assist in *
*         running student jobstreams. It should *
*         be attached, by the student, to the *
*         deck submitted for execution.       *
*********************************************
```

Execution requirements are listed on *Figure A.1,* which you should have received from the student's advisor.

Exercise Description

- this single step job catalogs a phase named ISPEXP1 into a Core Image Library.
  - ISPEXP1 is to remain in the library until the student completes Computer Exercises 5 and 6.
- student assumes that:
  - required system logical unit and workfile assignments for the linkage editor are already made.
  - required label information for the SYS001 workfile is in the standard label area.
  - required label information for the target Core Image Library is in the label information area, or has been supplied by the student's advisor.
  - access to the necessary Relocatable Library (or Libraries) has been provided.
- student does not provide VSE/POWER JECL in the job. If VSE/POWER is being used, please provide required JECL statements.

Computer Exercise 5

Executing a Cataloged Program Using UPSI And Generic Assignments

Objective

Upon completing this exercise you will have demonstrated the ability to:

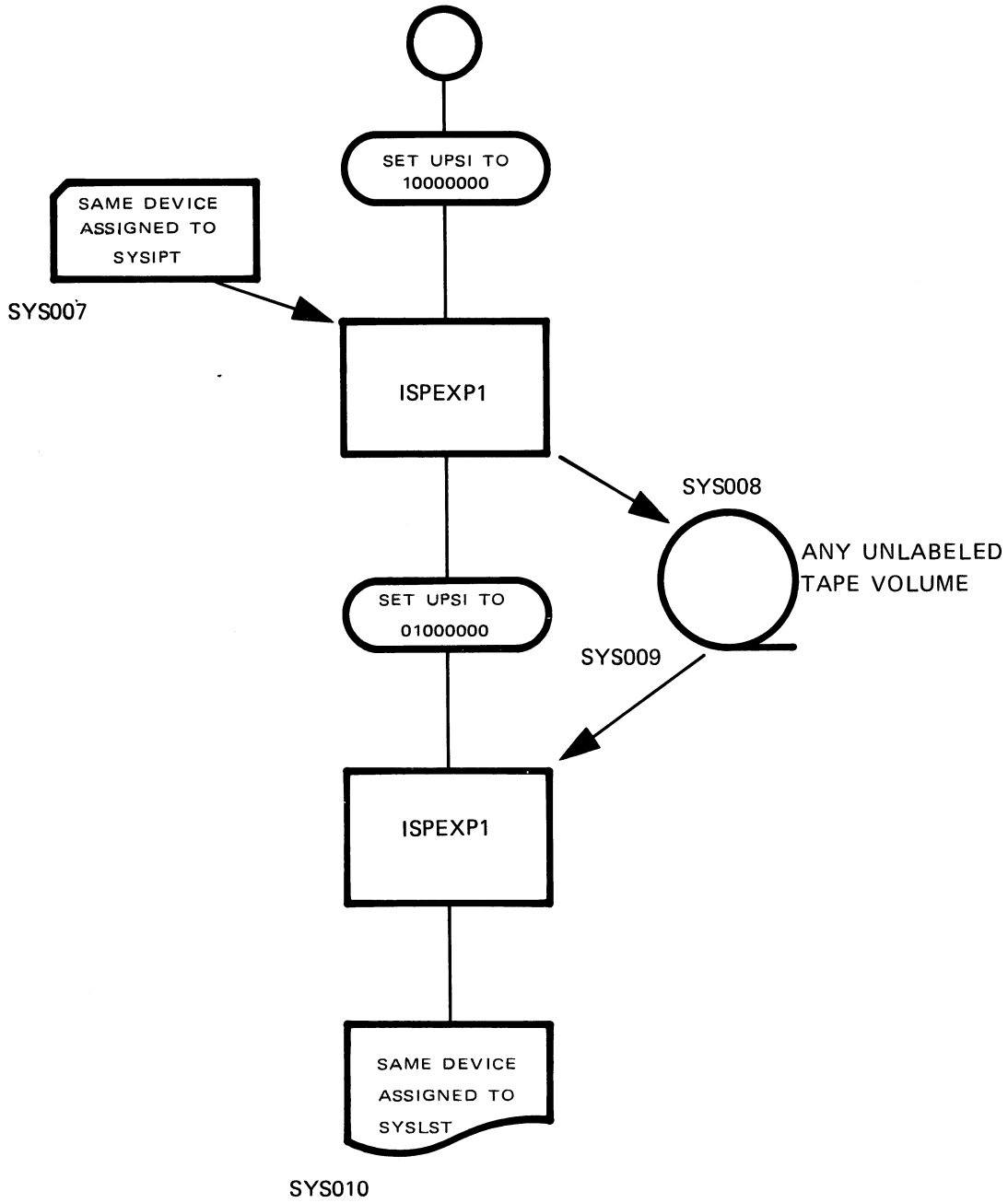
- reset the UPSI.
- generically assign a logical unit to a physical unit.
- pass a file created in one step of a job to a following step of the job.

Materials Required

VSE/Advanced Functions System Control Statements manual

Examine the following schematic and proceed with the instructions on the next page.

EXECUTING A CATALOGED PROGRAM USING THE UPSI AND GENERIC ASSIGNMENTS



Instructions

Code a two-step job.

Step 1.

Execute the phase named ISPEXP1 cataloged in Computer Exercise 4. Prior to executing the phase, set the user program switch indicators in the communications region to '10000000'. This UPSI setting will cause the program to write its output to SYS008 for temporary storage on an unlabeled magnetic tape. The tape will be input to the second step. Allow the system to select which physical tape device will be used.

The single data card (used in Computer Exercise 3) is to be processed from SYS007. It is:

008500001125070978

SYS007 should be assigned to the same device as SYSIPT.

Step 2.

Execute the phase ISPEXP1 again. This time, prior to execution, set the user program switch indicators to '01000000'. This UPSI setting will cause a tape file (created on SYS008 in step 1), to be read from SYS009 and listed on the printer. Assign SYS009 to the same device assigned in step 1 to SYS008.

The program will use SYS010 for printed output. SYS010 should be assigned to the same device as SYSLST.

Keypunch

your solution, remove the next page - labeled "PROGRAM RUN DESCRIPTION" - and attach it to your solution.

Submit your solution for execution if a computer is available. If not, compare it to the solution in *Computer Exercise Solutions* (SR20-7301).

PROGRAM RUN DESCRIPTION

Computer Exercise 5

Executing a Cataloged Program Using UPSI And Generic Assignments

```
*************************************************
*   NOTE: This page is for the use of the       *
*         operations department to assist in    *
*         running student jobstreams. It should *
*         be attached, by the student, to the   *
*         deck submitted for execution.         *
*************************************************
```

Execution requirements are listed on *Figure A.1,* which you should have received from the student's advisor.

Exercise Description

- a phase named ISPEXP1 is executed by both steps of this two-step job.
- step 1 reads a data card from SYS007, and writes a tape file to SYS008 (generically assigned). The file will be input to the second step (on SYS009).
- in step 2, the tape file created in step 1, is read from SYS009 and listed on the printer (SYS010). Student was requested to use for SYS009 the same drive used for SYS008 in step 1. This goal is to avoid the need to demount and mount the tape file between steps.
- return printed output to the student. The tape file is not to be saved.
- student does not provide VSE/POWER JECL in the job. If VSE/POWER is being used, please provide required JECL statements.

Computer Exercise 6

Creating and Accessing a Sequential Disk File

Objective

Your objective is to:

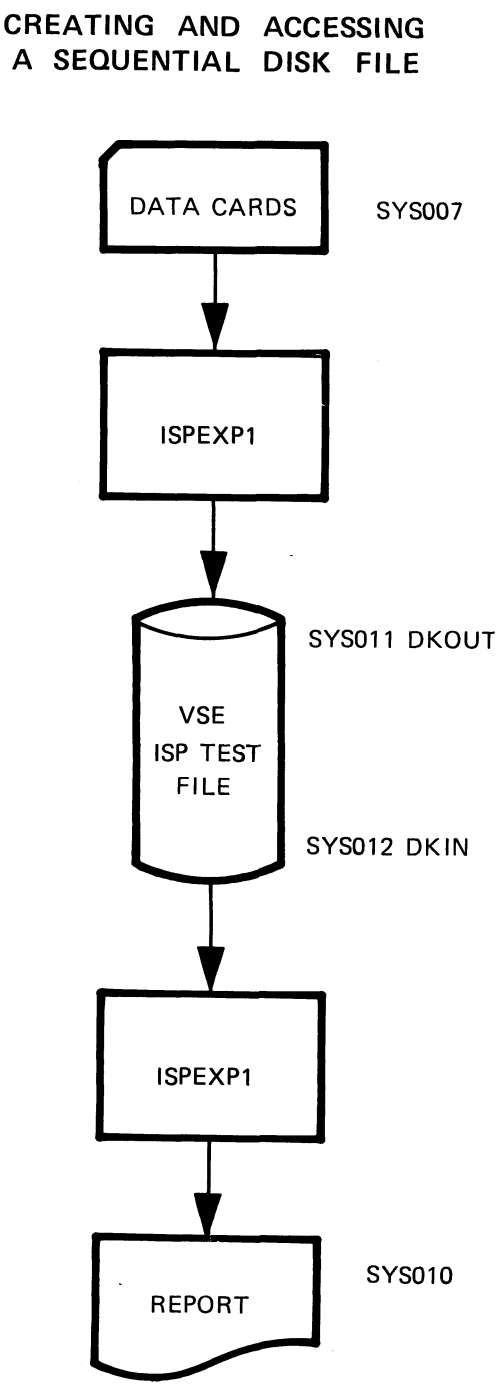
- code an ASSGN statement that generically assigns a Programmer Logical Unit to a disk drive, performs volume serial number checking, and shares the volume with other current users.
- code the DLBL and EXTENT statements needed to create a single extent sequential disk file and perform volume serial number checking at OPEN time.
- code the DLBL and EXTENT statements needed to access a sequential disk file.

Materials Required

Blank assembler coding forms

VSE/Advanced Functions System Control Statements manual

Examine the following schematic and proceed with the instructions that follow.



Instructions

This is a two-step jobstream.

Step 1

of this jobstream requires the execution of a phase named ISPEXP1. This is the phase you cataloged in Computer Exercise 4. Prior to executing the phase, the user program switch indicators in the communications region must be set to '00100000'. This will result in the output, from the program, being written to a sequential disk file. The ID for the file should be "VSE.ISP.TEST.FILE". The program will require that a disk unit be assigned to Programmer Logical Unit SYS011. The assignment should be made generically so that the pack may be shared with other users and the volume serial number will be checked when assigned. The volume serial number should also be checked when the file is opened.

The program will read the same data card that was read in Computer Exercises 3 and 5. The data card is:

008500001125070978

The program will read this card from Programmer Logical Unit SYS007. SYS007 should be assigned to the same device as SYSIPT.

For CKD, one cylinder of disk space will be required. You will need the following information from the operations department:

1. the disk volume number you may use, and
2. the cylinder number allocated to your disk file.

For FBA DASD, use 250 blocks of available disk space.

Step 2

of the jobstream again executes the phase ISPEXP1. Prior to execution, the user program switch indicators must be reset to '00010000'. This switch setting will result in the disk file that was written in step 1 being read and listed on the printer. The program will require that the pack be on a drive assigned to Programmer Logical Unit SYS012. To avoid having to make the operator unload and reload the pack or change drive addresses, be sure to assign SYS012 to the same drive used in running step 1. The printed output will be directed to Programmer Logical Unit SYS010. SYS010 should be assigned to the same device as SYSLST.

Keypunch

your solution, remove the next page - labeled "PROGRAM RUN DESCRIPTION" - and attach it to your solution.

Submit your solution for execution if a computer is available. If not, compare it to the solution in *Computer Exercise Solutions* (SR20-7301).

PROGRAM RUN DESCRIPTION

Computer Exercise 6

Creating and Accessing Sequential Disk

```
*****************************************************
* NOTE: This page is intended for the use of the    *
*       operations department to assist them in     *
*       running the student's jobstream. It should  *
*       be attached, by the student, to the card    *
*       deck submitted for execution.               *
*****************************************************
```

Execution requirements are listed on *Figure A.1,* which you should have received from the student's advisor.

Exercise Description.

- this is a two-step job.
- the program executed in this job is the one that was cataloged by the student in Computer Exercise 4. It was cataloged under the name ISPEXP1. Once the student has successfully completed the job, delete ISPEXP1 from the Core Image Library.
- the first step writes output to a sequential disk file. For CKD, one cylinder of disk space will be required. For FBA devices, 250 blocks of disk space will be required. You should provide the student with the volume serial number and a cylinder or block number for the file. This output file is read by the second step. There is no requirement to save the file.
- the program uses a card reader and a printer in addition to the disk.
- the problem statement does NOT assume that VSE/POWER is in control of your system. If it is being used, please provide the student with required JECL statements.

Computer Exercise 7

Catalog and Display a Source Program

Objective

Your objective is to code control statements to:

- catalog a program to the Source Statement Library
- list and punch the cataloged program from the library

Materials Required

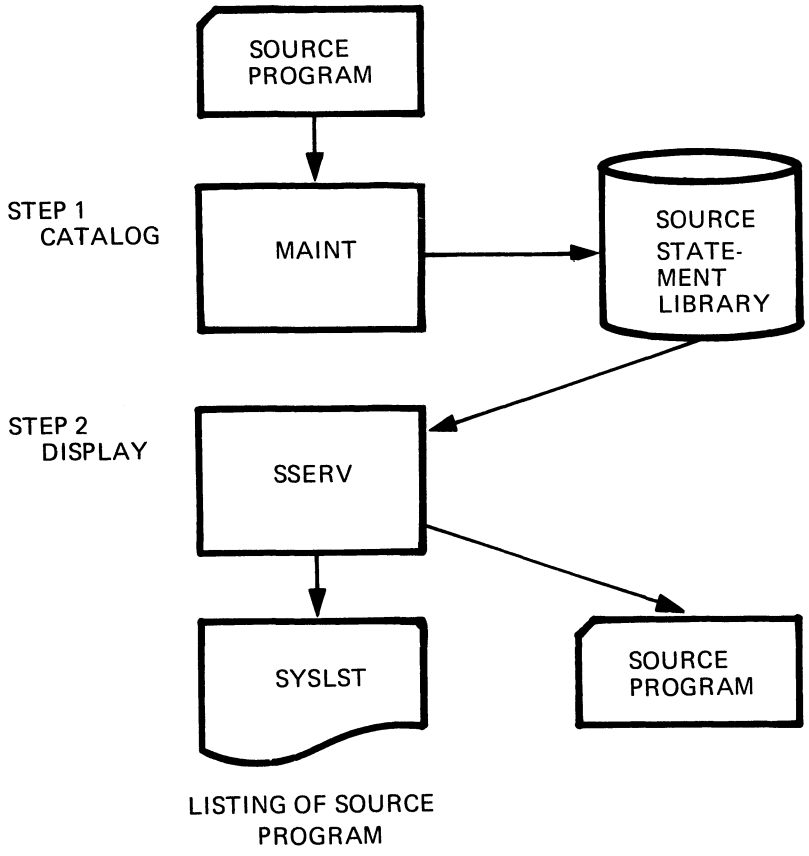
VSE/Advanced Functions System Control Statements manual

Computer Exercise Card Deck (SR20-7302) or equivalent

Coding Forms

Examine the following schematic and proceed with the instructions on the next page.

CATALOG AND DISPLAY A SOURCE PROGRAM



Instructions

Catalog the card deck (SR20-7302) to the Source Statement Library, then punch and list the program from the library. Name the book ISPEXP1.

Keypunch

your solution, remove the page - labeled "PROGRAM RUN INSTRUCTIONS" - and attach it to your solution.

Submit your solution for execution if a computer is available. If not, compare it to the solution in *Computer Exercise Solutions (SR20-7301).*

Here are some tips to help you get the most out of the computer exercise and gain confidence in doing similar coding dealing with libraries.

1. Don't look at the solution immediately when an exercise does not work.
2. Look at the message on the SYSLST output to find out which job step or job failed.
3. The *VSE/Advanced Functions Messages* manual usually gives a good indication where to start looking for the cause of the error. Check your input statements against the description of the statement in the System Control Statements manual. Compare examples of how the statements are used to do similar operations with how you did it.
4. If this does not help, compare your solution to the suggested solution in *Computer Exercise Solutions (SR20-7301).* If you do not understand this solution, discuss it with an experienced person.

PROGRAM RUN INSTRUCTIONS

Computer Exercise 7

Catalog and Display a Source Program

```
*********************************************
*   NOTE: This page is for the use of the   *
*         operations department to assist in *
*         running student jobstreams. It should *
*         be attached, by the student, to the *
*         deck submitted for execution.       *
*********************************************
```

Execution requirements are listed on *Figure A.1,* which you should have received from the student's advisor.

Exercise Description

- this is a two-step job that catalogs a program to the Source Statement Library and produces
  - a listing of the program cataloged
  - a card deck of the program cataloged
- student assumes that:
  - required system logical units are already assigned.
  - access to the designated Source Statement Library is provided.
- student does not provide VSE/POWER JECL in the job. If VSE/POWER is being used, please provide required JECL statements.
- Cataloged program should remain in the Source Statement Library for the next computer exercise.

Computer Exercise 8

Update Source Statement Library Program, Assemble and Execute Program

Objective

Your objective is to:

- Update a Source Statement Library book, which is an assembler language program.
- Assemble and execute the program.

Materials Required

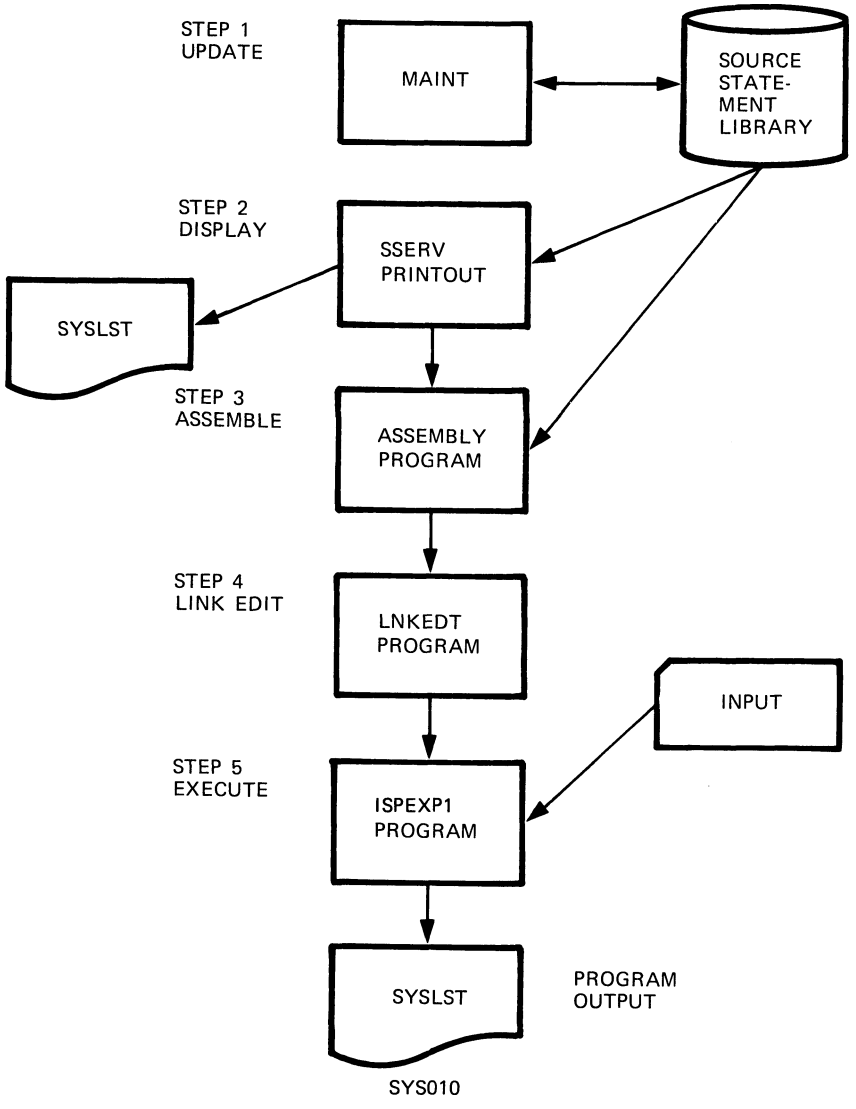
VSE/Advanced Functions System Control Statements manual

Coding Forms

Listing of cataloged program (computer exercise 7 output)

Examine the following schematic and proceed with the instructions on the next page.

UPDATE SOURCE STATEMENT LIBRARY PROGRAM
ASSEMBLE AND EXECUTE IT



DELETE BOOKS OLDEXP1 AND ISPEXP1 WHEN
YOU HAVE COMPLETED THIS COMPUTER EXERCISE

Instructions

In this exercise the heading lines of the amortization table are changed. The name of the original source program is changed to OLDEXP1.

Update the program that was cataloged in computer exercise 7 as follows:

- change name of the original cataloged source program from ISPEXP1 to OLDEXP1.
- resequence number updated source program in increments of 5
- include change level verification starting with 1.1.
- make changes in the source program as indicated in the figures below:

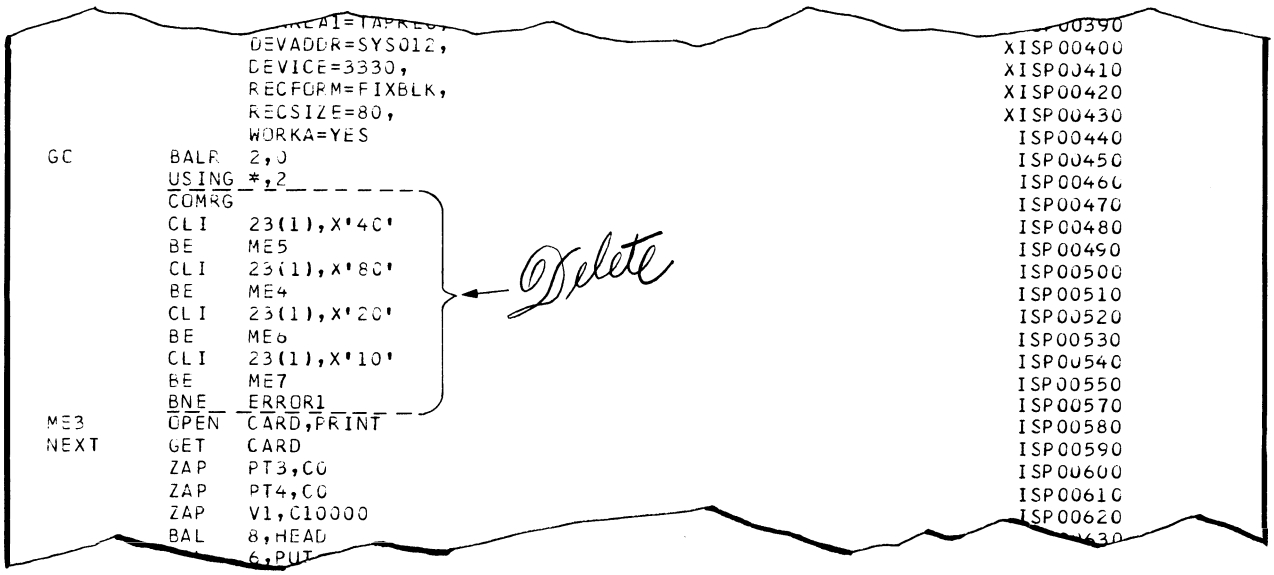
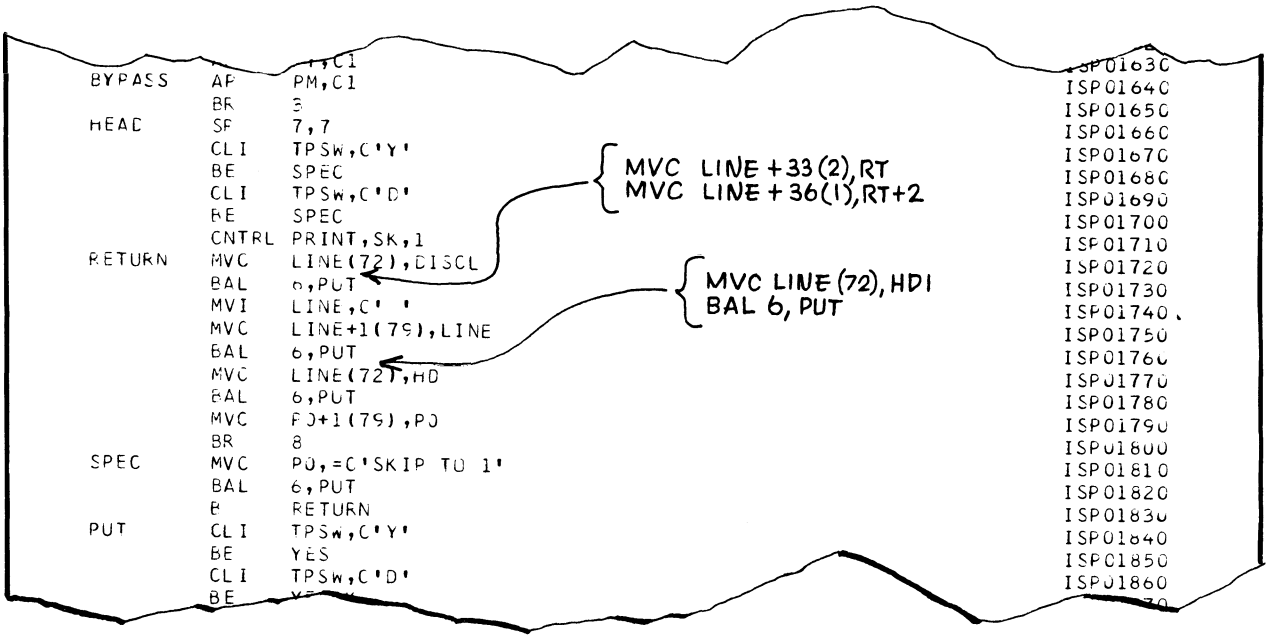
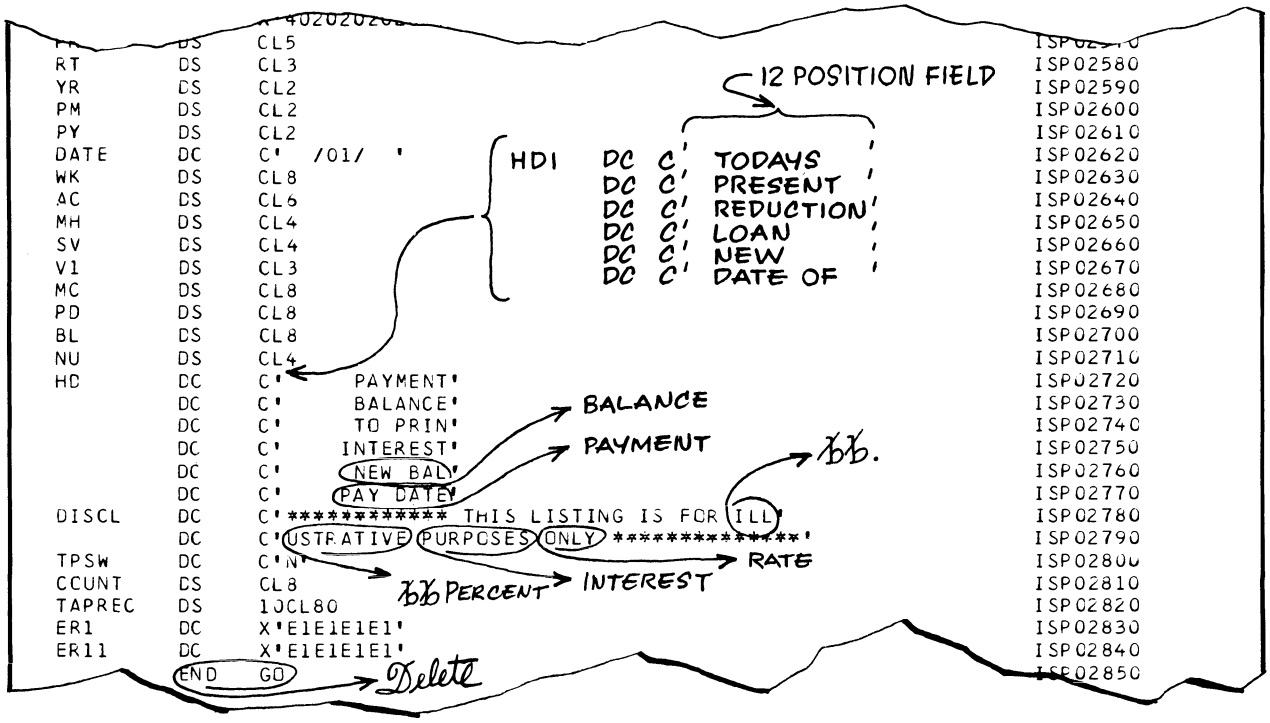


Figure 8.1 Computer Exercise 8




```
BYPASS   AP      PM,C1                                    ISP01630
         BR      3                                        ISP01640
HEAD     SP      7,7                                      ISP01650
         CLI     TPSW,C'Y'                                ISP01660
         BE      SPEC                                     ISP01670
         CLI     TPSW,C'D'                                ISP01680
         BE      SPEC                                     ISP01690
         CNTRL   PRINT,SK,1                               ISP01700
RETURN   MVC     LINE(72),DISCL                           ISP01710
         BAL     6,PUT                                    ISP01720
         MVI     LINE,C' '                                ISP01730
         MVC     LINE+1(79),LINE                          ISP01740
         BAL     6,PUT                                    ISP01750
         MVC     LINE(72),HD                              ISP01760
         BAL     6,PUT                                    ISP01770
         MVC     PD+1(79),PD                              ISP01780
         BR      8                                        ISP01790
SPEC     MVC     PD,=C'SKIP TO 1'                         ISP01800
         BAL     6,PUT                                    ISP01810
         B       RETURN                                   ISP01820
PUT      CLI     TPSW,C'Y'                                ISP01830
         BE      YES                                      ISP01840
         CLI     TPSW,C'D'                                ISP01850
         BE      YES                                      ISP01860
```

Figure 8.2 Computer Exercise 8




```
RT       DS      CL5                                      ISP02580
YR       DS      CL3                                      ISP02590
PM       DS      CL2                                      ISP02600
PY       DS      CL2                                      ISP02610
DATE     DC      C' /01/  '                               ISP02620
WK       DS      CL8                                      ISP02630
AC       DS      CL6                                      ISP02640
MH       DS      CL4                                      ISP02650
SV       DS      CL4                                      ISP02660
V1       DS      CL3                                      ISP02670
MC       DS      CL8                                      ISP02680
PD       DS      CL8                                      ISP02690
BL       DS      CL8                                      ISP02700
NU       DS      CL4                                      ISP02710
HD       DC      C'     PAYMENT'                          ISP02720
         DC      C'     BALANCE'                          ISP02730
         DC      C'     TO PRIN'                          ISP02740
         DC      C'   INTEREST'                          ISP02750
         DC      C'   NEW BAL'                           ISP02760
         DC      C'   PAY DATE'                          ISP02770
DISCL    DC      C'********** THIS LISTING IS FOR ILL'    ISP02780
         DC      C'USTRATIVE PURPOSES ONLY **********'    ISP02790
TPSW     DC      C'N'                                     ISP02800
CCUNT    DS      CL8                                      ISP02810
TAPREC   DS      1OCL80                                   ISP02820
ER1      DC      X'E1E1E1E1'                              ISP02830
ER11     DC      X'E1E1E1E1'                              ISP02840
         END     GO                                       ISP02850
```

Figure 8.3 Computer Exercise 8

- Obtain a listing of the updated program and the original program.
- Assemble, link-edit and execute the program.

The input card for the program starts in column 1 and must contain:

003000000600050877

Card input is on SYS007 and printer output is SYS010. Assign SYS010 to the same device assigned to SYSLST. The resultant output for this job stream will be the:

Listing of updates made (NOTE: Error message 3U31I is normal on this listing)
Status report for Source Statement Library
Listing of OLDEXP1
Listing of ISPEXP1
External Symbol Dictionary
Assembler listing
Relocation dictionary
Cross-reference
Diagnostics and statistics
Linkage editor diagnostic of input
Listing of an amortization table (job output)

NOTE:

If you rerun this program, delete ISPEXP1 (updated version of program cataloged with exercise 7) and rename OLDEXP1, ISPEXP1. Under some conditions, it may be necessary to recatalog the computer exercise card deck to run the problem.

When you have run this job successfully, code and submit a job to:

A. delete ISPEXP1

B. delete OLDEXP1

You'll also note that you are being asked to delete statement 2850 from the program (Figure 8.3). This is because the Assembler must find its END statement *on the SYSIPT FILE* and not in the SSL. You will have to include an END GO AFTER THE COPY STATEMENT in your solution job stream to satify the Assembler's requirements.

Keypunch

your solution, remove the page - labeled "PROGRAM RUN INSTRUCTIONS" - and attach it to your solution.

Submit your solution for execution if a computer is available. If not, compare it to the solution in *Computer Exercise Solutions* (SR20-7301).

PROGRAM RUN INSTRUCTIONS

Computer Exercise 8

Update Source Statement Library, Assemble and Execute a Program

```
*************************************************
*   NOTE: This page is for the use of the       *
*         operations department to assist in    *
*         running student jobstreams. It should *
*         be attached, by the student, to the   *
*         deck submitted for execution.         *
*************************************************
```

Execution requirements are listed on *Figure A.1,* which you should have received from the student's advisor.

Exercise Description

- this is a five-step job that does the following:
  - updates a program
  - lists the updated program
  - assembles cataloged program
  - link-edits the program
  - executes the program
- it produces:
  - status report
  - SSERV listing
  - assembler listing
  - program output listing
- student assumes that:
  - required system logical unit and workfile assignments are already made.
  - required label information for Assembler workfiles is in the standard label area.
- student does not provide VSE/POWER JECL in the job. If VSE/POWER is being used, please provide required JECL statements.

Computer Exercise 9

Using Procedure Library

Objective

Your objective is to use the procedure library by coding a jobstream to:

- create a procedure
- catalog a procedure
- display a procedure
- execute a procedure with modifications
- delete the procedure

Materials Required

VSE/Advanced Functions System Control Statements manual

VSE/Advanced Functions System Management Guide

Object deck output from the Assembler in Problem 2.

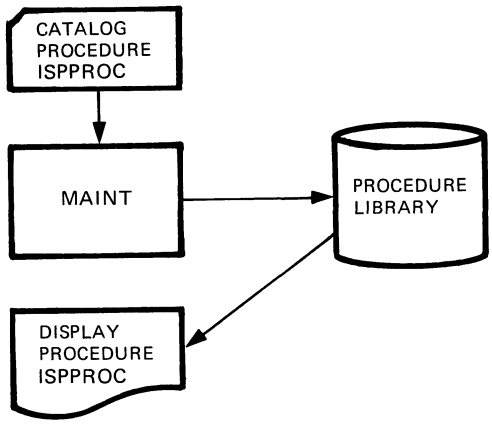
Coding Forms

Examine the following schematic and proceed with the instructions on the next page.

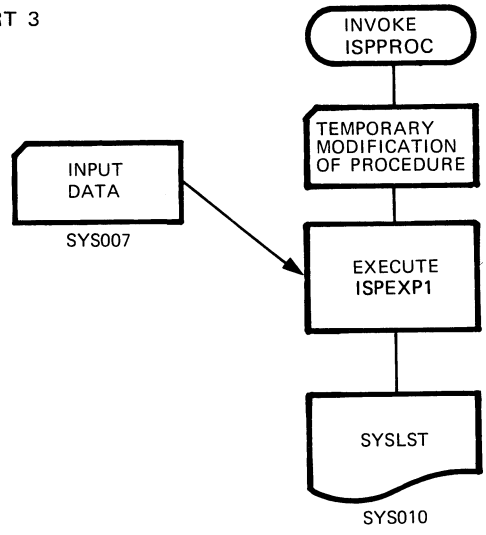
USING THE PROCEDURE LIBRARY

PART 1 RUN COMPUTER EXERCISE #4 (catalog object program to core image library)

PART 2



PART 3



PART 4 DELETE THE PROCEDURE ISPPROC and phase ISPEXP1

Instructions

In this exercise, you will create, catalog, use, and modify a procedure. There are four parts to the exercise. Part one is used to set up the conditions to create, use and modify a procedure. Part two is used to catalog a procedure and display procedure cataloged. Part three uses the procedure with modifications. Part four deletes the phase and procedure used in the exercise.

Part 1 -

Run Computer Exercise 4 to permanently catalog a phase.

Part 2 -

This part of the exercise creates, catalogs, and lists a procedure that includes data.

- Create and catalog a procedure named ISPPROC, to execute the Core Image Library phase, ISPEXP1. The procedure should include:
  - an assignment of SYS007 to device 02C
  - an assignment of SYS010 to device 02E
  - execute phase ISPEXP1
  - a comment statement - COMPUTER EXERCISE 9

  The statements within the procedure are to be modified during execution.
- Display the procedure cataloged

Part 3 -

This part of the exercise executes the procedure with the following modifications:

- change ASSGN cards to assgn SYS007 to SYSIPT and to assign SYS010 to SYSLST
- add comment statement - PROCEDURE ISPEXP9 before ASSGN statements
- add SIZE=AUTO to EXEC statement

The input card for the program starts in column 1 and must contain:

008500001125070978

NOTE: The intent of this part of the exercise is to use a procedure with modification rather than demonstrate good use of a procedure. This modification requires almost as many statements as the procedure itself, which is not a good use of a procedure.

Part 4 -

This part of the exercise handles "housekeeping." After the jobs for parts 1-3 have run successfully, code and submit a job to:

- delete phase ISPEXP1
- delete procedure ISPPROC

Keypunch

your solution, remove the page - labeled "PROGRAM RUN INSTRUCTIONS" - and attach it to your solution.

Submit your solution for execution if a computer is available. If not, compare it to the solution in *Computer Exercise Solutions* (SR20-7301).

PROGRAM RUN INSTRUCTIONS

Computer Exercise 9

Using Procedure Library

```
*************************************************
*   NOTE: This page is for the use of the       *
*         operations department to assist in    *
*         running student jobstreams. It should *
*         be attached, by the student, to the   *
*         deck submitted for execution.         *
*************************************************
```

Execution requirements are listed on *Figure A.1,* which you should have received from the student's advisor.

Exercise Description

- This computer exercise has four parts.

  The first part permanently catalogs a phase.

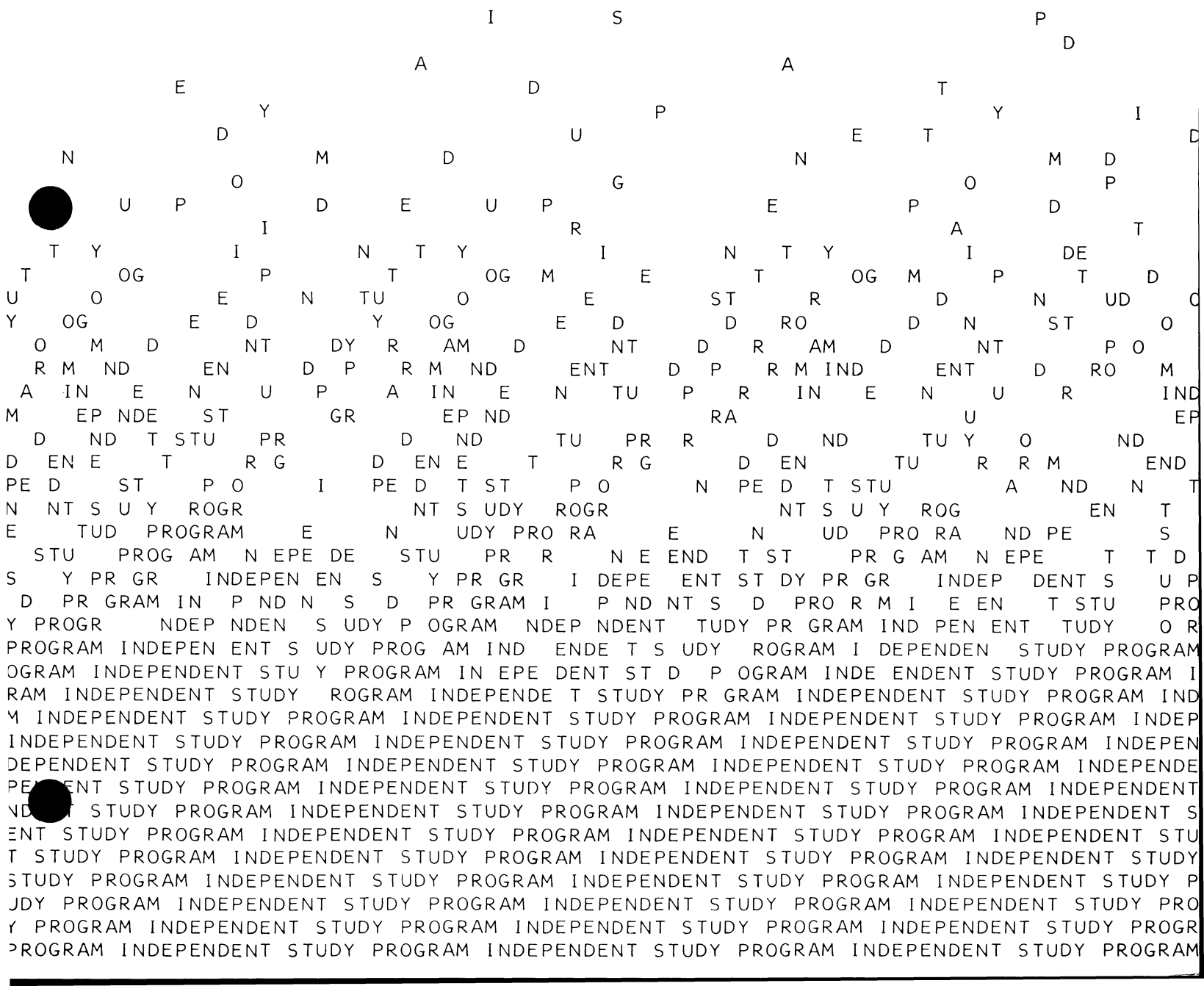
  Part two catalogs a procedure and then displays it.

  Part three executes a procedure with modifications.

  Part four deletes the cataloged procedure the the phase cataloged.

- student assumes that:
  - required system logical unit and workfile assignments are already made.
  - required label information for Assembler workfiles is in the standard label area.
- student does not provide VSE/POWER JECL in the job. If VSE/POWER is being used, please provide required JECL statements.

Appendix **B**



Appendix B:

Assembler Language Program

This appendix contains a copy of the program used in the Computer Exercises in Appendix A. It is a printed copy of the Assembler language statements punched in the Computer Exercise Card Deck (SR20-7302).

```
         PRINT NOGEN                                                    ISP00010
ISPEXP1  START                                                          ISP00020
CARD     DTFCD DEVADDR=SYS007,                                         XISP00030
               IOAREA1=CARDIN,                                         XISP00040
               DEVICE=2540,                                            XISP00050
               EOFADDR=LASTCD                                           ISP00060
PRINT    DTFPR DEVADDR=SYS010,                                         XISP00070
               IOAREA1=LINE,                                           XISP00080
               BLKSIZE=80,                                             XISP00090
               DEVICE=1403,                                            XISP00100
               CONTROL=YES                                              ISP00110
TAPOUT   DTFMT DEVADDR=SYS008,                                         XISP00120
               FILABL=NO,                                              XISP00130
               BLKSIZE=800,                                            XISP00140
               IOAREA1=TAPREC,                                         XISP00150
               RECFORM=FIXBLK,                                         XISP00160
               RECSIZE=80,                                             XISP00170
               TYPEFLE=OUTPUT,                                         XISP00180
               WORKA=YES                                                ISP00190
TAPIN    DTFMT DEVADDR=SYS009,                                         XISP00200
               FILABL=NO,                                              XISP00210
               BLKSIZE=800,                                            XISP00220
               EOFADDR=ENDTAP,                                         XISP00230
               IOAREA1=TAPREC,                                         XISP00240
               RECFORM=FIXBLK,                                         XISP00250
               RECSIZE=80,                                             XISP00260
               TYPEFLE=INPUT,                                          XISP00270
               WORKA=YES                                                ISP00280
DKOUT    DTFSD BLKSIZE=808,                                            XISP00290
               IOAREA1=COUNT,                                          XISP00300
               DEVADDR=SYS011,                                         XISP00310
               DEVICE=3330,                                            XISP00320
               RECFORM=FIXBLK,                                         XISP00330
               RECSIZE=80,                                             XISP00340
               TYPEFLE=OUTPUT,                                         XISP00350
               WORKA=YES                                                ISP00360
DKIN     DTFSD BLKSIZE=800,                                            XISP00370
               EOFADDR=ENDDK,                                          XISP00380
               IOAREA1=TAPREC,                                         XISP00390
               DEVADDR=SYS012,                                         XISP00400
               DEVICE=3330,                                            XISP00410
               RECFORM=FIXBLK,                                         XISP00420
               RECSIZE=80,                                             XISP00430
               WORKA=YES                                                ISP00440
GO       BALR  2,0                                                      ISP00450
         USING *,2                                                      ISP00460
         COMRG                                                          ISP00470
```

```
         CLI   23(1),X'40'                                    ISP00480
         BE    ME5                                            ISP00490
         CLI   23(1),X'80'                                    ISP00500
         BE    ME4                                            ISP00510
         CLI   23(1),X'20'                                    ISP00520
         BE    ME6                                            ISP00530
         CLI   23(1),X'10'                                    ISP00540
         BE    ME7                                            ISP00550
         CLI   23(1),X'00'                                    ISP00560
         BNE   ERROR1                                         ISP00570
ME3      OPEN  CARD,PRINT                                     ISP00580
NEXT     GET   CARD                                           ISP00590
         ZAP   PT3,C0                                         ISP00600
         ZAP   PT4,C0                                         ISP00610
         ZAP   V1,C10000                                      ISP00620
         BAL   8,HEAD                                         ISP00630
         BAL   6,PUT                                          ISP00640
         PACK  PR,PRIN                                        ISP00650
         PACK  RT,RATE                                        ISP00660
         PACK  YR,YEAR                                        ISP00670
         PACK  PM,PMCD                                        ISP00680
         PACK  PY,PYCD                                        ISP00690
         ZAP   SV,YR                                          ISP00700
         MP    SV,C12                                         ISP00710
         ZAP   MC,PR                                          ISP00720
         BAL   3,CAL                                          ISP00730
         ZAP   AC,PD                                          ISP00740
ITER     AP    AC,V1                                          ISP00750
FND      ZAP   MH,SV                                          ISP00760
         ZAP   BL,PR                                          ISP00770
         ZAP   NU,C1                                          ISP00780
LOOP     CLI   SW,C'T'                                        ISP00790
         BNE   BACK                                           ISP00800
         CP    NU,C1                                          ISP00810
         BE    BACK                                           ISP00820
         BAL   3,PRT                                          ISP00830
BACK     ZAP   MC,BL                                          ISP00840
         BAL   3,CAL                                          ISP00850
         ZAP   WK,AC                                          ISP00860
         SP    WK,PD                                          ISP00870
         ZAP   P1,NU                                          ISP00880
         ZAP   P2,BL                                          ISP00890
         ZAP   P3,WK                                          ISP00900
         ZAP   P4,PD                                          ISP00910
         CP    WK,BL                                          ISP00920
         BNL   APPR                                           ISP00930
         SP    BL,WK                                          ISP00940
         ZAP   P5,BL                                          ISP00950
         AP    NU,C1                                          ISP00960
         SP    MH,C1                                          ISP00970
         BP    LOOP                                           ISP00980
         B     ITER                                           ISP00990
NBL      CLI   SW,C'T'                                        ISP01000
         BE    SKIP                                           ISP01010
```

```
         MVI   SW,C'T'                                          ISP01020
         B     FND                                              ISP01030
SKIP     ZAP   P3,BL                                            ISP01040
         ZAP   P5,C0                                            ISP01050
         BAL   3,PRT                                            ISP01060
         MVI   P0,C' '                                          ISP01070
         MVC   P0+1(79),P0                                      ISP01080
         BAL   6,PUT                                            ISP01090
         MVC   P2,PAT                                           ISP01100
         MVC   P3,PAT                                           ISP01110
         ED    P2,PT3+7                                         ISP01120
         ED    P3,PT4+7                                         ISP01130
         BAL   6,PUT                                            ISP01140
         MVI   SW,C'N'                                          ISP01150
         B     NEXT                                             ISP01160
LASTCD   CLI   TPSW,C'Y'                                        ISP01170
         BE    CLTAPE                                           ISP01180
         CLI   TPSW,C'D'                                        ISP01190
         BE    CLDISK                                           ISP01200
         CLOSE CARD,PRINT                                       ISP01210
         EOJ                                                    ISP01220
CLTAPE   CLOSE CARD,TAPOUT                                      ISP01230
         EOJ                                                    ISP01240
CAL      ZAP   WK,MC                                            ISP01250
         MP    WK,RT                                            ISP01260
         DP    WK,C12                                           ISP01270
         ZAP   WK,WK(6)                                         ISP01280
         L     5,=F'-4'                                         ISP01290
         SRP   WK,0(5),5                                        ISP01300
         ZAP   PD,WK                                            ISP01310
         BR    3                                                ISP01320
PRT      MVC   P0+2(10),SPAT                                    ISP01330
         AP    PT3,P3                                           ISP01340
         AP    PT4,P4                                           ISP01350
         ED    P0+2(10),P1+7                                    ISP01360
         MVC   P1,PAT                                           ISP01370
         ED    P1,P2+7                                          ISP01380
         MVC   P2,PAT                                           ISP01390
         ED    P2,P3+7                                          ISP01400
         MVC   P3,PAT                                           ISP01410
         ED    P3,P4+7                                          ISP01420
         MVC   P4,PAT                                           ISP01430
         ED    P4,P5+7                                          ISP01440
         MVI   P5,C' '                                          ISP01450
         MVC   P5+1(11),P5                                      ISP01460
         UNPK  DATE(2),PM                                       ISP01470
         MVZ   DATE+1(1),DATE                                   ISP01480
         UNPK  DATE+6(2),PY                                     ISP01490
         MVZ   DATE+7(1),DATE                                   ISP01500
         MVC   P5+4(8),DATE                                     ISP01510
         BAL   6,PUT                                            ISP01520
         MVI   P0,C' '                                          ISP01530
         MVC   P0+1(79),P0                                      ISP01540
         CP    PM,C12                                           ISP01550
         BNE   BYPASS                                           ISP01560
```

```
         LA    7,1(7)                                          ISP01570
         C     7,=F'4'                                         ISP01580
         BNE   NOHEAD                                          ISP01590
         BAL   8,HEAD                                          ISP01600
NOHEAD   BAL   6,PUT                                           ISP01610
         ZAP   PM,C0                                           ISP01620
         AP    PY,C1                                           ISP01630
BYPASS   AP    PM,C1                                           ISP01640
         BR    3                                               ISP01650
HEAD     SR    7,7                                             ISP01660
         CLI   TPSW,C'Y'                                       ISP01670
         BE    SPEC                                            ISP01680
         CLI   TPSW,C'D'                                       ISP01690
         BE    SPEC                                            ISP01700
         CNTRL PRINT,SK,1                                      ISP01710
RETURN   MVC   LINE(72),DISCL                                  ISP01720
         BAL   6,PUT                                           ISP01730
         MVI   LINE,C' '                                       ISP01740
         MVC   LINE+1(79),LINE                                 ISP01750
         BAL   6,PUT                                           ISP01760
         MVC   LINE(72),HD                                     ISP01770
         BAL   6,PUT                                           ISP01780
         MVC   P0+1(79),P0                                     ISP01790
         BR    8                                               ISP01800
SPEC     MVC   P0,=C'SKIP TO 1'                                ISP01810
         BAL   6,PUT                                           ISP01820
         B     RETURN                                          ISP01830
PUT      CLI   TPSW,C'Y'                                       ISP01840
         BE    YES                                             ISP01850
         CLI   TPSW,C'D'                                       ISP01860
         BE    YESDK                                           ISP01870
         PUT   PRINT                                           ISP01880
         BR    6                                               ISP01890
YES      PUT   TAPOUT,LINE                                     ISP01900
         BR    6                                               ISP01910
APPR     CLI   SW,C'T'                                         ISP01920
         BE    SKIP                                            ISP01930
         CP    V1,C1                                           ISP01940
         BE    NBL                                             ISP01950
         SP    AC,V1                                           ISP01960
         SRP   V1,64-1,0                                       ISP01970
         B     ITER                                            ISP01980
ME4      OPEN  CARD,TAPOUT                                     ISP01990
         MVI   TPSW,C'Y'                                       ISP02000
         B     NEXT                                            ISP02010
ME5      OPEN  PRINT,TAPIN                                     ISP02020
NEXTTAP  GET   TAPIN,LINE                                      ISP02030
         CLC   P0,=C'SKIP TO 1'                                ISP02040
         BE    DOSKIP                                          ISP02050
         BAL   6,PUT                                           ISP02060
         B     NEXTTAP                                         ISP02070
DOSKIP   CNTRL PRINT,SK,1                                      ISP02080
         B     NEXTTAP                                         ISP02090
ENDTAP   CNTRL PRINT,SK,1                                      ISP02100
         CLOSE PRINT,TAPIN                                     ISP02110
```

```
         EOJ                                                    ISP02120
ERROR1   PDUMP ER1,ER11                                         ISP02130
         EOJ                                                    ISP02140
ME6      OPEN  CARD,DKOUT                                       ISP02150
         MVI   TPSW,C'D'                                        ISP02160
         B     NEXT                                             ISP02170
CLDISK   CLOSE CARD,DKOUT                                       ISP02180
         EOJ                                                    ISP02190
YESDK    PUT   DKOUT,LINE                                       ISP02200
         BR    6                                                ISP02210
ME7      OPEN  PRINT,DKIN                                       ISP02220
NEXTDK   GET   DKIN,LINE                                        ISP02230
         CLC   P0,=C'SKIP TO 1'                                 ISP02240
         BE    DKSKIP                                           ISP02250
         BAL   6,PUT                                            ISP02260
         B     NEXTDK                                           ISP02270
DKSKIP   CNTRL PRINT,SK,1                                       ISP02280
         B     NEXTDK                                           ISP02290
ENDDK    CNTRL PRINT,SK,1                                       ISP02300
         CLOSE PRINT,DKIN                                       ISP02310
         EOJ                                                    ISP02320
CARDIN   DS    0CL80                                            ISP02330
PRIN     DS    CL8                                              ISP02340
RATE     DS    CL4                                              ISP02350
YEAR     DS    CL2                                              ISP02360
PMCD     DS    CL2                                              ISP02370
PYCD     DS    CL2                                              ISP02380
FILL     DS    CL66                                             ISP02390
LINE     DS    0CL80                                            ISP02400
P0       DS    CL12                                             ISP02410
P1       DS    CL12                                             ISP02420
P2       DS    CL12                                             ISP02430
P3       DS    CL12                                             ISP02440
P4       DS    CL12                                             ISP02450
P5       DS    CL12                                             ISP02460
P6       DC    C'          '                                    ISP02470
PT3      DC    PL12'0'                                          ISP02480
PT4      DC    PL12'0'                                          ISP02490
C0       DC    P'0'                                             ISP02500
C1       DC    P'1'                                             ISP02510
C12      DC    P'12'                                            ISP02520
C10000   DC    P'10000'                                         ISP02530
SW       DC    C'N'                                             ISP02540
PAT      DC    X'40202020206B2020214B2020'                      ISP02550
SPAT     DC    X'40202020202020202020'                          ISP02560
PR       DS    CL5                                              ISP02570
RT       DS    CL3                                              ISP02580
YR       DS    CL2                                              ISP02590
PM       DS    CL2                                              ISP02600
PY       DS    CL2                                              ISP02610
DATE     DC    C'  /01/  '                                      ISP02620
WK       DS    CL8                                              ISP02630
AC       DS    CL6                                              ISP02640
MH       DS    CL4                                              ISP02650
SV       DS    CL4                                              ISP02660
```

```
V1        DS    CL3                                                    ISP02670
MC        DS    CL8                                                    ISP02680
PD        DS    CL8                                                    ISP02690
BL        DS    CL8                                                    ISP02700
NU        DS    CL4                                                    ISP02710
HD        DC    C'     PAYMENT'                                        ISP02720
          DC    C'     BALANCE'                                        ISP02730
          DC    C'     TO PRIN'                                        ISP02740
          DC    C'    INTEREST'                                        ISP02750
          DC    C'     NEW BAL'                                        ISP02760
          DC    C'    PAY DATE'                                        ISP02770
DISCL     DC    C'************ THIS LISTING IS FOR ILL'                ISP02780
          DC    C'USTRATIVE PURPOSES ONLY ***************'            ISP02790
TPSW      DC    C'N'                                                   ISP02800
COUNT     DS    CL8                                                    ISP02810
TAPREC    DS    10CL80                                                 ISP02820
ER1       DC    X'E1E1E1E1'                                            ISP02830
ER11      DC    X'E1E1E1E1'                                            ISP02840
          END   GO                                                     ISP02850
```

PROGRESS FORM

Directions: Use this form to help you plan your study time and to help you keep track of your progress through the course/module. You may want to consult with your advisor or manager in planning your study time. The form can also be used to inform your advisor and manager of your progress.

VSE System Control

Name_____ **Start Date:**_____

Unit	Study Hours	Scheduled Completion	Date Completed	Student/Mgr. Initials
1	____	_____	_____	_____
2	____	_____	_____	_____
3	____	_____	_____	_____
4	____	_____	_____	_____
5	____	_____	_____	_____
6	____	_____	_____	_____
7	____	_____	_____	_____
8	____	_____	_____	_____

IBM Independent Study Program
VSE System Control - Study Guide

READER'S COMMENT FORM

Order No. SR20-7300-1

This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

Cut or Fold Along Line

Name _____

Address _____

_____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

SR20-7300-1

Reader's Comment Form

Cut or Fold Along Line

Fold and tape Please Do Not Staple Fold and tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Publishing/Media Support - Department 78L
IBM Education Center, Building 005
South Road
Poughkeepsie, New York 12602

Fold and tape Please Do Not Staple Fold and tape

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

SR20-7300-1