GC33-5371-7
File No. S370/S4300-34

Systems

DOS/VSE
System Management Guide

IBM

Eighth Edition (February, 1979)

This is a major revision of, and obsoletes, GC33-5371-6 and TNL GN33-9227. This edition applies to the IBM Disk Operating System/Virtual Storage Extended, DOS/VSE, and to all subsequent versions and releases until otherwise indicated in new editions or Technical Newsletters.

Changes are continually made to the information herein; before using this publication in connection with operation of IBM systems, consult the latest *IBM System/370 Bibliography*, GC20-0001, for the editions that are applicable and current.

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

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Laboratory, Publications Department, Schoenaicher Strasse 220, D–7030 Boeblingen, Germany. 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.

Summary of Amendments

Edition GC33-5371-7 documents:

- New processor support
  - 4331 and 4341
  - 3031
- New device support
  - 3310 and 3370 Direct Access Storage Devices
  - PRT1 printers (3289 Model 4 and 3203-5)
  - 5424 MFCU
- Improved installation aid
  - Maintain System History Program (MSHP)
- Improved supervisor functions
  - Pageable supervisor options
  - Deletion of obsolete supervisor options
  - Improved method for loading SVA
- Extension of label information area
- Common EREP for DOS/VSE and OS/VS
- VSE/Advanced Functions
  - Asynchronous operator communication
  - Up to seven partitions
  - Partition balancing
  - Library device independence
  - Implicit linkage editor invocation
  - Switchable fast CCW translation
  - Fast B- and C-transient fetch
  - User label area definition by means of the DLA command
  - User job-to-job communication by means of the JOBCOM macro
  - Automated system initialization
  - Removal of LBLTYP statement
  - Improved access control
  - Alternate dump files
  - Use of up to 15 extents for page data set

With DOS/VSE the following programming support has been removed:

- QTAM support
- Support of the System/370 Model 20 Emulator on System/370
- Support of the IBM 2321 Data Cell
- Support of the IBM 2495 Tape Cartridge Reader

This manual has been completely reorganized and rewritten. Changes are therefore not marked by vertical bars in the left margin.

THIS MANUAL ...

... is a guide to using the functions available with the system control programming (SCP) support of the IBM Disk Operating System/Virtual Storage Extended (DOS/VSE) in its specified operating environment: with VSE/Advanced Functions installed. In addition, the manual provides guide information for using significantly improved or extended functions that have been implemented as part of VSE/Advanced Functions. Such information has been marked off in this publication by shading. Unless specifically stated, all unshaded information applies to both the specified environment and the SCP only (without VSE/Advanced Functions) environment.

Note: In this publication, DOS/VSE refers to the specified operating environment.

SCP support of DOS/VSE is discussed on a conceptual and functional level. System management refers not only to the way DOS/VSE is organized, but also to the way the user can efficiently manage the SCP facilities at his disposal. This manual, therefore, does more than describe the functions and interaction of the control programs that constitute the SCP. It also describes how you – as a system planner, systems programmer, or applications programmer – can use DOS/VSE to your best advantage.

Before you begin reading this manual, you should be familiar with the information contained in the *Introduction to DOS/VSE.*

This book is not a guide to data management; instead, a separate manual is provided for this purpose, called the *DOS/VSE Data Management Concepts*.

After reading this manual and the above mentioned manuals, you should be able to turn directly to the DOS/VSE library of reference manuals in order to work with your operating system. A reference manual is organized so that you can easily retrieve specific information on the formats of the control statements, macro instructions, labels, and messages, which you deal with daily.

This manual is divided into four chapters:

Chapter 1: DOS/VSE Overview provides conceptual information on multiprogramming, virtual storage and multitasking.

Chapter 2: Planning the System gives planning information for system generation.

Chapter 3: Using the System provides information on how to use the system, in particular on the use of the IPL, job control, linkage editor, and librarian programs.

Chapter 4: Using the Facilities and Options of DOS/VSE provides guidance information on how to use facilities and options of DOS/VSE; for example, writing IPL and job control user exit routines, checkpointing and restarting a program, or designing programs for virtual mode execution.

For reference purposes the organization of the system residence disk file (SYSRES) is shown in Appendix A.

The following IBM manuals are referred to in the text of this manual:

DOS/VSE Data Management Concepts	GC24-5138
DOS/VSE Macro User's Guide	GC24-5139
DOS/VSE Macros Reference	GC24-5140
Guide to the DOS/VSE Assembler	GC33-4024
DOS/VSE IBM 3800 Printing Subsystem Programmer's Guide	GC26-3900
Introduction to DOS/VSE	GC33-5370
DOS/VSE Tape Labels	GC33-5374
DOS/VSE DASD Labels	GC33-5375
DOS/VSE System Control Statements	GC33-5376
DOS/VSE System Generation	GC33-5377
DOS/VSE Operating Procedures	GC33-5378
System Library Supplement to DOS/VSE Operating Procedures	SD12-5007
DOS/VSE Messages	GC33-5379
DOS/VSE Serviceability Aids and Debugging Procedures	GC33-5380
DOS/VSE System Utilities	GC33-5381
DOS/VSE OLTEP	GC33-5383
DOS/VSE Maintain System History Program (MSHP) User's Guide	GC33-6060
Data Security under DOS/VSE*	GC33-6077
VSE/Advanced Functions System Information	SC33-6107

---

* available 2nd Quarter 1979

Table of Contents

List of Figures

Chapter 1: DOS/VSE Overview

Chapter 2: Planning the System

Chapter 3: Using the System

Chapter 4: Using the Facilities and Options of DOS/VSE

Appendix A: System Layout on Disk

Chapter 1: DOS/VSE Overview

DOS/VSE is a combination of programs that interact with user-written programs running on a System/370 or IBM 3031 or on a 4300 Processor. A reference to System/370 implies, in this manual, a reference to the IBM 3031. When installed on an 4300 Processor, DOS/VSE may run in either 370 mode or ECPS:VSE mode. DOS/VSE installed on a System/370 runs in 370 mode only.

This chapter expands on the conceptual information contained in *Introduction to DOS/VSE* about the following topics:

- Multiprogramming
- Virtual storage
- Multitasking

Multiprogramming

Multiprogramming is a technique that allows the concurrent execution of more than one program in a single computer system. Multiprogramming balances the difference between the speed of the central processing unit (CPU) and the relatively slower speed of the I/O devices, and improves the overall throughput of the system.

When a single executing program requests an I/O operation, it may not be able to continue processing until the I/O request has been satisfied. During this time, the CPU is idle. With multiprogramming, when one program stops processing, the CPU is put at the disposal of another program.

A program is said to be *in control of the system* when its instructions are being executed by the CPU. A program can voluntarily yield control of the CPU, or control can be withdrawn from it. Programs that share the use of the CPU in multiprogramming do not have an equal claim on the CPU. Instead, one program is given a greater priority than another.

When a program must wait for an event to occur before it can continue processing, it yields control of the CPU. DOS/VSE then passes control to a program of lower priority. Conversely, DOS/VSE withdraws control from a program whenever a program with higher priority is ready to resume processing. This generally happens when the I/O operation for which the program has been waiting is completed.

Multiprogramming, therefore, allows the I/O operations of one program to be overlapped by the processing of other programs. When a program has to wait for the completion of an I/O operation, DOS/VSE sets the program in the wait state and selects another program for execution on the basis of its priority and readiness to run. This process, called task selection, is performed by the supervisor program of DOS/VSE. The supervisor is always resident in storage and controls many functions of DOS/VSE. The

supervisor is discussed in detail in the section *Tailoring the Supervisor* in *Chapter 2, Planning the System.*

Partitions

Efficient use of the system relates not only to the degree of CPU activity but also to storage management. Storage is allocated to partitions to accommodate the programs that will be executed in them. At times, only a portion of the partition is used by the program being executed. Some programs require a large partition. DOS/VSE automatically balances the storage demands made by programs by making processor storage not being used by one program available to a program in another partition as required.

The number of partitions supported equals the number of problem programs that can be executed concurrently within the system. There is always support for one background (BG) partition and one foreground (F1) partition. Optionally, support for up to three additional foreground partitions (F2 through F4) can be requested if you operate in an SCP only environment, and up to five additional foreground partitions (F2 through F6) if you have VSE/Advanced Functions installed; see Figure 1-1. The actual number of partitions in a particular configuration is a supervisor generation option, and as such is described in the section *Tailoring the Supervisor* in *Chapter 2, Planning the System.*

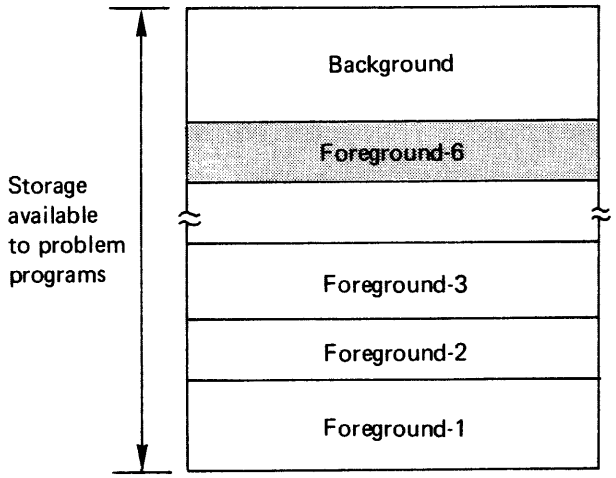


Figure 1-1. The Partitions of a DOS/VSE

The background partition differs from the foreground partitions in the following respects:

- The background partition is automatically activated by IPL. A foreground partition must be activated via the BATCH or START operator command. (The BATCH and START operator commands are discussed in detail in *DOS/VSE Operating Procedures.*)

- Certain IBM-supplied programs can be executed only in the background partition. These programs are CORGZ (the merge into SYSRES functions), and MAINT (except deleting, renaming and condensing functions for a private core image library). Refer to the section *Using the Libraries* in *Chapter 3, Using the System.*
- Link editing a program to the system core image library can be done only in the background partition.

Partition Priorities

During supervisor generation, priorities are established for each partition defined in the system. The default priorities are (from low to high): BG, F6, F5, F4, F3, F2, F1. Besides assigning a fixed priority to a certain partition, you can also specify two or more partitions for balancing. Balanced partitions are treated as a single entity within which the supervisor assigns priorities; that is, dynamically distributes CPU time to the individual partitions.

During processing the operator can display the partition priorities and change them dynamically by issuing the PRTY command. This can be used to accelerate the execution of a given program. However, the priorities should be reset to the installation standards as soon as possible to handle the normal flow of jobs through the system.

Changing priorities while jobs are being executed should be done with special care if the licensed program VSE/POWER or teleprocessing, which normally run in a high-priority partition, are active in the system.

Storage Protection

Storage protection, which is standard on all System/370 and 4300 processor models, ensures that the instructions and data of one program in a given partition do not interfere with those of another program in another partition.

Device Considerations Under Multiprogramming

Generally, the same physical I/O device (or extent of a direct access or diskette device) may not be used concurrently by programs being executed in different partitions. Exceptions to this are:

- The device or extents assigned to the system logical units:

SYSRES	for system residence
SYSREC	for the recording of system information such as console messages and hardware statistics
SYSLOG	for system-operator communication
SYSDMP	for alternate dump files
SYSCAT	for use with VSE/VSAM, a licensed DOS/VSE access method.

These devices (extents) are considered to belong to the system as a whole, rather than to individual partitions. (A description of these system logical units is contained in the section *Symbolic I/O Assignment* in *Chapter 3, Using the System*).

- The page data set.
- Private libraries which may be shared for read-only operations (for more information refer to *Using Private Libraries* in *Chapter 3, Using the System*).
- A file on a direct access device can be accessed across partitions, providing it is not being created simultaneously by programs in more than one partition (see *Track Hold Option* in *Chapter 2, Planning the System* for information on protection when updating a file concurrently by separate tasks).

If, for example, you wish to link-edit programs in different partitions concurrently, different physical devices or extents (except for SYSRES and SYSLOG) must be assigned for each partition to all logical units used by the linkage editor program. Figure 1-2 shows an example of the device assignments in order to link-edit in two partitions concurrently.

Logical Unit	F1 Partition	BG Partition
SYSIN	X'181'	X'00C'
SYSLST	X'182'	X'00E'
SYSLOG	X'01F'	X'01F'
SYSLNK	X'131'	X'132'
SYS001	X'131'	X'132'
SYSCLB	X'130'	—
SYSRES	X'130'	X'130'

Figure 1-2. Assigning Different Physical Devices to the Same Logical Units

In this case, the output on SYSLST in F1 is written on a tape. A listing of this output can be obtained by printing the tape after the job is completed. If VSE/POWER is used, the listing could be automatically obtained whenever a printer becomes available.

Virtual Storage

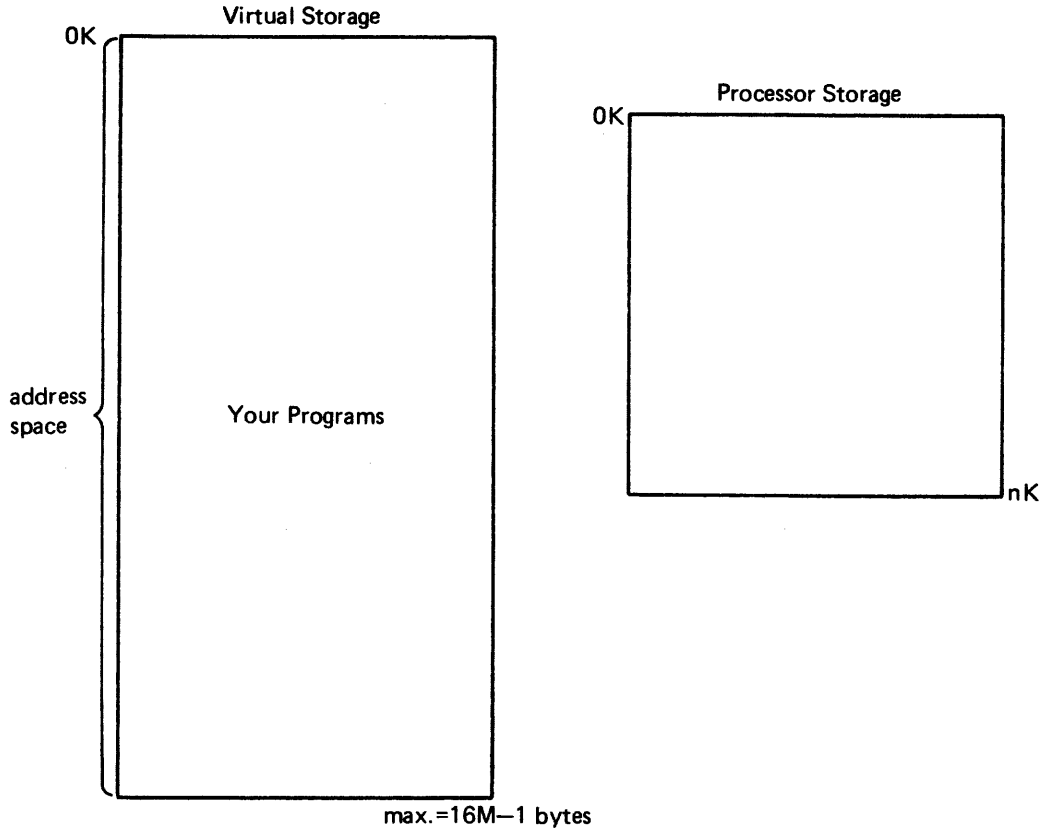
The objective of the virtual storage concept is to achieve greater throughput. Multiprogramming, for example, increases throughput by sharing CPU time between two or more partitions. Virtual storage enables you to improve real (processor) storage utilization.

In the previous multiprogramming discussion the statement is made that "Multiprogramming . . . allows the concurrent execution of more than one program . . . ". Note that concurrent does not mean simultaneous. Even in the multiprogramming environment, when two or more programs are executing in storage, the CPU (Central Processing Unit) can execute only one instruction at a time. Hence, the space in storage used by all other instructions, data areas etc. is temporarily not needed. All that must be in storage at any one point in time is the instruction (and its associated data areas) that is being executed. The Virtual Storage concept exploits this fact.

Virtual Storage in DOS/VSE

Through a combination of hardware design and programming support, DOS/VSE has an address space, called *virtual storage*, that can extend to the maximum allowed by the system's addressing scheme, which is 16,777,216 bytes (16M bytes).

How much of the maximum address space (16 M bytes) will be used in a particular system depends on a number of factors: the size of the computer's processor storage, the amount of disk storage available, the number of partitions, their sizes, and the characteristics of the installation's programs and operating environment.



It is in the address space that programs conceptually run.

Figure 1-3. Virtual Storage and Processor Storage

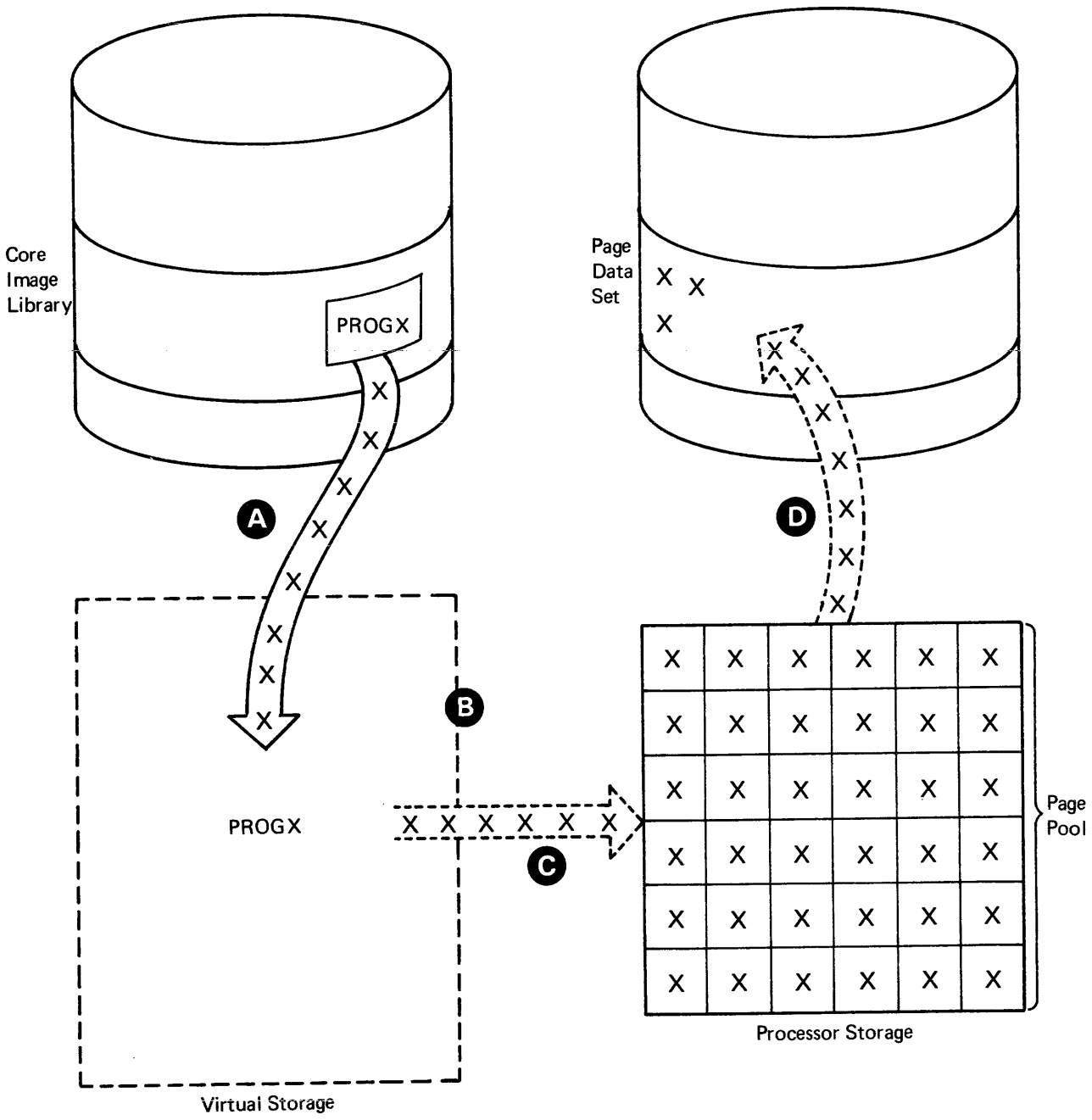
Your programs *are conceptually* loaded and run in *address space*. See Figure 1-3. Of course, each instruction of a program must be in processor storage when the instruction is executed, and so must the data this instruction manipulates. The other instructions and data of that program in virtual storage need not be in processor storage at that same moment; they can reside on auxiliary storage until needed. The file used for this purpose is called the *page data set*.

It would be inefficient, however, to bring every instruction and its associated data into processor storage individually. Virtual storage is manipulated in sections called *pages*; the size of a page in DOS/VSE is 2K bytes. Processor storage is also divided into 2K byte sections; these are called *page frames*. Page frames accommodate pages of a program during execution.

The resident routines of the DOS/VSE supervisor occupy the low address page frames, while the remaining page frames are available for the execution of processing programs and the pageable routines of the supervisor. These remaining page frames are collectively called the *page pool*.

When a program is loaded from the core image library into virtual storage, all its pages are brought into page frames of the page pool. If there

are not enough page frames available to contain all the pages of a program, DOS/VSE writes the contents of some page frames to the *page data set*. See Figure 1-4.



A program named PROGX (A) is "conceptually" loaded into virtual storage (B). DOS/VSE finds *page frames* in the *page pool* of processor storage (C). When there are not enough page frames to accomodate all of PROGX, DOS/VSE stores the contents of some page frames on the page data set (D). The remaining pages of the program can then be loaded.

Figure 1-4. Storage Management Concept – DOS/VSE

The following discussion amplifies the concept of DOS/VSE storage management shown in Figure 1-4.

When programs are loaded for execution they may be loaded in non-contiguous page frames of processor storage. DOS/VSE knows what processor storage locations pages of a given program occupy. If the program should cancel, due to an error, the listing produced by DOS/VSE reflects the virtual addresses where the program was conceptually running. In Figure 1-5, a 16K-byte program named INVEN, is conceptually loaded at the virtual storage location 1024K. As shown, DOS/VSE selected eight *page frames* of processor storage which are not contiguous. If the program were to end abnormally, and a listing representing storage was produced (on SYSLST), the INVEN program would be shown as occupying addresses 1024K through 1040K minus 1.

All of the information pertaining to the virtual storage and page frames is maintained within the system in a series of tables. It is through these tables that the virtual storage exists. Entries in these tables reflect the current status of a given page of virtual storage.

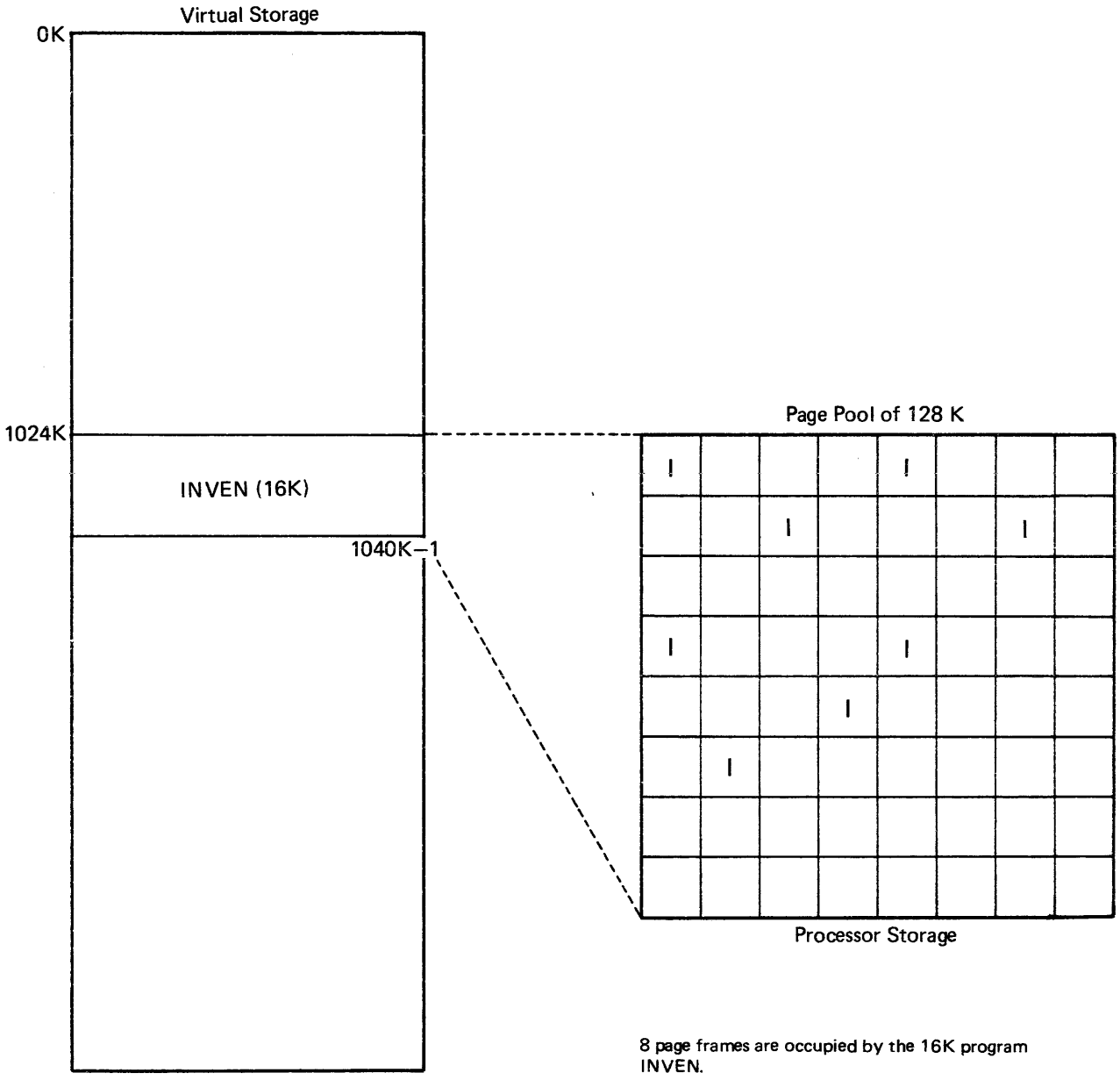


Figure 1-5. Running a Program in Virtual Storage

Relating Virtual Storage to Locations in Processor Storage

Since the system does not anticipate where in processor storage a page will be loaded, the virtual addresses must be translated into real addresses when required for execution. The address translation is performed by a combination of the system hardware and DOS/VSE.

If an entire program fits in processor storage, none of the program's pages will be placed on the page data set.

In the example shown in Figure 1-5, no page of INVEN will be *paged out* as long as the demand on processor storage does not exceed the number of available page frames.

If a second program were to be executed (multiprogramming) and this program together with INVEN were larger in size than the number of frames available in the page pool, DOS/VSE would store as many pages as necessary on the page data set to keep both programs running.

In Figure 1-6 a program called PAYROLL is being executed as well as INVEN. PAYROLL is a 118K program. As the *page pool* in this example is only 128K, the total demand (INVEN + PAYROLL) of 134K exceeds the processor storage resource by 6K or three page frames.

The program PAYROLL will not start executing until all of its pages have been loaded into processor storage. After having loaded 112K of program PAYROLL, DOS/VSE must make three page frames available for that program. It does this by selecting the three least recently used (LRU) pages and storing them on the page data set. See Figure 1-7. Once the pages have been saved on the page data set the page frames are available for the last three pages of the program PAYROLL. See Figure 1-8.

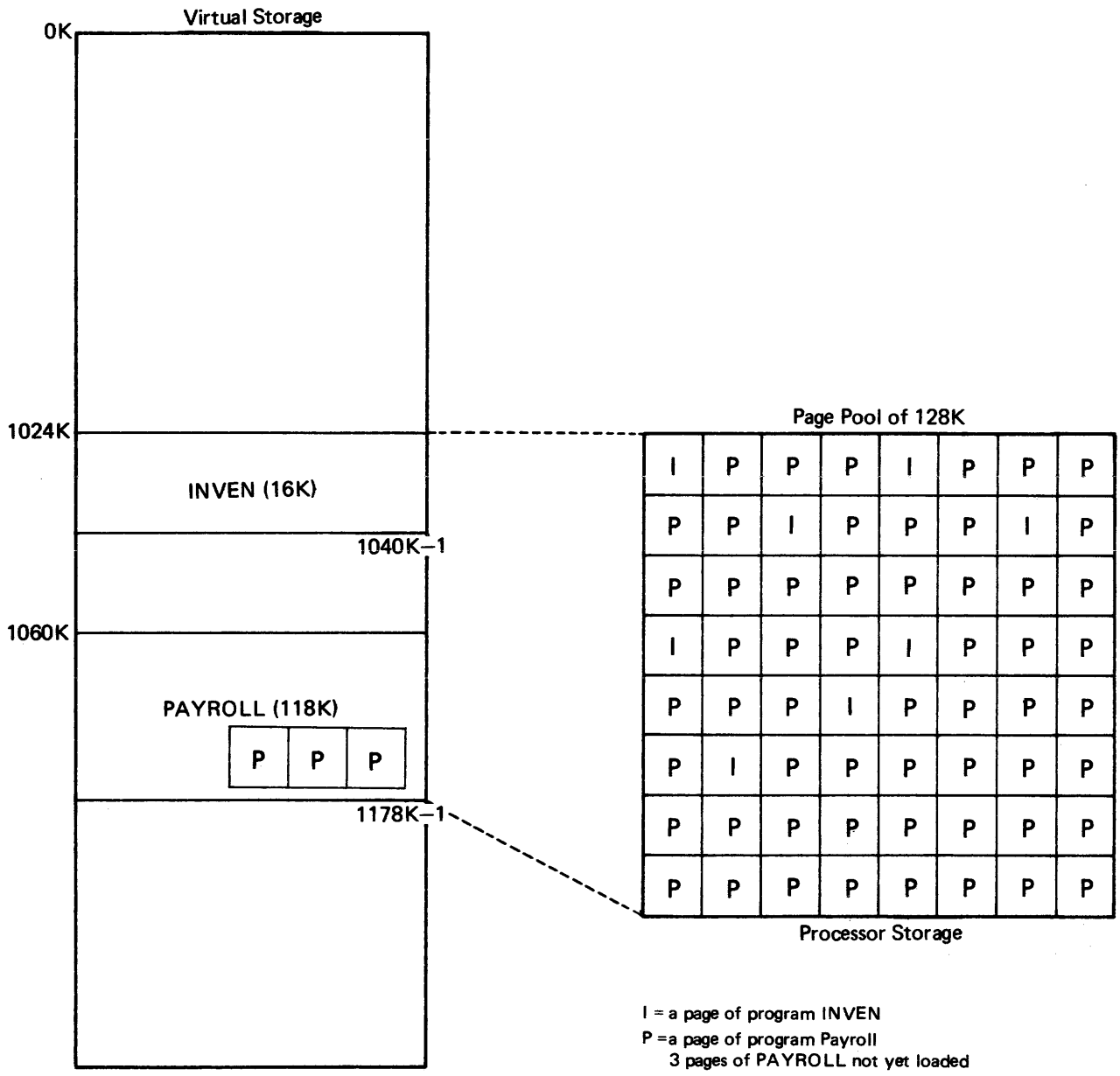


Figure 1-6. Loading Program Pages into Page Frames

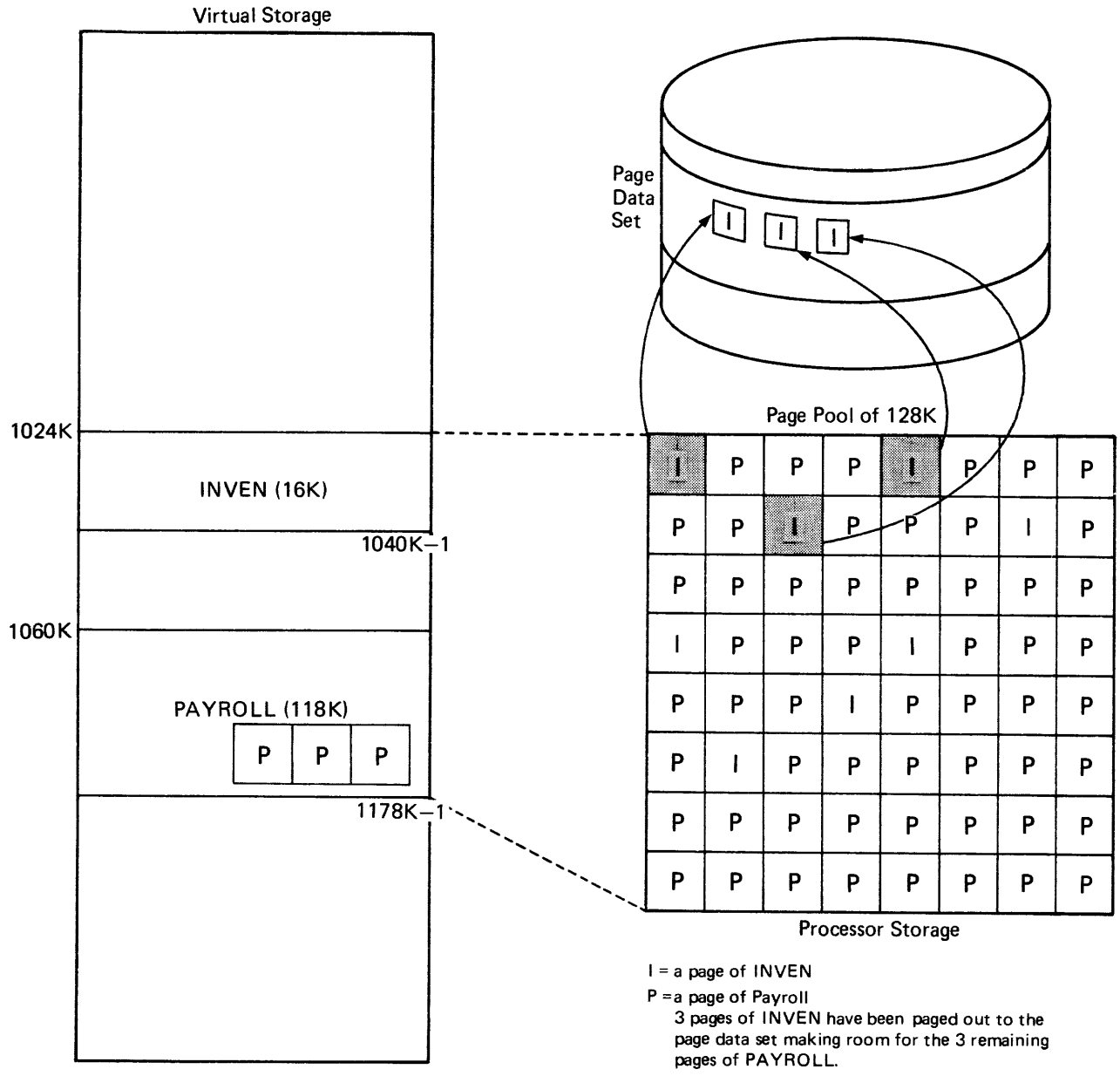
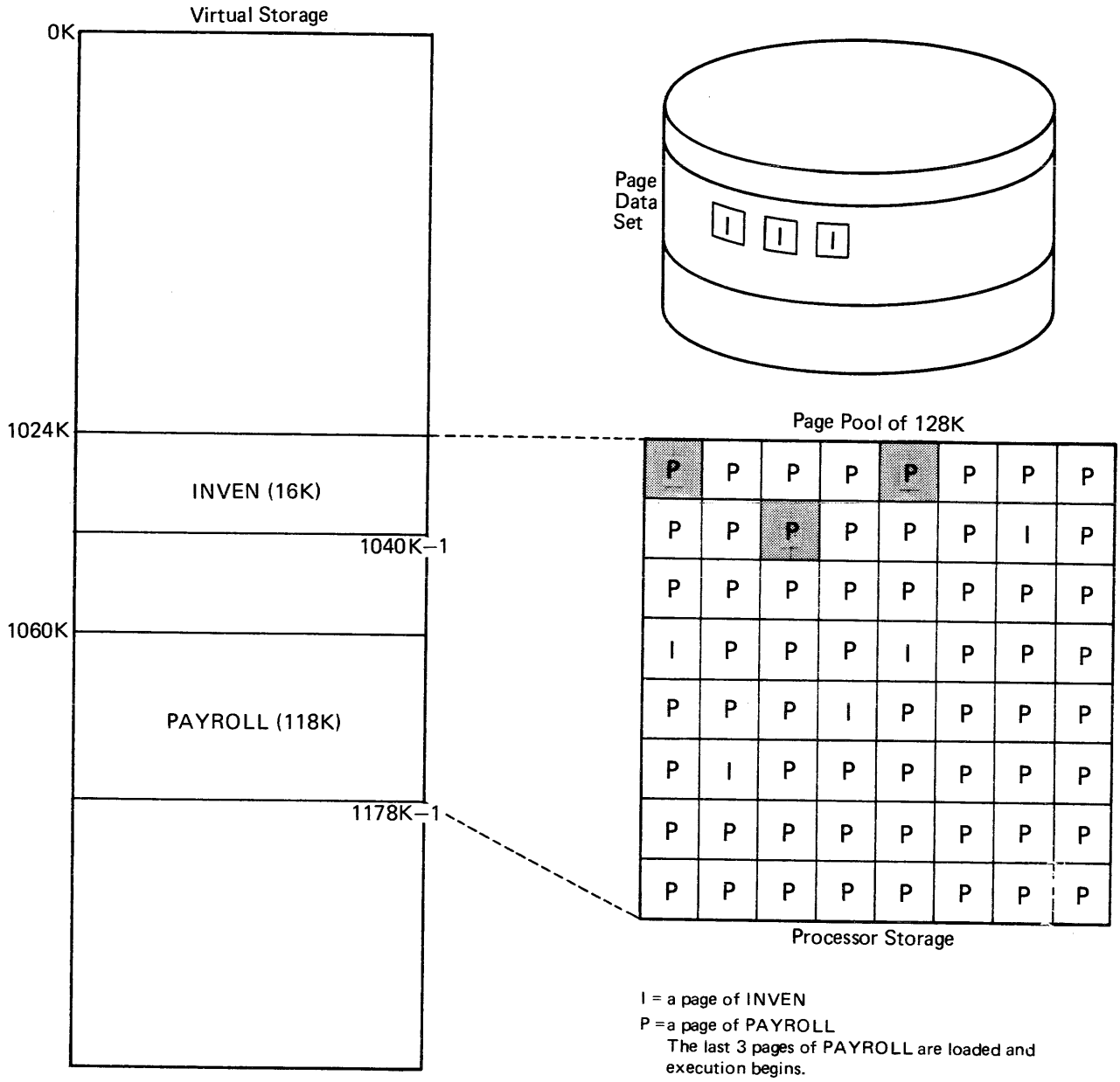


Figure 1-7. Storing Pages on the Page Data Set (pageouts)



During execution, whenever a required instruction or some data is not present in processor storage, execution is interrupted by a so-called *page fault*. DOS/VSE must then read the required page into processor storage.

Figure 1-8. Managing the Page Pool

Virtual Storage Implementation under DOS/VSE

Under DOS/VSE you may generate a system that will execute on 4300 or /370 hardware. Using the 4300 hardware, your DOS/VSE System may be generated to run in either ECPS:VSE mode or 370 mode. DOS/VSE on the System/370 hardware may only run in 370 mode.

The generated supervisor in 370 mode is functionally the same, whether the hardware is System/370 or a 4300 processor.

The concepts of virtual storage are the same in both modes of execution; however, the implementation differs slightly.

This section discusses: virtual storage, processor storage, and program execution (with and without paging). The implementation of most of these items is the same in both modes. The differences and figures showing the different execution modes (ECPS:VSE or 370) are at the end of this section.

Division of Address Space

As stated earlier, all programs, including the supervisor, run in an address space called virtual storage. This address space is divided into areas: for the supervisor, the partitions, a shared virtual area (SVA).

Supervisor Area. The address space reserved for the supervisor is the low addresses of your virtual storage. The supervisor area begins at location 0K and extends up to the size of your generated supervisor (see Figure 1-9).

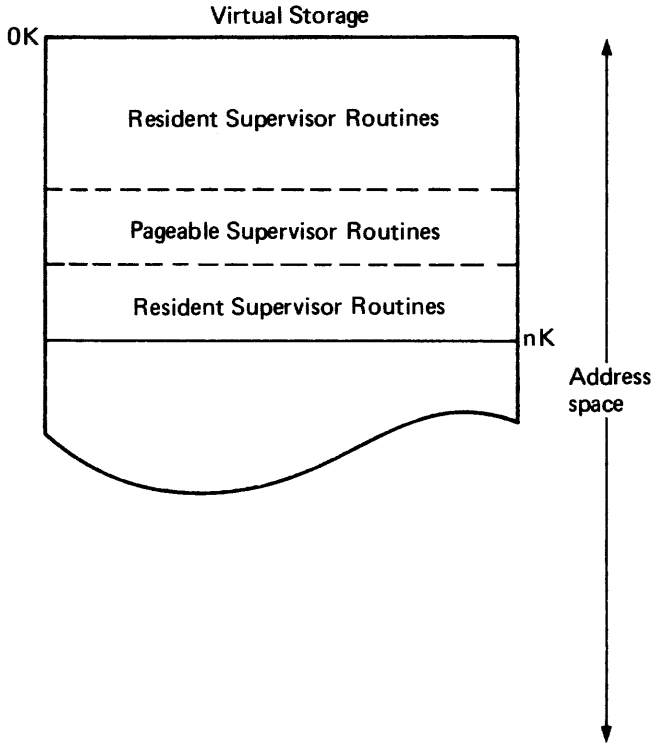


Figure 1-9. Supervisor Area in Virtual Storage Address Space

Partitions. The virtual storage contains the areas which are used by the DOS/VSE partitions. Programs will execute from these areas. The number of partitions is determined at system generation. See *Chapter 2, Planning the System*. The distribution of the partitions in the address space follows

the default partition priority scheme, that is the lower priority partitions have the lower addresses. The sequence is always BG, F4, F3, F2, F1 for a five partition system.

Figure 1-10 shows the layout of virtual storage for a 4-partition DOS/VSE system. In this figure each partition is 200K in size.

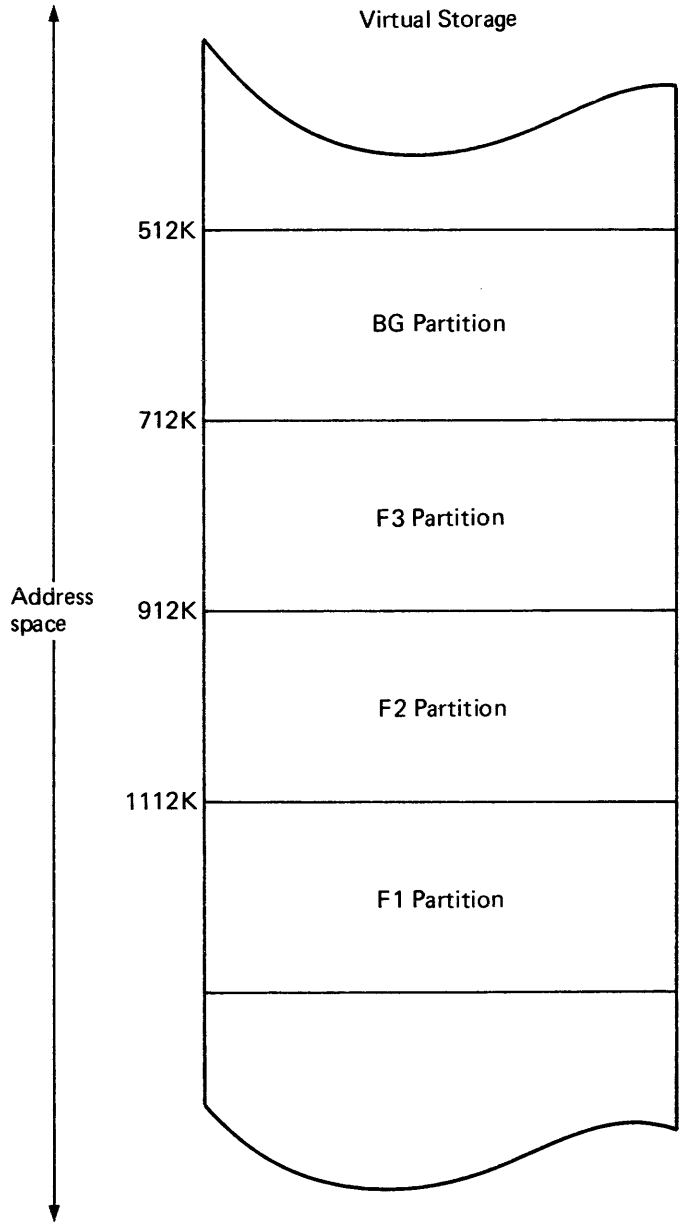


Figure 1-10. Partition Distribution in a Four-Partition System

The Shared Virtual Area (SVA). The SVA occupies the address space immediately following the partitions, see Figure 1-11. Certain frequently used programs are loaded into the SVA. Such programs (or parts of programs), which are relocatable and reenterable, are available for concurrent use by programs executing in any partition. Additional

information on the use of the SVA is contained in this guide where appropriate.

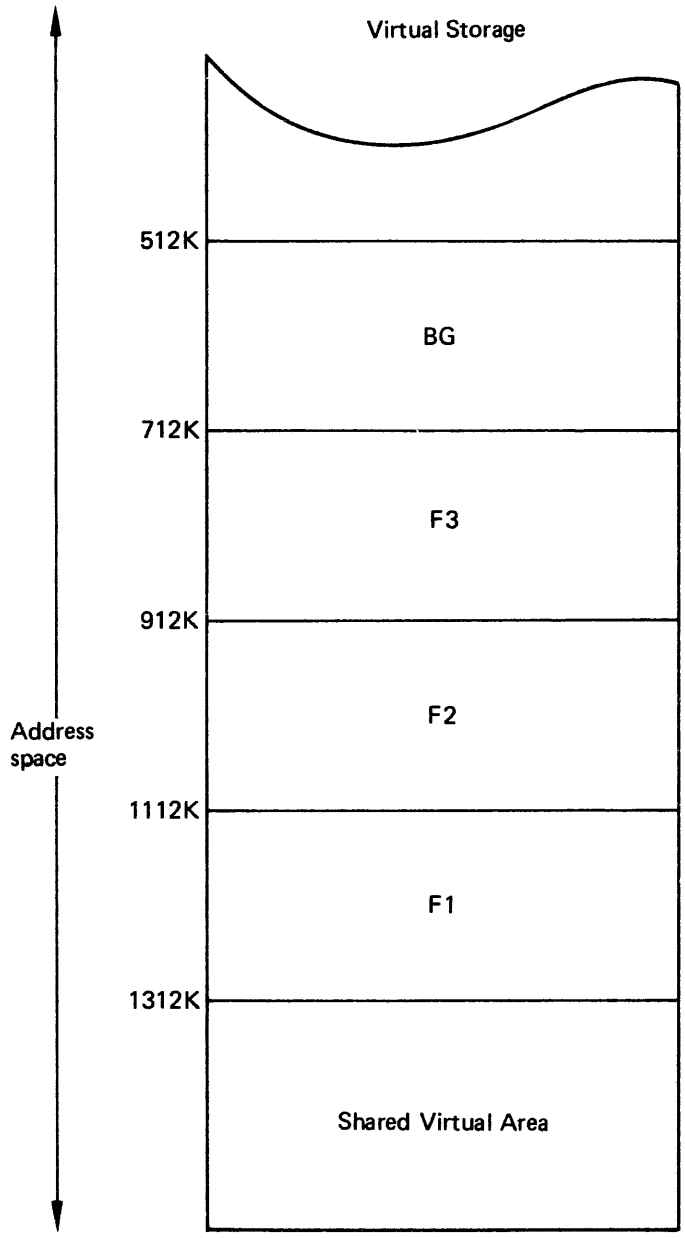


Figure 1-11. Shared Virtual Area in a Four Partition System

Processor Storage Utilization

Under DOS/VSE processor storage is used as follows:

- For the accommodation of the resident supervisor routines.
- For the loading and execution of the pageable supervisor routines.
- For the loading and execution of programs.

As shown in Figure 1-12, all page frames of processor storage not needed for the resident supervisor routines are available to the *page pool*. It is from this page pool that DOS/VSE selects page frames for pages of executing programs (including the pageable routines of the supervisor).

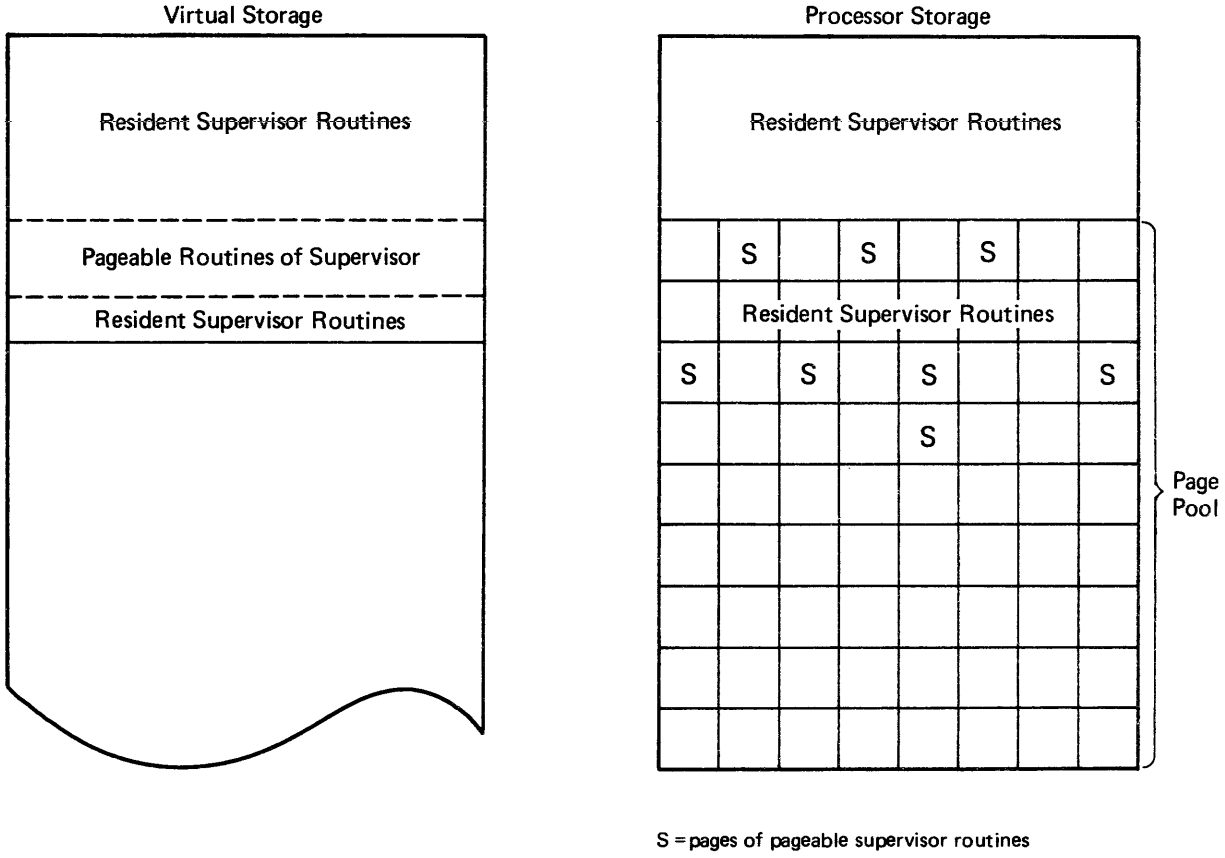


Figure 1-12. Supervisor Routines – Fixed and Pageable

Executing Programs in Virtual and Real Mode

All programs when executing are conceptually running in the address space associated with a partition. DOS/VSE selects page frames from the page pool for pages of the executing programs. The execution can be in one of two modes:

Execution in Virtual Mode: The page frames occupied by pages of programs running in virtual mode continue to be part of the page pool. DOS/VSE will manage the processor storage placing some pages on the page data set, when necessary, and retrieving those pages as required. Programs in virtual mode are *pageable*.

Execution in Real Mode: The page frames occupied by pages of programs running in real mode are taken out of the page pool for the duration of that program's execution; the page frames will not be selected by DOS/VSE for another program of higher priority; the program is fixed in processor storage and is *non-pageable*.

To have a program executed in real mode, an amount of processor storage must be allocated to the partition in which that program is to run. The allocated processor storage remains part of the page pool until real mode execution begins. Under DOS/VSE certain programs – such as those with critical time dependencies – may have to run in real mode. A partition may execute in only one mode at a given point in time; for example, the BG partition can not initiate both real and virtual execution at the same time.

Storage Allocation

From a storage management point of view, only minor differences exist in virtual and processor storage utilization techniques between ECPS:VSE and 370 mode. These differences are indicated as the following topics are being discussed:

- Address space layout
- Partition allocation
- Processor storage allocation for real mode execution
- Dynamic storage areas.

Address Space Layout. In *ECPS:VSE mode*, the virtual storage is one area whose size is determined at Initial Microprogram Load (IML).

In *370 mode*, the virtual storage is logically divided into two areas: real address space and virtual address space, see Figure 1-13. The virtual address space is defined when the supervisor is assembled. The size of the real address space is determined at the time of Initial Program Load (IPL); it is equal to the amount of processor storage installed. The supervisor resides in the low addresses of your virtual storage. In 370 mode, this is in the real address area. See Figure 1-14.

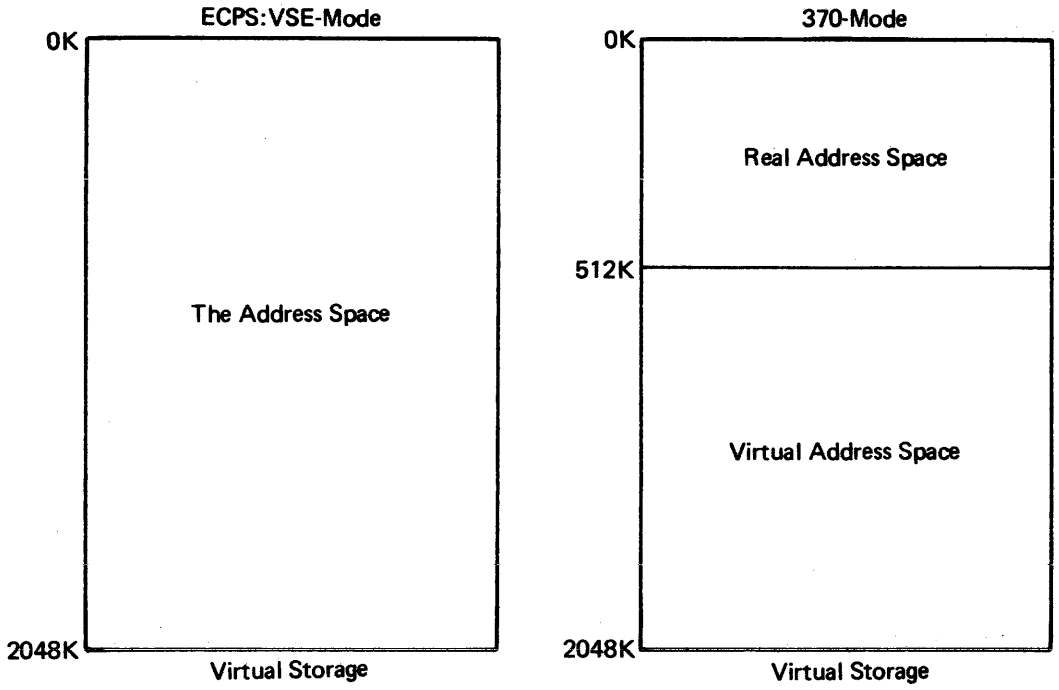
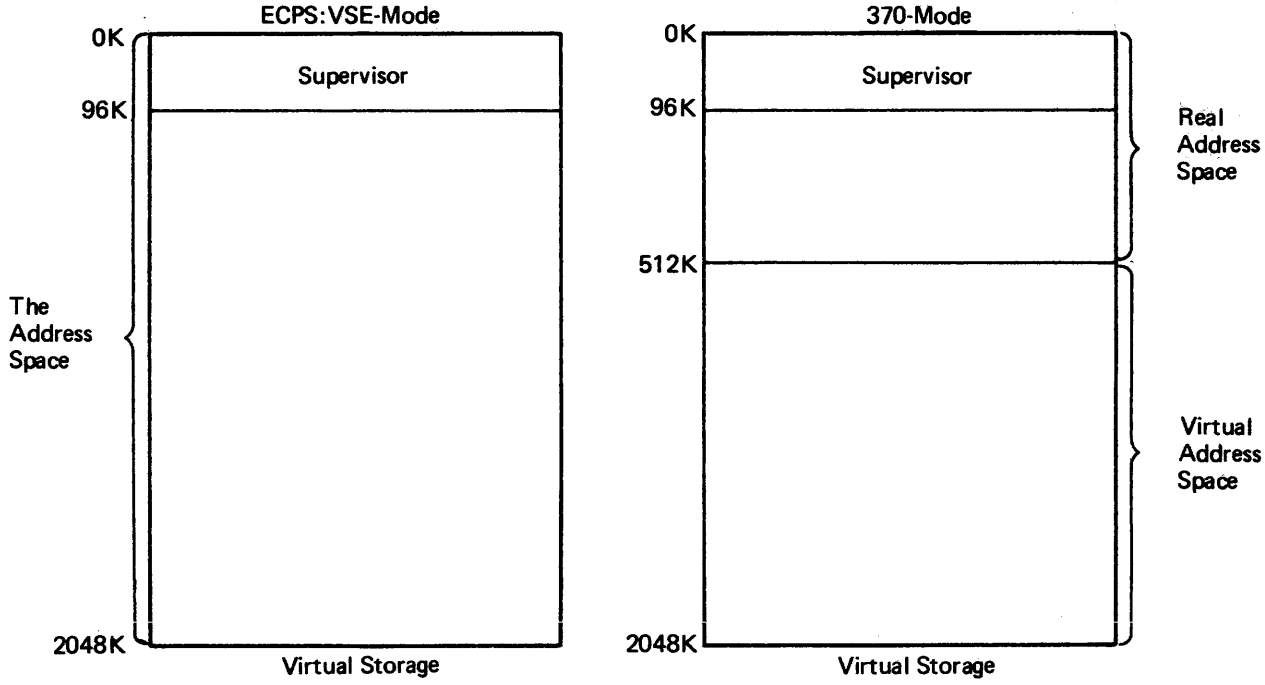


Figure 1-13. Address Space for 2048K Bytes of Virtual Storage and 512K Bytes of Processor Storage



96K as supervisor size is an arbitrary number, somewhere above the minimum supervisor size.

Figure 1-14. Supervisor Location in Both ECPS:VSE and 370 Mode

Partition Allocation. Only the number of partitions but not their sizes are defined when the supervisor is assembled. IPL allocates all of the address space available for the partitions to the Background (BG). After IPL, you allocate the foreground (FG) partition sizes. See *Chapter 3, Using the System.*

Figure 1-15 shows the layout of a 4-partition system after IPL and allocation, respectively, has taken place.

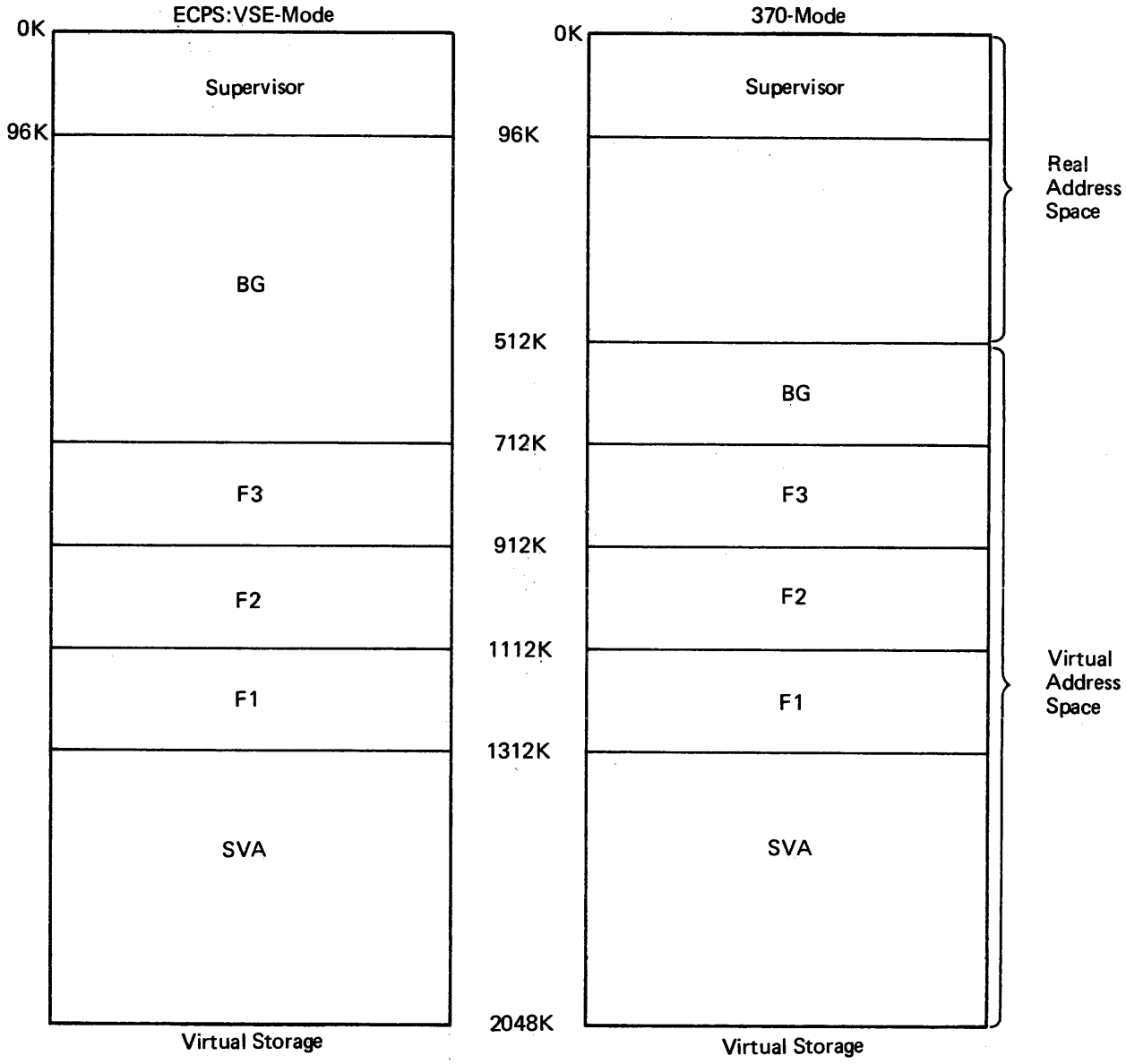


Figure 1-15 assumes a virtual storage size of 2048K and a processor storage size of 512K. The supervisor will occupy the low address 96K of this system.

In *ECPS:VSE mode*, the address space from the end of the supervisor to the beginning of the Foreground 3 partition belongs to the BG partition (616K).

In *370 mode* the BG partition's address space starts at the beginning of the virtual address space (512K). The real address space is the address space from which programs running in *real mode* are executed.

Figure 1-15. A 4-Partition System in ECPS:VSE and 370 Mode

Processor Storage Allocation for Real Mode Execution. A specific number of page frames of processor storage may be allocated to any of the partitions for real mode execution. The allocation may be done at any time with the ALLOCR command.

Submitting

ALLOCR BG=20K, F1=24K

for example, causes the following:

- **In ECPS:VSE mode:** DOS/VSE notes that 10 page frames and 12 page frames of processor storage are available to partitions background and foreground 1, respectively, for real mode execution.

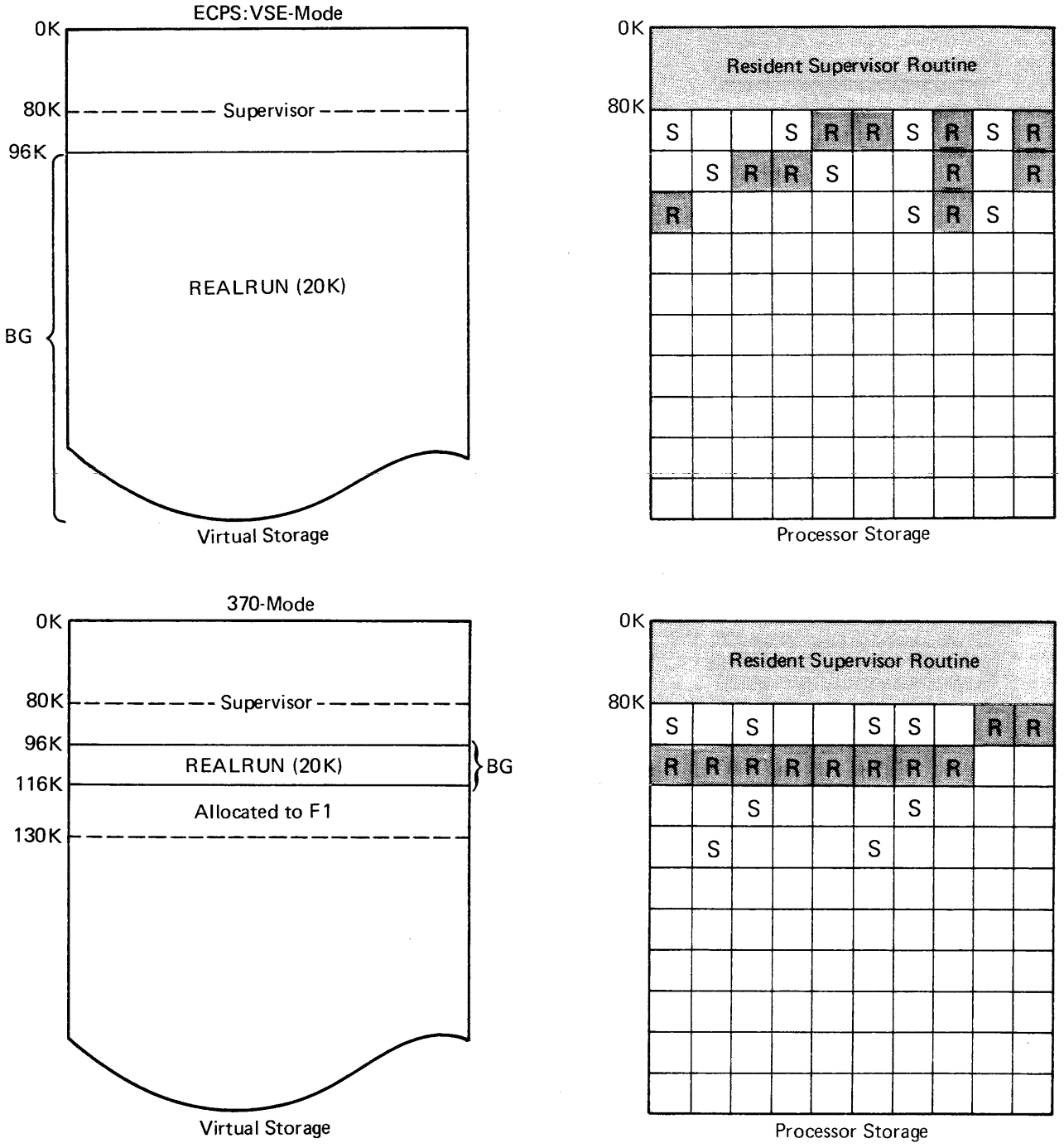
- **In 370 mode:** DOS/VSE allocates 20K and 24K of real address space to partitions background and foreground 1, respectively. In addition, when real mode execution takes place, the processor storage addresses used by DOS/VSE are the same as the addresses within the allocated real address space.

With the above ALLOCR command the largest program that can be executed real in the two partitions are 20K in BG and 24K in F1.

When not occupied by a program running in real mode, the page frames allocated to a partition are part of the page pool.

When a program running in real mode does not require all the allocated page frames, the unused page frames may be made available to the page pool by specifying the amount of storage required by the program in the SIZE operand of the EXEC job control statement for the program. In order to execute a program in real mode an EXEC statement with the REAL parameter must be used. For more details on the EXEC statement see *Chapter 3, Using the System.*

Figure 1-16 shows the results of the above discussed ALLOCR command with a 20K-program REALRUN executing in the BG partition in real mode.



The shaded portions of processor storage are not part of the page pool at this time. The illustration assumes a supervisor with 80K resident routines and 16K pageable routines. The program REALRUN is 20K in size and is executing in real mode in the BG partition. Note that in *ECPS:VSE mode* the page frames are selected randomly from the page pool, while in *370 mode* the page frames occupied by REALRUN have the same processor storage addresses as the pages that are occupied by REALRUN within virtual storage. The allocation for F1 has not affected the page pool.

Figure 1-16. Executing in Real Mode

Fixing Pages in Processor Storage. The allocated page frames are used not only for programs running in real mode, but may also be used for programs running in virtual mode.

Some programs that run in virtual mode contain instructions or data that must be in processor storage when needed and therefore cannot tolerate paging. The pages containing such code or data can be fixed via the PFIX macro instruction, and freed immediately after use via the PFREE macro instruction. The licensed program VSE/POWER is an example of an IBM program that uses PFIX/PFREE macros.

When pages of a program running in a given partition are fixed in response to the PFIX macro, they are fixed in the page frames allocated to the partition. If a PFIX macro is issued and enough storage is not allocated, the pages are not fixed, and a completion code indicating this is returned to the program.

Fixing pages in processor storage means that, in a multiprogramming environment, fewer page frames are available to other programs running in virtual mode, potentially degrading total system performance. When channel programs with large I/O areas are involved, the initial size of the page pool may be too small. Consider this effect carefully before allowing the use of the PFIX macro at your installation.

Dynamic Storage Areas. Under DOS/VSE there is a requirement for certain system functions to acquire virtual storage dynamically during program execution. An area called GETVIS area is used for this purpose. Each partition has its own *partition GETVIS* area, the SVA includes the system GETVIS area. The GETVIS areas occupy the high address space associated with each partition and the SVA. Figure 1-17 shows the virtual storage layout in ECPS:VSE and 370 mode with the GETVIS areas included. For further information on the size and use of GETVIS areas see *Chapter 3, Using the System*.

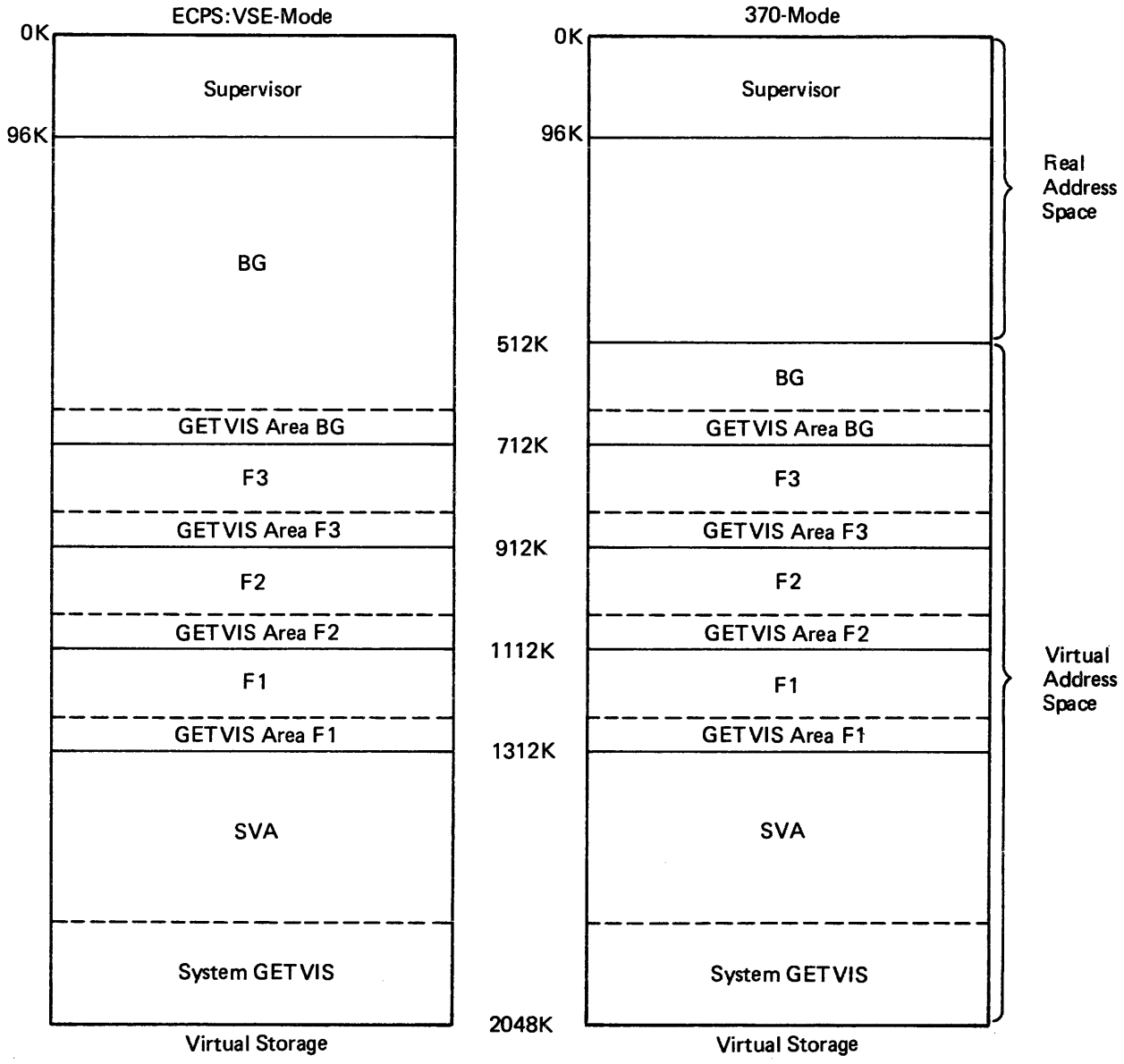


Figure 1-17. A 4-Partition System in ECPS:VSE and 370 Mode with the GETVIS Areas

Multitasking

At the beginning of this chapter, we defined multiprogramming as the ability to execute more than one program concurrently in separate partitions within a single computer system. Multitasking can be regarded as an extension of multiprogramming in that it provides the ability to execute more than one program concurrently in a single partition. In simple terms, therefore, multitasking can be regarded as multiprogramming within one partition.

Some installations using former versions of DOS, employed multitasking to run more than three programs in a 3-partition system. The additional partitions that DOS/VSE provides serve the same purpose. However, running programs concurrently in separate partitions usually requires less preparation than running programs concurrently in the same partition.

Two Types of Multitasking

Programs (or parts of a program) that are executed concurrently in a given partition are called tasks. A distinction is drawn between the main task in a partition and one or more subtasks in the same partition. The main task is that program (or program part) which is initiated by job control. The subtasks are programs (or program parts) that are initiated by the main task through the use of the ATTACH macro in an assembler language routine.

A subtask executed in a given partition may be (1) logically independent, or (2) logically dependent.

In the first case, the main task monitors the execution of the subtasks, treating them as independent programs. Such subtasks may be coded in any programming language. This type of multitasking is sometimes called multiprogramming within a partition. It is a suitable technique to use, for example, for concurrent execution of more programs than partitions are available.

In the second case, both the main task and the subtasks are program routines that are logically part of the same program. Thus, the tasks can communicate with one another. In this case the subtasks are likely to be coded in assembler language to allow the use of the task intercommunication macros. They can share code (in particular, an access method or subroutines), provided that it is of a read-only nature (that is, that the code or subroutines are not modified during execution). This technique is complex and can best be understood after studying the first type of multitasking.

Cross-Partition Event Control

Highly complex applications may have a need for communication between programs executing in separate partitions. For example, two such programs may need to perform operations on a common file, and the operations may require actual communication between the two programs.

Through cross-partition event control macros, one partition can delay the execution of part of a program until another partition signals the completion of a critical event. This allows synchronized multiprogramming in separate partitions – thus protecting programs against inadvertent destruction of each other – while at the same time providing for any necessary communication between them. IBM licensed programs require this support in certain complex applications. One example is the licensed program VSE/POWER generated with SPOOL=YES. For details about cross-partition event control, see the manual *DOS/VSE Macro Reference*.

Chapter 2: Planning the System

After a brief description of the system generation procedure in general, this chapter discusses in greater detail three major considerations during system generation, namely:

- Planning the libraries (planning the contents, the location and size of the libraries).
- Planning the system files and workfiles.
- Tailoring the supervisor (adding functions to those of the basic supervisor).

Because of the nature of this information, this chapter primarily addresses programmers who are responsible for planning the system.

System Generation Procedure

Proper and detailed planning is essential for efficient system generation and minimizes the need to modify the system after it is generated. You may want to contact your IBM marketing representative to set up a system generation planning meeting. IBM field engineering should be invited to attend the meeting to discuss the procedure to install the nonlicensed SCP (system control programs). Generating a system includes:

- *Planning the contents, organization, and size of the system and (optionally) private libraries.* This entails distributing the storage space available (on the disk packs) between the libraries desired for day-to-day use. You must consider the size of the system core image library and other system and private libraries.
- *Planning the location and size of system and workfiles.* This entails determining, what system files are required, how large they must be and where they shall be placed. Additionally workfile space needed to assemble the supervisor and to link-edit and catalog the components selected to the system core image library must be reserved.
- *Planning the options and estimating the approximate size of the supervisor.* This entails selecting from the programming services provided by IBM, those options you wish to include in the supervisor, and estimating the cost of these services in terms of bytes of storage.

Handling the Distribution System

To install the DOS/VSE SCP, you work with the IBM-supplied distribution medium (normally a magnetic tape), which is composed of four system libraries

core image
relocatable
source statement
procedure

and a system history file.

Unless you decide to operate in an SCP only environment, your system generation approach should be as follows:

1. Restore the SCP and also the supplied history file to disk. (This step does not apply if you receive the IBM supplied code on disk.)
2. Do an initial program load of the restored (nonlicensed) SCP supervisor and install VSE/Advanced Functions, the licensed DOS/VSE SCP support package.
3. Generate the supervisor by coding a set of supervisor generation macros, which define the system configuration and the services you wish the supervisor to contain. These are described in detail in the section *Tailoring the Supervisor.*
4. Delete from the libraries any components you do not require and then condense to free library space.
5. Assemble or compile and/or link-edit programs – both your own and IBM's – and catalog them into the appropriate libraries.

After you deleted any of the supplied components, you must update your history file by running the service program MSHP (Maintain System History Program). The usage of MSHP is described in *DOS/VSE Maintain System History Program (MSHP) User's Guide.*

Having determined what elements are to be contained in the system libraries, you may wish to retain additional elements in private libraries and therefore want to create private core image, relocatable, or source statement libraries. These choices are discussed in the section *Planning the Libraries.*

The system libraries, together with certain system work areas, constitute the system residence file (SYSRES), which is one extent of a direct access storage volume. The SYSRES file is described in *Appendix A: System Layout on Disk.*

After establishing your SYSRES file and the history file, you should copy those onto tape or disk for backup purposes. The utility programs Backup/Restore System and Fast Copy Disk, which are provided for this purpose, are described in *DOS/VSE System Utilities.*

For complete details on how to perform a system generation procedure refer to *DOS/VSE System Generation.*

Planning the Libraries

The components of DOS/VSE are shipped in four system libraries: the core image library, the relocatable library, the source statement library, and the procedure library. Most programs and procedures developed and used by your installation will also be stored in these libraries. In addition to the system libraries, DOS/VSE supports private libraries which you may use to either substitute for or supplement the corresponding system libraries.

Planning the size, contents, and location of these libraries according to the needs of your installation is an essential part of the system generation procedure. Such detailed planning will ensure that:

- No disk space is wasted by components not required in your installation.
- The libraries are large enough to allow for future additions.
- The libraries are accessed by the system with maximum efficiency.

Following a brief description of the purpose and contents of the individual libraries, this section discusses the major considerations involved in tailoring the libraries to the needs of your installation:

- Which libraries are required.
- How many disk drives are available and where on these devices should the individual libraries be placed.
- How large should each of the libraries be and what should they contain.

Note that this section is intended to give only general guidance for planning the libraries. More details about DASD space requirements for the libraries are contained in *DOS/VSE System Generation*. How to change the size of a library, how to insert elements into or delete elements from a library, and how to create private libraries is described in *Chapter 3, Using the System*.

Purpose and Contents of the Libraries

The following is a brief summary of the purpose and contents of the DOS/VSE system and private libraries.

Core Image Library

In order to be executed, all programs must be link-edited into phases and placed in the core image library (CIL). IBM supplies the DOS/VSE system control program (SCP) components pre-linked and cataloged in the CIL. A complete list of the supplied SCP components is shipped with the program directory documentation which accompanies your DOS/VSE SCP. Prior to receiving your DOS/VSE, consult *DOS/VSE System Generation* and the applicable VSE/Advanced Functions publication for a listing of the DOS/VSE components.

IBM also supplies cataloged distribution supervisors. Assembler source statements used to generate these supervisors are shown as part of the *Memorandum to Users* and are contained in the source statement library.

The entries in the CIL directory of phases are sorted in alphanumeric sequence. The phases themselves are cataloged in the next available space in the library.

You have to decide which of the IBM supplied SCP phases to retain in the CIL. To delete unwanted SCP components, use the delete procedures contained in the procedure library. See *DOS/VSE System Generation* for a list of these procedures.

Besides IBM SCP components you may add to the CIL your own application programs such as your payroll or accounts receivable programs, program packages obtained from IBM (for example, licensed programs), or program packages from other sources. If you wish to include such programs in the CIL, you must catalog them yourself. For information on how to do so, refer to the description of the linkage-editor in *Chapter 3, Using the System*.

Relocatable Library

The relocatable library as shipped by IBM uses a considerable amount of DASD space. The library contains:

- SCP component object modules
- Compiler logical input/output control system (LIOCS) modules

SCP Object Modules. These modules make up unlinked code of the executable SCP component phases in the CIL. The modules have been link-edited and cataloged into the CIL you receive. These modules are provided in the relocatable library for maintenance purposes only.

Compiler LIOCS Modules. The LIOCS modules needed by the various compilers are cataloged in the relocatable library. There are different modules for each device type and access method. Some modules can be used by more than one compiler. For a complete list of the LIOCS module names and device applicability, see *DOS/VSE System Generation*.

Source Statement Library

The elements in the source statement library are called books. A book is either a sequence of source statements or a macro definition.

You can catalog into the source statement library sets of source statements that are used by more than one program, and then include these statements in your source program by specifying a COPY (assembler, DOS/VS RPG II, and COBOL) or %INCLUDE (PL/I) statement.

The macro definitions in the source statement library include those macros supplied by IBM as well as any others which you may have written and cataloged yourself. When you issue a macro instruction in your program, the corresponding macro definition is retrieved from the source statement library and included in your program according to the parameters you specified.

Each book in the source statement library is classified as belonging to a specific sublibrary; for example, an assembler, a PL/I, or a COBOL sublibrary. Sublibraries are identified by a 1-letter prefix added to the book name. Letters A through I and the letters P, R and Z are reserved for sublibraries containing system components. You can use all other letters, the digits 0 through 9, and the special characters $, &, and #, to define your own sublibraries.

Procedure Library

Frequently-used sets of control statements can be cataloged into the procedure library. The elements of the procedure library, called cataloged procedures, can consist of IPL (Initial Program Load), job control statements and/or SYSIPT data. Included VSE/POWER JECL statements will be treated as DOS/VSE comment statements. If extended procedure support was included during supervisor generation (by specifying the SYSFIL option) you can also catalog procedures containing data that is to be read from SYSIPT under control of the device-independent sequential IOCS, by your program or by IBM-supplied service programs and language translators. SYSIPT in-line data can be, for example, the control statements processed by the librarian or the sort/merge program. Cataloged procedures are retrieved from the procedure library by a special form of the EXEC job control statement.

The procedures shipped in the procedure library are provided as system installation aids. They include:

- library-member-delete and module-link procedures
- MSHP history file update procedures
- standard label definition procedures

Delete and Link Procedures. The delete procedures are provided to assist you in tailoring your libraries. A complete list of the delete procedures is provided in the manual *DOS/VSE System Generation*. Once your system is installed these procedures themselves can be deleted.

The link procedures are provided to link-edit SCP modules contained in the relocatable library to the core image library. These procedures are provided for system-service purposes (the SCP's have been link-edited prior to your receiving the system).

MSHP History File Update Procedures. If you have installed a component without the use of MSHP (Maintain System History Program) there is no entry in the *history file* for that component. This can occur if, for example, you have a DOS/VS Release 34.0 or earlier with a licensed program, such as DOS/VS COBOL, running under it. The MSHP history file update procedures may be used to create a history file entry for the component, in this example DOS/VS COBOL. Now, you may use MSHP for subsequent modification (updates, maintenance etc.) of that component. For more details on the use of the program MSHP see *DOS/VSE Maintain System History Program (MSHP) User's Guide*.

Standard Label Procedures. These procedures are discussed in section *Label Information Area* in this chapter. A complete listing showing the contents of the procedures is included in the Program Directory Document shipped to all recipients of DOS/VSE.

Private Libraries

Private libraries can be defined for the core image, relocatable, and source statement libraries. The procedure library is supported as a system library

only. Private libraries are created by using the program CORGZ and have the same format as the system libraries.

You may establish private relocatable or source statement libraries either to supplement or to replace the system libraries on the SYSRES file, thereby extending the space available to the system core image library. Conversely, you may reduce the size of the system core image library by cataloging selected programs in a private core image library.

Private libraries are also useful in a testing environment where you may keep working copies of your programs intact on a system library while you test modifications of the same programs on a private library. Private libraries thus add a great deal of flexibility to your system.

You may define as many private core image, relocatable, and source statement libraries as desired, each serving a particular purpose. For instance, having a separate core image library for each partition, each on a separate disk drive, would reduce disk arm movement on the SYSRES volume, which means faster access to the libraries.

When you define the private core image library extent (associated with the logical name SYSCLB) it can reside on any disk volume that is supported by DOS/VSE. Multiple private core image libraries can reside on one volume or they can be created on separate volumes. They can be created on the same volume as SYSRES, but this is not recommended unless the access level is low. SYSCLB can be assigned only permanently (not temporarily).

In an SCP only environment, private relocatable (SYSRLB) and private source statement (SYSSLB) libraries are restricted to the same device type as the SYSRES device; the private core image library can be on a device type different from that of the SYSRES file. With VSE/Advanced Functions, you can place the private libraries on any disk device supported by DOS/VSE; the type of that disk device may be different from the device type of the SYSRES file.

Choosing the Libraries for an Installation

In an operational DOS/VSE, all SCP components must reside in the system core image library. Therefore, a system core image library must be present in every DOS/VSE installation. Which of the other libraries you need depends largely on the type and amount of work to be done and the resources available at your installation.

Relocatable and Source Statement Libraries

Although these libraries are optional, few installations can operate efficiently without them. If, for example, you work with a PL/I compiler and you need to have the PL/I resident library routines on-line at all times, these routines must be in the relocatable library. (The only – and very inefficient – alternative would be to include the physical card decks for such modules in-line with the linkage editor input.) Similarly, when you assemble programs that use IBM-supplied macros, the corresponding macro definitions *must* be present in the source statement library.

The same advantages as those gained by having IBM-supplied modules in a library can of course be obtained if you store your own object modules or source statement books in a relocatable or source statement library. The more information you have on-line in a library the less card handling is required and the more efficient your system will operate. Because the disk space available to the libraries is limited, you may prefer to reduce the number of SCP components in the relocatable and source statement libraries to a minimum to allow for sufficient space for the core image library. If additional disk drives are available, the space problem can be solved by creating private libraries.

Procedure Library

In most data processing installations there are a number of programs that are frequently executed. An inventory control program, for instance, may have to be run daily or weekly. Or a payroll program may have to be executed weekly or monthly. These programs are probably used for a long period of time without being changed.

For each of these programs, there would be one or more sets of job control statements which the programmer prepared and tested when the program was first run. These sets of job control statements can be cataloged as cataloged procedures in the procedure library; then, to retrieve a set, only one statement is required. This minimizes repetitive operator handling (which often includes the replacement of defective cards or reinsertion of diskettes) and reduces machine time and errors.

A cataloged procedure is exactly the same as what is described above as a fixed set of job control statements. But the individual procedure is no longer collected by the operator and selected manually for use; instead, it is cataloged and retrieved through a special form of the EXEC job control statement. Cataloged procedures can be modified as they are retrieved from the library.

The use of cataloged procedures is discussed in *Chapter 3, Using the System*.

Automated System Initialization (ASI) allows you to automate initial program load (IPL) of DOS/VSE and partition start up. If you plan to use ASI, you must catalog your IPL procedure(s) and your job control procedures (to start up particular partitions) into the procedure library. For more information about ASI, refer to *Starting the System* in *Chapter 3*.

Determining the Location of the Libraries

Having decided which libraries you want in your system, you must determine where on the available devices these libraries are to be placed. All system libraries must reside in the SYSRES extent of the system disk pack in a predefined sequence (see Figure 2-1). Although it is theoretically possible to have private libraries on the system pack, outside the SYSRES extent, this is not recommended because it involves increased movement of the disk arm.

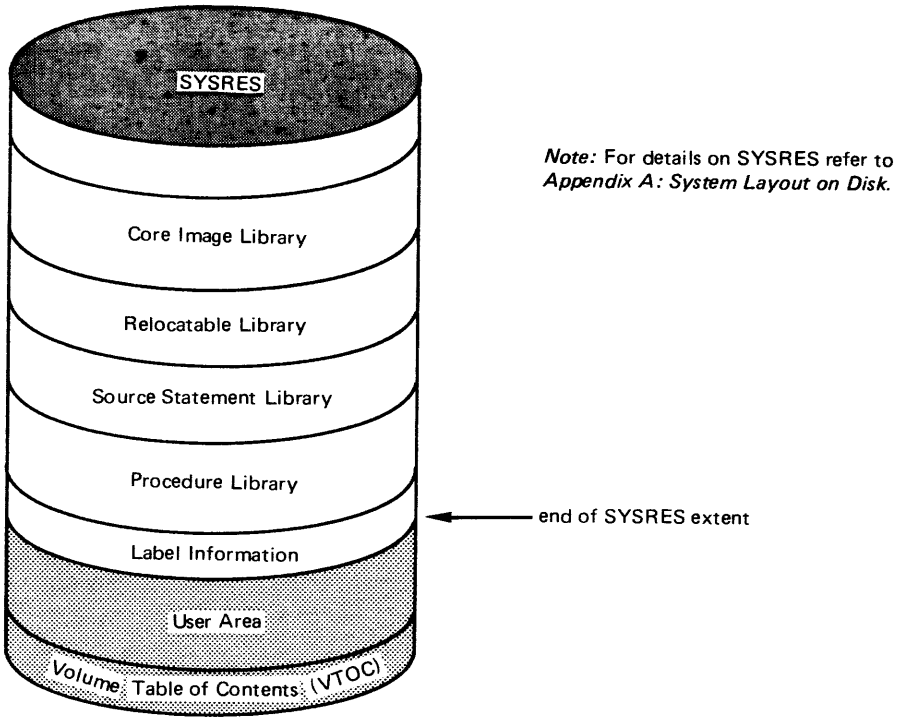
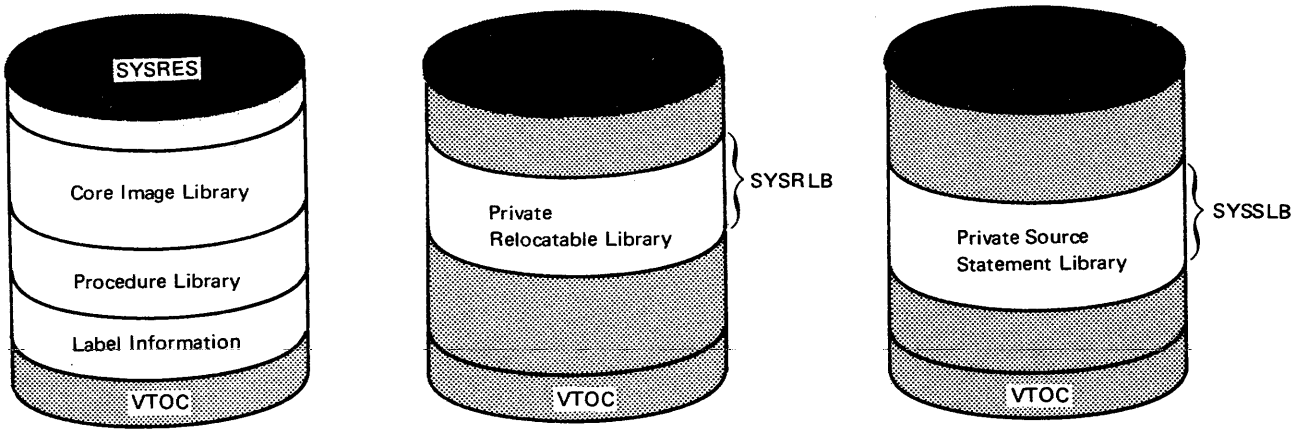


Figure 2-1. The Relative Location of the Four System Libraries

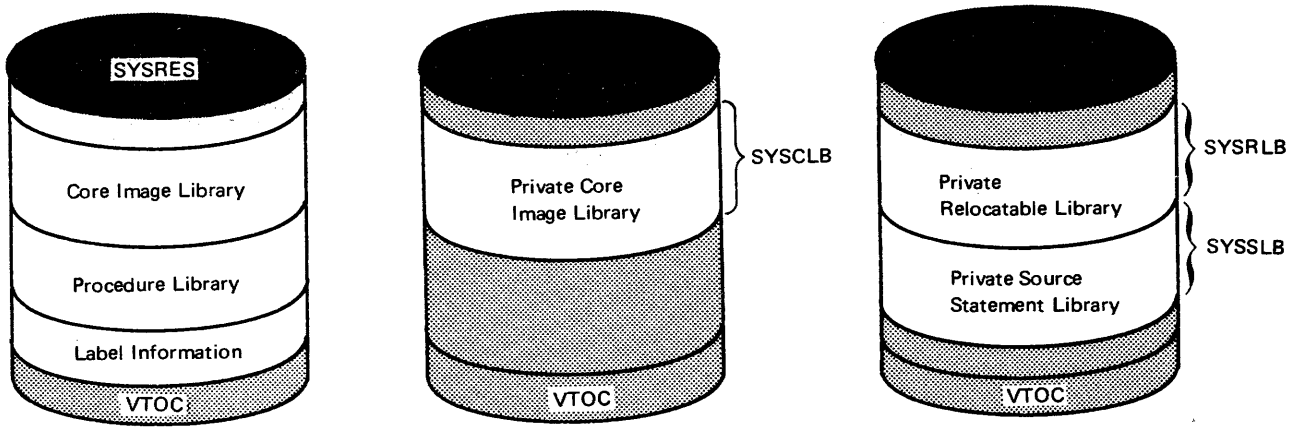
If you have additional disk drives, you can define private core image, relocatable, and/or source statement libraries on the extra volumes. Private relocatable and private source statement library volumes must be of the same type as the SYSRES pack, unless you have VSE/Advanced Functions installed. Private core image libraries can be on any disk device type supported by DOS/VSE. The system relocatable and system source statement libraries can be removed from SYSRES and established as private libraries; the system core image library, however, must always be present on SYSRES. It can be supplemented but not replaced by a private core image library. The procedure library is supported only as a system library; you cannot create a private procedure library.

Figure 2-2 shows two examples of how you can organize the libraries in a system with three disk drives. Any other combination of libraries on the available devices is possible.

The examples in Figure 2-2 are to demonstrate that you can distribute your private libraries among the available devices as you may see fit, provided you observe the existing restrictions regarding device types. A more practical example of how you can organize your libraries is given in Figure 2-3. The example assumes a system with three disk drives, but it is also applicable if you have only two drives or more than three. The organization of the libraries in this example is especially useful when you need large amounts of data on-line during execution.



If a private relocatable library and a private source statement library are to *replace* the corresponding system library, the core image library directly precedes the procedure library. These private libraries can also be used to supplement the system relocatable and source statement libraries, in which case the SYSRES file would appear exactly as shown in Figure 2-1.



A private core image library can only be used to *supplement* the system core image library, which must always be present on SYSRES. Several private libraries may reside on the same disk as illustrated.

Private core image, private relocatable, and private source statement libraries can reside on disk device types different from the SYSRES device type; for example, SYSRES 3330, private relocatable 2314.

Figure 2-2. Alternative Locations of the Libraries

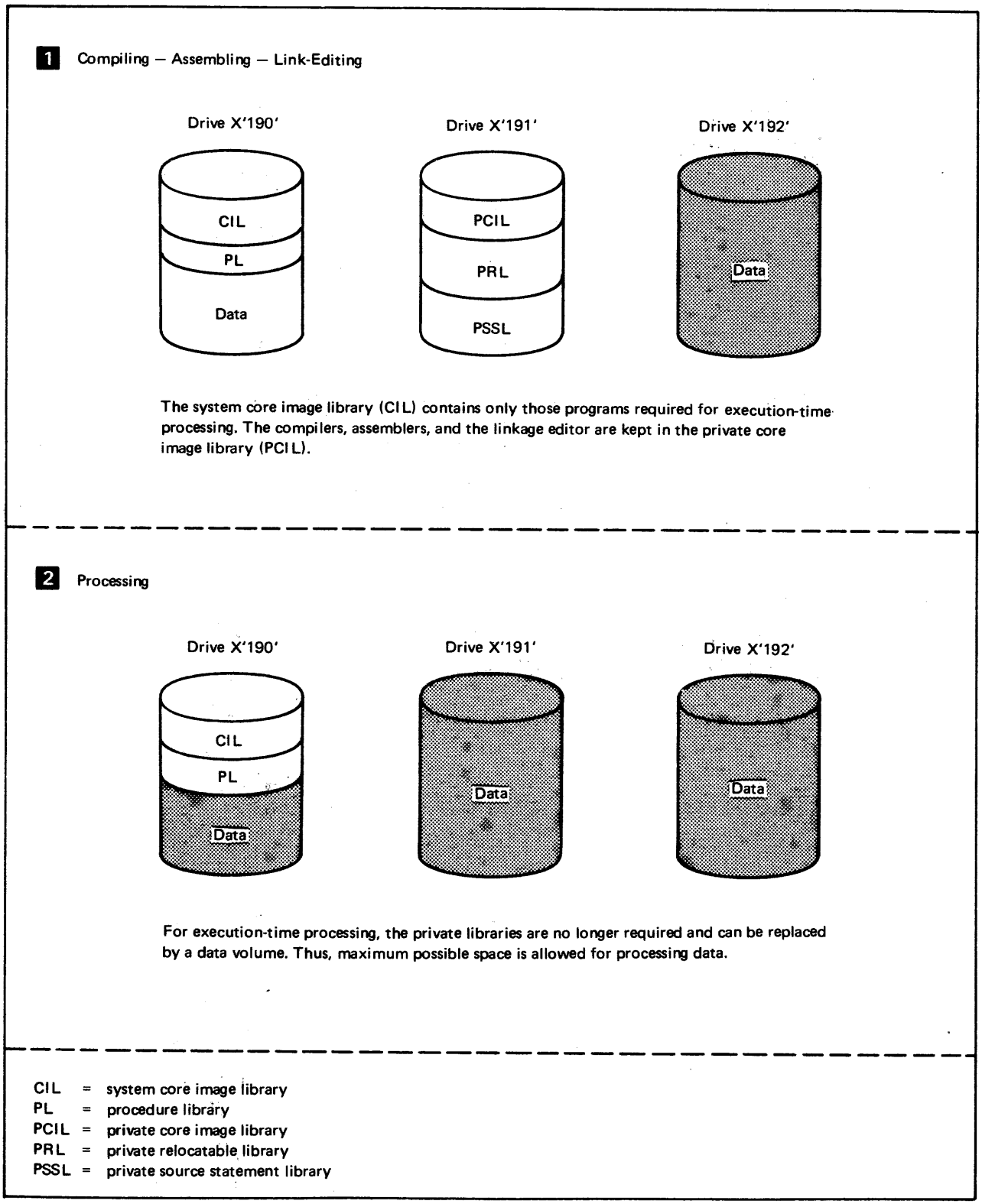


Figure 2-3. Example of Library Organization

Planning the Size and Contents of the Libraries

When planning the libraries for an operational system, you must decide on their precise contents and size for daily use. Although you can change the size of your system libraries at any time after system generation (by means of the librarian programs), you should try to anticipate future space requirements and, if possible, provide this space. Such detailed planning can eliminate the need for a complete reorganization of the libraries which would be necessary if the extension of a library results in an overflow on just one disk pack. Careful planning of the private libraries will save you additional work because you cannot easily redefine the extents of a private library once it has been created. To change the size of a private library you must create a new private library and copy the contents of the old library into it.

Consider the following factors before deciding on the contents and size of the libraries:

- The average size of a program in your installation.
- The number of programs you want on-line.
- The amount of space available.

The core image library, for example, is the library in which you normally keep most of your programs. (Otherwise, each program must be submitted to the linkage editor and placed in the core image library temporarily before it can be executed.) Therefore, ensure that your core image library is large enough to accommodate all programs that must be on-line; this includes your own programs as well as IBM-supplied components.

Special considerations apply when you work with an on-line private core image library:

- Phases whose names start with $ could be in a private core image library, but it is more efficient to keep them in the system core image library. When a $ phase is required, DOS/VSE first searches the system core image library and then, if it does not find the phase, searches the assigned private core image library.
- For all other phases (not beginning with $), first the private and then the system core image library is searched; thus, if you work with a private core image library, search time is reduced for these phases cataloged in the private core image library.

The system relocatable and source statement libraries initially contain more (IBM-supplied) members than you normally use for daily operation. By deleting from your system libraries those members which you do not need daily you are creating *operational libraries*. This reduces the disk space requirement of the SYSRES extent. In planning the contents and size of an operational relocatable library, determine which of the IBM-supplied modules can be deleted and how much space you need to store your own object modules on-line.

With one disk pack available for system files, you may prefer to maintain only enough free space in the relocatable library of the operational pack to contain the modules for the largest component in the system. This

small relocatable library permits temporary insertion of any component in relocatable format. The component can then be immediately link-edited into the core image library and deleted from the relocatable library.

Similar considerations apply to an operational source statement library. Determine which of the IBM-supplied components you need on-line, which should be transferred to a backup volume for future extensions of your system, and which can be deleted entirely.

If you intend to use the procedure library, you should allocate sufficient space for it on the SYSRES file during system generation. In estimating the amount of space required, consider the number of IPL commands, job control statements and SYSIPT data records (source modules, utility control statements, etc.) you expect to store in the procedure library.

After you have determined the space requirements for your libraries in terms of number and size of programs, you must define and allocate the amount of disk space needed to accommodate these programs. A set of formulas is available to calculate the disk space required for each library. These formulas are contained in *DOS/VSE System Generation*.

The contents of the libraries are identified in the *Memorandum to Users* (shipped with the distributed DOS/VSE system). The storage requirements (sizes) for these components and macro definitions are identified in the section for each component.

System and Workfiles

The SYSRES file is only one of the system files that must be planned. The location of the other system and workfiles and their sizes deserves some thought. The system files besides SYSRES are:

Page data set

Recorder file (SYSREC)

Hard copy file (SYSREC)

History file (SYSREC)

Alternate dump files (SYSDMP)

Page Data Set

If you have the licensed program VSE/POWER installed, the page data set should not be placed on the same drive as the VSE/POWER data files if this can be avoided. You should attempt to place the page data set on a pack that has relatively low activity yet is on-line all the time. Normal data files are not conducive to this approach as you probably do not want to leave these files on-line when they are not needed. In many cases the best place for the page data set is on the same pack that contains the SYSRES file. A user with only two disk drives should place the page data set on the pack that contains SYSRES. For information on the size of the page data set see *Defining the Page Data Set* in this chapter.

Recorder File

The recorder file contains recovery management support statistics provided primarily for IBM service personnel to analyze the performance of your system. The information collected is related, for example, to:

- I/O errors
- CPU errors
- IPL reason codes

The system logical name used for the recorder file is SYSREC. The file name is IJSYSRC. The SYSREC file must be defined as a disk extent on a DASD type that is supported by DOS/VSE as SYSRES.

The recorder file is created immediately after the first IPL for your DOS/VSE with the SET RF=CREATE command. The file is opened by the first occurrence of a // JOB statement after IPL. No // JOB statement may be submitted prior to the SET RF=CREATE command. See *Starting the System* in *Chapter 3, Using the System.*

Hard Copy File

The hard copy file, a disk extent, must be on the same device as the recorder file SYSREC. The system logical name is SYSREC and the file name is IJSYSCN.

The hard copy file contains all of the messages displayed on the display operator console (DOC). These messages can be retrieved on SYSLOG by using the operator redisplay (D) command, or on SYSLST by using program PRINTLOG. The hard copy file is created immediately after the first IPL with the SET HC=CREATE command. The file is opened by the occurrence of the first // JOB statement after IPL. See *Starting the System* in *Chapter 3, Using the System.*

History File

Each system needs a history file containing information about the components of the system and the fixes applied to those components. The history file is used by MSHP (Maintain System History Program) for the recording of information about your installed components. When DOS/VSE is shipped to you, a history file is also shipped. This file reflects the change level of the supplied DOS/VSE SCP components. An up-to-date history file eases maintenance of your system.

The history file is a disk extent and must be on the same device as the recorder and hard copy files. The system logical name is SYSREC and the file name is IJSYSHF.

For information on installing the supplied history file consult *DOS/VSE System Generation*. How MSHP uses the history file is described in *DOS/VSE Maintain System History Program (MSHP) User's Guide*. You should also consult *DOS/VSE System Utilities* for information on BACKUP/RESTORE and those programs' relationship with the history file.

Alternate Dump Files

Instead of SYSLST, one or two dump files on a direct access volume may be used to receive partition dumps. A partition dump may be produced, for example, when a program cancels.

The first (or only) dump file has the file name DOSDMPF. If you choose to have a second dump file (its file name is DOSDMPG), the two dump files are used alternatingly: while one is being filled, the other one could be processed by the DOSVSDMP program. Note that the two dump files must reside on the same DASD volume. Each dump file is a single extent file.

At the time of IPL, you must assign the dump file using the DEF command with the specification SYSDMP=cuu. The assignment cannot be changed until the next IPL. If you fail to assign the dump file, the dump will be printed on SYSLST.

You create the dump file(s) through the DOSVSDMP program. This program is also used for printing the dump from the dump file. For details on the usage of the DOSVSDMP program, refer to the publication *DOS/VSE Serviceability Aids and Debugging Procedures*.

DLBL and EXTENT job control information must be provided each time the dump file is to be accessed, that is, when

- the file is created
- a dump is written into the dump file
- a dump is printed with the dump file as input.

For each of these three cases, the EXTENT statement must specify the logical unit name SYS006.

Workfiles

Workfiles are temporary files that are used by a program during the execution of a given application. User-written programs as well as IBM-supplied programs can use workfiles. Workfiles used by your own programs must be defined, created, and named individually by you. They are not discussed here.

System workfiles are used in compiling (assembling) source statements and preparing input for the linkage editor. System workfile naming uses the following conventions:

Symbolic Name	File Name
SYSLNK	IJSYSLN
SYS001	IJSYS01
SYS002	IJSYS02
SYS003	IJSYS03
SYS004	IJSYS04
SYS005	IJSYS05
SYS006	IJSYS06

For example, the assembler requires three workfiles to translate source input and one workfile (SYSLNK) to prepare linkage editor input.

The workfiles are defined via // DLBL and // EXTENT statements. They are opened and created when needed.

Listed below are the symbolic device requirements for the Assembler, DOS/VS COBOL, and DOS/VS RPG II, the language translators, most frequently used under DOS/VSE.

	SYSLNK	SYS001	SYS002	SYS003	SYS004	SYS005	SYS006
Assembler	L	M	M	M			
DOS/VS COBOL	L	M	M	M	M	O	O
DOS/VS RPG II	L	M	M				

M = Mandatory
O = Optional
L = Required when link-editing

The size requirements of these files vary. Refer to *DOS/VSE System Generation* which gives the formulas for calculating the size requirements of the assembler and linkage editor workfiles. DOS/VS COBOL and DOS/VS RPG II workfile sizes are described in their respective installation guides.

To compile and link in two or more partitions simultaneously you will need a set of workfiles for each partition in which you plan to compile and link programs. A method for handling this situation is given in section *Label Information Area* which follows.

Label Information Area

The label information area is part of the SYSRES file and follows the last library in SYSRES. If SYSRES is an FBA device, the label information area comprises 200 blocks. For CKD devices the area is two cylinders. (For the 3340 disk, it is 3 cylinders and for the 3350 it is 1 cylinder).

For FBA devices, but not for CKD devices, you may change the size of the label information area using the RESTORE program. See *DOS/VSE System Utilities* for details on this program.

Using the DLA command during IPL, you may define or reference an additional label information area. This area is separate from the SYSRES file, but is located on the volume containing the SYSRES file. The need to define such an area may arise when two CPUs or two DOS/VSE systems under VM/370 share one SYSRES file. More information on the DLA command is provided in chapter *Using the System* under section *IPL commands*.

The size of a label information area that you define via the DLA command can deviate from the default size, regardless whether it is located on an FBA device or a CKD device.

Usage of the label information area is described in *Chapter 3, Using the System*.

Entries in the label information area point DOS/VSE to the appropriate files on a given disk pack. IBM provides standard label procedures in the procedure library for placing standard label information into the label information area for the following files:

File Name	File-ID	Symbolic Name
IJSYSRS	DOS.SYSRES.FILE	SYSRES
IJSYSRC	DOS/VS.RECORDER.FILE	SYSREC
IJSYSHC	DOS/VS.HARDCOPY.FILE	SYSREC
IJSYSHF	DOS/VS.HISTORY.FILE	SYSREC
IJSYSLN	DOS/VS.SYSLNK.FILE	SYSLNK
IJSYS01	DOS/VS.WORK-FILE.1	SYS001
IJSYS02	DOS/VS.WORK-FILE.2	SYS002
IJSYS03	DOS/VS.WORK-FILE.3	SYS003
IJSYS04	DOS/VS.WORK-FILE.4	SYS004
IJSYSIN*	DTTEPTF	SYSIN

* SYSIN labels for diskette cardless system.

The label information assumes you have taken the default library allocations when you restored your system from tape to disk. If you use different library allocations or if your page data set size is larger than the default, prepare your own label information and execute your own // OPTION STDLABEL run. If you wish to add standard label information, run the supplied standard label procedure(s) (or your own) and supply also the new entries.

The *Memorandum to Users* shipped with DOS/VSE lists the standard label procedure names and the contents of those procedures.

Planning for Compiling in More Than One Partition

Once the standard label area contains label information for the workfiles you can now assign the symbolic names (SYSnnn) to some physical drive and start compiling. Initially there is only one set of // DLBL and // EXTENT statements for each workfile (IJSYS01, IJSYS02, etc.), so you cannot run compiles simultaneously in two different partitions.

The DOS/VSE open routines always look for the label information in the label storage area in the following sequence:

1. partition userlabel area
2. partition standard label area
3. system standard label area

To cause each partition to have its own set of workfiles, place the necessary label information in the partition standard label area associated with that partition.

The job control program will write label information to the partition standard label area of the partition in which job control is running when it encounters the // OPTION PARSTD statement.

```
(a)   // OPTION PARSTD
      // DLBL IJSYS01,'BG-WORKFILE-1',0,SD
      // EXTENT SYS001,,1,0,12,12
      // DLBL IJSYS02,'BG-WORKFILE-2',0,SD
      // EXTENT SYS002,,1,0,24,12
      // DLBL IJSYSLN,'BG-SYSLNK',0,SD
      // EXTENT SYSLNK,,1,0,36,12

(b)   // OPTION PARSTD
      // DLBL IJSYS01,'F2-WORKFILE-1',0,SD
      // EXTENT SYS001,,1,0,48,12
      // DLBL IJSYS02,'F2-WORKFILE-2',0,SD
      // EXTENT SYS002,,1,0,60,12
      // DLBL IJSYSLN,'F2-SYSLNK',0,SD
      // EXTENT SYSLNK,,1,0,72,12
      // DLBL IJSYSCL,'PCIL-FOR-F2',0
      // EXTENT SYSCLB,,1,0,84,24
```

Job streams (a) and (b) above, when run in the BG and F2 partitions with appropriate ASSGN statements, will enable simultaneous use of the DOS/VS RPG II compiler in both partitions. When running the compiler in either partition, the OPEN routines will search for file names IJSYS01, IJSYS02, IJSYSLN. In the BG partition the compiler will use cylinder 1 through cylinder 3 of a 3340, and in the F2 partition cylinders 4 through 6.

Note: Label information for a private core image library (PCIL) has been provided in job stream (b). To link edit in a foreground partition a PCIL must be permanently assigned. See *Creating and Working with Private Libraries* in *Chapter 3, Using the System* for information on creating private libraries.

Tailoring the Supervisor

The IBM-shipped DOS/VSE includes three supervisors, one of which is used during system generation. Part of your system generation procedure is to plan and assemble your tailored supervisor. You may generate a system to run either in ECPS:VSE or 370 mode for the 4300 processor, or in 370 mode for the System /370 CPUs.

This section describes the optional and required parameters of the DOS/VSE generation macros in a topical sequence; that is, such that related options are presented together regardless of the macros in which they are contained. For the exact formats of these macros, refer to *DOS/VSE System Generation*. This section discusses, in addition, the advantages or necessity of specifying the support for the various features in the supervisor.

In tailoring your supervisor to the requirements of your installation, you can take into consideration future plans to add functions that require supervisor options by including their requirements in your supervisor generation macros. This allows you to upgrade your installation without having to regenerate your supervisor. In your library planning, you should include space for modules or components that will be required by a planned future configuration or functional upgrades. The storage cost of additional supervisor options may be estimated by consulting section *Storage Requirements* in *DOS/VSE System Generation.*

Storage Management Options

This section describes those supervisor options that relate to

- The size of virtual storage (applies to 370 mode only)
- The number of partitions and their priorities
- The layout of the page data set
- Facilities for improving the paging mechanism

Virtual Storage Size

The method of defining virtual storage is different for ECPS:VSE mode and 370 mode.

ECPS:VSE Mode Virtual Storage Definition.
In ECPS:VSE mode, the default value for the total size of your virtual storage is 16M (16,777,216) bytes. The operator may change this value at IML (Initial Microprogram Load). For details about IML on a 4300 processor, see the Operator's Library Procedures manual provided by IBM for the pertinent processor model. The value is used by DOS/VSE to determine the size of the page data set. How to define the page data set is discussed later.

370 Mode Virtual Storage Definition.
In 370 mode, virtual storage is composed of virtual address space and real address space. You specify the size of the virtual address space in the VSIZE operand of the VSTAB generation macro. The size of the real address space is determined automatically when you execute the Initial Program Load (IPL) program. The value you specify for VSIZE is equal to the sum of the virtual address space allocated to the defined partitions and the size of the shared virtual area.

The value specified for VSIZE cannot be changed without a new supervisor generation. The maximum size of virtual storage is 16M (16,777,216) bytes. The maximum value you can specify for VSIZE is 16M minus the size of the real address space. The value you specify for VSIZE must be equal to or greater than 704K bytes (the minimum for a two partition system).

The value you specify for VSIZE is used by DOS/VSE to determine the size of the page data set. Refer to *Defining the Page Data Set* later in this section.

The Shared Virtual Area.

The shared virtual area (SVA) is divided into subareas as follows; a system directory list (SDL), an area for phases, a system GETVIS area (see Figure 2-4).

You cannot define the SVA size at the time of supervisor generation; DOS/VSE determines the size during IPL at which time you may allocate additional space. Because the SVA space shortens the amount of virtual

storage that is left to the partitions, you should take the SVA and its size into your planning considerations.

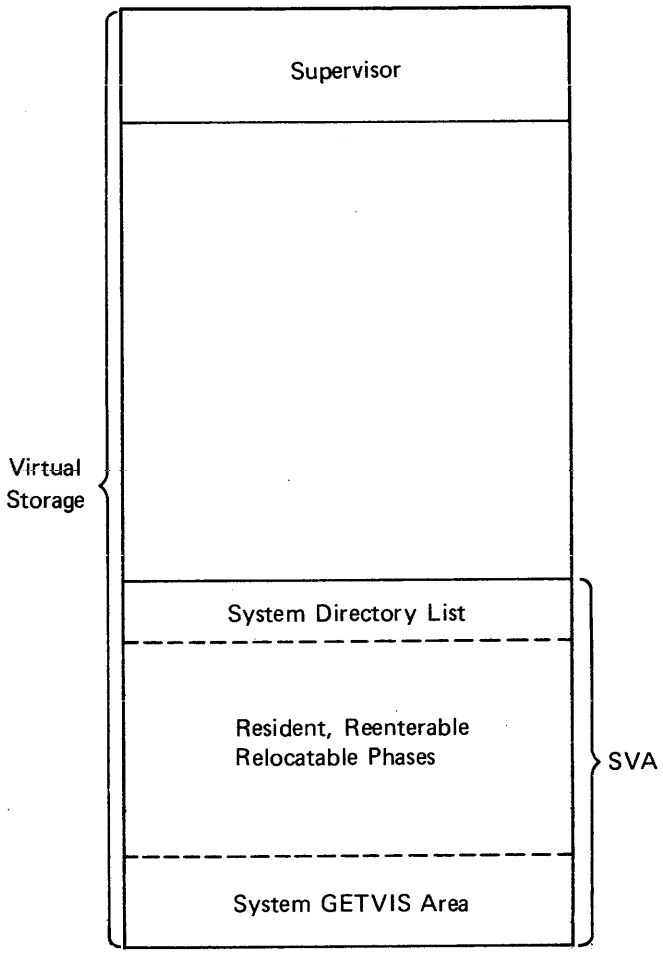
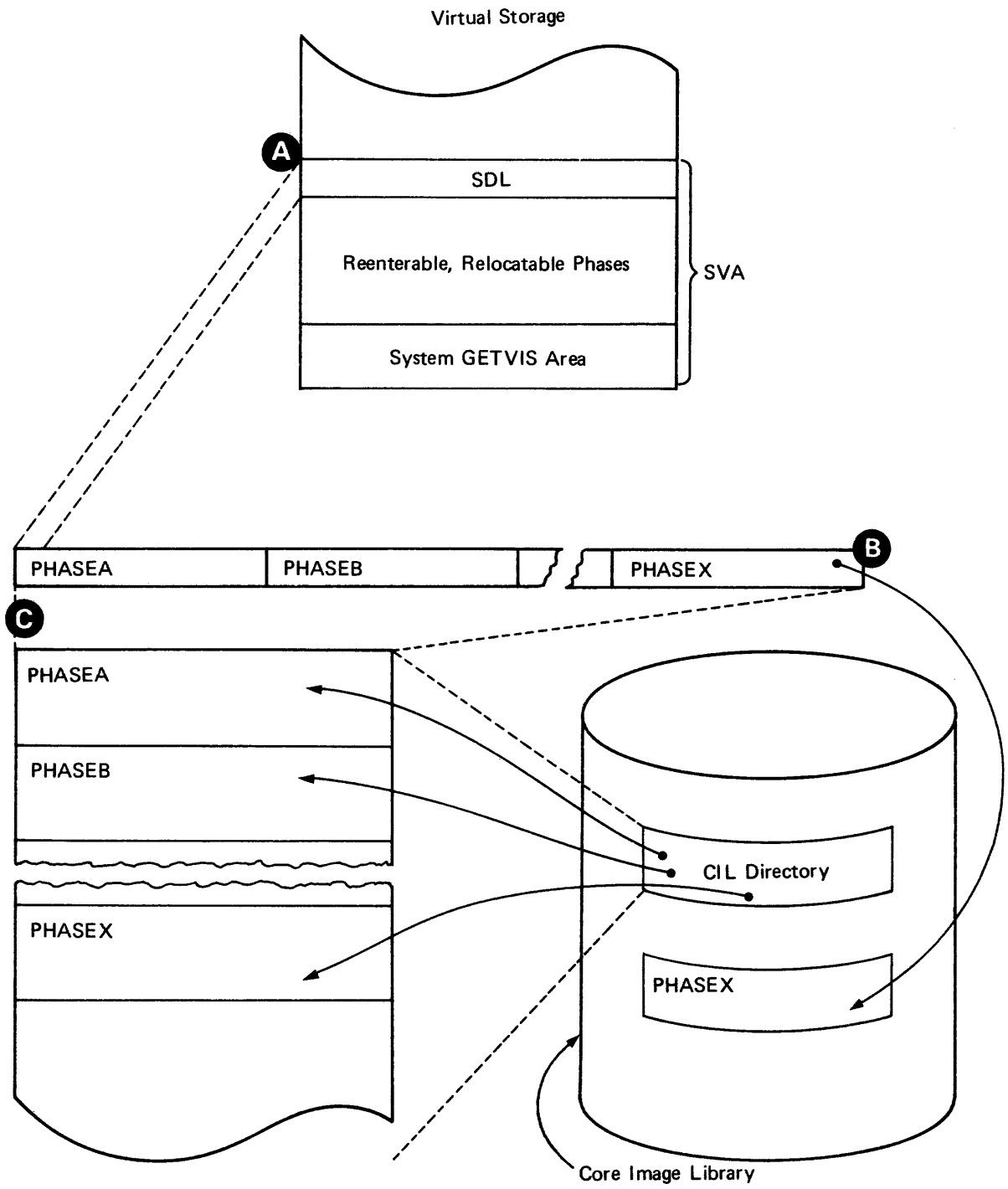


Figure 2-4. Layout of the Shared Virtual Area

The System Directory List. The system directory list (SDL) contains copies of selected entries of the system core image library directory. This provides fast retrieval of frequently used phases. (These phases may be resident in the SVA or in the system core image library.) Having SDL entries avoids searching the system core image directory (on disk) for each phase load request. Figure 2-5 shows the SDL and its relationship to the system core image library.



A The system directory list (SDL), built by DOS/VSE, provides for fast locating of frequently used phases either in the SVA or in the system core image library.

B The SDL entries point directly to a phase's location on disk.

C The SDL entries are copies of selected Core Image Library Directory entries.

Figure 2-5. System Directory List

The SVA Phase Area. The SVA phase area always contains DOS/VSE system phases; the area may, in addition, contain IBM licensed program phases and user-written phases.

Phases that are in the SVA may be used concurrently by more than one partition if the phases are reenterable and relocatable. Having phases in the SVA speeds processing by:

- *eliminating loading from a core image library* – When a phase is resident in the SVA, it does not have to be loaded from the library for each execution. This saves the disk I/O of a directory search. Additionally, even if the phase was *paged out* to the page data set, time is saved as paging is generally faster than loading from a core image library.
- *reducing processor storage demands* – If the phase is being shared between two or more partitions, the impact on the page pool is less than if two or more copies of the phase were loaded into storage.

The System GETVIS Area. The system GETVIS area is used by DOS/VSE to dynamically acquire virtual storage for its own use.

An example of the GETVIS area use is the initialization of the SDAID program. The SDAID program normally requires approximately 100K of system GETVIS space when it is being initialized. For more details on the SDAID program see *DOS/VSE Serviceability Aids and Debugging Procedures*.

Size of the SVA. At IPL, based upon the chosen supervisor options, DOS/VSE calculates the SVA size. The supervisor options and their cost in SVA space are shown in the manual *DOS/VSE System Generation*. Additional space requirements for installed licensed programs such as VSE/VSAM or DOS/VS SORT/MERGE are also automatically calculated by DOS/VSE at IPL. The space requirements for each licensed program are shown in the appropriate licensed program documentation. To support user-written programs in the SVA you must indicate the required SVA space. The parameters SDL, PSIZE and GETVIS of the IPL command SVA are used to increase the SVA size beyond the DOS/VSE defaults.

The loading of certain phases into the SVA, and the creation of SDL entries for them occur automatically at IPL. For information on how to increase the size of the SVA as well as loading items not automatically included by DOS/VSE, see the section *Starting the System* in *Chapter 3, Using the System*.

Defining the Number of Partitions

In the NPARTS parameter of the SUPVR generation macro, you define the maximum number of partitions for your system.

In selecting the appropriate number of partitions for your particular installation, you should consider the type of processing you require. Assume you want to run concurrently the following types of programs:

- Test cases (assemble/compile, link-edit, and execute)
- Daily application programs
- A spooling program, such as VSE/POWER
- Teleprocessing application program.

For this case, you should generate a system with four to five partitions, depending on the volume of application program processing. If, for example, your system includes the licensed program ACF/VTAM, at least two partitions must be specified: one for ACF/VTAM and one for your VTAM application programs.

Because you cannot alter the NPARTS specification unless you regenerate the supervisor, it may be advantageous to specify more partitions than you see an immediate need for.

Defining Partition Priorities

A processing priority is associated with each partition in a multiprogramming system. If you do not specify processing priorities during system generation, DOS/VSE establishes them by default following the concept indicated by the examples given below.

If you specify	The default processing is (from low to high)
NPARTS=2	BG – F1
NPARTS=3	BG – F2 – F1
NPARTS=4	BG – F3 – F2 – F1

In most cases, there will be no need to select another priority sequence; however, the PRTY parameter in the FOPT generation macro is provided for this purpose. In the PRTY parameter you can specify the partition identifiers in any desired sequence, and thus select another priority sequence.

Besides assigning a fixed priority to a certain partition, you can also specify two or more partitions for balancing.

Partition balancing is requested through use of the equal sign (=) between operands of the PRTY command. For example:

```
PRTY F3,F1=F2=F4,F5
```

specifies F1, F2, and F4 as balanced partitions for which the supervisor dynamically distributes CPU time. F5 has a higher priority, and F3 a lower priority than the three balanced partitions.

The operator can display and modify the priorities established during supervisor generation at any time during processing with the PRTY command. He may want to do this in order to accelerate the execution of a given job.

Defining the Page Data Set

The page data set, a sequentially organized set of records on a direct access device, is required to accommodate paged-out pages of programs that are being executed in virtual mode. The size of the page data set depends on the amount of pageable address space.

You define the page data set through the IPL command DPD. This command is discussed in section *IPL commands* in *Chapter 3, Using the System*. Among other items, you can specify the channel and unit number of the device, whether you want to treat the page data set as a data secured file, the size of a particular extent, and the lower limit address of the extent.

The page data set can reside on any disk device that is supported by DOS/VSE as a system residence device.

Your page data set may be spread over up to 15 extents (depending on the value specified for DPDEXT in the IOTAB generation macro). These extents may be allocated on different volumes, a maximum of three per volume; you must, however, stay within one disk architecture: FBA or CKD.

For all but the last extent, the size must be specified in the corresponding DPD command. If a command does not include the size specification, DOS/VSE considers the command as the last one of a series. As a result, DOS/VSE calculates the upper limit address according to the amount of pageable storage defined for your system. The usage of disk space is shown below:

Disk Device Type	Pages per Cylinder
2314	60
3330	114
3340	36
3350	240
FBA	see note

Note: Four FBA blocks contain one page of virtual storage; hence a 2M byte system (2048K) requires 4096 FBA blocks (2048K ÷ 2K x 4 blocks).

In *ECPS:VSE mode*, the virtual storage size to be mapped on the page data set, is a function of the hardware. The default system size is 16M bytes (16,384K). The default may be altered during Initial Microprogram Load (IML) to: 2048K, 4096K or 8192K. How to perform IML is described in the IBM provided *Operator's Library Procedures* manual for your central processor. If disk space is a concern, you might consider reducing the virtual storage size. For example, a 16M (16,384K) system requires 32,768 FBA blocks whereas a 4M (4096K) system requires 8192 FBA blocks.

In *370 mode*, DOS/VSE uses the value specified in the VSIZE parameter of the VSTAB macro to calculate the disk space requirements. If your supervisor includes pageable routines, DOS/VSE reserves space on the page data set for these routines.

Improving the Paging Mechanism

The page handling is controlled by the page management routines of the supervisor. You can, however, influence the paging mechanism in order to reduce the number of page faults, to minimize the page I/O activity, and to control the page traffic within a specific partition. You can do this by means of three macros: RELPAG, FCEPGOUT, and PAGEIN.

RELPAG (Release Page). This macro informs the page management routines that the contents of one or more pages is no longer required and need not be saved on the page data set when the page frames occupied by these pages are claimed for use by other pages. This saves unnecessary page I/O.

FCEPGOUT (Force Page-out). The macro informs the page management routines that one or more pages will not be needed until a later stage of processing, and that they should be given highest page-out priority. This prevents page-out of other pages which, should they be paged-out, might be needed again immediately after being written onto the page data set.

PAGEIN. This macro requests one or more pages to be paged-in in advance, so that page faults can be avoided when the specified pages are needed in processor storage. If the specified pages are already in processor storage when the macro is issued, they are given lowest priority for page-out.

If you anticipate the use of one or more of the above macros in any of your programs, specify PAGEIN=n in the SUPVR generation macro. This generates the support for the three macros. The value of n must be 1 or greater. It specifies the number of page-in requests that can be queued if more than one PAGEIN macro is issued concurrently in the system.

Library Options

You may generate support for special applications in the procedure library and for reserved supervisor space to achieve better fetching performance. These options are described below.

Extended Support for the Procedure Library

Normally, cataloged procedures can consist of job control statements or linkage editor control statements or both. However, with the extended support, cataloged procedures can include data that is to be read from SYSIPT. Such data may be, for instance, utility control statements to be processed by a utility program.

To include the extended support for the procedure library, specify the SYSFIL parameter in the FOPT generation macro, which is discussed in the section *Disk Options* in this chapter.

More information on the procedure library is contained in the section *Planning the Libraries.*

Second Level Directory for Core Image Libraries

The directory entries for phases in the core image library are sorted by phase name in alphameric sequence.

An index of the directory entries is kept in the supervisor in a second level directory (SLD). The SLD speeds the retrieval of phases from the system core image library. You may specify the number of entries the SLD will contain through the SLD parameter of the FOPT generation macro. The value specified depends on the type of disk device that contains the system core image library:

For CKD devices – the number of directory tracks.
For FBA devices – the number of directory blocks.

There is a PSLD parameter in the FOPT macro which specifies the number of entries for a private second level directory (PSLD) for the private core image libraries. You have one PSLD for each partition specified in NPARTS in the SUPVR generation macro. You should specify a PSLD value that accommodates for your largest private core image library; the size of each PSLD will be based on one specification in the FOPT macro.

Telecommunication

DOS/VSE provides facilities for telecommunication, the interchange of data between an application in the system and terminals connected via telecommunication lines. These facilities provide the ability to define such lines for supervisor assembly and to specify one or more access methods for input/output services between an application and terminals.

Teleprocessing devices (terminals) are normally attached to the CPU through transmission control units or communications controllers. The control unit must be defined via the IPL command ADD. In some cases there is a direct local attachment.

The access methods, defined in the TP parameter of the SUPVR generation macro, are the licensed programs:

- Advanced Communication Function/VTAM (ACF/VTAM)
- Basic Telecommunication Access Method – Extended Support (BTAM-ES)

Supervisor support for BTAM-ES is standard, also the support for TP balancing (teleprocessing balancing).

For detailed information on generating and using a teleprocessing access method, refer to the appropriate teleprocessing publications. Teleprocessing users should also pay particular attention to section *I/O Options* later in this chapter and read section *Balancing Telecommunication* in *Chapter 4, Using the Facilities and Options of DOS/VSE.*

BTAM-ES Support

Applications using BTAM-ES can execute in either virtual or real mode. If you have used BTAM under DOS or DOS/VS in the past, you have to reassemble and catalog BTMOD before submitting your applications to DOS/VSE for execution. If BTMOD and the application program were assembled together, the application program must also be reassembled and re-linkedited.

ACF/VTAM Support

ACF/VTAM executes in virtual mode in its own partition. When VTAM is specified, RMS support (370 models 115 and 125 only) is automatically generated.

As ACF/VTAM uses the PFIX macro, processor storage page frames must be allocated to the partition in which ACF/VTAM is to run. A separate partition is required for VTAM application programs. For information on installing this licensed program refer to the ACF/VTAM documentation.

Interactive Computing and Control

The licensed program VSE/Interactive Computing and Control Facility (VSE/ICCF) offers interactive timeshared computing and control services to terminal users.

VSE/ICCF provides a collection of tools for

- Online library maintenance
- Context editing and text manipulation
- Development and execution of interactive problem programs
- Job entry
- Monitoring of time-shared job processing.

VSE/ICCF runs in a DOS/VSE partition. Support for VSE/ICCF is generated in the supervisor by specifying ICCF=YES in the SUPVR generation macro.

Access Authorization Checking and Security Event Logging

DOS/VSE provides a service to check against unauthorized usage of your data and your programs.

Support for this function is available if you assigned a positive value to the SEC parameter in the FOPT generation macro. In addition, your supervisor must have been generated with support for the VSE/Interactive Computing and Control Facility (VSE/ICCF), by specifying ICCF=YES in the SUPVR generation macro.

Access Control

DOS/VSE provides access control for the following resources:

- your data
- your private libraries
- individual programs (phases) within any of the core image libraries.

Access control is not available for DOS/VSE system libraries. However, it is available for phases of the system core image library.

Security profiles. To do this checking, DOS/VSE uses the 'System Security Table'. You build this table through the DTSECTAB macro; usage of this macro is described in *Data Security under DOS/VSE*. DOS/VSE loads this table into the SVA at the time of IPL.

The system security table has two groups of entries:

- *User profile entries*. Anyone who uses your data processing installation and wants to access secured programs or data or both must submit a user-id and a password; the batch user through the // ID job control statement, the terminal user through logon procedures. User-id and password have to match the corresponding parameters within one particular user profile entry. In addition, each user profile entry may contain up to 32 security classes.
- *Resource profile entries*. There is one entry for each named resource which is defined as 'secure'. Such a resource may be a file name, a library name, or the phase name of a program.

  Associated with each resource is a security class. When a user program attempts to access a protected resource, DOS/VSE compares the security class in that user's profile with the security class assigned to the resource. If the security classes don't match, access to the particular resource is denied to the user program.

For more information about data security implementation, refer to *Data Security under DOS/VSE*.

Logging and Reporting

If you have the licensed program *VSE/Access Control – Logging and Reporting* installed, the security related events are recorded on the logging file. For details on the creation of and access to the logging file, refer to the documentation available with that program.

What constitutes a *security related event*, is determined at the time you build the resource profile entry. Depending on your installation's requirements, you may want to trace only security *violations* of a protected resource; or, you may want to trace all permitted accesses to that resource.

Use the Reporting Program to get a formatted listing of the logging file.

ASCII Support

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

ASCII tape files may be processed in any partition. Input files containing ASCII data are translated to EBCDIC as records are read into the I/O area. Output files described as ASCII are translated from EBCDIC to ASCII just prior to writing the records. No user translation tables or instructions are required.

If your DOS/VSE is required to process ASCII files, specify ASCII=YES in the SUPVR generation macro.

Job Accounting

The job accounting interface facility provides job and job step information that can be used for charging system use, supervising system operation, planning new applications, etc.

When this option is selected (JA=YES in the FOPT generation macro), job accounting tables are built in the supervisor to accumulate accounting information. One DOS/VSE job accounting table is maintained per partition. The format of these tables and information on how to write a DOS/VSE job accounting routine is given in *Chapter 4, Using the Facilities and Options of DOS/VSE.*

To utilize this job accounting information, you must write a routine to store or print the desired portions of the table. This routine must be cataloged in the core image library under the name $JOBACCT.

If the user I/O routine ($JOBACCT) is written using LIOCS with label processing, the JALIOCS parameter of the FOPT macro must be specified in addition to the JA parameter. JALIOCS indicates that a user save area and a label area in the supervisor are to be reserved. The label area replaces the one normally used by LIOCS label processing routines.

If the licensed program VSE/POWER job accounting is desired, support for the job accounting interface is required. No user-written data collection routine is then necessary. Refer to the *VSE/POWER* documentation.

Timer Services

The following timer services are available to DOS/VSE users:

- Time-of-day clock
- Interval timer
- Task Timer

The time-of-day clock is a standard hardware feature, while the task timer and the interval timer require other hardware features (the clock comparator and the CPU timer) which are standard on all System/370 and 4300 processors, except the 370 models 135 and 145. Utilization of these timer services in DOS/VSE is briefly discussed below. Except for the task timer, the timer services are automatically provided in DOS/VSE. Support for the task timer is a supervisor generation option.

Time-of-Day Clock

The time-of-day (TOD) clock provides a consistent measure of elapsed time suitable for time-of-day indication.

The TOD clock support also enables programs to issue the GETIME macro instruction, which causes the exact time-of-day to be stored in general register 1. A description of the use of the GETIME macro instruction is given in *DOS/VSE Macro User's Guide*.

The time-of-day and the date are automatically included with each // JOB and / & job control statement that is printed on SYSLST or SYSLOG.

The ZONE parameter in the FOPT macro is associated with the TOD support. In the ZONE parameter you indicate the difference between Greenwich mean time (GMT) and local time in hours and minutes. If the local time to be specified is GMT, the ZONE parameter can be omitted.

During the IPL procedure, if IPL is performed from SYSLOG, a message is printed on the operator console to inform the operator of the status of the date, clock, and zone. If necessary, the operator can correct this information in the SET command.

Interval Timer

The interval timer can be used by programs (main tasks or subtasks or both) that need to schedule certain processing based on discrete time intervals. If a problem program is written with the appropriate macros and routines, the interval timer causes an external interrupt when the time limit established by the program has elapsed.

Several problem program macros relate to interval timer support. For information about using these macros, refer to *DOS/VSE Macro User's Guide*.

Task Timer

The task timer can be used by the main task of the partition owning the task timer to escape from processing and enter an exit routine after a specified period of time. This discrete time interval is decremented only when the main task is executing. If support for the task timer is included in the supervisor, and the owning partition's main task is written with the appropriate macro instructions and routines, the specified task timer routine is entered when the time interval has elapsed.

To include support for the task timer in the supervisor, specify the TTIME parameter in the FOPT generation macro.

If an exit routine is not specified in the STXIT TT macro, the interrupt is ignored. The SETT macro is used to set the time interval, and that interval can be tested or canceled by means of the TESTT macro. The EXIT TT macro is used to return control from a task timer exit routine.

Console Buffering

In an installation with a relatively slow console device, the entire system can be held up while messages are being issued to the operator. Console buffering support builds a queue of output messages and returns control immediately to the partition requesting the output. The messages are then written as soon as the console becomes available.

Console buffering is useful in two cases:

- when your console device is a 3210/3215 printer keyboard, or
- when your console is a display operator console and a printer is used to produce a hard copy of messages while they are displayed on the screen.

In an installation without such printers, a performance improvement cannot be obtained by requesting console buffering support. On the contrary, console buffering may in that case even work to your disadvantage: certain DOS/VSE tasks such as error recovery routines issue high priority messages. If your console is a display operator console, and a DASD rather than a printer is used as a hard copy file, then, depending on the size of your console buffer, messages may be issued to the screen in such rapid succession that a message like *INTERVENTION REQUIRED ...* can easily be overlooked by the operator.

Support for console buffering is indicated by the CBF=n parameter in the FOPT generation macro (where n is the number of I/O requests to be buffered). If you decide to use console buffering, at least one buffer should be specified for each partition or task issuing messages so that buffers are available and the task can continue processing while the message is being printed. Two per partition is recommended. Console buffering is not split per partition, but used by the whole system.

Asynchronous Operator Communication

With asynchronous operator communication, operator requests (action or decision messages) and the corresponding replies need no longer be in series. They can be asynchronous; that is, the operator can defer replies to messages while the system continues processing. One reply per active task in the system may be outstanding at a time.

To enter a reply, the operator must key in the reply-ID that the system has assigned to the corresponding message. The asynchronous operator communication support is activated by specifying ASYNOC=YES in the FOPT macro. For details, refer to *DOS/VSE Operating Procedures*.

User Exit Routines

If required, the supervisor can permit user routines to gain control when any of the following types of events occurs:

- Interval Timer Interrupt (IT)
- Program Check Interrupt (PC)
- Abnormal Termination (AB)
- Operator Communication Interrupt (OC)
- Task Timer Interrupt (TT)
- Page Fault Handling Overlap (PHO)

Both the supervisor and the problem program that contains the user routine must have the proper code to establish an interface.

The problem program that wants to utilize the options must contain code to set up the interface. For the first five events, code can be generated by the STXIT macro. For the last event, code is generated by the SETPFA macro. This code is assembled in the main line of a problem program.

Short descriptions of the support for each of the types of user exit routines follow, indicating the associated problem program macros. For information on how multitasking affects this support and what happens if multiple events coincide, refer to *DOS/VSE Macro User's Guide*. Some high-level languages offer similar facilities, for details of which see the appropriate programmer's guide.

Interval Timer Exit

Suppose you want to take a checkpoint on a job at a certain time after it has started. Code the STXIT to set up the interface of your user-exit routine with the supervisor; use the SETIME macro to set a time interval. When that interval elapses, an interval timer interrupt occurs and control is given to your user routine. The user routine need not be entered immediately. For instance, if the user routine is in the background partition, and a foreground partition is active, the user routine will not be entered until the background partition becomes active.

To find out the time remaining in an interval, a program can issue the TTIMER macro instruction. The supervisor then loads this value in general register 0. This macro can also be used to cancel the remaining time in the interval.

Program Check Exit

Programs can establish linkage from the supervisor to a user program-check exit routine by coding an STXIT macro. If a program check occurs within the program, the supervisor gives control to the user routine instead of discontinuing the program. The user routine can analyze the program check and choose to ignore, to correct, or to accept it.

If the check is ignored, control can be given back to the supervisor by executing an EXIT PC macro; if the user routine can correct the error

condition, the routine can request via the EXIT macro that processing of the main line program continue.

If the problem cannot be resolved, the program check is accepted as valid. The user routine can then terminate further processing of the program by issuing a CANCEL, DUMP, JDUMP, or EOJ macro.

The ability to include a user routine to process program checks can be especially advantageous when using LIOCS. In that case, I/O housekeeping such as closing files and freeing tracks can be performed before termination of the job or task.

Abnormal Termination Exit

Programs can establish linkage from the supervisor to an abnormal termination exit routine by issuing an STXIT AB macro.

The macro allows a user routine to get control from the supervisor before an abnormal end-of-job condition discontinues the processing of the program. The user routine normally ends with one of the termination macros (CANCEL, DUMP, JDUMP or EOJ) to terminate the problem program and to return control to the supervisor, rather than by initiating the continuation of the problem program.

Operator Communications Exit

DOS/VSE allows problem programs to provide a routine for handling external interrupts from the operator. This support is useful in a number of applications, for example:

- A change in the environment is needed. A message is then issued by the program. For example: MOUNT TAPE XXX on unit xxx and press the interrupt key.
- In telecommunication, the OC exit allows the operator to start and stop activities on certain lines or terminals, or to invoke diagnostic procedures. In this case, program run sheets with explicit instructions may be required to ensure understanding between programmer and operator.

The external interrupt that links to an OC user exit routine is caused by pressing the request key and, when the attention routine identifier AR appears, replying *MSG* followed by the partition identifier (such as BG or F2).

Task Timer Exit

Task timer support is included in the supervisor by the TTIME parameter of the FOPT generation macro. This parameter also identifies the partition owning the task timer. Only the main task in the owning partition can utilize the task timer.

The time interval is specified in the SETT macro and is decremented only when the main task is executing. The exit routine specified in the

STXIT TT macro is entered when the interval has elapsed, provided linkage between that routine and the supervisor has already been established, at that point of program execution.

To find out the time remaining in an interval, the task can issue a TESTT macro. This causes the time remaining in the interval to be returned in register 0. The task can also issue a TESTT CANCEL to cancel the remaining interval time. In this case the exit routine is not entered.

Page Fault Handling Overlap Exit

A user routine can continue processing during the time a page fault is being handled by the system, provided this page fault occurs in the same task and not in a supervisor routine invoked by this task. This support is of interest only for programs executed in virtual mode and making use of user-developed subtasking rather than IBM-supplied multitasking.

Such programs may issue the SETPFA macro instruction to establish linkage from the page management routines in the supervisor to a user routine, called the page fault appendage routine. The usage of the SETPFA macro is described in *DOS/VSE Macro User's Guide*.

Disk Options

Options are provided for some DASD devices. These options are:

- System files on disk (or diskette)
- DASD file protection
- Track hold
- Rotational position sensing

System Files on Disk or Diskette

The system logical units SYSRDR, SYSIPT, SYSLST, and SYSPCH can be assigned to a disk or diskette device. When a system logical unit is assigned to a disk, it must have only one extent.

For example, you may want to catalog the output from a language translator to the relocatable library. During the language translation step, SYSPCH could be assigned to a disk extent. The resultant object module would then be cataloged via the librarian program MAINT by assigning SYSIPT to the same disk extent.

Support for system files on disk or diskette is specified in the SYSFIL parameter of the FOPT generation macro.

The SYSFIL option is required to implement extended support for the procedure library. This means that cataloged procedures may contain in-line SYSIPT data. The sets of control statements that can be cataloged into the procedure library are, therefore, not limited to job control or linkage editor control statements. (See also *Library Options* earlier in this chapter.)

For systems without magnetic tapes, the SYSFIL option is required in order to install IBM programs and apply program maintenance, which, in this case, must be distributed on disk packs in SYSIN format.

DASD File Protection

This feature is provided to prevent user programs, which utilize DAM or user-written channel programs for writing onto DASD, from writing data outside of the limits of the DASD file currently being accessed. This might happen if, for example, a randomizing algorithm produces an unexpected DASD address which is outside the file limits.

DASD file protection support is indicated in the DASDFP parameter of the FOPT generation macro.

DASDFP gives protection on the basis of programmer logical units. If two DASD files are open in the same partition and use the same programmer logical unit, the DASDFP option does not give any protection to either of the two files.

If you are using physical IOCS, you must use the DTFPH macro to define the file. The file must be opened using the OPEN or OPENR macro, and each channel program must commence with a long seek (X'07') command, and contain no chained long seeks.

Specifying DASDFP does not prevent file contention between partitions, or within partitions if the same symbolic unit is used. Thus, more than one partition may access the same file at the same time and may even attempt to update the same record simultaneously. The track hold option (TRKHLD) is provided to solve this problem. Note, however, that all DASD writes (DAM and others) will be checked for being within the file-protect range.

Note that, for CKD devices, no protection is given to partially allocated cylinders; files to be protected should begin and end on cylinder boundaries.

Track Hold Option

The track hold option is used to ensure that, while data in a DASD file is being modified by one task, no other task in the system can access that data. The facility is available to all DOS/VSE disk access methods, including VSE/VSAM (and also ISAM Interface, except the LOAD function).

The track hold option can be selected by specifying the TRKHLD parameter in the FOPT generation macro.

Additionally, user programs must invoke the track hold feature. For the track hold feature to be effective all programs accessing the same file must request its use.

The track hold feature is requested in the DTF of the user program by specifying HOLD=YES. For VSE/VSAM files the track hold feature is

specified at the time the file is defined via the SHARE OPTION 4 parameter.

For FBA devices, the track hold facility protects the range of blocks which contains the accessed data. For CKD devices, the facility protects the track that contains the data being accessed.

Deadlock occurs if one task is waiting for a DASD area held by a second task and the second task is waiting for a DASD area held by the first. This can be prevented by establishing the convention that every task must be programmed so that it will not attempt to hold more than one DASD area at a time. Deadlock may also occur if the maximum number of DASD areas demanded to be held by all tasks combined exceeds the maximum specified in the TRKHLD parameter.

Rotational Position Sensing

Rotational Position Sensing (RPS) is a feature on all IBM CKD disk storage devices except 2311, 2314, and 2319; it is optionally available on IBM 3340. It provides the ability to overlap positioning operations on one device with service requests for other devices on a block multiplexer channel (or its equivalent on System/370 Model 115 or 125).

DOS/VSE makes use of the feature if you specify RPS=YES in the FOPT generation macro. However, you should not request RPS support if you use the 23xx emulator on a Model 115 or 125.

Better channel utilization can increase system throughput, especially in large multiprogramming systems with heavy concurrent I/O activity. Because a selector channel is monopolized once a channel program has been initiated, no other device on this channel can be accessed until the data has been transferred. With block multiplexer channels and the RPS feature of DASD devices, however, the device can disconnect from the channel during positioning operations. The channel is then available for other requests so that other devices on the channel can be accessed.

Overlap of positioning to a record on a track requires adding RPS CCWs to the direct access storage device channel programs. DOS/VSE system control and service programs that support RPS, dynamically build these CCWs during program execution provided that the supervisor is generated with RPS support and that the direct access storage device has the feature.

RPS support for DOS/VSE is provided in all access methods which support RPS DASD devices and in the DOS/VSE system control and service programs where the implementation benefits total system performance. Implementation of RPS support in DOS/VSE utilizes virtual storage to enable you to use RPS to avoid recompiling or relink-editing your problem programs. The partition GETVIS area is used to generate an extension to the DTF, and the shared virtual area is used to hold the RPS phases which are used in lieu of the logic modules of LIOCS.

Efficient use of RPS depends on each channel program's ability to free that channel so that it can service requests for other devices. Programs using DOS/VSE DASD LIOCS access methods will have RPS channel

programs built by the access method. Programs using PIOCS for DASD access have to be recoded to include Set Sector CCWs and to establish arguments for the CCWs. If this is not done, these programs will destroy the effectiveness of RPS by monopolizing the channel.

The RPS phases are loaded into the SVA by IPL if you have specified RPS=YES in the FOPT generation macro.

Figure 2-6 shows the organization of a user's program running in virtual storage without RPS support.

Figure 2-7 shows how, with RPS support, this organization will be modified when the pertinent file is opened to put the DTF extension in the partition GETVIS area. The pointers to the RPS phases which are used in lieu of the logic module and channel program will be put into the DTF while the non-RPS logic module and channel program addresses will be saved in the DTF extension. The DTF extension will be freed and the pointers restored to their original values when the file is closed.

DASD File Support for 3330-11 and 3350. Sequential or direct-access files can be created and accessed on the 3330-11 and 3350 disk devices in the same way as on other IBM DASDs if the pertinent program has so been written. However, to access these devices from programs that were written to create and access sequential and direct-access files on other IBM DASD types (such as 2314 or 3330-1), do one of the following without recompiling or reassembling your programs:

- Specify RPS=YES in the FOPT generation macro for supervisor assembly. With RPS support, DOS/VSE is able to access the 3330-11 or 3350 even if the original program specified a different device.
- If you have DOS/VSE or Release 34 of DOS/VS installed, relink your programs. This makes them capable of accessing the 3330-11 or 3350. The sequential disk and direct access logic modules are able to handle all CKD disk types.

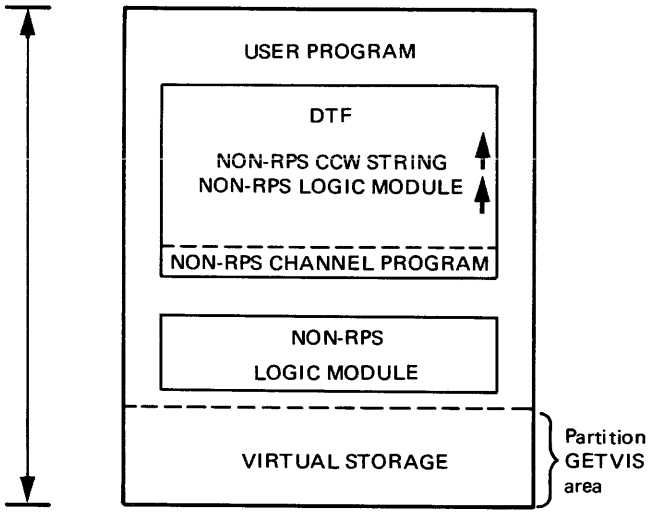


Figure 2-6. User Program Running in Virtual Storage without RPS Support

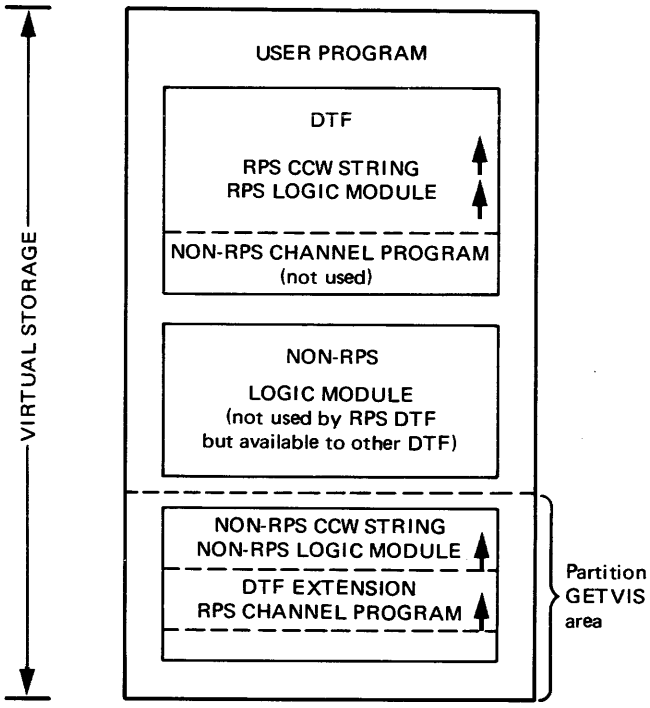


Figure 2-7. User Program Running in Virtual Storage using RPS Versions of Logic Module and Channel Program

Channel Queue

The channel queue (CHANQ) is used by DOS/VSE to schedule I/O operations. DOS/VSE builds an entry in the channel queue whenever a request is made for an I/O operation and the entry remains in the queue until the operation has completed. Thus, at any point in time, the queue consists of entries for I/O operations in progress and I/O operations waiting for initiation. Whenever an I/O event completes, the queue is examined to see if another entry exists for the channel, and if so, the operation is initiated. The number of channel queue entries to be reserved in the supervisor can be specified in the CHANQ parameter of the IOTAB macro.

The number of occupied entries in the channel queue depends on the activity in the system and no accurate formulas for determining the optimum size can be given.

Specifying too small a channel queue will cause performance degradation, too large a channel queue value will waste storage space.

Tasks or programs that request an I/O operation when the channel queue is full will be set in the wait state until an entry becomes free.

To avoid performance degradation it is better initially to specify ample channel queue space, and reduce the allotted space later, if desired. Given below is a *rule-of-thumb* that you may follow:

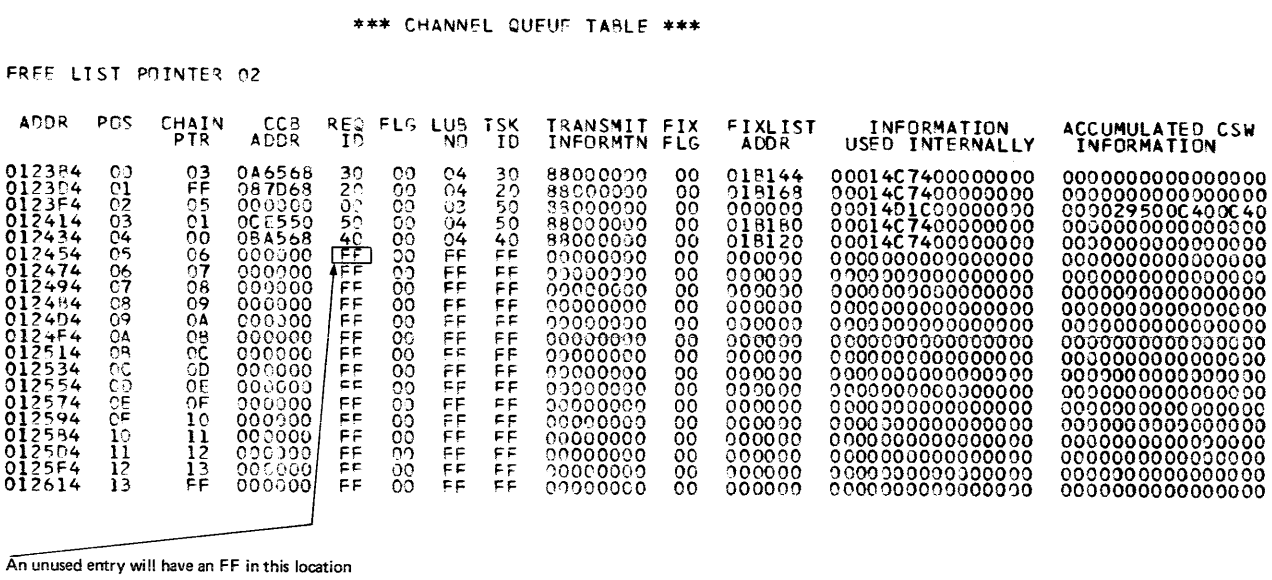
- Specify at least one queue entry for each I/O request that can be issued concurrently (open files per job step per partition).
- Specify one entry for the SYSRES file and one for the page data set.
- Specify one entry for each task or partition in the system.
- Specify one entry for each console buffer in the system.
- If multiple volume files are used on the system, specify one entry for each file being accessed at the same time.
- Add two entries per tape drive.
- Specify one entry for each teleprocessing line that could solicit input. If IBM 2260 local or 3270 local video display units are to be supported by BTAM-ES, specify one entry for each display.
- Add five entries to the total for contingencies.

When the system has been generated, run as many programs as represent the heaviest work load; in particular, run any teleprocessing programs. Then, before the next IPL, obtain a formatted dump of virtual storage.

An analysis of the channel queue should show that entries near the beginning of the table have been used, whereas those near the end are unused. Although the unused entries are normally redundant, a few surplus entries should be retained to allow for exceptional cases. If all the entries

have been used, then the channel queue was almost certainly too small, and a process of experimentation will show the correct size.

Figure 2-8 shows the channel queue as displayed in a formatted dump. Refer to *DOS/VSE Serviceability Aids and Debugging Procedures* for information on obtaining a formatted dump.



*** CHANNEL QUEUE TABLE ***

FREE LIST POINTER 02

ADDR	POS	CHAIN PTR	CCB ADDR	REQ ID	FLG	LUB NO	TSK ID	TRANSMIT INFORMTN	FIX FLG	FIXLIST ADDR	INFORMATION USED INTERNALLY	ACCUMULATED CSW INFORMATION
0123B4	00	03	0A6568	30	00	04	30	88000000	00	01B144	00014C7400000000	0000000000000000
0123D4	01	FF	087D68	20	00	04	20	88000000	00	01B168	00014C7400000000	0000000000000000
0123F4	02	05	000000	00	00	02	50	88000000	00	000000	00014D1C00000000	0002950DC400C40
012414	03	01	0CE550	50	00	04	50	88000000	00	01B1B0	00014C7400000000	0000000000000000
012434	04	00	0BA568	40	00	04	40	88000000	00	01B120	00014C7400000000	0000000000000000
012454	05	06	000000	FF	00	FF	FF	00000000	00	000000	0000000000000000	0000000000000000
012474	06	07	000000	FF	00	FF	FF	00000000	00	000000	0000000000000000	0000000000000000
012494	07	08	000000	FF	00	FF	FF	00000000	00	000000	0000000000000000	0000000000000000
0124B4	08	09	000000	FF	00	FF	FF	00000000	00	000000	0000000000000000	0000000000000000
0124D4	09	0A	000000	FF	00	FF	FF	00000000	00	000000	0000000000000000	0000000000000000
0124F4	0A	0B	000000	FF	00	FF	FF	00000000	00	000000	0000000000000000	0000000000000000
012514	0B	0C	000000	FF	00	FF	FF	00000000	00	000000	0000000000000000	0000000000000000
012534	0C	0D	000000	FF	00	FF	FF	00000000	00	000000	0000000000000000	0000000000000000
012554	0D	0E	000000	FF	00	FF	FF	00000000	00	000000	0000000000000000	0000000000000000
012574	0E	0F	000000	FF	00	FF	FF	00000000	00	000000	0000000000000000	0000000000000000
012594	0F	10	000000	FF	00	FF	FF	00000000	00	000000	0000000000000000	0000000000000000
0125B4	10	11	000000	FF	00	FF	FF	00000000	00	000000	0000000000000000	0000000000000000
0125D4	11	12	000000	FF	00	FF	FF	00000000	00	000000	0000000000000000	0000000000000000
0125F4	12	13	000000	FF	00	FF	FF	00000000	00	000000	0000000000000000	0000000000000000
012614	13	FF	000000	FF	00	FF	FF	00000000	00	000000	0000000000000000	0000000000000000

An unused entry will have an FF in this location

Figure 2-8. Channel Queue Table

Supervisor Buffers for I/O Processing

Supervisor buffer space is used for the handling of I/O requests from programs that execute in virtual mode. You specify the number of buffers via the BUFSIZE parameter of the VSTAB generation macro.

The amount of buffer space required is dependent on the number and type of concurrent I/O requests. The number of entries that you specify in the channel queue table can be used as a guide. Generally three times the number of channel queue table entries will give a sufficient number of buffers. If ISAM is the predominant access method used or if you have generated RPS support, you should increase the number of buffers by 20%.

Because your supervisor must end on a 2K boundary, any space between the end of the supervisor and the next 2K boundary will be used for I/O buffers in addition to the amount you specify in the VSTAB generation macro.

To determine whether or not you specified a sufficient number of buffers, use a technique similar to the one suggested for an analysis of the channel queue. While running as many programs as represent your heaviest work load, issue the DUMP command specifying the begin and end addresses of the buffer area in the supervisor; if all blocks have been used, then probably too few buffers were specified.

The use of the buffers is different in ECPS:VSE and 370 mode.

ECPS:VSE Mode. The buffers are called work blocks, and they have a size of 36 bytes each. DOS/VSE uses the work blocks to store information about your channel program and the I/O areas for that channel program. The information will be used to fix in processor storage your I/O areas, channel program and control blocks until the I/O request has been satisfied. The information stored is referred to as a *fixlist*. For example, DOS/VSE needs one workblock per I/O request for an FBA type DASD and two or more such blocks per I/O request for a CKD type DASD.

If you are writing your own channel programs it is suggested that you use the IORB macro rather than the CCB so that your channel program will contain a fixlist; processing will then be faster. For more information about these two macros, refer to *DOS/VSE Macro Reference*.

370 Mode. In 370 mode the buffers are called copyblocks and have a size of 72 bytes each. DOS/VSE uses the copy blocks to keep a copy of your channel program and control blocks in the supervisor area.

Your channel program refers to virtual addresses and these addresses must be translated to reflect the processor storage locations that your I/O area(s) actually occupy. (The translation is necessary since 370 mode does not support relocating channels which can do the address translation.) Once your channel program is translated, the I/O area(s) are fixed in processor storage and the translated channel program is given to the channel for execution. If you have installed the licensed program VSE/VSAM the minimum number of buffers you should specify is 40. To execute DOS/VSE system utility programs, DOS/VSE needs up to 38 copy blocks.

Bypassing System Translation of I/O Addresses. In most instances, double buffering techniques and an increase in block size can significantly reduce the system overhead associated with channel program translation. However, in extreme cases, you may wish to perform your own translation of channel programs and thereby avoid system CCW translation overhead. Programs that might require this are EXCP programs that have very high start I/O rates and that repeatedly use the same channel programs.

DOS/VSE provides support that assists in the translation of channel programs. This support allows you to use the VIRTAD and REALAD macros as well as the REAL parameter of the EXCP macro. You must obtain processor storage by means of the PFIX macro and then translate the channel program. For detailed information see *DOS/VSE Macro User's Guide* and *DOS/VSE Macro Reference*.

The Fast Translate or Fast Function Option. You may specify FASTTR=YES in the FOPT generation macro. This creates a supervisor with *fast-function* support in ECPS:VSE mode and *fast-translate* support in 370 mode.

The feature works essentially the same way in both ECPS:VSE and 370 mode. That is, the supervisor buffers used for an I/O request are not released when the I/O request is completed. The buffers are saved and the referenced I/O areas are fixed in processor storage until the end of job. This can speed I/O processing if your program has frequent repetitive I/O requests. The overall effect on your DOS/VSE system is subjective, however.

The page pool is decreased in size because the I/O areas remain fixed. Additionally, more supervisor buffers are required than without this support. In ECPS:VSE mode specify, as a rule of thumb, a number of buffers that is 9 times the number of channel queue entries and in 370 mode 6 times the number of channel queue entries.

If you do not specify enough buffers or the page pool becomes too small, the saved buffers and fixed I/O areas are released as required by DOS/VSE.

Specification of FASTTR=YES may cause degradation of performance when CICS/VS accesses SAM, ISAM and DAM files.

FASTTR can be switched off for the duration of a job by specifying NOFASTTR in the OPTION job control statement. Specifying this option is meaningful if, for a job, it is unlikely that buffers and fixed I/O areas will be reused.

Error Queue

The error queue option is of value to installations using a large number of I/O devices, for instance, teleprocessing systems. The ERRQ parameter of the FOPT generation macro allows you to specify the number of error queue entries within the error recovery block of the supervisor. These entries are used to record information on I/O device errors, and this information is used by the ERP and RMSR routines. The normal default value is five entries, but in ERRQ you can specify up to 25.

Reliability/Availability/Serviceability

DOS/VSE includes routines that analyze and record CPU, channel, and device errors and attempt to recover from them. The data is stored on the system recorder file (SYSREC). The information obtained from this file serves not only as an aid in diagnosing machine errors, but also helps IBM customer engineers to increase reliability, availability and serviceability (RAS) of your system.

If on-line recovery is impossible, the system may be placed in a hard wait state. A message is then issued to the system operator to run either the SEREP or EREP program to obtain the diagnostic data.

On the IBM System/370 Models 115 and 125, errors in the CPU and natively attached input/output devices (for example, card reader/punch, disk and printer) are recorded on the system diskette (see note). This hardware error recording is independent of the software routines. The recorded hardware statistics may be obtained on the video display unit (DOC), on advice of the IBM customer engineer, through the LOG ANALYSIS displays. Error recovery for channel-attached input/output devices for these CPU models requires the use of software routines with error recording on SYSREC, known a recovery management support (RMS). *DOS/VSE Serviceability Aids and Debugging Procedures* contains more detailed information about the various RAS features discussed below in context with a discussion of RMS.

Note: *IBM System/370 Model 158 and the IBM 3031 have a similar hardware error recording feature in addition to a software error recording facility.*

Recovery Management Support

These routines, referred to as RMS, are standard for all /370 and 4300 processors except for the System/370 Models 115 and 125. For these models, specify the RMS parameter of the SUPVR generation macro to obtain the RMS support of your choice.

If full RMS support is included (RMS=YES is specified or forced for models 135 and above), the following RAS facilities are provided:

- Machine Check Analysis and Recovery
- Channel Check Handler

These facilities provide hardware error analysis and attempt recovery. Another RAS facility, the Recovery Management Support Recorder (RMSR) provides for recording of error and operational statistics on SYSREC as follows:

- Machine Check (CPU)
- Channel Check
- Unit check
- Tape/disk error statistics by volume
- MDR (Miscellaneous Data Recorder)
- IPL information
- End-of-Day statistics held in main storage

Device ERP routines are standard for all CPU models. For models 115 and 125, if full RMS support is not desired, RMSR support for channel attached devices, tape units, and TP devices is still included even if you specify RMS=NO. Specification of RMS=NO will cause the system to enter a hard wait on the occurrence of a hardware failure with no recording on SYSREC. However, the system diskette will contain error recordings for the CPU and natively attached devices. If your installation plans to use DASD switching, RMS=YES must be specified.

RMSR has several options that may be specified during supervisor assembly. These options involve the tape error statistics and error volume analysis.

Tape Error Statistics. As a standard feature the DOS/VSE system gathers tape error statistics. This information is accumulated in the PUB2 table for each tape unit and stored in the system recorder file SYSREC. For tapes with standard labels the information is accumulated and stored per volume. When error statistics are required to enable the monitoring of nonstandard or unlabeled tapes, the TEBV parameter of the FOPT generation macro gives you two options: the parameter can be specified as IR (individual

recording) or as CR (combined recording). IR refers to the accumulation of error statistics between two consecutive OPENs on the same tape unit. CR refers to the accumulation of error statistics on the same tape unit until a standard labeled tape is opened on that unit or until a ROD command is issued.

Error Volume Analysis. This option of RMSR enables you to specify the number of temporary read/write errors that occur on a tape volume before an informatory message is printed on SYSLOG. The threshold value of temporary read/write errors is specified in the EVA parameter of the FOPT generation macro.

Defining the System Configuration

During supervisor generation you must code various generation macros that relate to the central processing unit, to the portability of your system and to the I/O devices installed (or planned to be installed).

Central Processing Unit

In ECPS:VSE mode, there is no need for specifying a CPU model. In 370 mode, you must specify which central processing unit is to be used. The specification is made in the MODEL parameter of the CONFG generation macro.

If you plan to run your generated system on more than one CPU model, you may specify any of the models you plan to use. However, you must take the following precautions:

- restrict yourself to one CPU-mode. For example, it is not possible to generate a supervisor to be run either on a System/370 or on a 4300 processor in ECPS:VSE mode.
- ensure disk support that fits all configurations you plan to use.
- if you have a display operator console (DOC) installed, this DOC must be compatible with the DOC support of your generated supervisor. Note that the 3210/3215 printer keyboard is always supported. Therefore, if you specify DOC=3277 in the FOPT generation macro, for example, your actual console must not be a 125D; it can only be a 3277 DOC or a 3210/3215 printer keyboard.
- ensure RMS support, except if you plan to stay with /370 Models 115/125. For example, if you specify MODEL=125 and you plan to run your supervisor on a Model 138, specify RMS=YES.
- specify the larger range of channels in the DASDFP parameter, if you use DASD file protection.

You should keep in mind one other option that takes effect at the time of IPL: if your page data set resides on multiple extents and you plan to use the same pack(s) with the same extents on the alternate system, then specify DSF=N in the DPD command at your last IPL before moving to

the new system. The DPD command is explained in *Chapter 3, Using the System*, section *Initial Program Loading (IPL)*.

Display Operator Console Support

In ECPS:VSE mode, 3277 is the standard operator console support. In 370 mode, the DOC parameter of the FOPT generation macro determines the operator console support in the supervisor. The following discussion pertains to 370 mode only.

If you do not specify the DOC parameter, the following defaults apply:

MODEL (in CONFG macro)	Default
115, 125	125D
138, 148, 158, 3031, 4300 (in 370 mode)	3277
135, 145, 155-II	NO (3210/3215 printer keyboard support)

A specification other than the default is not checked or changed to the default. For example, MODEL=158, DOC=NO gives a supervisor that is generated for the 158 with console support in 3210/3215 printer keyboard mode.

I/O Devices

The supervisor generation macros that relate to the I/O devices attached to the CPU are: PIOCS and IOTAB. These macros are discussed below.

The PIOCS generation macro defines the I/O configuration that is to be supported. The associated parameters involve the channel switching and disk device support.

The IOTAB generation macro, in general, defines the area for the necessary device tables for the system. The parameters involved refer to:

- The number of programmer logical units for each partition defined by the NPARTS parameter in the SUPVR macro.
- The maximum number of extents that may be defined for the page data set.
- The number of job information blocks for the system. One is required whenever a temporary or alternate assignment is made.
- The number of DASD devices separate for each IBM DASD type included in your system configuration.
- The number of tape devices.
- Other devices like the IBM 3800 Printing Subsystem, the IBM 3886 Optical Character Reader, or the IBM 3540 Diskette Reader.
- The number of telecommunication devices (Models 115/125 only).
- The estimated number of physical I/O devices.

Before you can actually use your I/O devices, you must define each unit to the system, specifying its characteristics such as channel and unit address, device type, its mode (if applicable). You do this via the ADD command at the time of Initial Program Load (IPL).

A supervisor generation macro is not available for this purpose. Nevertheless, because the definition of your I/O devices is likely to remain stable over a longer period, you should already at the time of system generation give some thought to the sequence of ADD commands you are going to use. The total number of ADD commands must not exceed the total number of devices specified in the IODEV parameter of the IOTAB generation macro.

Furthermore, physical I/O device addresses must be assigned to logical unit names, via the // ASSGN job control statement or job control command (no //). You cannot make these assignments at the time of supervisor generation, even though you may want to have them remain unchanged for a longer period of time.

The Automated System Initialization (ASI) facility allows you to place all your IPL commands in a procedure. This procedure is automatically invoked each time you IPL the system. Additionally the ASI facility allows you to place job control commands in a procedure which would be automatically invoked whenever the pertinent partition is started.

Definition and assignment of I/O devices is described in sections *Starting the System* and *Controlling Jobs* within *Chapter 3, Using the System*.

Emulators

Through emulation, a program written for execution on a 1400 series machine can be executed under DOS/VSE. The emulator program, serving as the interface between the user program and the DOS/VSE supervisor, runs together with the user program in the same partition.

Several emulators can be executed concurrently. One exception, however, is the Model 125, which cannot execute two 1400-series emulator jobs concurrently. To use a 14xx emulator on a Model 125, RPQ SU002 is required.

Tape reading and writing on 1400-series machines can operate with odd or even parity checking. If you will be using a 1400-series emulator and mixed-parity tape processing, you must specify EU=YES in the SUPVR generation macro.

Prior to executing emulator jobs, you must generate the emulator program and catalog it into the core image library. This can be done when the system is generated or at a later time.

For more information about available emulator programs see the publication *Introduction to DOS/VSE*.

Chapter 3: Using the System

This chapter is intended primarily for programmers who are responsible for optimum system throughput and for servicing the installation's libraries. The topics discussed are:

Starting the System — describes the initial program load (IPL) procedure. It also describes how to create the file required for recording error information, how to allocate storage to a partition, and how to start a foreground partition.

Controlling Jobs — describes the required input to the job control program, which controls the execution of a job; it includes a brief discussion of label processing.

Linking Programs — describes the input to the linkage editor program, which links the modules produced by language translators, produces executable phases and places them in the core image library.

Using the Libraries — provides the information on how to alter, copy, and inspect the contents of the libraries. It also describes how to allocate space to the libraries and how to create private libraries.

Starting the System

Before a job can be submitted to DOS/VSE for execution, the supervisor must be read into processor storage and the job control program must be loaded into the background partition. To do this, the operator starts DOS/VSE by following the initial program load (IPL) procedure.

On a 4300 processor the amount of virtual storage available can be altered during IML (Initial Microprogram Load) which is done prior to the IPL procedure. Refer to section *Virtual Storage Size* in *Chapter 2, Planning the System*, and also to the Operator's Library Procedures manual for the pertinent CPU model.

This section describes the use of the IPL commands. The exact formats of these commands are contained in *DOS/VSE System Control Statements* and *DOS/VSE Operating Procedures*. This chapter also provides a summary of the automatic functions of IPL; descriptions of how to load the shared virtual area, and how to create the system recorder file (SYSREC) and the hard copy file; a section on the optional user exit routine for user-defined processing after IPL; and a section on entering data into SYSREC.

You must perform the IPL procedure each time you have to do one of the following:

- Load a new supervisor (for normal system start-up, for different supervisor options, or to recover from a system malfunction. For the last, refer to *DOS/VSE Serviceability Aids and Debugging Procedures*).
- Modify the shared virtual area size.

- Add devices to or delete them from the system configuration.
- Set or change the time-of-day clock value.
- Set or change the system's time zone value.
- Change the channel and unit assignment of the system residence (SYSRES), the VSE/VSAM master catalog (SYSCAT), SYSREC, or the page data set due to hardware problems with the channel or disk drive.
- Create SYSREC (for the first time or because of hardware problems).
- Replace SYSRES or the page data set because of a hardware problem with the pack.

Initial Program Loading (IPL)

For IPL, you place the system residence disk pack on a disk drive and set the address of that drive in the load unit switches, ready SYSLOG and the device containing the page data set and press LOAD on the console (on the video display/keyboard console, type in the address of the drive and press ENTER).

Now, the Automated System Initialization (ASI) is ready to control the IPL process. If you want to prevent ASI from executing your cataloged IPL procedure, press the INTERRUPT key immediately after you pressed LOAD. This allows you either to specify different ASI procedures or to leave ASI and continue with an interactive IPL. ASI is discussed in more detail under *Automatic Functions of IPL*, below. The remainder of this section describes the interactive IPL process.

Next, DOS/VSE enters the wait state. You now must indicate to DOS/VSE the device that is to be used as the operator console (SYSLOG). To do so, press the Request key (or END/ENTER) on the selected device. This causes an interrupt and automatically transmits the address of this device to DOS/VSE. (If you have installed an IPL communication device list, DOS/VSE accepts the interrupt only if the address of the device is contained in the list). IPL assigns SYSLOG to the device. This assignment remains valid until the next IPL.

At this point, DOS/VSE requests you to specify the supervisor you want to be used. You indicate this by one of the following:

- pressing ENTER or the Request key
- entering supervisorname[,P | N][,LOG | NOLOG]

Pressing ENTER or the Request key indicates that the pageable default supervisor is to be loaded ($$A$SUP1,P,LOG).

Specifying P causes the loaded supervisor (default or your own) to have certain routines pageable; specifying N causes the loaded supervisor (default or your own) to be non-pageable. If, on entering the supervisor name, you specify neither P nor N, P will be assumed.

By setting the list-option to NOLOG, you can prevent IPL from listing the IPL commands on SYSLOG. If you don't specify the list-option, LOG will be taken as default; that is, all IPL commands are listed on SYSLOG. Invalid commands are always listed.

IPL now reads the supervisor into low processor storage from the core image library. If an irrecoverable error is sensed while reading the supervisor, an error message is displayed on SYSLOG; the hard wait status is entered and an error code is set in the first four bytes of processor storage. The IPL procedure must then be restarted. For more information on wait states, refer to *DOS/VSE Serviceability Aids and Debugging Procedures*.

Establishing the Communications Device for IPL

DOS/VSE again goes into a wait state with all interrupts enabled (see Note). At this time you must indicate which device is to be used to communicate the IPL commands to the system. The specific manual operation you must perform depends on the selected device:

- If you wish to use the console (SYSLOG), press the Request key on the console. (On the video display/keyboard console, you can press the Enter key, the Request key, or the Cancel key.)
- If you wish to use a card reader, ready this card reader. DOS/VSE then assigns SYSRDR to this device for the duration of IPL.
- If you wish to use an IBM 3540 Diskette I/O Unit, ready it. DOS/VSE assumes that the file IJIPL is part of the diskette and that it contains the IPL commands in card image format (unblocked 80 byte records).

Note: *Because any interrupt will (on a first-come basis) establish the issuing device as the IPL communication device, it is advisable that TP installations and terminal-oriented installations with locally attached terminals, (for example, IBM 3277) install the IPL-phase $$A$CDL0. (See* IPL Communication Device List.*)*

IPL Commands

IPL commands serve to set or change various characteristics of your system. They operate on the following items:

I/O configuration	–	ADD and DEL commands
System date and time	–	SET command
System disk file assignments	–	DEF and DPD commands
Label information area outside the SYSRES extent	–	DLA command
Shared Virtual Area size	–	SVA command

ADD and DEL commands should precede all other commands; the SVA command is the last command to be submitted.

The ADD Command. Use the ADD command to define all your input and output devices to your system. This definition specifies for a device the channel and unit address, the device type, the mode (if applicable), and whether automatic channel switching is desired.

Each individual drive of a DASD (of a 3333/3330 or 3310, for example) requires an ADD command. Note that if one physical spindle contains two or more logical spindles, ADD commands must be issued for each of these logical spindles.

The following requirements should be kept in mind:

- You can add a device only if sufficient device table space was provided via the IOTAB generation macro during supervisor assembly.
- If DASD file protection was generated in the supervisor and you add a CKD DASD, the DASD must conform to the channel range specified in the DASDFP parameter of the FOPT generation macro.

If any of these requirements is not satisfied, you will get an appropriate error message. You must then provide space in the control blocks for the additional device by:

- deleting unnecessary devices of the type you want to add and then re-issuing the ADD command, or
- re-assembling the supervisor.

Note: For an IBM 3031 CPU, one service record file 7443 must be defined. This allows DOS/VSE to access the system diskette on the service support console. After having created the system recorder (SYSREC) file and encountered the first // JOB statement, DOS/VSE reads machine check frames and channel check frames from the service record file and writes them onto the SYSREC file. Those frame records will be available as input for the Environmental Recording Editing and Printing (EREP) program when that program is executed.

The DEL Command. Use the DEL command to drop an I/O device from the configuration you had established via ADD commands; this may be necessary if, for example, you defined (ADDed) more devices than you had allowed yourself in the IOTAB generation macro, or if you want to correct the device type for one of the preceding ADD commands. Because all references to the device are removed, any subsequent ASSGN job control statement that refers to a deleted device will not be accepted.

The Set Command. You can use the SET command to set the system date, the time-of-day clock, and the system time zone. If you specify a time-of-day clock setting, set the time-of-day clock switch to the "enable set" position at the exact time specified in the SET command. The SET command is required only if the time-of-day clock has not been set. If this is the case, a message at IPL will prompt the operator.

The DEF Command. You use the DEF command to assign the SYSCAT, SYSDMP, and SYSREC files. This command is mandatory.

The SYSCAT file, the VSAM master catalog, is required if you have the licensed program VSE/VSAM installed. If you don't have VSE/VSAM installed, specify DEF SYSCAT=UA. SYSREC is the symbolic name used for the *system recorder file*, the *hard copy file* and the *system history file*. Creation of these files is discussed later in this section. As described in section *System and Workfiles* of *Chapter 2, Planning the System*, the SYSDMP file can be used instead of SYSLST to hold system dumps, dump command output, and the output of your installation's stand-alone dump program.

The DEF command must be submitted after any ADD and DEL commands and prior to the SVA command. The ASSGN job control statement or command is not valid for SYSDMP, SYSCAT or SYSREC assignments.

The DPD Command. The DPD command is used to define the disk attributes of your page data set. The operands of the command allow you to specify

- a device address.
- whether the page data set resides on multiple extents.
- the size of a particular extent.
- whether the page data set is treated as a data secured file.
- the beginning address of the disk extent.
- the disk volume ID.
- whether or not the page data set should be formatted.

Because formatting the page data set is time-consuming, you should request it only if the pack was damaged. The first time you use the page data set, it will be formatted automatically.

The page data set can reside on any DASD supported by DOS/VSE as a system residence device. To help ensure better performance, the page data set should not reside on a pack that is subject to heavy I/O requests.

The DPD command is mandatory. It must be submitted after any ADD and DEL commands and prior to the SVA command.

If your page data set is to be allocated to multiple extents, you submit the corresponding number of DPD commands. After accepting the first DPD command, DOS/VSE prompts for additional DPD commands until either the entire virtual storage is covered by the specified extents or you submitted a number of commands that is equal to the limit specified in the DPDEXT parameter of the IOTAB generation macro.

The DLA Command. Use the DLA command to define or reference a label information area separate from the one within the SYSRES file. When, for example, two CPUs or two DOS/VSE systems under VM/370 share a SYSRES file, two separate label information areas enable the two systems to distinguish between dedicated system file names.

The additional label information area must be located on the same volume that contains the SYSRES file. Its format and layout are identical to the format and layout of the SYSRES label information area.

When you define the area, you specify its beginning address by the CYL or BLK parameter of the DLA command. By specifying NCYL or NBLK you may deviate from the default size of a SYSRES label information area. At the time of definition you supply a name by which this label area is referenced during subsequent IPLs.

To define a label area of 300 blocks on an FBA device, you might submit the following DLA command:

```
DLA NAME=MYLABEL,BLK=125000,NBLK=300
```

At subsequent IPLs, you refer to this area by issuing the command

```
DLA NAME=MYLABEL
```

If the DLA command is used, it must be submitted after any ADD and DEL commands and prior to the SVA command.

The SVA Command. This command must be the last IPL command submitted. The SVA command may be given with or without parameters.

The command's parameters (SDL, PSIZE, GETVIS) are used to increase the SVA size beyond the size set by the DOS/VSE IPL routine. They serve to add space for

- System Directory List (SDL) entries
- phases that you want to have loaded into the SVA
- the system GETVIS area.

If the parameters are not specified during IPL, no user SDL or phase space is reserved in the SVA for user phases. An SVA will be allocated which is large enough to contain:

- Phases required for use by DOS/VSE.
- Phases required for installed licensed programs.
- The default system GETVIS area.
- Required SDL entries.

Automated System Initialization (ASI). The facility allows you to place all your IPL commands into a procedure. In addition to IPL commands, you include a specification of your SYSLOG device and of the supervisor name you intend to use. After you have cataloged this procedure into the procedure library, you may let DOS/VSE execute the procedure whenever you IPL your system. Figure 3-1 shows a typical ASI IPL procedure (the first record specifies SYSLOG and a supervisor name; the ADD command preceding the DEF command defines the SYSLOG device type):

```
01F,$$A$SUP3,P,NOLOG
ADD 280,3420T9
ADD 281,3420T9
  .
  .
  .
ADD 162,3330
ADD 163,3330
ADD 00C,3505
ADD 00E,1403U
ADD 00D,3525P
ADD 01F,125D
DEF SYSREC=160,SYSCAT=160,SYSDMP=161
DPD UNIT=161,VOLID=PDSWRK,CYL=300,DSF=N
SVA SDL=100,PSIZE=150K,GETVIS=150K
/+  END OF IPL PROCEDURE
```

Figure 3-1. Example of an ASI IPL Procedure

Other ASI procedures contain job control information that serves to prepare partitions for operation: they allocate partition space, store label information, assign devices to logical units etc. Therefore, DOS/VSE performs the entire system initialization without your intervention.

A detailed description of how to set up ASI procedures is given in section *Automated System Initialization*.

Automatic Functions of IPL

Apart from the Automated System Initialization, IPL performs the following operations automatically:

- Builds the required control blocks and device tables.
- In 370 mode, determines the size of the real address space.
- Unassigns any DASD assignments for devices that are not operational at this time (so as to prevent the error recovery routines from trying to establish error recording statistics for these devices).
- Loads the printer-control buffers with the installation defined standard buffer images.
- Initializes the DOS/VSE RMS routines.
- Loads into the SVA required system phases and licensed program phases.

After IPL completes these operations, the system loader loads the job control program into the background partition and places the system in the problem program state. The message "READY FOR COMMUNICATIONS" appears on the console immediately after IPL is complete.

IPL Communication Device List

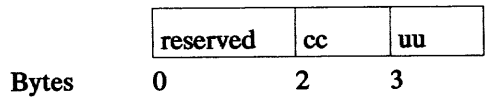
For telecommunication installations and for installations with locally

attached terminals (such as the IBM 3277), devices allowed to present an interrupt during IPL should be restricted because an unsolicited interrupt might interfere with your system start-up procedures. By installing an IPL communication device list, you can avoid that a device outside the operator's control establishes itself as the device used for submitting IPL commands.

To build a restrictive pool of IPL communication devices, you assemble an IPL communication device list (CDL) and catalog the list under the phasename $$A$CDL0 in the system core image library. During IPL, DOS/VSE loads this phase (if present) into storage. When DOS/VSE enters the wait state and an interrupt occurs, the CDL can now be searched for the address of the device issuing the interrupt. If the address is listed, the interrupting device is accepted as an IPL communication device and processing continues. If the address is not found, DOS/VSE remains in the wait state. Installation of the CDL is optional.

For IPL to be successful, once $$A$CDL0 is installed, the SYSLOG device address must be present in the CDL. If you intend to submit IPL commands from card reader or diskette, you must enter their addresses in the CDL as well. To ensure backup in case of hardware errors during IPL, consider stand-by devices, such as another card reader, diskette, or even an additional SYSLOG device in the CDL.

The CDL may have up to eight entries each of which is four bytes long:



where: cc = channel number
uu = unit number

You create the CDL by submitting a job that catalogs $$A$CDL0 into the system core image library. The example in Figure 3-2 creates a CDL with five entries:

```
// JOB CATALOG CDL
// OPTION CATAL,NODECK
   PHASE $$A$CDL0,+0
// EXEC ASSEMBLY
$$A$CDL0 CSECT
            DC   XL4'00C'      card reader
            DC   XL4'009'      1052
            DC   XL4'01F'      SYSLOG (DOC)
            DC   XL4'0BD'      3277
            DC   XL4'240'      diskette
            END
/*
// EXEC LNKEDT
/&
```

Figure 3-2. Example for the Creation of a CDL

Once phase $$A$CDL0 has been cataloged, the CDL addresses remain effective for subsequent IPLs. However, you may:

- Replace the phase by another one, either by assembling and link-editing a new phase or by using the DOS/VSE MAINT program to rename an already cataloged CDL that has a name other than $$A$CDL0.
- Override any CDL entry by manual intervention, which is the suggested approach should an erroneous CDL be cataloged in the core image library. The procedure for manually overriding the CDL is given in *DOS/VSE Serviceability Aids and Debugging Procedures*.

Building the SDL and Loading the SVA

Automatic SVA Loading

DOS/VSE builds a fresh copy of the SVA at each IPL. It loads phases into the SVA from the system core image library. DOS/VSE uses pre-defined load lists to find the appropriate phases. The load lists that identify required system phases are shipped in the system core image library ready for use at IPL. *DOS/VSE System Generation* contains a listing of the required system phases.

If you install an IBM licensed program that includes SVA eligible phases, you must catalog a load list for that licensed program. The licensed program documentation will describe this procedure and tell you how much space in the SVA the loaded phases require. Although DOS/VSE automatically allocates sufficient SVA space (by checking the load lists), you should know how much virtual storage will remain to be allocated to the partitions. (In 370 mode, your specification in the VSIZE parameter of the VSTAB generation macro is dependent on this information.)

Building the System Directory List (SDL). Entries in the SDL are copies of specific system core image library directory entries. Having entries in the SDL speeds up the loading of the corresponding phases.

At the time of IPL, DOS/VSE builds entries in the SDL for each phase that it automatically loads into the SVA. Each of those entries contains a pointer to the associated phase in the SVA.

User Options for the SVA

In order to load user chosen elements into the SVA (phases or SDL entries or both) the SVA space must be made large enough to accommodate the new entries. Space for user entries may be defined at IPL via the SVA command (see *The SVA Command* earlier in this section). The SET SDL command is available for having DOS/VSE both build an SDL and load phases into the SVA.

You should build SDL entries for certain frequently used DOS/VSE phases that are not SVA eligible. DOS/VSE provides a procedure (its name is SDL) that you should execute at the time of IPL. For a listing of the phases referenced by procedure SDL, refer to *DOS/VSE System Generation*. DOS/VSE does not automatically reserve SVA space for these SDL entries. In order to do that, you must define space with the IPL command SVA.

The SET SDL Command. The command used to create SDL entries and to load phases in the SVA is the SET SDL job control command. This command can be given only in the background (BG) partition. The command may be given at any time after IPL. There is no limit to the number of times it may be given.

It is recommended that you create a SET SDL job stream, catalog it as a procedure in the procedure library and run that procedure immediately after IPL. For compatibility with DOS/VS, SET SDL=CREATE will be accepted by DOS/VSE. If the SET SDL job stream is not being entered through a procedure, it may be submitted to job control through SYSRDR or SYSLOG (depending on the device from which job control is reading). This job stream can be entered via the IPL communications device. Figure 3-3 illustrates such a job stream.

Following the SET SDL command the input should be in the format of:

```
name[,SVA]
```

where name is any valid phase name and SVA indicates whether or not the phase is to be loaded into the SVA. If you specify SVA and the phase is SVA eligible, DOS/VSE loads that phase.

The phases need not be currently cataloged in the core image library and, if they are not, DOS/VSE issues a message on SYSLST (or SYSLOG if SYSLST is not available). If you subsequently catalog a phase into the system core image library under a name listed in the SDL as uncataloged, the entry in the SDL is activated. In this case, if the phase is also identified in the SDL as eligible for the SVA, it is loaded there immediately after it has been link-edited.

If duplicate phase names are submitted, only the first one is valid. For example:

```
SET SDL
    PHASEONE
    PHASEONE, SVA
/*
```

If PHASEONE exists in the system core image library, DOS/VSE builds an SDL entry. The phase is not loaded into the SVA; DOS/VSE recognizes statement PHASEONE, SVA as a duplicate and rejects it. In the above situation no message is issued to the operator. If the statements were submitted in two separate job streams;

```
SET SDL
    PHASEONE
/*
SET SDL
    PHASEONE,SVA
/*
```

an error message would be issued saying that an entry already exists in the SDL.

You will be able to place PHASEONE in the SVA the next time you IPL. The SVA (and of course the SDL) is rebuilt at each IPL.

It is recommended that you run the librarian program DSERV after a SET SDL job stream to be certain that all entries have been entered the way you wish. Include the DSERV control statement DSPLY SDL.

Fast B/C-transient Fetch. You have to issue the SET SDL command if you want to utilize the Fast B/C-transient Fetch feature. Normally, a request to load or fetch a logical transient routine results in an I/O operation. The Fast B/C-transient Fetch avoids this I/O operation by obtaining a copy from the SVA and moving it into the supervisor's logical transient area. Even if this action necessitates a page I/O operation, a performance improvement can be gained because no directory search operation is involved.

The transient routine must be self-relocating, the first character of its name must be a '$', and it must have been loaded into the SVA by the SET SDL command. To build an SDL entry for the transient and to load it into the SVA, supply the following statement (behind a SET SDL statement):

```
phasename,MOVE
```

DOS/VSE provides a SET SDL procedure, called 'FASTFTCH', which performs the above operation for certain B- and C-transients. A listing of these transient phases is provided as part of the documentation of VSE/Advanced Functions.

Replacing Phases Stored in the SVA. Occasionally, a phase stored in the SVA needs to be changed; that is, it must be replaced by an updated version. To replace a phase in the SVA, linkedit the updated version of the phase to the system core-image library. Immediately after this linkedit operation, DOS/VSE loads the updated phase into the SVA. The old version of the phase remains in the SVA, but is not addressable. Linkediting for inclusion of a phase in the SVA is further discussed in *Linking Programs* in this chapter.

Creating the System Recorder File

The DOS/VSE recovery management support requires a disk extent on which to record statistical information about machine errors and environmental information. This disk extent is called the system recorder file and is identified by the symbolic name SYSREC. The SYSREC file must exist before job control encounters the first // JOB statement after IPL. Usually, you create the SYSREC file only after the first IPL following a system generation (not after each IPL). If the SYSREC file has been damaged, however, you must re-IPL and re-create SYSREC.

If your DOS/VSE is running on an IBM 3031, the SYSREC file must be evaluated via program IFCEREP1 and recreated each time a hardware (microcode) change is installed which affects the frame records on the 3031's Service Record File. For details on IFCEREP1, refer to *OS/VS, DOS/VSE, VM/370 Environmental Recording Editing and Printing (EREP) Program.*

On a CKD device the SYSREC file requires a minimum of ten tracks (not including an alternate track), and it cannot be a split cylinder file. On an FBA device the SYSREC file requires a minimum of 72 blocks of 512 bytes each. You must define SYSREC as an extent of a permanently online disk device that DOS/VSE supports as a system residence device.

The IBM 3031 requires additional space on the recorder file to accomodate machine check frames and channel check frames (these frames are peculiar to the IBM 3031). On an IBM 3330, for example, this space amounts to approximately 9 tracks. If the SYSREC file resides on an FBA device with blocksize of 512 bytes, add 164 blocks. The exact amount of additional space needed for the recording of those frames can be calculated after the first // JOB statement has been processed and message '1I93I RECORDER FILE IS XX% FULL' is issued.

The SYSREC file label information must be included in the standard label portion of the label information area on the SYSRES file. Therefore, submit a // OPTION STDLABEL statement when you create the SYSREC file. Since the label information you submit is written at the beginning of the standard label area, which overwrites label information that was present there, you must resubmit all the information needed for subsequent operation. A more detailed description of preparing standard label information is given under section *Controlling Jobs* later in this chapter.

Figure 3-3 illustrates a job stream (via SYSLOG) to create the system recorder file. The IPL commands are included in the figure to show the proper placement of the statements that create the SYSREC file. Be sure that you do not submit a // JOB statement until you have supplied all the information applicable to SYSREC. This is because the SYSREC file is opened when the first // JOB statement is encountered. Note that the file name IJSYSRC is required in the DLBL job control statement.

```
0130I DATE=../../..,CLOCK=../../..
0110A GIVE IPL CONTROL COMMANDS
ADD . . .
ADD . . .
  .
  .
  .
ADD . . .
SET . . .
DEF SYSREC=190
DPD
SVA
0120I IPL COMPLETE FOR DOS/VSE REL. nn.n ECLEVEL=01
BG 1100A READY FOR COMMUNICATIONS
BG SET SDL
1S51I ENTER PHASE NAME OR /*
BG USERONE
1S51I ENTER PHASE NAME OR /*
BG USERTWO,SVA
1S51I ENTER PHASE NAME OR /*
BG   . . .
BG   . . .
BG   . . .
BG /*
BG ASSGN
      .
      .
      .
BG SET RF-CREATE
BG / / OPTION STDLABEL
BG / / DLBL IJSYSRC,'DOS.SYSTEM.RMSR.FILE'
BG / / EXTENT SYSREC,,,,1700,43
      .
      .
      .
BG /*
BG / / JOB FIRST
      .
      .
```

Submit with the rest of the STDLABEL statements

Figure 3-3. Example for the Creation of the SYSREC File and for Loading User Phases in the SVA

When the system is to be shut down, you should issue the Record On Demand (ROD) command to ensure that no statistical data is lost. For a 370 Model 115 or 125, the U command of the mode select display, should also be issued to save disk usage statistics on the system diskette. These commands are not valid for recording statistics on telecommunication operation; refer to the appropriate telecommunication guides for more information.

To obtain a listing of the SYSREC file, run the EREP program as described in *OS/VS, DOS/VSE, VM/370 Environmental Recording Editing and Printing (EREP) Program*. During execution of the EREP program, recording on SYSREC is suppressed.

Creating the Hard Copy File

On a system that supports a video display/keyboard console, all messages displayed on the screen and all information typed in by the operator are saved in a file on the device assigned to SYSREC. This file, called the hard copy file, can be used to obtain hard (printed) copies of the file whenever required.

You must create the hard copy file after the first IPL procedure and before you submit the first // JOB statement to DOS/VSE.

The control statements and commands needed to create the hard copy file are the same as those shown in Figure 3-3 for the SYSREC file with the exception that you specify HC=CREATE in the SET command, and the filename IJSYSCN in the DLBL job control statement. More information about creating and printing the hard copy file is given in *DOS/VSE Operating Procedures* and *DOS/VSE System Utilities*.

User-Defined Processing after IPL

At large DOS/VSE installations, it may be desirable to perform certain processing at the end of an IPL procedure. It may, for instance, be important to know who performed the procedure, whether the right system pack was mounted, and whether the correct date was entered for the new work session. Moreover, if you work with labeled data files it is important that they bear the correct creation date, so as to guarantee that data files are protected until their expiration date.

After the IPL procedure has been completed, control can be passed to a user exit routine (phase name = $SYSOPEN) that you may include for the purpose of checking system security and integrity. This routine is entered once after every IPL procedure. The DOS/VSE distribution volume contains a dummy phase $SYSOPEN in the system core image library. If you do not use the facility, that phase has no effect on your system. Conventions for writing this kind of user exit routine, together with an example, are contained in the section *Writing an IPL User Exit Routine* in *Chapter 4, Using the Facilities and Options of DOS/VSE*.

Entering RDE Data

Standard DOS/VSE support includes the reliability data extractor (RDE), and DOS/VSE requests you by a message to SYSLOG to provide a 2-character IPL reason code when the first // JOB statement after IPL is processed. The system may have been started at the beginning of normal operation or restarted because of a machine error, a program error, an operator error, etc. In addition, DOS/VSE requests you to supply a subsystem identifier, a code which identifies the device type or program type that failed. On the basis of these replies job control will build a record for SYSREC.

Before shutting down at the end of the day (or processing period), you must ensure that no environmental data is lost, by issuing the ROD command. This command also causes the RDE end-of-day record to be written on the disk assigned to SYSREC. To obtain a listing of this file, run

the EREP program as described in *OS/VS, DOS/VSE, VM/370 Environmental Recording Editing and Printing (EREP) Program.*

RDE information can be very valuable to your operations management. By replying with the exact reason code that applies in each case, you are in fact ensuring a permanent record of the reason why you had to re-IPL.

Refer to the *DOS/VSE Operating Procedures*, for more information on the RDE messages and the valid replies to them.

Allocating Address Space to the Partitions

For each partition specified in the NPARTS parameter of the SUPVR generation macro, address space must be allocated. The address space available to the partitions is all of the address space from the end of the supervisor area to the beginning of the SVA. The minimum size of that address space is 512K.

Allocation of address space to a foreground partition must be done explicitly. Space not allocated to a foreground partition belongs to the BG partition. If no allocations are made, for example immediately after IPL, then all available address space belongs to the BG partition. In this case, the BG partition has the following size:

ECPS:VSE mode:	Virtual storage size (16M default or as specified at Initial Microprogram Load)
	minus supervisor size
	minus SVA size;
370 mode:	Virtual address space size (VSIZE value of VSTAB macro)
	minus SVA size.

Through the use of the job control ALLOC command you allocate the foreground partitions. Address space allocations are in multiples of 2K. The minimum amount of address space that may be allocated to a partition (explicitly or implied) for execution in virtual mode is 128K. This 128K size includes a minimum partition GETVIS area of 48K.

If a foreground partition is defined (via the NPARTS parameter of the SUPVR generation macro), but not needed for a while, you can set its size to 0K by submitting an appropriate ALLOC command.

During certain periods of processing, the operator can modify the allocations to the individual partitions, again by using the ALLOC command. Details on the ALLOC command are given in *DOS/VSE Operating Procedures* and in section *Starting the System* in *Chapter 3*.

Allocating Processor Storage to the Partitions

Processor storage is allocated to the partitions to enable the following:

- Program execution in real mode.
- Fixing pages by means of the PFIX/PFREE macros.

When processor storage is used for running a program in real mode or for fixing pages of a program running in virtual mode (for example, VSE/POWER), the page pool is reduced by the number of page frames required for real mode execution or page fixing, respectively. Because reducing the page pool may reduce total system throughput, the use of real mode execution and PFIX/PFREE macros should be carefully considered.

Processor storage is allocated to the partitions via the ALLOCR command. For a partition's allocation to be affected the partition identifier (BG, F1, F2, ...) must be specified. The allocation is made in multiples of 2K, with 2K being the smallest allocation permissible. Absence of the partition identifier means: do not change the current allocation. An allocation of 2K allocates one page frame, 20K allocates 10 page frames etc.

Note: In 370 mode, when the ALLOCR command is issued, the system delineates real address space as well as allocating processor storage frames. In 370 mode, programs executing real execute in the real address space.

The size of a given processor storage allocation for a partition is determined either by the largest program you must run in real mode, or by the maximum number of pages a program may fix. The number of pages that can be fixed by the PFIX macro is limited by the amount of processor storage allocated to that partiton.

Given: ALLOCR BG=20K, F3=10K
with the above allocation you could PFIX 10 pages in BG (while executing in BG) or 5 pages in F3 (while executing in F3). You could not PFIX 15 pages from one program in either partition without reallocating processor storage.

Page Pool. The page pool is all processor storage beyond the resident supervisor routines. When you use the ALLOCR command you are potentially reducing the size of the page pool. The page pool is not reduced until the processor storage page frames are taken for real mode execution or for PFIX use in virtual mode. The minimum page pool size is 24K. If you allocate processor storage to partitions you must ensure that at least 24K remain unallocated. A program running in virtual mode that needs more than 6K for its I/O processing requires a corresponding increase of the minimum page pool size.

Initiating Foreground Partitions

An ASI procedure may be used to start foreground partitions by including, in the appropriate procedure, the required partition start-up statements.

In order to initiate a foreground partition, at least 128K of virtual storage must be allocated to that partition. The allocation is made after IPL with the ALLOC job control command.

Since DOS/VSE automatically determines the size of the SVA at IPL, it is recommended that you issue the MAP command prior to any virtual storage allocation. The MAP command will display the current allocations and you can determine the amount of virtual storage available for allocation to the foreground partitions.

The ALLOC command is both a job control and an attention routine command. (The attention routine is loaded when you press the Request key on the console keyboard; that routine is in control of the system when AR is displayed on SYSLOG.) When the ALLOC command is given through the attention routine it cannot decrease the size of an active partition. The initial allocation of foreground partitions decreases the size of the BG partition because all available virtual storage is allocated to BG at IPL. Since, after IPL, the BG partition is active, the ALLOC command must be given through job control.

Once virtual storage is allocated to the foregound partitions, they may be made "active" through the attention routine. Issuing the BATCH or START command, specifying a foreground partition, causes DOS/VSE to initiate that foreground partition. For example:

```
AR BATCH F1
```

causes the job control program to be loaded into the virtual storage allocated to the F1 partition.

Input may now be submitted to the F1 partition. Submitting jobs is described in the following section, *Controlling Jobs*.

Automated System Initialization (ASI)

During IPL and during the subsequent setting up of the system environment, normally the same commands, the same prompting messages and replies, the same job control information are processed.

ASI allows to place the required control information in procedures that are cataloged in the procedure library and to let DOS/VSE execute those procedures, without operator intervention, each time an IPL and a partition start-up occurs. The ASI procedures can be reused as long as your system environment remains unchanged. Thus, your effort for a total system bring-up is reduced to merely activating the initial microcode load. In exceptional situations, you may have to bypass ASI and perform a nonautomated, that is: an interactive system initialization.

The Procedure Library. Your system residence (SYSRES) file must contain the procedure library. To catalog the ASI procedures into the procedure library, use the librarian program MAINT and its CATALP function. The librarian programs are described in Section *Using the Libraries*, later in this chapter.

The Set of Procedures. ASI requires one procedure for IPL (ASI IPL procedure), and one job control procedure per partition (ASI JCL procedure) if this partition is to be started under control of ASI.

Procedure Names. ASI assumes certain default names unless you instruct it to use different names. The defaults are:

IPL:	$IPL370	(for 370 mode)
	$IPLE	(for ECPS:VSE mode)
JCL:	$BJCL370	(for 370 mode)
	$1JCL370	
	$2JCL370	
	.	
	.	
	.	
	$BJCLE	(for ECPS:VSE mode)
	$1JCLE	
	$2JCLE	
	.	
	.	
	.	

You might want to use different names. For example, the initialization of your system during the day deviates from that of the night shift: the day shift runs a 5-partition DOS/VSE (including VSE/POWER, ACF/VTAM, CICS/VS) whereas the night shift runs only simple batch jobs in 3 partitions. In this case, you might prefer to use procedure names as follows: $IPLD, $BJCLD, $1JCLD, $2JCLD, $3JCLD, $4JCLD for the day shift, and $IPLN, $BJCLN, $1JCLN, $2JCLN for the night shift.

If you catalog ASI procedures by names other than ASI's default names, be sure to delete procedures with ASI's default names if they are cataloged; ASI looks for those names first and, upon finding them, executes the pertinent procedure. When the default procedures are not present, ASI prompts the operator to specify an ASI procedure; in the above example, he may then enter $IPLD and $$JCLD, or $IPLN and $$JCLN.

When you catalog your ASI JCL procedures, you must observe the same naming rule as when you catalog a partition-related procedure. The first character must be a $. The second character identifies the partition: B for the BG-partition, 1 for the F1-partition etc. The remaining characters must be identical for all procedures belonging to one set.

ASI Master Procedure. If two or more CPU's share one SYSRES file, it may be advisable to have a separate set of procedures cataloged for each CPU by a separate set of procedure names. ASI still performs a completely

automated system initialization if you have the ASI master procedure $ASIPROC cataloged. Each record within this procedure describes the ASI procedure set to be used for a specific CPU and the processing mode of that CPU.

An ASI master procedure is also useful

- if you have only one procedure set, but want to use other than default names, or
- if you plan to use the ASI STOP facility; for example when you are still 'debugging' your ASI procedures.

The STOP facility allows you to specify, via the STOP parameter (see below), up to four different IPL commands. Upon encountering the first of a particular command type, the automatic IPL process interrupts itself and gives the operator a chance to enter or update IPL commands via SYSLOG.

To build the master procedure, submit one statement per procedure set. The statement allows you to specify the following parameters, separated by commas and terminated by a blank.

CPU=cpu-id	specifies 12 hexadecimal digits to identify the CPU on which an ASI procedure is to be run.
IPL=proc-name	specifies the ASI IPL procedure.
JCL=proc-name	specifies the name of the JCL procedure set; the name must start with $$. Default: $$JCLE in ECPS:VSE mode $$JCL370 in 370 mode.
MODE=370|E	indicates the processing mode of CPU. Default: 370.
STOP=stoplist	a list of up to four different IPL commands, in arbitrary sequence. If more than one is specified, the commands must be enclosed within parentheses and separated by a comma. The first of a specified command type that is encountered during IPL initiates an interrupt; before the command is processed, the operator may enter additional IPL commands.

The parameters may be specified in any sequence. Parameters CPU and IPL are mandatory. proc-name starts with an alphabetic character and may consist of up to eight alphameric characters.

Following is an example of how to catalog the master procedure:

```
// JOB CATALP $ASIPROC
// EXEC MAINT
   CATALP $ASIPROC
CPU=FF0713800138,IPL=IPLX,JCL=$$JCLX,STOP=(DEF,DPD)
CPU=00071380O138,IPL=IPLE,MODE=E
/+
/&
```

Contents of ASI IPL procedures

The ASI IPL procedure contains all IPL commands that you want to have executed by the IPL routines. Use the same format as in an interactive IPL.

In addition to IPL commands, you must submit a first record which specifies in

- columns 1 through 3: SYSLOG device address
- columns 4 through 20: (optionally) ,supervisor name, paging option, list option (for a description of these parameters, refer to section *Initial Program Loading* at the beginning of this chapter.)

The address you specify in columns 1 through 3 must be a DOS/VSE supported console device. Specification of an address which does not represent a DOS/VSE supported console device may produce unpredictable results. The address is meaningful only

- in IPL procedures referenced in $ASIPROC
- in procedure $IPL370 or $IPLE.

All other situations cause ASI to prompt for a procedure name from SYSLOG. This can be done only when SYSLOG has been defined via REQUEST/ENTER; the SYSLOG device address specified in the ASI procedure will be ignored then.

Following is an example of a skeleton ASI IPL procedure:

```
01F,$$A$SUPX,N,NOLOG
ADD 180,3330
ADD 04C,2540R
.
.
.
DPD UNIT=180
.
.
.
DEF SYSREC=180
SVA
```

If your page data set is allocated to multiple extents, you should place all DPD commands necessary to define the extents into the procedure. This prevents DOS/VSE from prompting the operator during IPL to define the remaining extents.

The SET command should not be part of the ASI IPL procedure. The command must be given only if the time-of-day clock is inoperative or is not set; if this is the case, the operator has to provide the actual date values.

Contents of ASI JCL procedures

ASI JCL procedures should contain all those job control commands or statements that you would normally submit during an interactive system start-up. Complete conceptional information on the use of job control commands is given in section *Controlling Jobs*, later in this chapter.

ASI Background Procedure. This procedure must contain all job control statements and commands necessary to initialize the BG partition and the system as a whole.

- ALLOC and ALLOCR commands to allocate space to the foreground partitions you intend to start.
- // OPTION STDLABEL, together with label information, to set up the system standard label subarea if it was not set up during a previous system initialization.
- // OPTION PARSTD, together with label information, to set up the background partition standard label subarea if it was not set up during a previous system initialization.
- All permanent assignments of logical units needed in the BG partition.
- The SIZE command if needed.
- // STDOPT command for the definition of standard (permanent) options.
- // JOB jobname for the initialization of RSMR recording.
- START Fn for each foreground partition to be started from this BG partition.
- STOP if the BG partition is to be spooled by VSE/POWER. The STOP command should immediately follow the START command for the VSE/POWER partition.

Note: The placement of the STOP and START commands, as given here for VSE/POWER, applies also to other permanently running programs such as VSE/ICCF or CICS/VS.

ASI Foreground Procedure. This procedure must contain job control statements and commands necessary to initialize a particular foreground partition:

- // OPTION PARSTD, followed by label information, to set up the foreground partition standard label subarea if it was not set up during a previous system initialization.
- All permanent assignments of logical units needed in the particular foreground partition.

Note that a foreground partition can be started through execution of the ASI BG-procedure or via VSE/POWER or via an attention routine START command.

Automatic Invocation of a Never Ending Program. A never ending program such as VSE/POWER or ACF/VTAM must not be started within a procedure; the procedure wouldn't come to an end while that program is

running. The procedure library is locked up for updates as long as a procedure is processed. Therefore, the program to be called must be indicated after the procedure is finished. This is done by specifying an EXEC statement in the comment portion of the end-of-procedure statement:

```
/+ // EXEC progname
```

SYSRDR or SYSIN cannot be assigned within a procedure. To cause automatic assignment of these logical units, specify the required ASSGN statement in the comment portion of the end-of-procedure statement:

```
/+ // ASSGN SYSIN,...
/+ // ASSGN SYSRDR,...
```

Only one // EXEC or one // ASSGN statement can be specified as a comment. The command form (no //) is not allowed.

Example of an ASI JCL Procedure Set

Figure 3-4 shows a skeleton example of an ASI JCL procedure set. It assumes a 3-partition system with VSE/POWER running in the F1-partition. Figure 3-5 shows the associated sequence of VSE/POWER AUTOSTART commands on SYSIPT.

```
   * ASI PROCEDURE FOR BG
   ALLOC F1=150K,F2=200K
   ALLOCR F1R=80K,F2R=24K
   // OPTION STDLABEL
   // DLBL IJSYSRS
   // EXTENT SYSRES,.............
   // ...........
   // ...........
   // OPTION PARSTD
   // DLBL IJSYS01
   // EXTENT SYS001,.............
   // ...........
   // ...........
   ASSGN SYSLNK,SYSRES
   ASSGN SYS001,SYSRES
1  // JOB ADAM
2  START F1
3  STOP
8  /+ // ASSGN SYSIN,00C,PERM
```

```
4  *  ASI PROCEDURE FOR F1
5  // OPTION PARSTD
   // DLBL IJSYSIN
   // EXTENT SYSIPT,,,,1000,20
   // .........
   // .........
6  ASSGN SYSIPT,131
   ASSGN .....
7  /+ // EXEC POWER,SIZE=AUTO
```

```
   * ASI PROCEDURE FOR F2
   // OPTION PARSTD
   // DLBL IJSYS01
   // EXTENT SYS001,.........
   // ...........
   // ...........
   ASSGN SYSLNK,130
   ASSGN SYS001,130
9  /+ // ASSGN SYSIN,00C,PERM
```

1 This // JOB statement initializes RSMR recording.

2 Activates the F1 partition where VSE/POWER is to run.

3 Deactivates the BG partition which is to be spooled by VSE/POWER.

4 When the F1 partition becomes active, the ASI JCL procedure for F1 is called automatically.

5 Label information is written to the F1 partition standard label subarea.

6 Assigns SYSIPT to a disk file in which VSE/POWER AUTOSTART statements had been recorded in an earlier run.

7 Calls VSE/POWER which starts to read the AUTOSTART statements from the SYSIPT file. VSE/POWER starts the F2 partition at which point the F2 JCL procedure is executed. VSE/POWER also reactivates the BG partition.

8 The BG partition continues; it assigns SYSIN to a spool device. The same happens

9 at the end of the F2 JCL procedure.

Figure 3-4. Example of an ASI JCL Procedure Set

```
    FORMAT=NO
    PSTART RDR,00C
    PSTART LST,00E
    PSTART PUN,00D
1   PSTART F2,2          *** F2 ***
    READER=00C
    PRINTERS=00E
    PUNCHES=00D
2   PSTART BG,0          *** BG ***
    READER=00C
    PRINTERS=00E
    PUNCHES=00D
    /*
```

1 Starts the F2 partition.

2 Reactivates the BG partition from where the VSE/POWER partition was started.

Figure 3-5. Example of VSE/POWER AUTOSTART Statements

Controlling Jobs

After DOS/VSE has been successfully started by means of the IPL program, the following messages are displayed on the console:

```
BG 1I00A READY FOR COMMUNICATIONS
BG
```

This shows that the job control program is in the background partition ready to accept input.

At this point, the job control program will accept commands submitted through the console (SYSLOG). Job control's normal input source, however, is the logical unit SYSRDR.

Job control reads from SYSRDR if, at this point, you depress the ENTER key on the console without entering any commands. Normally, SYSRDR is standardly assigned to a card reader or diskette device.

The unit of work that is submitted to DOS/VSE for execution is called a *job*. A job, and the environment in which it is to run, must be defined to the system through job control statements and commands. These job control statements and commands are processed by the job control program which DOS/VSE loads into storage automatically as required.

The job control program runs in virtual mode in any partition. It performs its functions only between jobs and job steps, and is not present in the partition while a problem program is being executed.

After each job control statement or command is read, control can be given to a user exit routine for examining and altering the input before it is processed by DOS/VSE. For a description of this facility refer to *Chapter 4, Using the Facilities and Options of DOS/VSE.*

The difference between job control statements and commands are not discussed here because there is no need for a distinction in this section. Whenever applicable, it is simply stated whether the function can be performed using statements, commands, or both. The description of the job control statements and commands in this section is limited to their use and functions; formats and characteristics of statements and commands are detailed in *DOS/VSE System Control Statements.*

This section describes how to define a job, how to relate files to a program, and how to work with cataloged procedures.

Defining a Job

The beginning and end of a job are defined by the JOB and / & (end-of-job) statements.

If you have the Access Authorization Checking service of DOS/VSE implemented, you must also submit an ID statement which specifies your user identifier together with a password. For more information about this service of DOS/VSE, see the publication *Data Security Under DOS/VSE.*

The program to be executed in a job is requested through an EXEC statement. The occurance of an EXEC statement is called a *job step*. Each job may consist of one or more job steps.

You may include as many job steps in a job as you wish. However, it is not advisable to execute, in one job, several programs that are completely independent of one another because, if one step terminates abnormally, the job control program ignores the remaining job steps up to the next / & statement. A typical example of related job steps that should form a single job are assembling, link-editing, and executing a program, where correct execution of one job step depends on successful completion of the preceding one. Figure 3-6 shows an example of a multistep job.

```
1  // JOB jobname
     .
2    .
     .
3  // EXEC PAYROLL
     .
     .
     .
3  // EXEC CHEX
     .
     .
     .
4  /&
```

1 Defines the beginning of a job. For jobname, you may specify a name of your own choosing.

2 Additional job control statements if required.

3 The two job steps. Job control is reloaded into storage at the end of each job step, enabling the reading of subsequent job control statements.

4 At the end of the CHEX program's execution job control is reloaded and reads the end of job indicator.

Figure 3-6. Control Statements Defining a Job Consisting of Two Job Steps

Following are some additional details about the job and end-of-job (/ &) statements. The EXEC statement is discussed later in this chapter.

The JOB Statement. The JOB statement indicates the beginning of control information for a job. The specified job name is stored in the communications region of the corresponding partition and is used, for example, by job accounting and to identify listings produced during the execution of the job.

If the JOB statement is omitted, DOS/VSE uses NO NAME as the job name. If the JOB statement is without a job name it is rejected by job control as an invalid statement. The JOB statement should not be omitted, as many DOS/VSE functions assume its presence. If, for example, the operator cancels a job using the attention routine CANCEL command, the job control program normally bypasses all statements on SYSRDR until encountering a / & . However, if the job in question was submitted without a JOB statement, no statements in the job stream are bypassed even though job NO NAME was canceled.

Having JOB statements with specific job names is useful when you issue the MAP command in a multiprogramming environment. The MAP command displays on SYSLOG the storage allocations for each partition, together with the name of a job that is currently active in the corresponding partition.

The JOB statement is always printed in positions 1 through 72 on SYSLST and SYSLOG; also, the time of day is printed. The JOB statement causes a skip to a new page before printing is started on SYSLST.

The End-of-Job (/&) Statement. This statement is the last one for each job (not job step). It signals the end of the input stream for the job. When job control encounters / & on SYSRDR during normal operation, the permanent assignment for SYSIPT becomes effective and SYSIPT is checked for an end-of-file condition.

If the / & statement is omitted, the next JOB statement will cause control to be transferred to the end-of-job routine to simulate the / & statement, except for abnormal job termination.

When a job terminates abnormally all statements on SYSRDR are bypassed until / & . A JOB statement preceding that / & statement does not stop this bypass operation.

When a / & statement is encountered, the job control program performs such operations as the following:

- Resets all job control options for the partition to standard: either as established by the STDOPT command, or the system default if the particular option was not set through a STDOPT command.
- Resets all system and programmer logical unit assignments for the partition to the permanent assignment established by job control commands. Logical unit assignment is discussed under *Relating Files to Your Program* later in this chapter.
- Modifies the communications region as follows:

  1. Resets the date from the DATE statement to the one specified in the SET command during IPL.
  2. Stores the job name NO NAME.
  3. Sets the user area and the UPSI byte to zero.

- Displays an end-of-job (EOJ) message on SYSLST and SYSLOG with the time and duration of the job.

- Ensures that end-of-file has been reached on SYSIPT.
- Deletes the temporary labels in the label information area on SYSRES. (See *Storing Label Information*, later in this chapter.)
- Checks whether the condense limits of any of the libraries have been reached (if library maintenance has been done in the job).

Job Streams

The job control program provides automatic job-to-job transition. In other words, an unlimited number of jobs can be submitted to the system in one batch, and job control processes one job after the other without requiring intervention by the operator. The job or jobs submitted are referred to as a *job stream* (see Figure 3-7 for an example of a payroll jobstream).

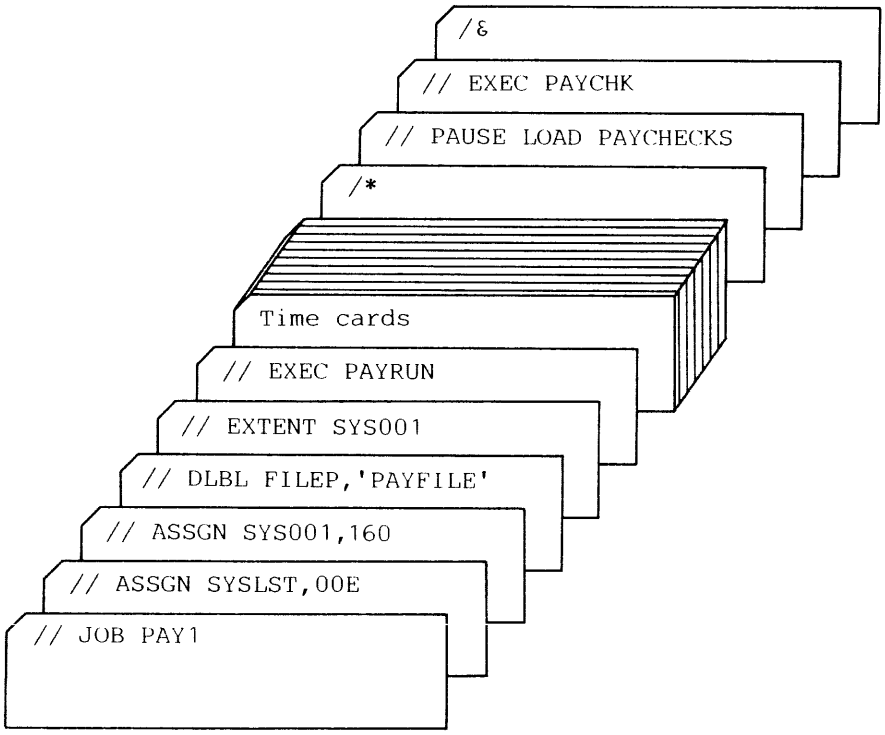


Figure 3-7. Example of a Job Stream

When setting up a job stream for a partition, you should bear in mind that all jobs will get the priority of that partition. The selection of the jobs for a particular partition in a multiprogramming system can help to improve the efficiency of your installation. For example, jobs which have a relatively low CPU usage and a relatively high rate of I/O activity, and which therefore spend most of their time waiting for the completion of I/O operations, should run in a high priority partition. Conversely, CPU-bound jobs should be in a partition with a lower priority.

The operator may interrupt the processing of a job stream in any partition to make last-minute changes to one of the jobs or to squeeze in a special rush job. He does this by using the PAUSE statement or command.

A PAUSE statement may be included anywhere among the job control statements of a job stream (see Figure 3-7). It becomes effective at the point where it was inserted; processing is suspended in the affected partition, and the operator console is unlocked for input. The PAUSE statement can contain instructions to the operator and is always displayed on SYSLOG.

The PAUSE statement may also be helpful when SYSIN is assigned to a 5424 or 5425 card reader (neither of which have an end-of-file button). Place the // PAUSE card after the last / & card; this will force control to be given to the console-keyboard, which enables the console operator to control subsequent system operation.

A PAUSE command may be entered either through the operator console (after pressing the request key), or within a job stream together with the job control statements for a job. If entered through the console to the attention routine, the command must specify the partition that is to pause (if the background partition is intended, however, no operand is required). After encountering a PAUSE command, DOS/VSE passes control to the operator (through the console) into the specified partition, at the end of the current job step (which may also be the end of the job). If that PAUSE command specifies the EOJ operand, control passes to the operator at the end of the current job, regardless of the number of steps needed to reach that point.

The macro JOBCOM allows you to do job-to-job communication. You may store information (up to 256 bytes) in one job to be passed to and retrieved by a subsequent job running in the same partition. *DOS/VSE Macro Reference* provides a detailed description of the JOBCOM macro.

Relating Files to Your Program

Most programs perform some kind of input/output operation (that is, they process files) on auxiliary storage devices. Before such files can be processed, certain information about them must be provided to DOS/VSE. This information includes:

- The address of the I/O device on which each of the files resides.
- For files on direct access storage devices (DASD), the exact location of the file on the storage medium.
- For files on DASDs, on diskettes, or on labeled magnetic tape, a description of the file, called a label, which is used for checking and protection purposes.

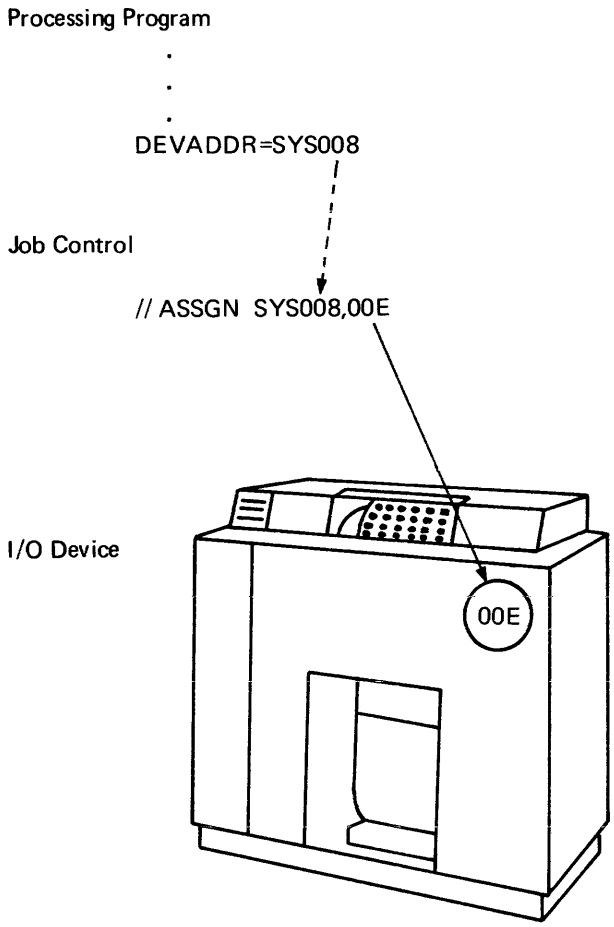
The above information, specified in job control statements, is stored in the system by the job control program for use by the DOS/VSE data management routines. How this is done is described below.

Whenever a processing program needs access to a file on auxiliary storage the program need not specify an actual device address, but only a symbolic name which refers to a logical, rather than physical, unit. Before the program is executed that logical unit must be associated with an actual device. This is done by DOS/VSE when it executes an ASSGN job control statement or command which specifies the symbolic name of the logical unit and one of the following:

- A general device class or specific device type, with or without volume serial number.
- The physical address (channel and unit number) of the I/O device.
- A list of physical addresses.
- Another logical unit.

See Figure 3-8 for an illustration of some of these combinations.

ASSGN statements may be submitted as part of ASI JCL procedures or between jobs or job steps.

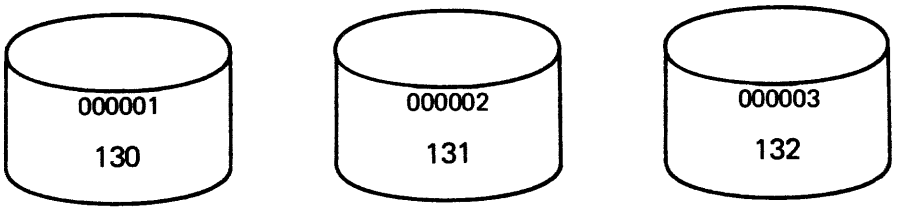


1. The logical unit specified in the processing program (via DTF or CCB or IORB) is a print file referred to by the symbolic device name SYSLST.
2. An ASSGN statement is used to associate SYSLST with the physical address 00E of a printer. This information is stored in the system by job control and can be accessed when a program is executed.

Figure 3-8. Example of Symbolic I/O Assignment (Part 1 of 2)

```
Processing Program
        .
        .
        .
      DEVADDR=SYS002

Job Control
// ASSGN SYS002,(130,131) (A)
// ASSGN SYS003,3330,VOL=000003 (B)
// ASSGN SYS004,TAPE (C)
```



(A) Device list – if drive 130 is unassigned SYS002 will be assigned to it, if it is assigned DOS/VSE tries 131.

(B) Device type – DOS/VSE searches for the device type (3330 in this case) that is available and has the volume-id 000003.

(C) Device class – DOS/VSE searches for an available tape device.

Figure 3-8. Example of Symbolic I/O Assignment (Part 2 of 2)

Logical Units

There are two types of logical units: *system logical units,* primarily used by the system control and service programs, and *programmer logical units,* primarily used by the processing programs. The following list shows the names, logical units and the I/O devices that each of these logical units can represent. In the case of disk devices, the logical unit is not assigned to the entire volume mounted on the device but only to the referenced extent(s). Refer to the section *Files on Direct Access Devices* for more information on disk files.

Logical unit name	Type of I/O device
SYSRDR	Card reader, magnetic tape unit, disk device, or diskette used as input unit for job control statements or commands.
SYSIPT	Card reader, magnetic tape unit (single volume), disk, or diskette extent used as input unit for programs.
SYSPCH	Card punch, magnetic tape unit, disk, or diskette extent used as the unit for punched output.
SYSLST	Printer, magnetic tape unit, disk, or diskette extent used as the unit for printed output.
SYSLOG	Operator console used for communication between the system and the operator.
SYSLNK	Disk extent used as input to the linkage editor.
SYSRES	System residence extent on a disk pack.
SYSCLB	Disk extent used for a private core image library.
SYSSLB	Disk extent used for a private source statement library.
SYSRLB	Disk extent used for a private relocatable library.
SYSREC	Disk extent used to store error records collected by the recovery management support recorder (RMSR) function. If a display operator console (DOC) is installed, messages to or from the operator are stored in the hard copy file, a separate SYSREC extent so that a hard copy listing of these messages can be produced. A third SYSREC extent holds the system history file.
SYSDMP	Disk extent(s) for alternate dump file(s).
SYSCAT	Disk extent used to hold the VSAM master catalog.
SYSCTL	Used by DOS/VSE.
SYSnnn	Format for coding programmer logical units which are discussed later in this section.

System Logical Units. All of the above logical unit names, except SYSnnn, represent system logical units. Of these system logical units, user-written programs may use SYSIPT and SYSRDR for input, SYSLST and SYSPCH for output, and SYSLOG for communication with the operator. All other system logical units may not be used within user-written programs (or EXTENT statements, which are discussed later in this section).

Two additional symbolic names, SYSIN and SYSOUT, are used under certain conditions:

SYSIN	*Can* be used if you want to assign SYSRDR and SYSIPT to the same card reader or magnetic tape unit. You should not assign SYSRDR and SYSIPT to the same disk or diskette extent, assign SYSIN to that extent instead.
SYSOUT	*Must* be used if you want to assign SYSPCH and SYSLST to the same magnetic tape unit. SYSOUT *cannot* be used to assign SYSPCH and SYSLST to disk or diskette because these two units must refer to separate extents.

SYSIN and SYSOUT are valid only to job control and cannot be referenced in a user-written program. Examples for the use of SYSIN and SYSOUT are given in the section *System Files on Tape, Disk, or Diskette* later in this chapter.

Programmer Logical Units. Programmer logical units may be assigned to any device installed on the system used for processing program input and output. Each partition has a minimum of 10 programmer logical units (SYS000–SYS009) and a maximum of 241 (SYS000–SYS240). The number of programmer logical units is a supervisor generation option.

Types of Device Assignments

Device assignments are either permanent or temporary, depending on the time of the assignment and the type of ASSGN statement or command used.

Permanent Device Assignments. A permanent assignment is set up between jobs or job steps any time after IPL by the ASSGN job control command (no //) or the // ASSGN job control statement with the PERM operand. It is valid until the next IPL procedure unless superseded by another ASSGN job control command. A permanent assignment can be changed for the duration of a job or job step by a // ASSGN statement or by an ASSGN command with the TEMP option.

Temporary Device Assignments. A temporary assignment is established either by a // ASSGN statement or by an ASSGN command with the TEMP option. It is valid for a single job only, unless superseded by another temporary or permanent assignment. Temporary assignments are reset to permanent by

- a / & or JOB statement, whichever occurs first, or by
- a RESET job control statement or command.

Restrictions: The type of device assignment is restricted under certain conditions:

1. If one of the system logical units SYSRDR, SYSIPT, SYSLST, or SYSPCH is assigned to a disk device or diskette, the assignment must be permanent. If SYSCLB is assigned, its assignment must also be permanent.
2. If SYSRDR and SYSIPT are to be assigned to the same disk or diskette extent, SYSIN should be assigned instead, and this assignment must be permanent.
3. SYSOUT, if used, must be a permanent assignment.
4. The SYSLOG assignment is restricted when IPL was done from either a 125D or 3277 device. You may not assign SYSLOG to a 125D if IPL was done from a 3277 and vice-versa.

Device Assignments in a Multiprogramming System

Each partition has its own set of system logical units. For example, the BG partition has a SYSRDR, SYSLST, SYSIPT etc. as do all the other generated partitions. As each partition is started, assignments must be made for the system logical units. Some assignments need be made only in one partition and are valid for all partitions. These are logical units that service the system rather than one partition. The page data set (assigned via the DPD command) and the following units fall into this category:

logical name	how assigned
SYSLOG	ASSGN job control command
SYSREC	DEF IPL command
SYSDMP	DEF IPL command
SYSCTL	automatically assigned by DOS/VSE
SYSRES	Entering disk address at IPL
SYSCAT	DEF IPL command

All of the other system logical unit assignments must be made for each individual partition.

Each partition also has its own set of programmer logical units (SYS000 through SYSnnn) where nnn is the number of programmer logical units specified for the partition minus 1.

You must make assignments of the programmer logical units as needed by the programs running in each partition. Certain IBM supplied programs require specific programmer logical unit assignments. For example the linkage editor requires SYS001 and the assembler requires SYS001, SYS002, and SYS003.

Sharing Assignments. Within the same partition, different logical units may be assigned to the same physical device. For example:

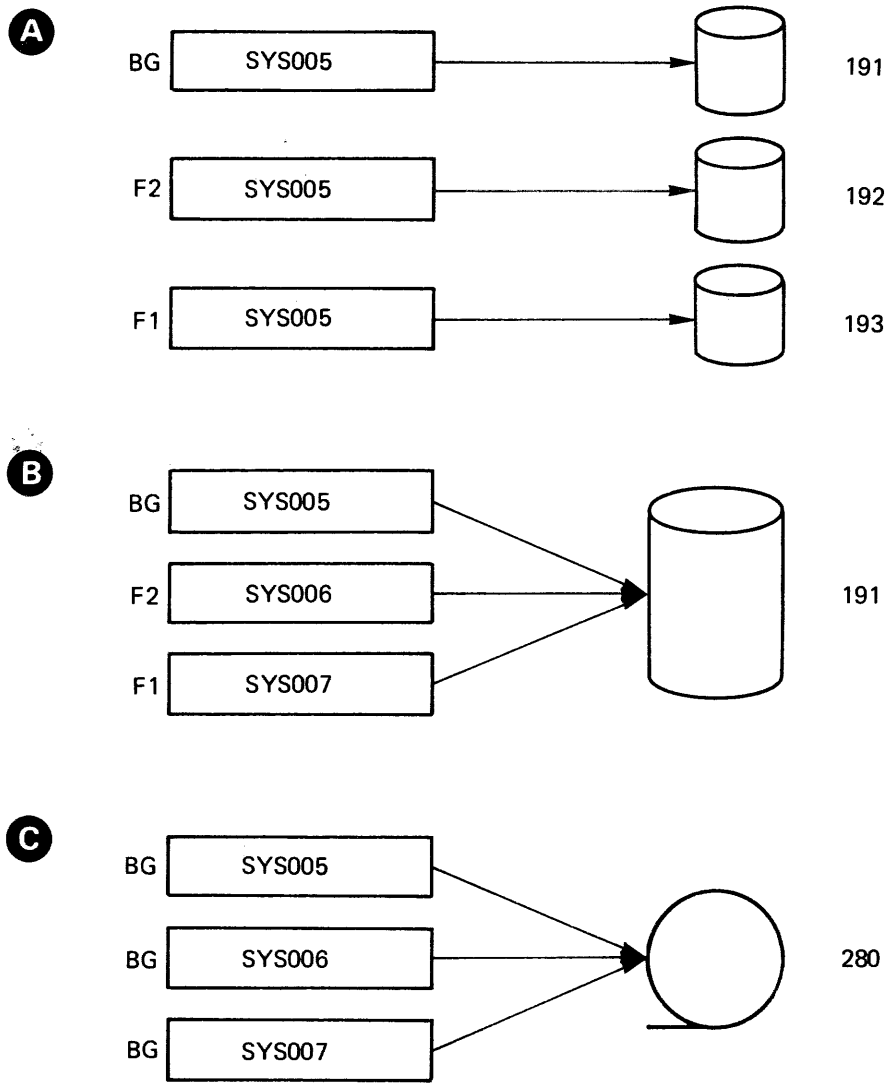
```
// ASSGN SYSLST,00E
// ASSGN SYS007,00E
```

Both logical names SYSLST and SYS007 are assigned to the device at address 00E.

Normally it is not possible to share physical devices (except DASD) between partitions. For example if you have a tape drive assigned to the BG partition, but not used by that partition, you must first unassign it in BG before attempting to assign it in F2. If, however, you use a spooling package, such as the licensed program VSE/POWER, you can share unit record devices (card reader, card punch, for example) and diskette between partitions (see the licensed program VSE/POWER documentation for more details).

With direct access devices this problem does not exist because each extent on a disk can be thought of as a separate device. It is not possible, however, to share a diskette between partitions.

Figure 3-9 illustrates possible device assignments.



A Each partition has its own set of programmer logical units.

B Each assignment must be for a separate extent on the disk for this example to be valid. Extents are discussed under *Processing of File Labels* in this chapter.

C These assignments allow access to the tape volume by three different logical unit names. No assignments to this tape are valid from a partition other than BG at this time.

Figure 3-9. Possible Device Assignments

Figure 3-10 shows the logical units needed for an assembly. The illustration shows that the ASSGN statements must always precede the EXEC statement of the job step for which they are to be effective. (The device assignments for compilers are similar to the device assignments shown in this assembler example; any variations are documented in the applicable programmer's guides.)

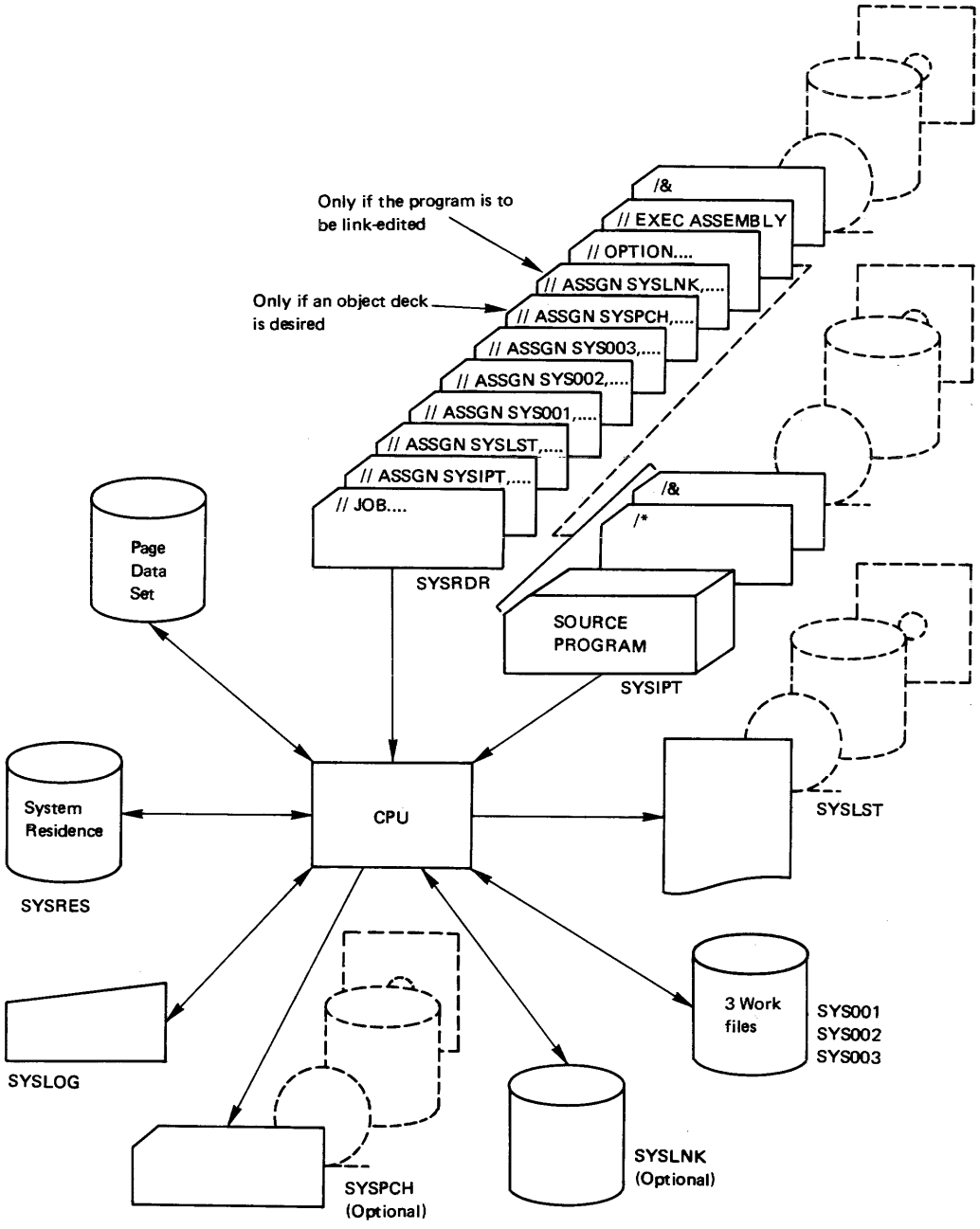


Figure 3-10. Device Assignments Required for an Assembly

The following summarizes the functions of the job control ASSGN statement (or command). Also included are statements (commands) that can be used with logical unit assignments.

The ASSGN Statement/Command. The ASSGN statement or command is used to connect a logical I/O unit to a general device class, a specific device type, a physical device or a list of physical devices, or another logical unit. An ASSGN statement or command can also be used:

- to specify a temporary or permanent assignment.
- to specify a volume serial number for a tape, disk, or diskette.
- to specify that a disk is shareable by more than one partition or logical unit.
- to unassign a logical unit to free it for assignment to another partition.
- to ignore the assignment of a logical unit, that is, program references to the logical unit are ignored (useful in testing and certain rerun situations).
- to specify an alternate tape unit to be used when the capacity of the original is reached.

The assignment routines check the operands of the ASSGN statement/ command for the relationship between the physical device, the logical unit, the type of assignment (permanent or temporary), etc. The following list summarizes the most pertinent items to remember when making assignments:

- Assignments are effective only for the partition in which they are issued.
- Apart from the operator console, no physical device except DASD can be assigned to more than one active partition at the same time.
- All system input and output file assignments to disk or diskette must be permanent.
- SYSIN must be assigned if both SYSRDR and SYSIPT are to be assigned to the same extent.
- SYSOUT cannot be assigned to disk or diskette; it must be a permanent assignment if assigned to tape.
- SYSLNK must be assigned before issuing the LINK or CATAL option in an OPTION statement; otherwise, the option is ignored and the message 'PLEASE ASSIGN SYSLNK' is issued to the operator.
- Before a tape unit is assigned to SYSLST, SYSPCH, or SYSOUT, all previous assignments to this tape unit must be permanently unassigned. This may be done by using a DVCDN command as discussed below.
- The assignment of SYSLOG cannot be changed while a foreground partition is active.

- SYSRES, SYSCAT, SYSREC, SYSDMP and the page data set can never be assigned by an ASSGN statement or command. An IPL is required to change these assignments.

The RESET Statement/Command. The RESET statement or command can be used to reset temporary assignments of a partition to permanent. With one RESET statement or command you can reset

- all logical units.
- all system logical units.
- all programmer logical units.
- one specific system or programmer logical unit.

The LISTIO Statement/Command. With the LISTIO statement or command you can obtain a listing of the current status of the I/O assignments in your system. This may be done for all devices or individual devices as required. If the LISTIO command is used (no //), the output goes to SYSLOG, otherwise the output is on SYSLST.

The DVCDN Command. The DVCDN (device down) command informs the system that a device is no longer physically available for system operations. This command releases all logical assignments to the device.

When the device becomes available again for system operations, a DVCUP (device up) command must be given and new assignments made, before the device may be used.

The DVCUP Command. The DVCUP (device up) command informs the system that a device is available for system operations after it has been down.

Processing of File Labels

As shown above, DOS/VSE relates physical devices to logical names, used in programs, via the ASSGN job control statement (or command). Certain device types (magnetic tape, disk, and diskette) have removable volumes. It is important to ensure that the volume(s) containing the file(s) to be processed are present on the assigned device(s). Magnetic tape, disk and diskette files are identified through file *labels* which are processed by the DOS/VSE data management routines. Magnetic tape file labels are optional, though desirable for reasons of data integrity. Disk and diskette file labels are required.

DOS/VSE writes file labels when a file is created based on label information submitted through job control statements.

To write a file label on magnetic tape, DOS/VSE uses the // TLBL statement. This label is written by DOS/VSE immediately preceding the associated file.

To write a file label on disk or on diskette, DOS/VSE uses the // DLBL and // EXTENT statements. The label is written into the volume table of contents (VTOC), and a utility program, LVTOC, is available to

list all labels included in this VTOC. Details on the DLBL and EXTENT statements are given in *DOS/VSE System Control Statements*. When a labeled file is to be processed, you must submit the required // TLBL, // DLBL and // EXTENT statements, so that DOS/VSE can perform the desired label checking on your existing file. Figure 3-11 shows the relationship of label information that you provide by the above mentioned statements to file labels and programs. For a detailed discussion of label processing, refer to *DOS/VSE DASD Labels* and *DOS/VSE Tape Labels*.

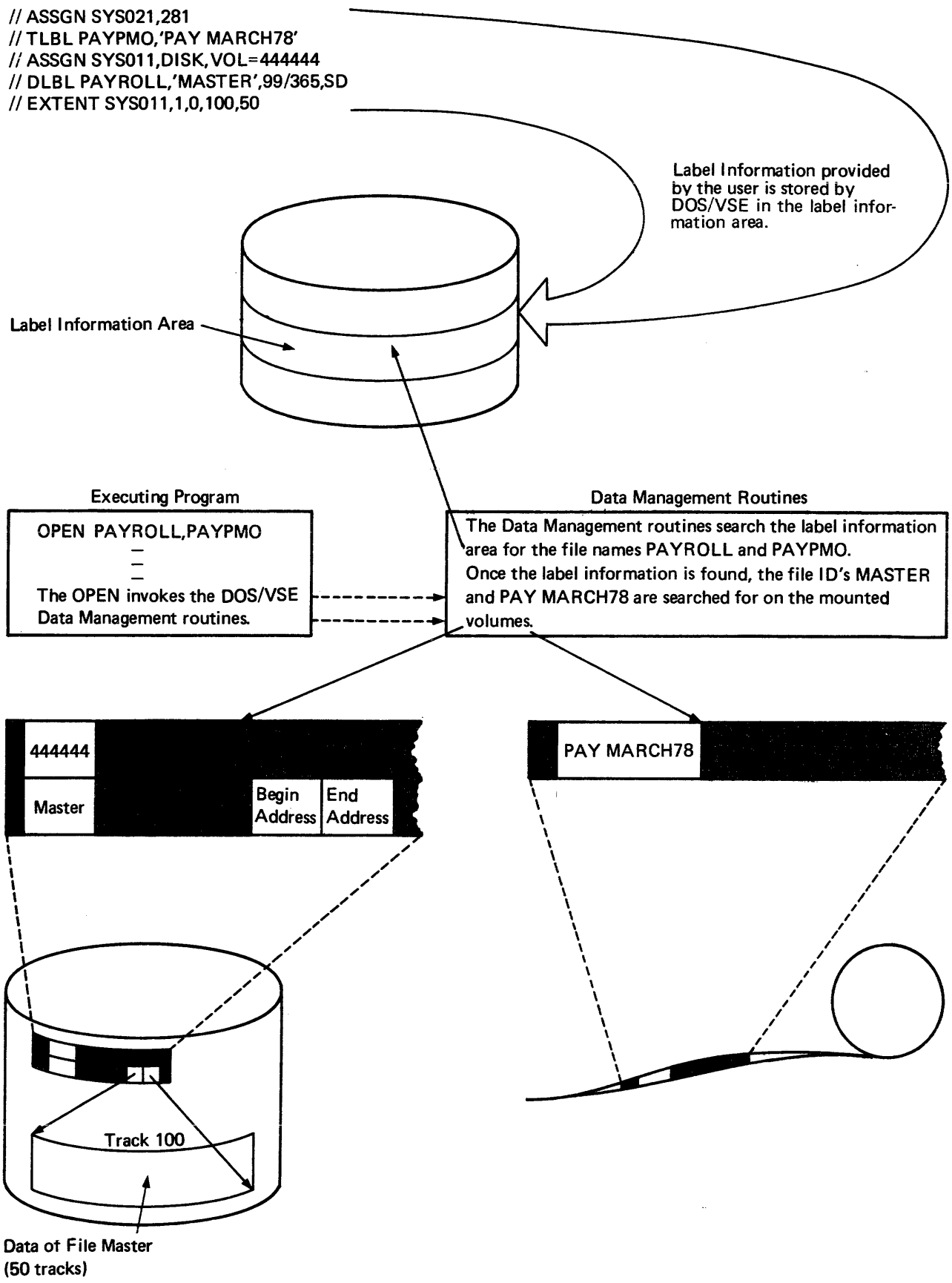


Figure 3-11. File Label Processing

The // TLBL, // DLBL, and // EXTENT job control statements may be submitted with each execution of a given program that processes labeled files. Job control temporarily stores these statements in the label information area. A recommended alternative for frequently accessed files is to permanently store the label information in the label information area. The section *Storing Label Information* later in this chapter describes how to permanently store label information.

When the program that processes the file is executed, the data management routines of DOS/VSE access the label information

- to write the appropriate labels onto the storage volume, and to check that no unexpired files are overwritten, if the file is to be created, or
- if an existing file is to be processed, to check the contents of the label information area against the label(s) of the file to ensure, for example that the correct volume is mounted.

The first two parameters of both the // TLBL and // DLBL statements are the same:

```
// TLBL filename,'file-id'
// DLBL filename,'file-id'
```

The filename is *not* part of the file label. You code a filename in your program to identify your file.

- In assembler language it is the DTF (Define The File) name.
- In DOS/VS RPG II it is the FILENAME.
- In DOS/VS COBOL it is the name specified in the SELECT clause.
- In PL/I it is the identifier (with the FILE attribute) in the DECLARE statement.
- In FORTRAN it is the file name associated with the data set reference number.

The filename from your program is used as a search argument by the data management routines in searching for label information in the label information area. Accordingly you must code a matching filename in your // TLBL or // DLBL statements.

The file-id *is* part of the file label. After the DLBL or TLBL statements are located (based on filename), the file-id is used to:

- create a label for an output file.
- locate and check the labels of an input file.

Example of label checking:

```
// JOB UPDATE
// ASSGN SYS007,00C
// ASSGN SYS008,280
*  PLEASE MOUNT CURRENT ACCOUNTS RECEIVABLE TAPE
// PAUSE
// TLBL ACCT,'ACCTS.REC.FILE'
// EXEC UPDATE
data cards
/*
// MTC REW,SYS008
// ASSGN SYS010,280
// ASSGN SYS007,00E
// TLBL ARFILE,'ACCTS.REC.FILE'
// EXEC ARREPORT
/&
```

The two programs UPDATE and ARREPORT access the same file 'ACCTS.REC.FILE'. The two programs happen to use different file names and different programmer logical units.

UPDATE opens a file named ACCT on logical unit SYS008 and ARREPORT opens a file named ARFILE on SYS010. In both cases the file accessed is 'ACCTS.REC.FILE'. If the two programs had used the same file name and programmer logical units, one ASSGN statement and one // TLBL statement permanently stored in the label information area would suffice.

Label Information for Files on Diskette Devices

After you have informed the system, via the ASSGN statement or command, on which physical device the file is to reside, you must supply the following information to allow the creation and checking of diskette labels:

1. A description of the characteristics of the file. You specify this in the DLBL job control statement.
2. The volume(s) the file is contained on. You specify this in one or more EXTENT job control statements.

The label information you supply in the DLBL job control statement may include the following:

- The name of the file. This name must be identical to the corresponding file name specified in your program. For programs written in assembler language, this would be the name of the DTF.
- An identification of the file. This name is the one contained in the file label on the diskette. It is associated with the file name via the DLBL statement.
- The expiration date of the file.
- The type of access method used to process the file; always coded as DU.

A diskette file consists of a data area on one or more volumes; each volume contains only one data area for a particular file. For each of these data areas, called extents, you must supply the following information on an EXTENT job control statement:

- The symbolic name of the device on which the volume containing the file is mounted.
- The serial number of the volume.
- The type of extent; always coded as 1.

In the following example, the program CREATE creates a diskette (DU) file named SALES that has a file-id of MONTHLY and is to be retained for 30 days. The file comprises up to three diskettes. The diskettes have the volume serial numbers 111111, 111112, and 111113, and are mounted on the drive assigned to the symbolic device named SYS005.

```
// JOB EXAMPLE
// ASSGN SYS005,060
// DLBL SALES,'MONTHLY',30,DU
// EXTENT SYS005,111111,1
// EXTENT SYS005,111112,1
// EXTENT SYS005,111113,1
// EXEC CREATE
/&
```

The job control program checks the DLBL and EXTENT statements for correctness and stores the supplied information in the label information area on SYSRES for the duration of the job (see *Storing Label Information* later in this chapter).

Label Information for Files on Direct Access Devices

After you have informed the system, via the ASSGN job control statement or command, which volume or physical device you want, you must supply the following information to allow the creation and checking of DASD labels:

1. A description of the characteristics of the file. You specify this in the DLBL job control statement.
2. The exact location of the file on the storage medium. You specify this in one or more EXTENT job control statements.
3. For non-sequential DASD files, the amount of storage in the partition to be reserved for label processing. Specification of this value is not required if you have VSE/Advanced Functions installed. You specify that value in the LBLTYP job control statement. Since this information is needed by the linkage editor, the LBLTYP statement is discussed in *Linking Programs* later in this chapter.

The label information you supply in the DLBL job control statement may include the following:

- The name of the file. This name must be identical to the corresponding file name specified in your program. For programs written in assembler language this would be the name of the DTF.

- An identification of the file which may include generation and version numbers of the file. This name is the one contained in the file label on the storage device. It is associated with the file name via the DLBL statement.
- The expiration date of the file.
- The type of access method used to process the file.
- An indication of whether or not a data secured file is to be created.
- The blocksize to be used for this file on an IBM 3330-11 or 3350 device.
- The control interval size (CISIZE) if your file is a sequential disk file and resides on an FBA device.

A DASD file can consist of one or more data areas on one or more volumes. For each of these data areas, called extents, you must supply the following information on an EXTENT job control statement:

- The symbolic name of the device on which the volume containing the file extent is mounted.
- The serial number of this volume.
- The type of the extent. An indexed sequential file, for instance, can consist of data areas, index areas, and overflow areas. For each of these areas an extent must be defined, and its type (data, index, or overflow) must be specified.
- The sequence number of the extent within the file.
- For CKD devices:

  The number of the track (relative to zero) on which the file extent begins.

  The amount of space (in tracks) the file occupies.
- For FBA devices:

  The block number on which the file extent begins.

  The amount of space (in blocks) the file occupies.

Examples for Submitting Label Information for DASD Files. Here are a number of examples of how to code the job control statements required to create or access the labels for the various types and organizations of DASD files. It is helpful if you are familiar with the formats of the DLBL and EXTENT job control statements as described in *DOS/VSE System Control Statements.* Detailed information on the possible organizations and access methods for DASD files is given in *DOS/VSE Data Management Concepts.*

Sequentially Organized Disk Files (Single Drive, Single Volume). In the following example, the program CREATE creates a sequential disk (SD) file named SALES that is to be retained until the end of 1979. The file comprises one extent of 190 tracks on a CKD device, starting on relative track number 1320. The disk pack has the volume serial number 111111 and is mounted on the drive assigned to the symbolic device name SYS005:

```
// JOB EXAMPLE
// ASSGN SYS005,DISK,VOL=111111,SHR
// DLBL SALES,'ANNUAL SALES RECORDS',79/365,SD
// EXTENT SYS005,111111,1,0,1320,190
// EXEC CREATE
/&
```

The job control program checks the DLBL and EXTENT statements for correctness and stores the supplied information in the label information area on SYSRES for the duration of the job or job step.

Sequentially Organized Disk Files (Single Drive, Multivolume). Assume that a program PROG100 needs a sequential disk file located on three different disk packs that are to be mounted successively on the same device (SYS005). The file consists of four extents on an FBA device: two on the pack with serial number 000020, one on pack 000100, and one on pack 000006. The following job stream shows the label statements required:

```
   // JOB SAMLABEL
   // ASSGN SYS005,DISK,VOL=000020,SHR
1  // DLBL FILNAME,'FILE ID',99/365,SD
   // EXTENT SYS005,000020,1,0,10,2010
   // EXTENT SYS005,000020,1,1,4000,1510
   // EXTENT SYS005,000100,1,2,64,1300
   // EXTENT SYS005,000006,1,3,50,636
2  // EXEC PROG100
3  /&
```

1 Only one DLBL statement is required. For each extent one EXTENT statement must be supplied in the sequence in which the extents are processed.

2 Logical IOCS in PROG100 opens the first extent using the file name and file ID in the DLBL statement, and the logical unit and volume serial number in the first EXTENT statement to locate the actual label on the disk pack. After PROG100 has processed the first extent, logical IOCS opens the second extent, based on the extent sequence number.

For the third extent, volume serial number 000100 is specified while the volume currently mounted on SYS005 has the number 000020. The OPEN routine of LIOCS notifies the operator of this discrepancy, and the operator can mount the correct volume, at which time the OPEN routine regains control. The same is true for the fourth extent.

3 The /& statement causes the label information stored in the label information area to be cleared. Thus, if the next job requires the same file, the label statements must be resubmitted (see *Storing Label Information* later in this chapter).

Sequentially Organized Disk Files (Multiple Drives). This example has the same requirements as the preceding 'Single Drive' example except that the three volumes are mounted on three different drives. The required job control statements are as follows:

```
   // JOB SAMLABEL
   // ASSGN SYS005,DISK,VOL=000020,SHR
   // ASSGN SYS006,DISK,VOL=000100,SHR
   // ASSGN SYS007,DISK,VOL=000006,SHR
1  // DLBL FILNAME,'FILE ID',99/365,SD
   // EXTENT SYS005,000020,1,0,10,2010
   // EXTENT SYS005,000020,1,1,4000,1510
   // EXTENT SYS006,000100,1,2,64,1300
   // EXTENT SYS007,000006,1,3,50,636
2  // EXEC PROG100
   /&
```

1 All label statements submitted are identical to the 'Single Drive' example except for SYSnnn in the EXTENT statements.

2 Logical IOCS opens each extent in the same way as described in the 'Single Drive' example except that processing does not stop for removal and mounting of packs, because enough devices are online to contain the file. A combination of this and the 'Single Drive' example could be used to reduce handling time without excessively increasing the total drive requirements.

DA Files. The program PROG101 processes a direct access file consisting of four extents contained on three CKD disk packs. The three packs must be ready at the same time. The following job stream shows the label statements required to process the file:

```
   // JOB DALABEL
   // ASSGN SYS005,DISK,VOL=000065,SHR
   // ASSGN SYS006,DISK,VOL=000025,SHR
   // ASSGN SYS007,DISK,VOL=000002,SHR
1  // DLBL FILNAME,'FILE ID',99/365,DA
   // EXTENT SYS005,000065,1,0,1320,190
   // EXTENT SYS005,000065,1,1,80,740
   // EXTENT SYS006,000025,1,2,50,906
   // EXTENT SYS007,000002,1,3,1275,64
   // EXEC PROG101
   /&
```

1 The label statements follow the same pattern as for sequential files (described in the preceding examples) except that the DLBL statement must specify DA to indicate direct access.

Label Information for Files on Magnetic Tape

Files on magnetic tape can be processed with or without labels. For tape files with IBM standard labels, the label information must be submitted through the TLBL job control statement. (A tape file can also have standard-user or non-standard labels; for these labels no job control statements are required. More information on tape labels is given in *DOS/VSE Data Management Concepts*).

The standard label information submitted in the TLBL statement may include the following:

- The name of the file. This name must be identical to the corresponding filename (DTF name) specified in your program.
- An identification of the file.
- Creation date for input and expiration date (or retention period) for output files.

- The volume serial number of the tape reel that contains the file.
- For files that extend over more than one volume, the sequence number of the volume.
- For volumes that contain more than one file, sequence number of the file.
- The version and modification number of the file.

When a program that processes tape files with standard labels is to be link-edited, you must supply a LBLTYP job control statement to define the amount of storage required in the partition for label processing (see also *Linking Programs* later in this chapter). Supplying this information is not required if you have VSE/Advanced Functions installed.

As with DASD files, the label information you supply in the TLBL job control statement is checked and stored in the label information area on SYSRES (see *Storing Label Information*, below).

Storing Label Information

Job control stores label information in the label information area. The label information is stored temporarily (for the duration of one job or job step) or permanently.

As label information is submitted, DOS/VSE acquires a portion of the label information area which is referred to as a label subarea.

The minimum size of a label subarea is one track for a CKD device and 2K for an FBA device, the maximum size is the entire label information area. There are three types of label subareas:

- partition temporary subarea
- partition standard subarea
- system standard subarea

Label information stored in either of the two types of *partition* subareas may be accessed only from the partition in which the label information was originally submitted. Label information stored in the *system* subarea may be accessed from all partitions. The type of subarea used is controlled by the following three options of the OPTION job control statement:

Default → USRLABEL causes all DASD, diskette, and tape label information to be stored temporarily for one job or job step. Label information submitted between job steps overlays the label information from the former job step. The label information is written to a partition temporary subarea (one per partition) and is accessible only by the partition in which it was submitted. It is a good idea to include all TLBL, DLBL, and EXTENT statements in the first step of a job (preceding the // EXEC statement). If no option is specified, or if the OPTION statement is omitted, USRLABEL is assumed.

PARSTD causes all DASD, diskette, and tape label information to be stored permanently for all subsequent jobs. The label

information is written to a partition standard subarea (one per partition) and is accessible only by the partition in which it was submitted.

STDLABEL causes all DASD, diskette, and tape label information to be stored permanently for all subsequent jobs. The label information is written to *the* system standard subarea and is accessible by all partitions but can only be submitted in the background partition. This ensures that the system standard label information is not updated simultaneously by two partitions. Logical unit numbers contained in the submitted label information must not be greater than the highest logical unit number specified for background at system generation.

Note: When SYSRES is on an FBA disk device, DOS/VSE blocks user-supplied label information before writing that information to disk. Therefore, you should terminate your // OPTION PARSTD or // OPTION STDLABEL job stream with a // OPTION USRLABEL statement. This ensures that all label information is actually written to the label information area as permanent partition or system standard labels. Labels in the system standard subarea are accessible from other partitions only after they have been written completely. The OPTION statement with USRLABEL specified indicates to DOS/VSE that no further partition or system standard labels will follow. The same effect is accomplished by a / &, // JOB, or // EXEC statement.

A partition can have only *one* temporary and *one* standard subarea at any point in time. As the subareas are variable in size it is possible that disk space is not available in the label information area when DOS/VSE attempts to write label information. When this occurs a message will be displayed on the console stating that the label area is exhausted. To clear a subarea (in order to run the current job), you can do one of the following:

- Submit a / & in another partition to clear that partition's temporary subarea.
- Submit a // OPTION PARSTD followed by a / & in any partition to clear that partition's standard subarea.

Do not clear the system standard subarea. If you find that the system standard subarea is using more disk space than you want, reorganize your label information area. For example if you have an application that always runs in the same partition (such as the licensed program VSE/POWER) the labels for that application should be put on that partition's standard label subarea, not the system standard subarea.

During program execution, the DOS/VSE data management routines search the label information area in the following sequence:

(1) user label information (partition temporary subarea)

(2) partition standard information (partition standard subarea)

(3) system standard information (system standard subarea).

It is important to distinguish between the conditions under which a label *option* remains in effect and the conditions that govern the retention of the

label *data* in the label information area. For example, the label data submitted following an OPTION statement with the PARSTD option is retained for all subsequent jobs until overwritten by another PARSTD option, but the PARSTD option is canceled at the end of the job or job step in which it was specified. This is shown in the summary of label options in Figure 3-12.

Option in search sequence	Type of label information	Option in effect until	Label information retained	For
USRLABEL[1]	temporary	STDLABEL or PARSTD is specified.	for one job. The / & statement causes the temporary label area to be cleared.[4]	the partition in which the option was specified.
PARSTD	permanent	a) end of job step b) end of job c) USRLABEL or STDLABEL is specified.[5]	for all subsequent jobs until another PARSTD option is used.[2]	the partition in which the option was specified.
STDLABEL	permanent	a) end of job step b) end of job c) USRLABEL or PARSTD is specified.[5]	for all subsequent jobs until another STDLABEL option is used.[2]	all partitions.[3]

[1] If no option is given or if the OPTION statement is omitted, USRLABEL is assumed.

[2] All label information submitted following a PARSTD or STDLABEL option is written at the beginning of the label area thus destroying any previously stored information. Therefore, if you want to add label data for another file, all previously stored label information that is to be kept must be resubmitted.

[3] Label information stored with the STDLABEL option is available to all partitions but can be submitted only through the background partition.

[4] Additional label information from a subsequent job step will overlay previous label information.

[5] It is recommended that a USRLABEL option be submitted following the PARSTD or STDLABEL job stream when SYSRES is on an FBA device.

Figure 3-12. Summary of Label Option Functions

By permanently storing the label information for a disk file in the label information area, DOS/VSE relates that file to the type of the device which is assigned to the pertinent logical unit when this file is processed for the first time. A later attempt to use this label information for the same file (and extent) on a different device type causes DOS/VSE to cancel the job. If a different device type has to be used for this file, DOS/VSE requires that the label statements be resubmitted and the pertinent logical unit assigned to the device of the new type.

Remember that, when adding to or altering permanently stored label information, all of the partition's (or system's) permanently stored label information must be resubmitted along with the updates. Stored label information may be displayed using program LSERV as follows:

```
// JOB
// EXEC LSERV
/*
/&
```

Controlling Magnetic Tape

The MTC job control statement or command controls certain magnetic tape operations, for example, file positioning. Files on magnetic tape are almost invariably processed sequentially. This means, for example, that if you have five files on one tape reel and you want to process the last one, you have to read four files before you can access the one you need. You can, however, instruct the job control program to position the tape at a particular file.

The MTC job control statement or command controls operations such as:

- Spacing the tape backward or forward to the required file.
- Spacing the tape backward or forward a specified number of records.
- Rewinding the tape to the beginning.
- Writing a tapemark to indicate the end of a file.

In the following example, program PROGA creates a labeled tape file named RATES on tape volume 222222. At the end of the first job step, an MTC job control statement is used to rewind (REW) the tape to the beginning of the tape volume so that the newly created file can be processed by PROGB.

```
// JOB TAPE
// ASSGN SYS004,TAPE,VOL=222222
// TLBL RATES,'MASTER',75/365,222222
// EXEC PROGA
// MTC REW,SYS004
// EXEC PROGB
/&
```

Controlling Printed Output

Most of the DOS/VSE supported printers use a forms control buffer (FCB) to control the length of forms skips. In addition, printers may be equipped with the universal character set feature, which is controlled by a universal character set buffer (UCB). Examples of printers equipped with these buffers are the 3203 and 3211 printers.

The buffers of these printers must be loaded during or immediately after IPL, and they may have to be reloaded later between job steps or, occasionally, while a job step using the printer is being executed.

The following methods for loading the buffers are available:

To load the FCB

- Automatic loading during IPL
- Using the SYSBUFLD program between job steps or immediately after IPL
- Using the LFCB command
- Using the LFCB macro in the problem program
- Using the FCB parameter in the VSE/POWER * $$ LST statement.

To load the UCB

- Automatic loading during IPL (applies to PRT1 and 5203U printers)
- Using the SYSBUFLD program between job steps or immediately after IPL
- Using the LUCB command
- Using the UCS command (applies only to a 1403 UCS printer).

The method of loading the buffers by using the SYSBUFLD program offers the advantage that hardly any operator activity is involved; on the other hand, loading the buffers by using the LFCB or LUCB command does not require the operator to wait for a partition to finish processing.

When the contents of an FCB or a UCB are replaced by a new buffer image, the system uses this new image to control printed output until the buffer is reloaded (or until the next IPL). None of the above methods provides automatic resetting of the buffer load to the original contents. It may be necessary to reset the buffer to the original contents before taking a storage dump, to ensure that the dump is printed in the correct format, without any part of it being left out.

Details on how to load the FCB and UCB are contained in *DOS/VSE System Control Statements*.

The 3800 Printing Subsystem. The 3800 Printing Subsystem is a noimpact, high-speed, general-purpose system printer that uses an electrophotographic technique with a low-powered laser to print output. It provides more features than current impact printers.

The following methods of controlling the 3800 are available:

- The SETPRT job control statement or command, which allows you to set the 3800 with user-specified control values. These values are reset at the end of the current job to the installation's default control values as specified in the SETDF operator command, or to the hardware defaults if SETDF is not specified.

- The SETDF operator command, which allows the operator to set and/or reset default control values for the 3800. A SETDF command can set default control values for the following:
  - One character arrangement table
  - The forms control buffer
  - The copy modification phase
  - The paper forms identifier
  - The forms overlay name
  - Bursting and trimming or continuous forms stacking
  - The setting of all hardware defaults with one command.
- The SETPRT macro instruction, which is generally invoked via the preceding statements but can also be used directly by the programmer to initialize or dynamically change the setup of the 3800.

For information on available techniques for controlling the 3800, see *DOS/VSE IBM 3800 Printing Subsystem Programmer's Guide*.

Executing a Program

After you have properly defined the I/O requirements of your program to the system you can instruct job control to prepare your program for execution. How this is done and how the supplied information is processed is described in the following section.

Assembling/Compiling, Link-Editing, and Executing a Program

In DOS/VSE, three processing steps are necessary to obtain results from a problem program once the source program has been written:

1. Assembly or compiling of the source program into an object module. (Object modules are discussed in section *Linking Programs* later in this chapter.)
2. Link-editing of the object module to form an executable program phase.
3. Execution of the program phase.

Each of these steps is initiated by the job control program in response to an EXEC job control statement. The EXEC statement must be the last of the job control statements submitted for any one job step. Figure 3-13 shows an example of the job control statements needed to assemble, link-edit, and execute a source program.

```
   // JOB EXECUTE
1  // OPTION LINK
2  // EXEC ASSEMBLY
3  // EXEC LNKEDT
4  // EXEC
   /&
```

1 To link-edit a program, the LINK option must be set ON.

2 The assembler is fetched from the core image library and starts execution.

3 The linkage editor is fetched from the core image library and starts execution.

4 When an EXEC statement without a program name is encountered, the program last stored (if stored within the same job) in a core image library by the linkage editor is fetched for execution.

Figure 3-13. Job Control Statements to Assemble, Link-Edit, and Execute a Program in one Job

Instead of submitting three EXEC statements, you may invoke all three steps by one EXEC statement. Specifying the GO parameter in the statement which invokes the assembler (compiler) causes the linkage editor and your executable program to be invoked automatically once the assembly (compilation) is finished. Only the source program and any additional data required by your program must be submitted.

Language translators read their input from SYSIPT. If SYSRDR and SYSIPT are assigned to the same device, the source statements of your program must follow the corresponding EXEC job control statement. In this example, the assembler language statements would have to follow the // EXEC ASSEMBLY statement. The end of the input data submitted for one program must be indicated by a /* (end-of-data) statement. The /* statement is not processed by job control; it is read by the logical IOCS routines of DOS/VSE. (Note: For an input file on an IBM 5424 MFCU, the /* card must be followed by a blank card.) The placement of input data and the /* statement is shown in Figure 3-14.

```
// JOB INPUT
// OPTION LINK
// EXEC ASSEMBLY
    .
    .
source program
    .
    .
/*
// EXEC LNKEDT
// EXEC
    .
    .
input data for user program
    .
    .
/*
/&
```

Figure 3-14. Submitting Input Data on SYSIPT

How the job shown in Figure 3-14 is processed by the system is illustrated in Figure 3-15. The numbers to the left of the subsequent paragraphs refer to the encircled numbers in that illustration. The inclusion of SYSIPT data in job streams in the procedure library is described in the section *SYSIPT Data in Cataloged Procedures.*

1 Job control reads the JOB statement and stores the job name in the supervisor. Other functions of the JOB statement are described under *Defining a Job,* earlier in this chapter.

2 Job control reads the OPTION statement with the LINK option and sets the LINK bit in the supervisor. This indicates

   a) to the assembler, that the assembled object module is to be written onto SYSLNK,

   b) to job control that link editing is allowed in this job,

   c) to the linkage editor, that the executable program is to be stored in the core image library only temporarily for execution in the same job.

3 On encountering the // EXEC ASSEMBLY statement, job control transfers control to the supervisor passing it the name of the assembler program.

4 The supervisor loads the assembler into the partition, replacing job control.

5 The assembler reads the source program, assembles it, and stores the object module on SYSLNK (not shown).

6 The assembler transfers control to the supervisor.

7 The supervisor loads job control into storage, replacing the assembler.

8 Job control reads the // EXEC LNKEDT statement, as well as any preceding linkage editor statements, and transfers control to the supervisor, passing it the name of the linkage editor.

9 The supervisor loads the linkage editor into storage, replacing job control.

10 The linkage editor reads the object module from SYSLNK and link-edits it.

11 The linkage editor stores the executable program in the core image library.

12 The linkage editor transfers control to the supervisor.

13 The supervisor loads job control into storage.

14 Job control reads an EXEC statement without a program name and transfers control to the supervisor.

15 The supervisor loads the program last stored in the core image library by the linkage editor replacing job control.

16 The user program is executed. It reads and processes the data from SYSIPT and, at end-of-job, returns control to the supervisor.

17 The supervisor loads job control.

18 When job control reads the / & statement, it turns off the LINK option and replaces the jobname stored in the supervisor by NO NAME. Other functions of the / & statement are described under *Defining a Job*, earlier in this chapter.

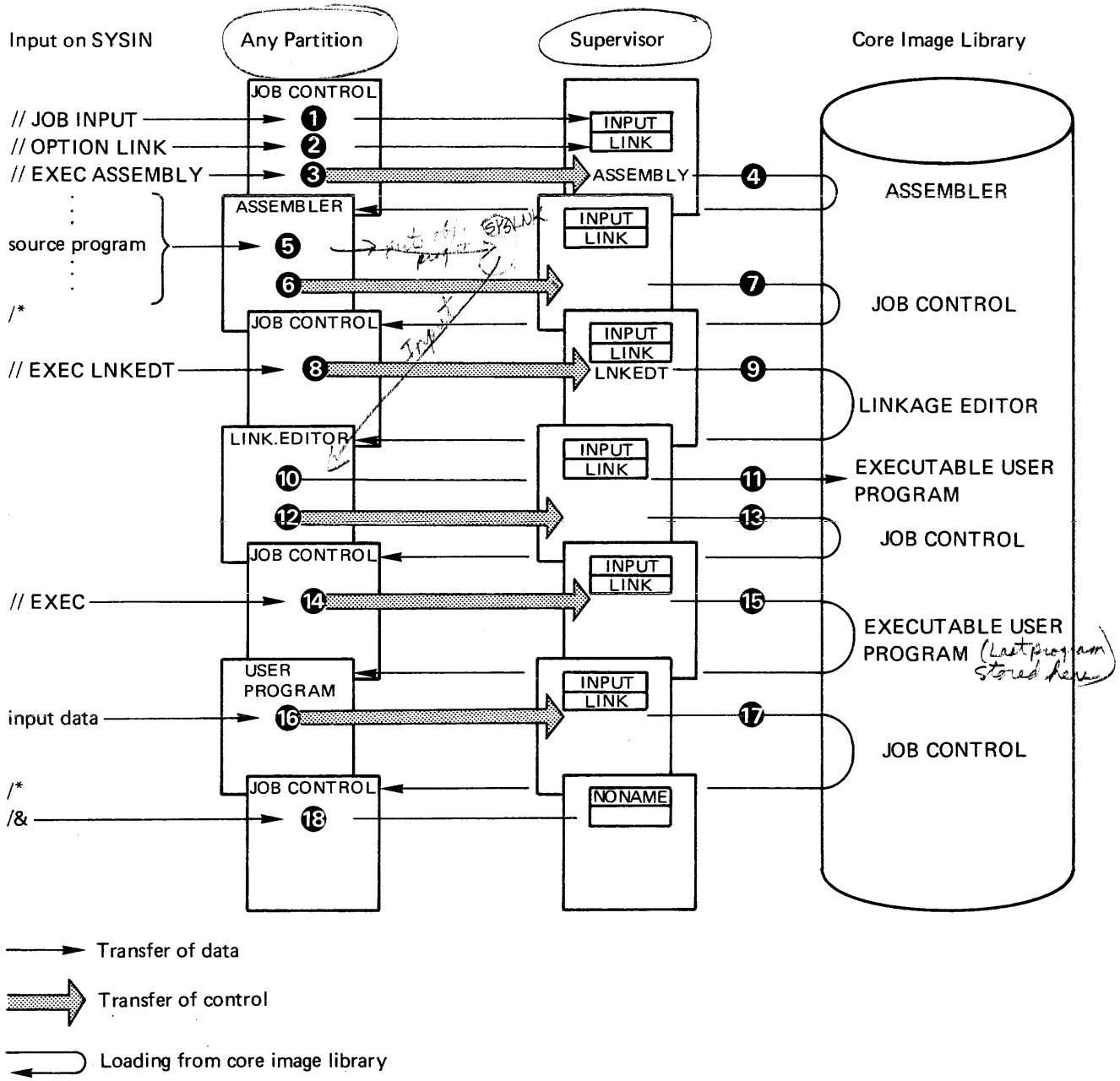


Figure 3-15. System Operation of an Assemble, Link-Edit and Execute Job

Executing Cataloged Programs. Programs may be cataloged permanently in the core image library after they have been assembled and link-edited. This saves assembling and link-editing a program for every run.

Cataloging into the core image library is done by the linkage editor in response to an OPTION job control statement with the CATAL option (see *Linking Programs* later in this chapter).

To execute a cataloged program you use an EXEC job control statement specifying the name under which the program was cataloged (as shown for the assembler and linkage editor in the preceding example).

For example, the following job executes a program that was cataloged in the core image library under the name PROGA; data cards are submitted on SYSIPT:

```
// JOB CAT
    .
    .
assignment and label
statements, if required
    .
    .
// EXEC PROGA
    .
    .
input data
    .
    .
/*
/&
```

Defining Options for Program Execution

In the preceding section, it was shown how the OPTION job control statement can be used

- to specify the type of label information to be stored for a file (USRLABEL, PARSTD, STDLABEL options), and
- to define whether a program is to be link-edited (LINK option).

There are a number of additional functions which you can invoke through the OPTION job control statement. The most important ones are:

// OPTION LOG
Logs all job control statements submitted to the system on SYSLST. This facilitates diagnosing the job control statements in case of an error.

// OPTION PARTDUMP
Dumps the contents of the registers, a formatted portion of the supervisor area, and the current partition on SYSLST in case of abnormal program termination. To obtain the entire supervisor area unformatted, // OPTION DUMP may be used.

// OPTION DECK
Puts an object module on SYSPCH. The object module can then be combined with other object modules by the linkage editor to form one executable program, or it can be used as input to the library maintenance program to catalog it into the relocatable library.

// OPTION LIST, LISTX, SYM, XREF, ERRS
Prints various listings produced by the language translators (compilers) on SYSLST. These listings include object code, symbol table, cross-reference, and error lists which are useful debugging aids during the test period of a program. SXREF may be specified instead of XREF to obtain a cross reference listing that includes only the referenced labels in the assembled program.

These (and other) options may be permanently set by using the STDOPT command. The specified options become effective after the next / & or // JOB statement.

Permanent options are valid for all jobs unless overridden by an OPTION job control statement. Options specified in an OPTION statement remain in effect until (1) a contrary option is read or (2) a JOB or / & statement is encountered which resets the options to permanent.

Certain of these options can be suppressed by specifying the prefix NO (for example, NOLIST, NODUMP). A complete list of the available options is given in *DOS/VSE System Control Statements*.

Communicating with Problem Programs via Job Control

Via job control a program can be instructed to take a specific path of action. This instruction is given by setting program switches which can be tested by the problem program at the time of program execution.

These program switches, called UPSI (user program switch indicator), can be set "on" (1) or "off" (0). They are set by job control in response to the UPSI job control statement. The specific meaning attached to each bit in the UPSI byte depends on the design of the program. The statement

```
// UPSI 10000001
```

for example, sets bits 0 and 7 of the UPSI byte to 1, and bits 2 through 6 to zero. A program can inspect these switches and take a specific path based on their setting. Since the // JOB statement sets the eight bits of the UPSI byte to zero, the // UPSI statement should follow the // JOB statement.

UPSI switches might be useful, for example, in an accounting application that prepares reports of daily, weekly, and monthly accounts. Through the program switches, the application can be instructed as to when the daily, weekly, or monthly reports are due.

For more details on the UPSI statement see *DOS/VSE System Control Statements*.

Executing in Virtual or Real Mode

All programs invoked for execution through an EXEC job control statement are normally executed in virtual mode. To run a program in real mode, you specify the REAL operand in the EXEC statement. Example:

```
// JOB NAME
   .
   .
// EXEC PROGA,REAL
/&
```

If, for the above example, job control runs in partition F2, then the program PROGA will be loaded and executed in real mode if there is sufficient processor storage allocated to the F2 partition to hold the entire program PROGA.

If a program executing in real mode is smaller than the allocated processor storage, the unused allocated processor storage should remain part of the page pool. Specifying the size of the program in the SIZE operand of the EXEC statement accomplishes this. Example:

```
// JOB NAME
    .
    .
// EXEC PROGA,REAL,SIZE=30K
/&
```

If the F2 partition has 50K of processor storage allocated and the program PROGA has a size of 30K bytes, the remaining 20K bytes of that partition will remain in the page pool.

If you specify SIZE=AUTO, job control automatically uses the information in the program's core image directory entry to calculate the size of the program to be loaded.

Running programs in real mode implies temporarily forfeiting a number of page frames in the page pool, which may lead to degradation of system throughput. Therefore, real mode execution should be used sparingly.

With a few exceptions, all IBM-supplied and user-written programs can be executed under DOS/VSE either in virtual or real mode. These exceptions are listed in the following section.

Programs that Must Run in Real Mode. The IBM-supplied program OLTEP (On-line Test Executive Program) must be executed in real mode.

User-written programs must be executed in real mode if they contain channel programs for devices not supported by DOS/VSE.

User-written programs must be executed in real mode or modified if they

- contain MICR stacker selection routines or other time-dependent code for execution of I/O requests.
- contain channel programs that are modified during command execution.
- contain I/O appendage routines causing page faults.

A program may request to obtain additional storage from the partition GETVIS area (this area is described in the following section, *Dynamic Allocation of Storage*). During real mode execution, that storage is obtained from the unused allocated processor storage. Specifying a SIZE value, therefore, allows you to issue GETVIS requests from a program running in real mode (contrary to execution in virtual mode, DOS/VSE does not provide a default partition GETVIS area for real mode execution). For a program that is executed in real mode, allow 16K per open file, and allow additional processor storage if double buffering is used or if FBA files with large CI-sizes or VSE/VSAM files are opened. For most IBM-supplied programs that you want to run real, an allowance of 48K for GETVIS requests suffices.

Note that the FREEVIS macro releases GETVIS space which was obtained through a GETVIS macro; that space is again available for subsequent GETVIS requests. When issued from a program running in real mode, however, the space is not returned to the page pool until the execution of the particular job is finished.

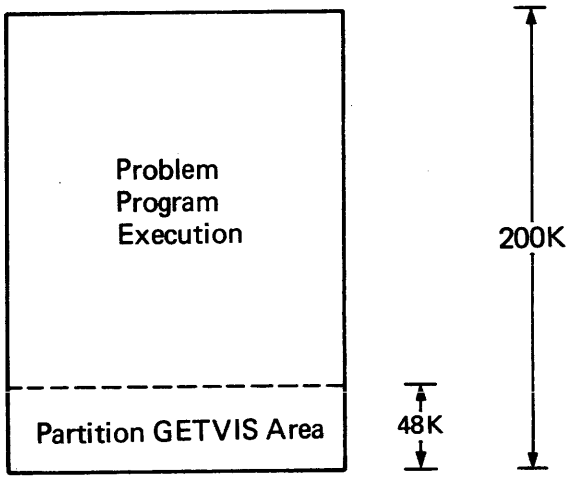
Dynamic Allocation of Storage

DOS/VSE dynamic storage areas, called GETVIS areas, are part of the virtual storage. The system GETVIS area (in the SVA) is discussed in *Chapter 2, Planning the System*. Each partition has an area called the partition GETVIS area. These areas occupy the high address space of a partition's virtual storage. The minimum GETVIS area for a partition is 48K, which is the IBM-set default. This default is not applicable to real mode execution; in this case, you have to reserve storage yourself (as described in the preceding section).

The partition GETVIS area is used by certain DOS/VSE system components for items such as opening of files, label processing etc. Programs using rotational positional sensing (RPS) require 256 to 512 bytes in the partition GETVIS area for each open file. This value should be added to the minimum system requirement of 48K.

Programmers writing in assembler language may request space from the partition GETVIS area via the GETVIS macro. When no longer needed by the requesting program, area so acquired can be released by issuing the FREEVIS macro. For details about using these macros, refer to the publication *DOS/VSE Macro User's Guide*.

Figure 3-16 shows the virtual storage layout of a 200K partition with a default-size partition GETVIS area.



The largest size program that could execute in the shown partition is one that is 152K.

Figure 3-16. Storage Layout of a Partition with Default GETVIS Area

You may increase the size of a partition GETVIS area through:

- the SIZE job control or attention routine command.
- the SIZE parameter of the job control EXEC statement.

With the SIZE command, you specify the amount of virtual storage available for program execution in a given partition. The balance of that partition's allocation is the partition GETVIS area.

Given SIZE BG=140K, the result is a storage layout for the partition as shown in Figure 3-17.

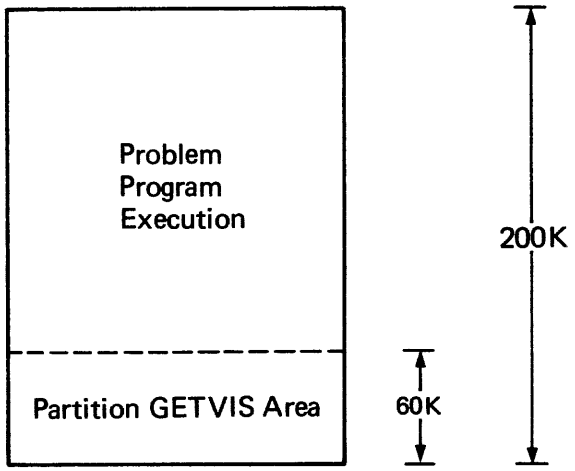


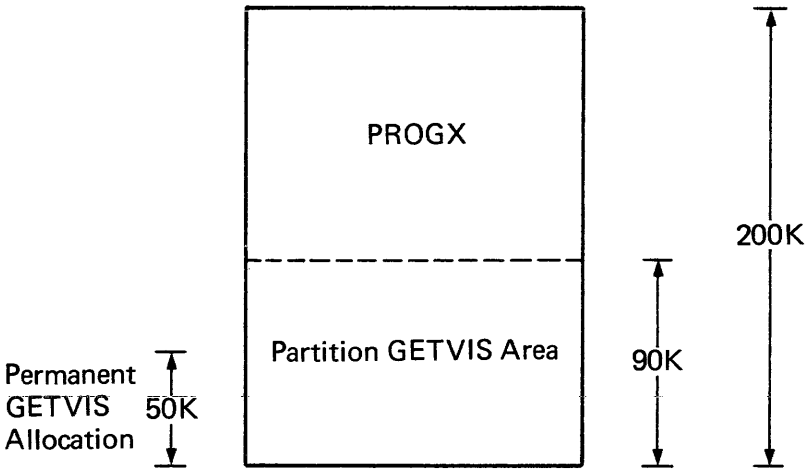
Figure 3-17. Storage Layout of a Partition after the SIZE Command is given

The boundaries set by the SIZE command are permanent until (1) another SIZE command for the same partition or (2) the next IPL.

You may temporarily alter the partition GETVIS area by using the SIZE parameter on the job control EXEC statement. The SIZE parameter establishes boundaries in the same way as the SIZE command, except that the parameter value holds only for one job step (the EXEC). At the end of the job step, the GETVIS size is set to the DOS/VSE default (48K) or the amount established by a preceding SIZE command. See Figure 3-18.

Given:

```
// EXEC PROGX,SIZE=110K
```



When PROGX is finished executing the partition GETVIS area size returns to its permanent allocation.

Figure 3-18. Program Execution with the SIZE Parameter

With the SIZE parameter you may also specify SIZE=AUTO, in which case job control uses the information available in the associated core image library directory to determine the amount of storage needed by the program and then allocates the remainder of the partition as GETVIS area.

IBM licensed programming support (for example VSE/VSAM) may have partition GETVIS requirements beyond 48K bytes. Consult the appropriate licensed program documentation to determine the partition GETVIS area size requirements.

System Files on Tape, Disk or Diskette

As mentioned earlier in this chapter, I/O devices (except DASD) cannot be assigned to more than one active partition at the same time. This means, for instance, that in an installation with only one card reader the input job stream on SYSRDR and SYSIPT for one partition must have been completely processed by job control and unassigned for that partition before job streams can be read by another partition. This also applies accordingly to the system output on SYSLST and SYSPCH if only one printer and one card punch are available.

Since this situation can cause a considerable decrease of system throughput, DOS/VSE permits storing the input job streams and the system output on a direct access device or, if enough tape units are available, on magnetic tape. This allows several partitions simultaneously to read system input from or to write system output to high-speed devices, thus increasing

system throughput and, due to reduced CPU wait time, improving the overall performance.

Note: *If system logical units (SYSIPT, SYSLST, SYSPCH, SYSRDR) are to be device independent, DTFDI must be used in application programs that refer to any of these system logical units.*

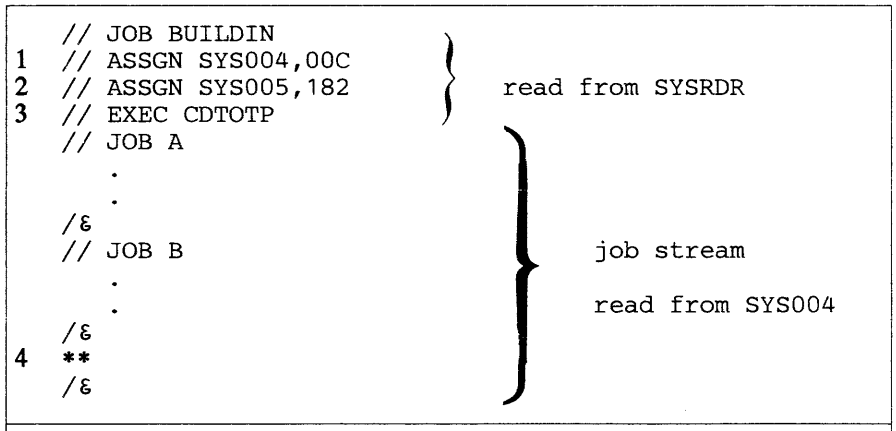
The following section describes how to store system input and output on high-speed devices and to read and process the job streams from these devices.

The same improvements as those gained by having system files on high-speed devices - but far more efficient and easier to use - can be achieved by using a spooling program such as VSE/POWER. The spooling program stores the job streams on disk, transfers the jobs to the partitions for execution, and stores list and punch output on disk before it is finally printed or punched.

System Files on Tape

If the system input units SYSRDR and SYSIPT are assigned to the same magnetic tape unit, they may (but need not) be referred to as SYSIN. If the system output units SYSLST and SYSPCH are assigned to the same magnetic tape they must be referred to as SYSOUT. The tapes may be unlabeled or they may have standard labels. If SYSLST or SYSPCH is assigned to a standard label tape and no new label information is supplied, the old labels will remain on the tape. SYSIPT assigned to a magnetic tape cannot be a multiple-volume file.

To store the input stream on magnetic tape you must write your own program that transfers the job stream to the tape. Assume, in the following example, that you have written such a program and cataloged it in the core image library under the name CDTOTP; the program CDTOTP uses SYS004 to read the input job stream, and SYS005 for the tape onto which the job stream is to be written; the end of input data for CDTOTP is indicated by **. The example in Figure 3-19 shows how to use the program CDTOTP to create a combined system input file on tape.



1	SYS004 is assigned to the card reader from which CDTOTP reads the job stream.
2	SYS005 is assigned to the tape which is to receive the job stream.
3	The CDTOTP program is executed and writes the job stream onto tape.
4	** (or any other significant character combination) signals end-of-data to CDTOTP

Figure 3-19. Creation of SYSIN on Tape

After completion of the job BUILDIN shown in Figure 3-19 you can assign SYSIN to the tape containing the job stream; job control will then read and process the jobs A and B from the tape just as it would have done from the card reader.

In the same way you can direct the system output on SYSLST and SYSPCH to go on magnetic tape and then use your own or an IBM-supplied program to print or punch the contents of the tape on the printer or card punch, respectively.

System Files on Disk

System files on disk can be used only if the SYSFIL parameter was specified in the FOPT generation macro during supervisor generation. Systems without tape units should specify the SYSFIL parameter to facilitate system maintenance.

When both SYSRDR and SYSIPT are assigned to disk, they *must* refer to the same disk extent, and should be referred to as SYSIN. Since the output units SYSLST and SYSPCH have different record lengths, they must be assigned to separate disk extents; SYSOUT therefore *cannot* be used if SYSLST and SYSPCH are assigned to disk. Note that only single extent system files are supported.

For system files on disk, you must provide the required label information by means of DLBL and EXTENT job control statements. In those statements, use the following predefined filenames:

IJSYSIN for SYSRDR, SYSIPT, SYSIN
IJSYSPH for SYSPCH
IJSYSLS for SYSLST

For example, the label information for SYSIN assigned to a disk extent could be submitted by the following job control statements:

```
// DLBL IJSYSIN,'DISKINFILE'
// EXTENT SYSIN,DOSRES,1,0,1260,30
```

The assignment of a system file to a disk extent must always be permanent, and it must follow the DLBL and EXTENT statement. Example:

```
// DLBL IJSYSIN,'DISKINFILE'
// EXTENT SYSIN,DOSRES,1,0,1260,30
   ASSGN SYSIN,131
```

After a system file on disk has been processed, it must be closed by a CLOSE job control command (no //). The second (optional) operand of the CLOSE command can be used to unassign a system logical unit or reassign it to another device. The following command closes the SYSIN file on disk and reassigns SYSIN to the card reader at address 00C:

```
CLOSE SYSIN,00C
```

The CLOSE command can either be entered on SYSLOG by the operator or it can be included at the end of the job stream on disk.

If SYSIPT is assigned to a disk extent, the CLOSE command must precede the / & . Multiple SYSIPT data files can be read via multiple job steps with one / & at the end of the job stream.

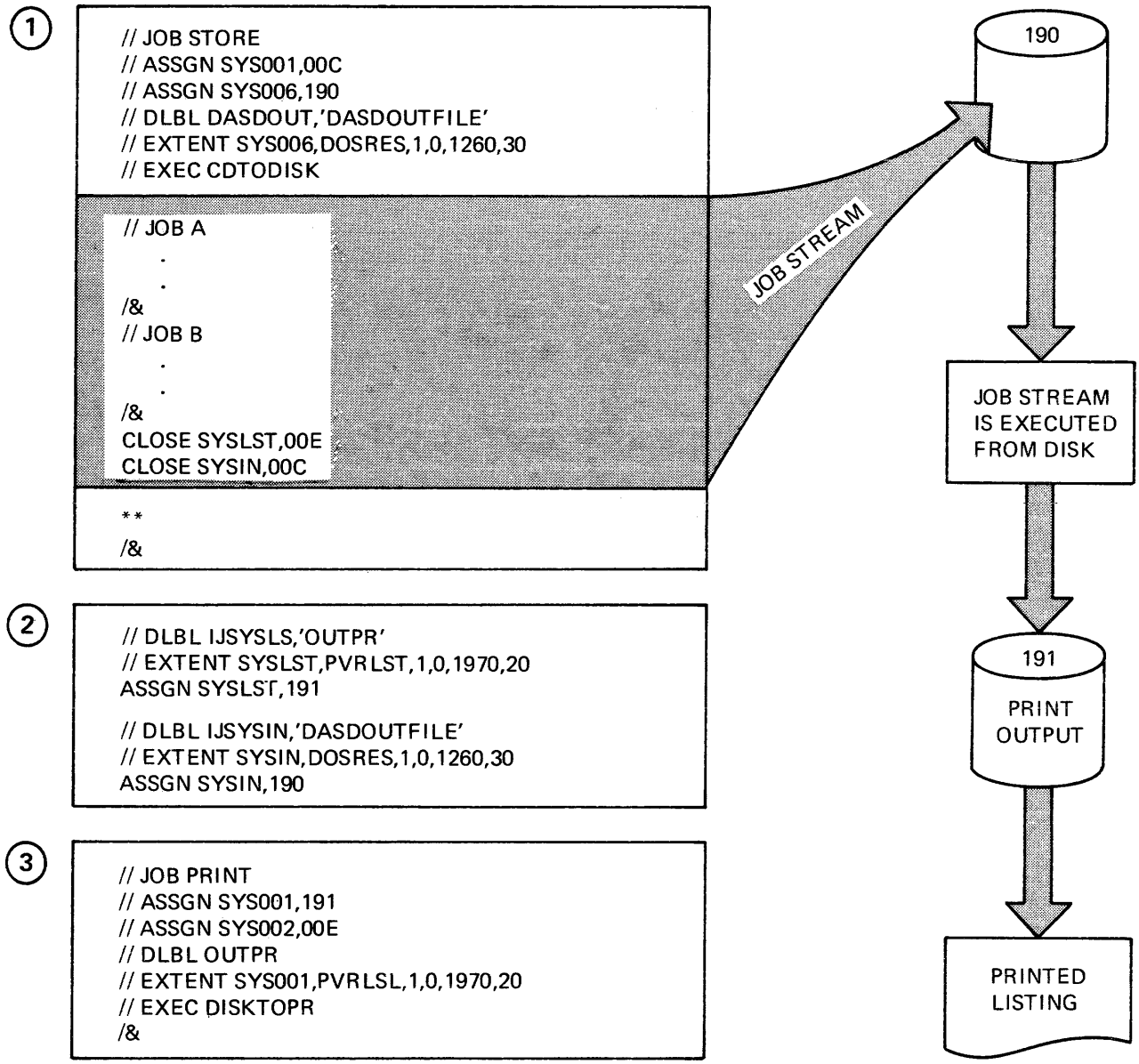
The example in Figure 3-20 shows the job control statements needed to

1. write a job stream on disk,
2. execute the job stream from disk and store the print output on disk, and
3. print the output from disk on the printer.

The example assumes that you have written your own programs to write the job stream on disk (CDTODISK) and to list on the printer the print output stored on disk (DISKTOPR).

System Files on Fixed Block Architecture (FBA) DASD. If an FBA DASD has a system logical unit assigned to it, the supervisor SYSFIL support will block and deblock system file records into the FBA Control Interval-based data format, handle all special conditions, and update the Disk Information Block (DIB). This permits existing DTFDI and DTFCP programs to process system files on FBA devices without making logic changes to handle the FBA blocking.

Note, however, that the DTFSD SYSFIL support is limited to sequential GET or PUT for fixed unblocked records. (That is, the UPDATE=YES parameter is not supported.)



(1) The program CDTODISK reads the following job stream from the card reader (SYS001) and stores it on disk (SYS006). The end of the job stream is indicated to CDTODISK by **.

(2) SYSLST and SYSIN are switched to disk. Job control now reads the job stream from the disk on device 190. The job stream is executed and the print output is stored on the disk on device 191. The CLOSE commands at the end of the job stream will close the system files on disk and reassign them to the printer and card reader, respectively.

(3) The program DISKTOPR reads the print output from disk (SYS001) and lists it on the printer (SYS002).

Figure 3-20. Processing System Input and Output Files on Disk

System files on diskette can be used only if the SYSFIL parameter was specified in the FOPT generation macro during supervisor generation.

If the system input units SYSRDR and SYSIPT are assigned to a diskette extent, they *must* be referred to as SYSIN. Since the output units SYSLST and SYSPCH have different record lengths, they must be assigned to separate diskette extents; SYSOUT therefore *cannot* be used if SYSLST and SYSPCH are assigned to diskette.

For system files on diskette, you must provide the required label information by means of DLBL and EXTENT job control statements. In those statements, use the following predefined filenames:

IJSYSIN for SYSRDR, SYSIPT, SYSIN
IJSYSPH for SYSPCH
IJSYSLS for SYSLST

For example, the label information for SYSIN assigned to a diskette extent could be submitted by the following job control statements:

```
// DLBL IJSYSIN,'DISKETTE',,DU
// EXTENT SYSIN,DSKETE,1
```

The assignment of a system file to a diskette extent must always be permanent, and it must follow the DLBL and EXTENT statement.

Example:

```
// DLBL IJSYSIN,'DISKETTE',,DU
// EXTENT SYSIN,DSKETE,1
   ASSGN SYSIN,060
```

After a system file on diskette has been processed, it must be closed by a CLOSE job control command (no //). The second (optional) operand of the CLOSE command can be used to unassign a system logical unit or reassign it to another device. The following command closes the SYSIN file on diskette and reassigns SYSIN to the card reader at address 00C.

```
CLOSE SYSIN,00C
```

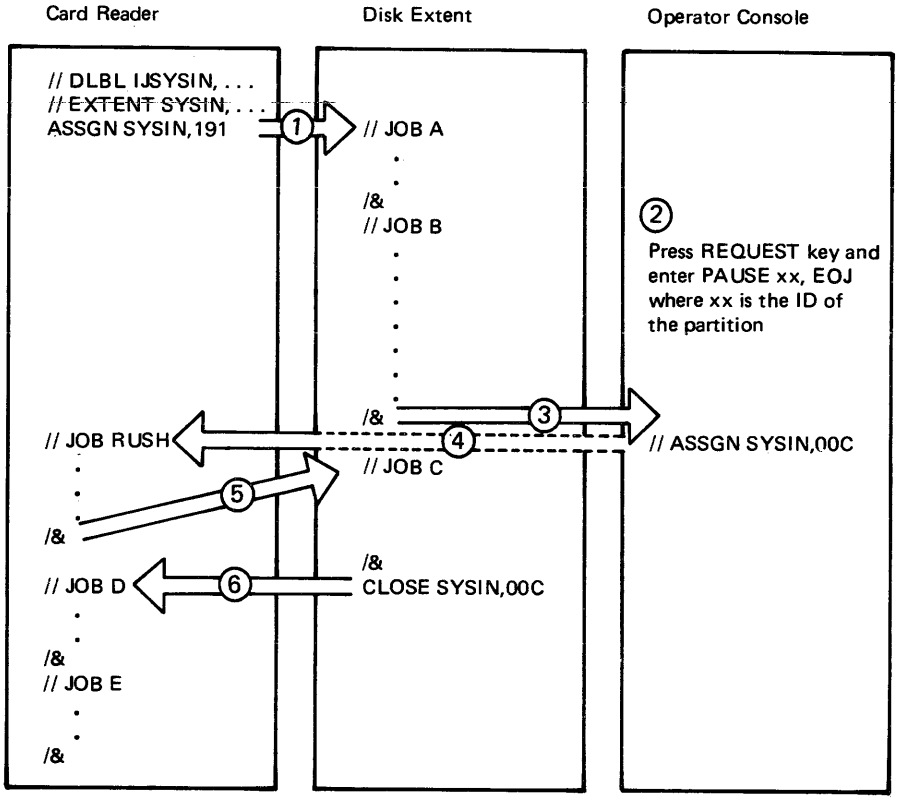
The CLOSE command can either be entered on SYSLOG by the operator or it can be included at the end of the job stream on diskette.

If SYSIPT is assigned to a 3540 diskette, the CLOSE command must precede the / & . Multiple input data files can be read via multiple job steps with one / & at the end of the job stream.

When job control encounters / & on SYSRDR during normal operation, the standard assignment for SYSIPT becomes effective and SYSIPT is checked for an end-of-file condition. If the standard assignments for SYSRDR and SYSIPT are not to the same device, SYSIPT is advanced to the next /* statement.

Interrupting SYSIN Job Streams on Disk, Diskette, or Tape

After a SYSIN or SYSRDR job stream has been prepared on tape, diskette, or disk, it may be necessary to interrupt the normal schedule to execute a rush job. To do this, press the Request key on the operator console and enter a PAUSE command with the EOJ operand causing the corresponding partition to suspend processing at the end of the current job. At this point you can make a temporary assignment for SYSIN to a card reader to execute the rush job. At the end of this job, processing of the job stream on disk, diskette, or tape will resume at the point of interruption. This is illustrated in Figure 3-21. Starting an urgent job that uses a cataloged procedure by means of a single EXEC statement is discussed in the section *Partition-Related Cataloged Procedures* later in this chapter.



1. SYSIN is assigned to disk and processing of the jobstream on disk begins.
2. While job B is being executed a PAUSE command is entered at the operator console.
3. At the end of job B control comes to the operator who can now enter a temporary assignment for SYSIN to the card reader.
4. The job RUSH is read and processed from the card reader. Note that the temporary assignment of SYSIN is not reset by the //JOB RUSH statement but is retained to end of the job.
5. The /& statement resets the temporary assignment of SYSIN to permanent (190) and the next job in the stream on disk is read and executed.
6. The CLOSE command closes the system file on disk and reassigns SYSIN to the card reader to process jobs D and E.

Figure 3-21. Interrupting a Job Stream on Disk

SYSLST records are 121 characters and SYSPCH records 81 characters in length. From SYSRDR and SYSIPT, job control accepts either 80- or 81-character records.

The first character of the SYSLST and SYSPCH records is assumed to be an ASA carriage control or stacker selection character. SYSIPT, SYSRDR, SYSPCH, and SYSLST records assigned to DASD have no keys, and record lengths are the same as stated above. (For CKD devices the records are unblocked; for FBA devices, DOS/VSE automatically blocks records into the FBA format and also deblocks them.)

Using Cataloged Procedures

This section describes how to retrieve a cataloged procedure from the procedure library and how to temporarily modify the contents of a cataloged procedure. How a procedure is cataloged in the procedure library is discussed in *Using the Libraries* later in this chapter.

Note: *The procedure library should not be updated in a running multiprogramming system.*

Retrieving Cataloged Procedures

To retrieve a cataloged procedure from the procedure library you use the PROC parameter in the EXEC job control statement, specifying the name of the cataloged procedure. Assume that a program called PAYROLL uses the following job control statements (in addition to the // JOB and / & statements) and that these statements have been cataloged in the procedure library under the name PAY.

```
// ASSGN SYS017,SYSRDR
// ASSGN SYS018,SYSPCH
// ASSGN SYS019,00E
// ASSGN SYS020,TAPE
// ASSGN SYS021,DISK,VOL=111111
// TLBL TAPFLE,'FILE-IN'
// DLBL DSKFLE,'FILE-OUT',99/365,SD
// EXTENT SYS021,111111,1,0,200,400
// EXEC PAYROLL
```

If the program PAYROLL is to be executed, the programmer or operator would simply prepare the following job control statements:

```
// JOB USER1
// EXEC PROC=PAY
/&
```

When the job control program starts reading the job control statements in the input stream on SYSRDR and finds the EXEC statement, it knows by the operand PROC that a cataloged procedure is to be inserted. It takes the name of the procedure to be used (PAY) and retrieves the procedure with that name from the procedure library. At this point SYSRDR is temporarily

assigned to the procedure library. Job control reads and processes the job control statements in its normal fashion. The statement

```
// EXEC PAYROLL
```

causes the program PAYROLL to be loaded and given control. When execution of PAYROLL is complete, the job control program reads the next statement from the procedure library and, in this example, would find an end of procedure indicator (/+). The end of procedure indicator returns the SYSRDR assignment to its permanent device, where the job control program finds the / & statement and performs end-of-job processing as usual.

Note: *The listing of job control statements on SYSLOG and/or SYSLST will show the message EOP PAY at the end of the inserted procedure.*

Temporarily Modifying Cataloged Procedures

The preceding example is the simplest case of the use of cataloged procedures. It will work as long as the requirements of the program do not change.

It may happen, however, that some of the statements in a cataloged procedure must be modified for a specific run of a program. For example, the printer normally used (00E in the preceding example) may be temporarily unavailable and a different printer must be assigned. It does not make much sense to delete the old procedure and to catalog a new one because the old procedure will be needed again as soon as the normal printer becomes operational again.

Likewise, it may be necessary to add or remove certain statements to or from a cataloged procedure for a specific run of a program. You may wish, for example, to process a different copy of the file FILE-OUT (see the preceding example). You must therefore temporarily suppress the corresponding DLBL and EXTENT statements in the cataloged procedure and replace them by statements that identify the file you want to process instead.

For cases like this, DOS/VSE permits one or more statements in a cataloged procedure to be

- temporarily modified (thus, overriding what was present).
- temporarily suppressed (deleted) without modifying them.
- temporarily incorporated at desired locations in a cataloged procedure.

You can request temporary modification of statements in a cataloged procedure by supplying the corresponding modifier statements in the input stream.

Since normally not all statements need be modified, you must establish an exact correspondence between the statement to be modified and the modifier statement by giving them the same symbolic name. This symbolic name may have from one to seven characters, and must be specified in columns 73 through 79 of both statements.

Note: *An unnamed statement cannot be modified. Therefore, to be able to modify any statement in a cataloged procedure for any usage of the procedure you should name each statement when cataloging. Moreover, the modifier statements must be in the sequence in which modification is to be performed on the cataloged statements. The JOB statement cannot be modified; also, job control continuation statements cannot be overriden.*

A single character in column 80 of the modifier statement specifies which function is to be performed:

A - indicates that the statement is to be inserted *after* the statement in the cataloged procedure that has the same name.

B - indicates that the statement is to be inserted *before* the statement in the cataloged procedure that has the same name.

D - indicates that the statement in the cataloged procedure that has the same name is to be *deleted.*

Any other character or a blank in column 80 of the modifier statement indicates that the statement is to replace (override) the statement in the cataloged procedure that has the same name.

If the LOG function is active (by having issued the LOG job control command), statements to be deleted are printed, with a D in column 80, on the console, but not 'executed'.

In addition to naming the statements and indicating the function to be performed, you must inform the job control program that it has to carry out a procedure modification. This is done

(1) by specifying an additional parameter (OV for overriding) in the EXEC statement that calls the procedure, and

(2) by using the statement // OVEND to indicate the end of the modifier statements.

Placement of the // OVEND statement is as follows:

- directly behind the last modifier statement or,
- if the last modifier statement overwrites a // EXEC statement and is followed by data input, between the /* and the /&.

The following examples show how you can temporarily modify a cataloged procedure.

Assume that a procedure named PROC5 for the program PAYROLL contains the following statements:

```
                                                                  73--79
// ASSGN SYS017,SYSRDR                                           PAY001
// ASSGN SYS018,SYSPCH                                           PAY002
// ASSGN SYS019,SYSLST                                           PAY003
// ASSGN SYS020,181                                              PAY004
// ASSGN SYS021,DISK,VOL=111111,SHR                              PAY005
// TLBL TAPFLE,'FILE-IN'                                         PAY006
// DLBL DSKFLE,'FILE-OUT'                                        PAY007
// EXTENT SYS021,111111,1,0,200,200                              PAY008
// EXEC PAYROLL                                                  PAY009
/+
```

Assume further that the programmer wants to use tape unit 183 instead of 181. The input stream on SYSRDR, in this case, would have to be as follows:

```
                                                                  73--80
// JOB USER
// EXEC PROC=PROC5,OV
// ASSGN SYS020,183                                              PAY004R
// OVEND
/&
```

The form of the EXEC statement in the input stream indicates that (1) the procedure PROC5 is to be used and (2) this procedure is to be modified in some way. The first three procedure statements are processed without change. The procedure statement named PAY0004 is replaced by the corresponding statement in the input stream. (As any character other than A, B, or D specifies override, an R was used to indicate this.) The remaining procedure statements are again processed without change.

As another example, assume that the program PAYROLL is to use file FILE-OUT1 instead of FILE-OUT and that this file resides on two extents of a disk pack that has the volume serial number 111112. The input stream might then look as follows:

```
                                                                Col.73--80
// JOB USER
// EXEC PROC=PROC5,OV
// ASSGN SYS021,DISK,VOL=111112,SHR                            PAY0005R
// DLBL DSKFLE,'FILE-OUT1'                                     PAY0007R
// EXTENT SYS021,111112,1,0,100,200                            PAY0008R
// EXTENT SYS021,111112,1,1,500,200                            PAY0008A
// OVEND
/&
```

Processing would be as follows: The JOB statement and all procedure statements up to the statement named PAY0004 are processed without modification. The procedure statements labeled PAY0005, PAY0007, and PAY0008 are replaced by the corresponding statements in the input stream. The second EXTENT statement in the input stream has the character A in column 80, which indicates that the statement is to be inserted after the (replaced) statement named PAY0008. The procedure statement named PAY0009 is processed without modification.

The possibility of modification as described above makes the use of cataloged procedures more flexible. Often, however, it is simpler and more economical to have different procedures for the same program than to have a single procedure and modify it.

SYSIPT data in a cataloged procedure cannot be overridden by the procedure override facility.

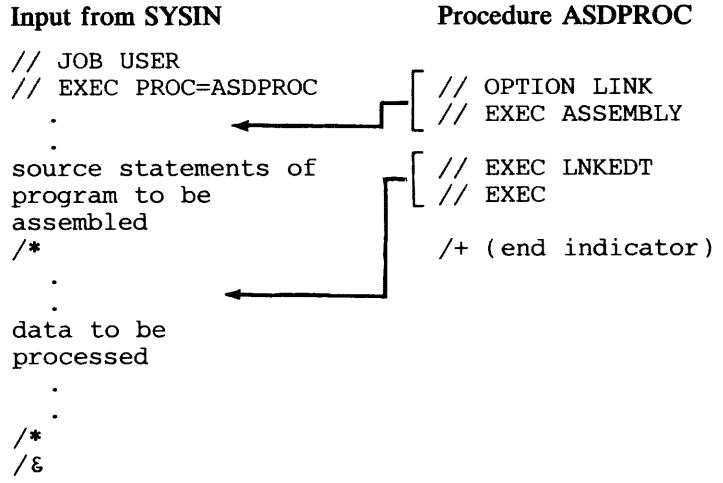
Several Job Steps in one Procedure

A cataloged procedure may contain more than one EXEC statement, that is, it may contain control statements for more than one job step (within the same job). However, as the number of job steps in a procedure increases, so does the time required to re-execute the whole procedure after an error occurs.

A program written in assembler language, for instance, requires three job steps to assemble, link-edit, and execute the program. For the use of a cataloged procedure, your input stream for the entire job (on SYSIN for simplicity) would contain the following:

```
// JOB USER
// OPTION LINK
// EXEC ASSEMBLY
source deck of program to be assembled
/*
// EXEC LNKEDT
// EXEC
data for program to be executed
/*
/&
```

If the OPTION statement and the three EXEC statements were cataloged under the name ASDPROC, the input stream could be simplified as shown below.



The same can be done for any number of job steps that logically belong together and are frequently executed. A stock control program STOCK, for instance, may be run daily to compile statistics that can be used to prepare the following lists:

1. An exception list that shows which items are low in stock. Required daily.
2. A list that shows the sales in currency for a certain item or group of items. Required weekly.

3. A list that shows the sales in number of units for each item or group of items. Required monthly.

4. An inventory list. Required semi-annually.

To simplify processing, four procedures may have been cataloged:

STKPR1 - two job steps: the first to execute STOCK, the second to prepare list 1.

STKPR2 - three job steps: the first two are the same as for STKPR1, the third to prepare list 2.

STKPR3 - four job steps: the first three the same as for STKPR2, the fourth to prepare list 3.

STKPR4 - five job steps: the first four the same as for STKPR3, the fifth to prepare list 4.

Which lists are printed after every run of STOCK then depends on what cataloged procedure is used.

Modifying Multistep Procedures

Multistep procedures may be modified in the same way as the single-step procedure described earlier. However, a number of considerations apply to the ordering of the modification statements in the input stream when a logical unit used for data input is assigned to the same physical unit as SYSRDR.

- It is advisable to avoid using identical symbolic names for the statements in the procedure.
- The modifier statements must be in the same sequence as the statements in the referenced procedure.
- Modifier statements are normally placed immediately following the EXEC PROC=procedure,OV statement. When input data is read by a job step (EXEC statement) executed from the procedure, the following cautions should be observed:

  1. The first statement following the EXEC PROC=procedure,OV must be a modifier statement (see "1" in Figure 3-22).

  2. Modifier statements that take affect after the input data is read are placed following the input data *except for the first modifier which must precede the input data* (see "1" and the modifier statement ASSGN SYSSLB,UA in Figure 3-22).

  3. An exception to point 2 above is when the input data is processed by a job step that itself was modified (see "3" and "4" in Figure 3-22). In this case the next modifier must follow the data (see statement "3" and the modifier ASSGN SYSCLB,UA in Figure 3-22).

Figure 3-22 shows an example of modifying the second and third steps of a three-step procedure.

In the example given in Figure 3-22, it is assumed that SYSRDR and SYSIPT are assigned to the same physical unit.

	SYSIN Input Stream	Column 73–79	Procedure CAT01 Containing JCL Only	Column 73–79
	// JOB EXAMPLE			
	// EXEC PROC=CAT01,OV			
❶	// ASSGN SYSRLB,UA	STMT3	// EXEC PSERV	STMT1
❷	DSPLY CAT01			
	/*		ASSGN SYSCLB,130	STMT2
			// ASSGN SYSRLB,130	STMT3
	// ASSGN SYSSLB,UA	STMT4	// ASSGN SYSSLB,130	STMT4
❸	// EXEC DSERV,REAL	STMT5	// EXEC DSERV	STMT5
❹	DSPLY CD,RD,SD			
	/*			
	ASSGN SYSCLB,UA	STMT6	// ASSGN SYSSLB,UA	STMT6
	// OVEND		// EXEC DSERV,REAL	STMT7
❺	DSPLY CD, PD		/+	
	/*			
	/&			

❶ This is the first modifier statement. It refers to the second job step.

❷ This statement provides SYSIPT data for PSERV.

❸ This modification overwrites the EXEC statement.

❹ This statement provides SYSIPT data for DSERV (STMT5).

❺ This statement provides SYSIPT data for DSERV (STMT7).

Figure 3-22. Example of Modifying a Three-Step Procedure

SYSIPT Data in Cataloged Procedures

In the example shown in Figure 3-22 the librarian service programs PSERV and DSERV accessed data from the logical unit SYSIPT. This 'SYSIPT' data may be made part of your cataloged procedure if SYSFIL=YES was specified in the FOPT generation macro at supervisor assembly. System utility, system service programs, and language translators all read their input from SYSIPT.

When you catalog a procedure containing SYSIPT data, the directory entry for the procedure indicates this. When you execute such a procedure, job control checks to see whether or not it contains SYSIPT data. If it does, both SYSRDR and SYSIPT are assigned to the procedure library until the end of the procedure. SYSIPT data in a cataloged procedure cannot be overriden by the procedure library override facility.

SYSIPT inline data in procedures may also be any data that is processed under control of the device independent IOCS used by your

program or IBM-supplied programs. Normally, though, you would not catalog source programs or data for your problem programs in the procedure library.

SYSIPT inline data in procedures is useful and convenient mainly in the case of control information for system utility and service programs.

A job stream for an initialize disk utility run could, for instance, contain the following control statements (the statements are shown in skeleton format only):

```
// ASSGN ...
// EXEC INTDK
// UID IR,C1,R=(0027003)
// VTOC STANDARD
VOL1111111
// END
/&
```

The job control statements are read from SYSRDR, the utility control statements are read from SYSIPT. If, however, both the job control and utility control statements had been cataloged (for example, under the name INITDK), only the following statements would be required on SYSRDR:

```
// JOB NAME
// EXEC PROC=INITDK
/&
```

If two or more programs in a procedure read SYSIPT data, the SYSIPT data must be handled in a consistent manner, that is, if the SYSIPT data is included in the procedure for one job step, it must be included for all job steps in that procedure which require SYSIPT data.

Partition-Related Cataloged Procedures

Although a given procedure may be executed in any partition, a particular job may need a specific set of job control statements, dependent on the partition of execution. For example, you may want to run a job to store DLBL and EXTENT statements in the partition label subarea for each partition (OPTION PARSTD). Since each partition requires a different set of label information, you would need a cataloged procedure for each of your partitions. Partition-related cataloged procedures then allow you to retrieve and execute the appropriate procedure with one version of the EXEC statement, no matter which partition you are running in. One benefit of this feature lies in the ease with which unscheduled jobs can be started.

To use the feature, you must first create separate procedures that conform to the specific partitions in your system. Most probably, the difference in these procedures will be in the EXTENT and DLBL statements because of the different device and DASD space assignments from partition to partition. Next, in order to distinguish between the procedures and relate them to the appropriate partitions, the following naming convention must be used for cataloging these procedures:

First character of name	–	$
Second character	–	B for BG partition
	–	1 for F1 partition, 2 for F2 partition, etc.
Third-eighth characters	–	any alphameric character

In the EXEC statement used to start the job, the first two characters of the procedure name must be $$, with the remaining characters identical to the last six characters of the cataloged name.

To continue the previous example, the procedures may be named $BPARSTD for the BG partition, $1PARSTD for the F1 partition and so on. The statement thus needed to invoke the appropriate procedure is // EXEC PROC=$$PARSTD.

Use of Cataloged Procedures by the Operator

Partition related procedures or procedures for the starting of urgent jobs are of great help to the operator. Full details on the use of cataloged procedures by the operator are given in *DOS/VSE Operating Procedures*.

Linking Programs

Prior to execution in storage, all programs must be placed in the core image library by the linkage editor. This section describes the role of the linkage editor and how you can communicate with it through control statements.

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 one or more executable 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 into one or more phases 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 linkage editor is no longer required for that phase*, because the supervisor can load it directly from the library in response to an EXEC job control statement, or a FETCH or LOAD macro. On the other hand, if the phase is cataloged temporarily and executed immediately, the linkage editor is required again the next time the phase 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 phase that is link edited.

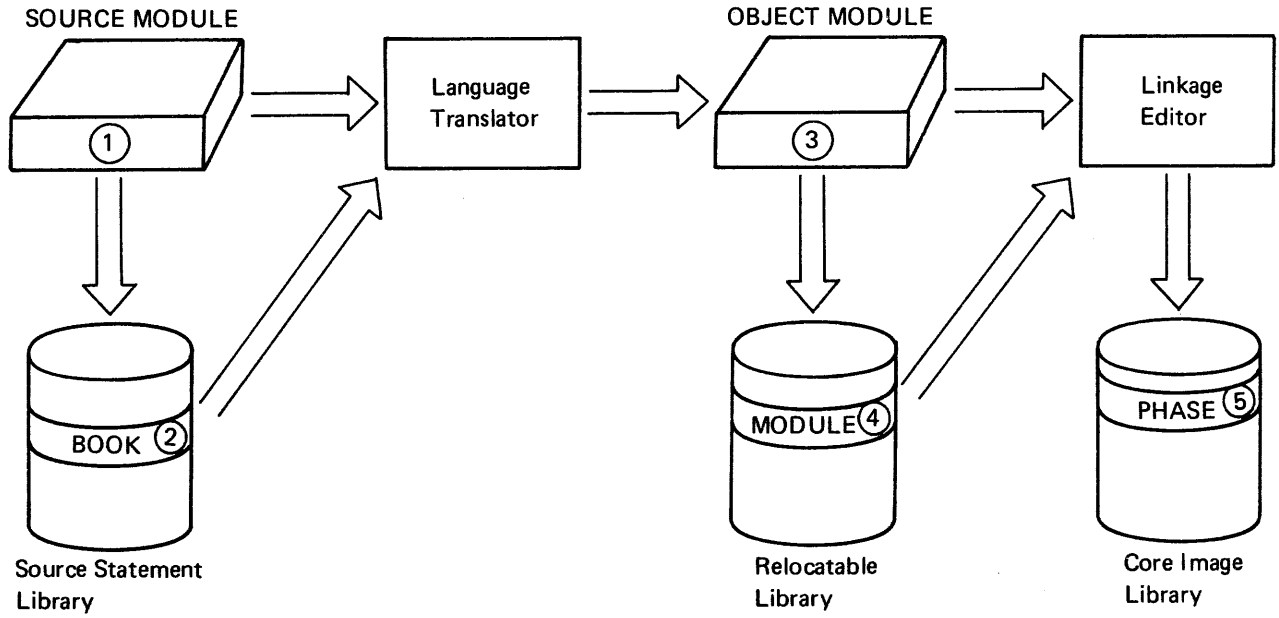
```
// OPTION CATAL
```

If the CATAL option is specified, the phase is stored permanently and can be executed any time after the link-edit run.

Phases produced by the linkage editor while running in a foreground partition are cataloged into a private core image library. To catalog a phase into the system core image library, the linkage editor must execute in the background partition. You may, optionally, use a private core image library in the background partition by ensuring that it is assigned during execution of the linkage editor. For more information on using private libraries, refer to *Using the Libraries* later in this chapter.

*If a phase is non-relocatable and the partition boundaries change so that the cataloged program's start and end addresses no longer fall within the partition, the program must be link-edited again.

To understand the functions of the linkage editor, you must understand the structure of a program during the various stages of its development. Figure 3-23 summarizes the three sections that follow, which discuss source modules, object modules, and program phases.



A set of source statements, or source module (1), must be processed by a language translator, but can first be cataloged as a book (2) into the source statement library. The output of the language translator is called an object module (3), which must be processed by the linkage editor, but can first be cataloged as a module (4) into the relocatable library. The output of the linkage editor is called a phase (5), which is cataloged into the core image library temporarily or permanently, and can also be loaded into the shared virtual area.

Figure 3-23. Stages of Program Development

Source Modules

After planning the most logical approach to your application, you write a set of source statements in a programming language. Your set of source statements, called a source module, is processed by a language translator. The language translator *assembles* source modules written in assembler language, or it *compiles* source modules written in a high-level language (for instance, COBOL, PL/I, or RPG II). The language translator transforms the source module into an object module, which is in machine language.

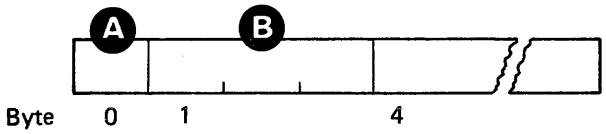
You can either submit your source module directly to the language translator for processing, or you can catalog it into a sublibrary of the source statement library for processing at a later time by the language translator.

Source modules are written in one or more control sections (CSECTs). Using assembler language the programmer defines the control sections. Source modules written in a high-level language have their control sections defined by the various compiler options used.

Object Modules

An object module, the output of a language translator, consists of the dictionaries and text of one or more control sections. The dictionaries contain the information needed by the linkage editor to modify portions of the text for relocation and to resolve cross-references between different object modules. The text consists of the actual instructions and data fields of the object module. You can either submit your object module directly to the linkage editor for processing, or catalog it into the relocatable library for later inclusion in a linkage editor job stream.

For each object module the language translator produces four types of records as illustrated and summarized in Figure 3-24. For more information about these records see *DOS/VSE System Control Statements*.



A Contains X'02'. Identifies the record as one of an object module.

B Indicates the record type and can be one of the following:

C'ESD' -- External symbol dictionary. Contains symbols defined in this module and referred to by one or more other modules and symbols referred to in this module but defined in another module.

C'TXT' -- Text. Contains actual code plus control information needed by the linkage editor.

C'RLD' -- Relocation list dictionary. Identifies those portions of the text which must be modified when the program is relocated for execution.

C'END' -- End of module. Indicates the end of a module. The record may contain an address where execution is to begin (transfer address) or the length of the control section or both.

Figure 3-24. Record Types of an Object Module

If you want to change information in a TXT record, you can prepare a REP record (user replace record) and submit it with your object module for cataloging into the relocatable library or for linkage editor processing. A REP record must be submitted between the TXT record it modifies and the END record; otherwise, the TXT record is not modified. Usually, you place the REP record(s) immediately before the END record.

The linkage editor produces a program phase from the object module(s) you identify in linkage editor control statements. A phase is the functional unit (consisting of one or more control sections) that the system loader can load into a partition in response to a single EXEC job control statement (or a FETCH or a LOAD macro instruction in an assembler language program).

In the PHASE control statement you instruct the linkage editor to produce one of three types of phases: relocatable, self-relocating, or non-relocatable.

Relocatable Phases. A phase is relocatable if it can be loaded for execution in any partition's address area. The linkage editor produces a relocatable phase unless you specify an absolute origin (load) address instead of a relative address. However, IBM recommends that you always specify a relative origin address. An address, in order to be relative, is represented by a symbol with or without a displacement; for details see *DOS/VSE System Control Statements*.

If a relocatable phase is also designed as a reenterable phase, it is eligible to be loaded into the shared virtual area (SVA). Phases resident in the SVA can be shared concurrently by programs running in either real or virtual mode.

Self-Relocating Phases. Prior to the availability of a loader with the relocating capability some users coded self-relocating programs in order to gain the advantages of relocatability. If you have to perform maintenance on such a program, you must write this program in assembler language according to the rules described in *DOS/VSE Macro User's Guide*. In the PHASE control statement you indicate an origin address of +0. The program must relocate all its addresses at execution time to correspond with the addresses available in the partition where the program is loaded.

Non-Relocatable Phases. A non-relocatable phase is link-edited to be loaded at a specific location (absolute address) associated with a partition. When you request execution of a non-relocatable phase in a given partition, the starting and ending addresses of the phase must be included within that partition. Otherwise, the job is canceled. If you wish to execute a non-relocatable phase in more than one partition, you must catalog a separate copy of the phase for each partition.

The Three Basic Applications of the Linkage Editor

The three basic applications of the linkage editor are referred to as:

- cataloging phases into the core image library
- link-edit and execute
- assemble (or compile), link-edit, and execute

The following sections include a discussion of the system flow during each of these applications.

Cataloging Phases into the Core Image Library

When you have an operational program (as an object deck in cards or on tape, for example) and you expect to use that program frequently, you should catalog it into a core image library. You can do this in a single job step, which is shown in Figure 3-25, and described below.

Job control copies, onto SYSLNK, the linkage editor control statements present on SYSRDR. The INCLUDE statement, without operands, signals job control to read any object modules that are to be included from SYSIPT. If an ENTRY statement is not encountered before the // EXEC LNKEDT statement, job control writes one on SYSLNK. An ENTRY statement signals termination of the input to the linkage editor.

The linkage editor is loaded into the partition where the job stream was submitted; it uses SYS001 as a work file.

Because the CATAL operand of the OPTION statement was specified, the linkage editor places the executable program permanently into a core image library. If a private core image library is assigned to this partition, the program is cataloged there; otherwise, (for the background partition) it is cataloged into the system core image library. The library descriptor entry in the core image directory for cataloged phases is updated.

If the phase is eligible for the shared virtual area and is indicated as SVA eligible in the system directory list, the phase is also loaded into the SVA.

Note: *System and work files such as SYSLNK and SYS001 must be defined.*

Cataloging a Supervisor. Supervisors may also be cataloged permanently into the core image library as described above. Be sure, when doing this, to specify a unique name (eight alphameric characters) for each supervisor.

Link-Edit and Execute

You do not always need to catalog a permanent copy of your program into the core image library in order to execute the program. For instance, you have modified parts of your program and want to test these modifications with the entire program. In this case, you can specify the LINK option, which requests that the linkage editor place a temporary copy of the program into the core image library. Again, the INCLUDE statement signals job control to read the following input from SYSIPT. The shaded portions of Figure 3-26 illustrate how this job stream differs from Figure 3-25.

By specifying an EXEC statement without a program name operand after the EXEC LNKEDT statement, the program just link-edited is loaded for execution. The space temporarily occupied by this program in the core image library is overwritten the next time a program is link-edited.

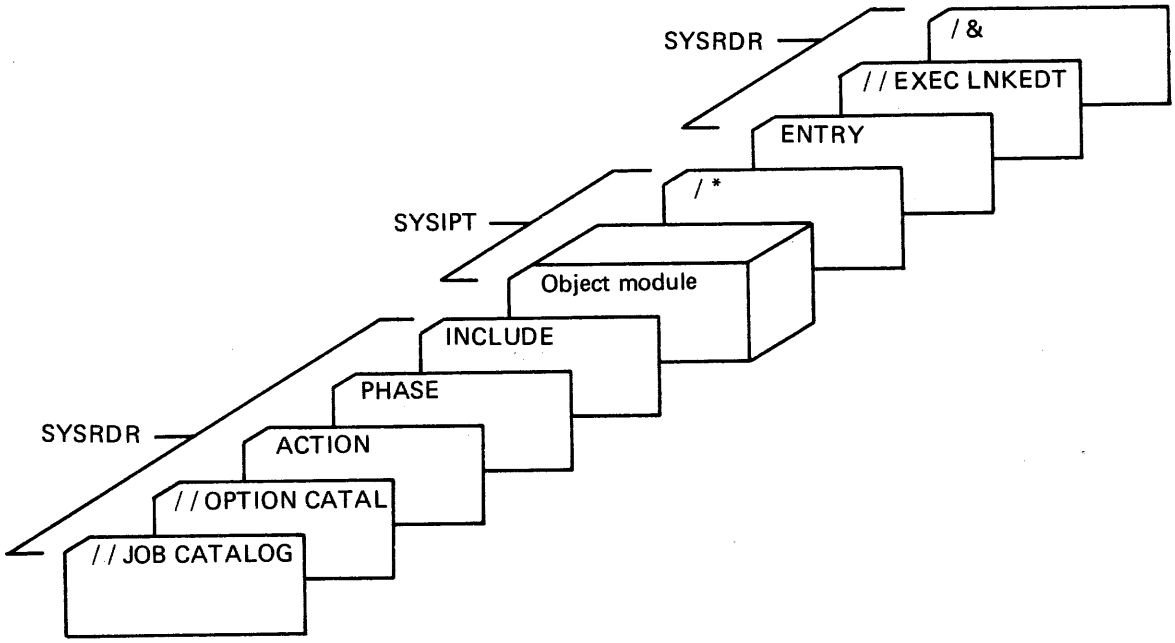


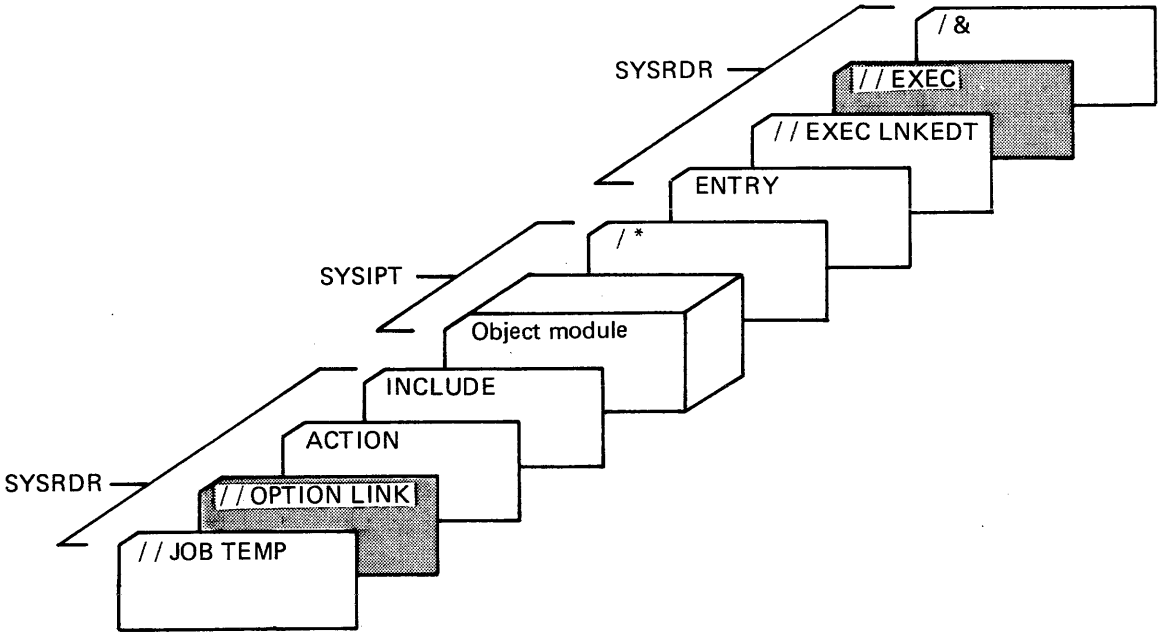
Figure 3-25. A Job Stream to Catalog a Program into the Core Image Library

Assemble (or Compile), Link-Edit, and Execute

You can also combine the job steps described above with a job step for assembly (or compilation) of your source program. This is especially useful when you are developing a program. Figure 3-27 shows how your job stream should be set up. The shaded portions of the figure illustrate how this job stream differs from that shown in Figure 3-26. Linkage editor control statements are not required when linking single-phase programs temporarily into the core image library.

You direct the language translator to write the object module directly onto SYSLNK by specifying the LINK option at the beginning of the job. After the linkage editor processed the input from SYSLNK, your program is loaded for execution.

Instead of submitting three job steps, you may specify the GO parameter in the EXEC statement that invokes the assembler (compiler). This causes the linkage editor and your executable program to be invoked automatically. Only the source program and any additional data for the go step are required. For multiple assemblies (compilations), an OPTION LINK statement must precede the first EXEC statement for an assembly or compilation. This is true also when linkage editor control statements like INCLUDE or PHASE are used. If no LINK option is set, the GO parameter will be in effect only for the EXEC statement it appears on, and the ACTION default will be set to NOMAP (linkage editor control statements are described below, under section *Preparing Input for the Linkage Editor*).



The // EXEC statement (without a program name operand) causes this program to be loaded for execution immediately.

The // OPTION CATAL statement may also be used in this job stream. In this case, the program that was cataloged (permanently) is executed immediately. When // OPTION CATAL is specified a PHASE statement is required.

Figure 3-26. A Job Stream to Link-Edit a Program for Immediate Execution

When you make use of the GO parameter, your executable program has to run in virtual mode, and the partition GETVIS area available to this program will be of the IBM set default size unless you overrode that value through the SIZE command.

If errors occur in one job step causing an abnormal termination, the remaining job steps are ignored. Certain linkage editor errors do not cause job step termination. If you do not want to execute the program when these errors occur, you may specify ACTION CANCEL after the // OPTION LINK.

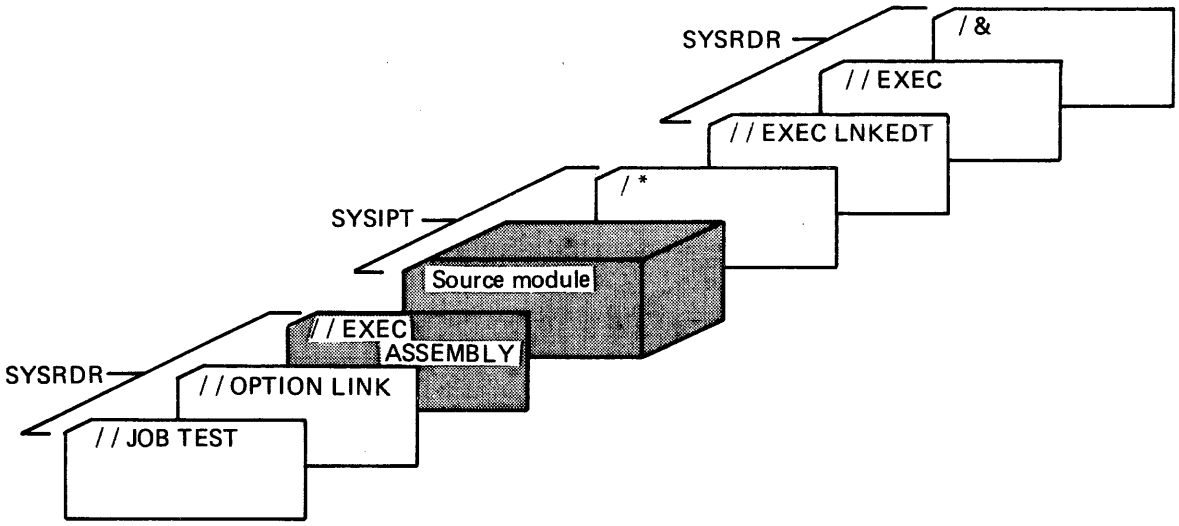


Figure 3-27. A Job Stream to Assemble, Link-Edit, and Execute

Processing Requirements for the Linkage Editor

The linkage editor can be executed in any partition. When the linkage editor is executed in a foreground partition, a private core image library (SYSCLB) must be uniquely assigned to that partition and phases are placed there. When the linkage editor is executed in the background partition where no private core image library is assigned, phases are placed into the system core image library; phases are placed into a private core image library also from the background partition if such a library is uniquely assigned to that partition.

Symbolic Units Required

The linkage editor requires the following symbolic units:

SYSIPT	Module input (if any)
SYSLST	Programmer messages and listings (if SYSLST is not assigned, no map is printed and programmer messages appear on SYSLOG)
SYSLOG	Operator messages
SYSRDR	Control statement input (via job control)
SYSLNK	Input to the linkage editor
SYS001	Work file.

Note that SYSRDR and SYSIPT may contain input for the linkage editor. This input is written on SYSLNK by job control.

If output from the linkage editor is to be placed in a private core image library, the following symbolic unit is required:

SYSCLB The private core image library. It may be assigned anywhere in the job stream but before job control reads the // EXEC LNKEDT statement.

If object modules from a private relocatable library are to be link-edited, the symbolic unit SYSRLB must be assigned.

Preparing Input for the Linkage Editor

The input you prepare for the linkage editor consists of job control statements, linkage editor control statements, and object modules. Job control reads the job control statements and the linkage editor control statements from the device assigned to SYSRDR and object modules from SYSIPT. The linkage editor control statements and object modules are copied onto the disk extent assigned to SYSLNK.

The linkage editor control statements direct the execution of the linkage editor. The statements are: ACTION, ENTRY, INCLUDE, and PHASE. A description of how to prepare these control statements is given on the following pages. Here, the various operands of the control statements are described under headings that indicate their function.

Assigning a Name to a Program Phase

Each program phase the linkage editor is to produce should have a name, which you specify in the PHASE statement. When a phase is cataloged in the core image library, the phase name identifies that phase for subsequent retrieval. In other words, the same phase name you supplied in the PHASE statement when permanently cataloging the initial or only phase of a program must be used as the operand in the EXEC job control statement or in a FETCH or a LOAD macro instruction.

When you catalog a phase with the same name as a phase already residing in the core image library, the earlier entry with the same phase name is deleted from the core image directory (and, if applicable, the system directory list in the SVA) and cannot be accessed again.

The choice of a phase name has a bearing on retrieval efficiency and the subsequent use of the librarian programs. Job control scans the directory of the appropriate library for all phases starting with the same four characters as the program name specified in the EXEC statement.

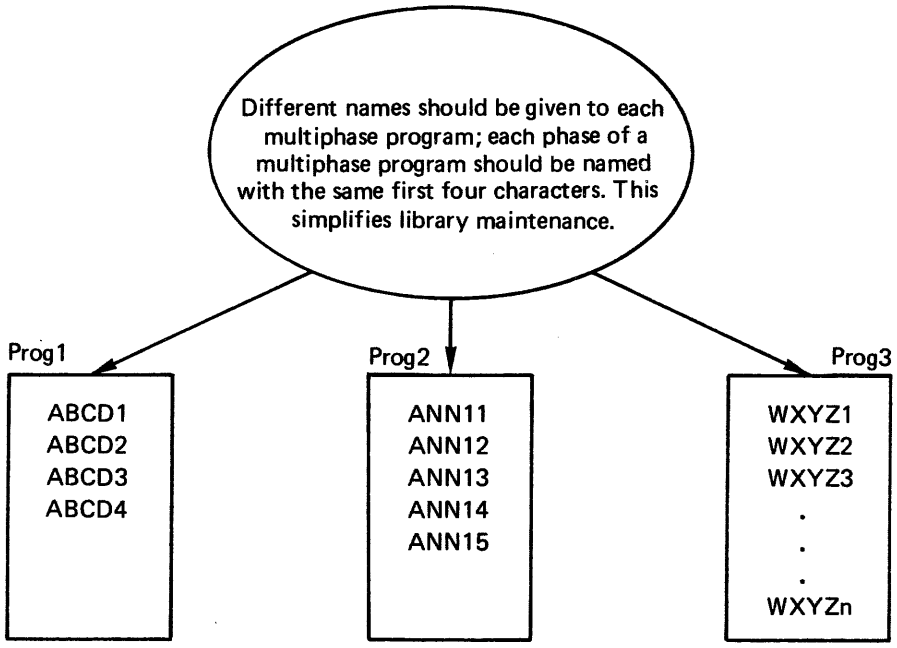
Any phases with the same first four characters of their phase name will be classified as a multiphase program. When a phase of a multiphase program is fetched, the available address space must be large enough to contain the largest of those phases even if that phase is not part of the program which is being executed.

Phase names may be formed only from characters 0-9, A-Z, /, #, $, and @. Otherwise, the phase statement is invalid. The names "S", "ALL", and "ROOT" are invalid phase names.

In choosing a name for any multiphase program, make sure that the first four characters are the same for all phases of that program but

different from those of other programs. Such names simplify the deleting, displaying, punching, merging, and copying of the entire program. Figure 3-28 summarizes the above recommendations.

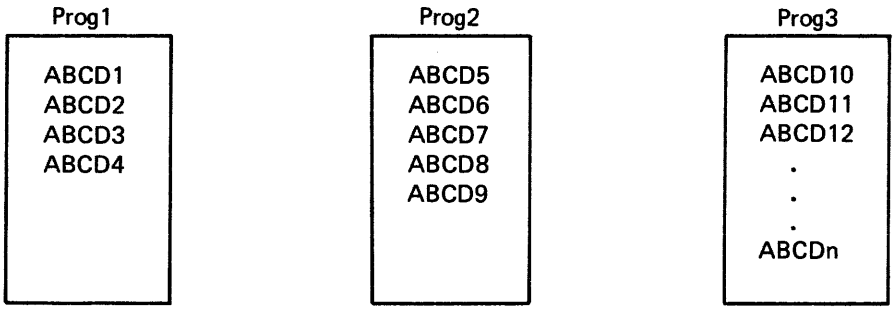
Note: *A phase name "//" cannot be placed into the System Directory List via the job control command SET SDL.*



Simplified library maintenance means, for example, that one simple control statement deletes all phases of Prog1:

```
DELETC ABCD.ALL
```

If the programs had been named:



the statement required to delete Prog1 would be:

```
DELETC ABCD1, ABCD2, ABCD3, ABCD4
```

Figure 3-28. Naming Multiphase Programs

Defining a Load Address for a Phase

For link-editing, you specify where your program is to be loaded for execution. You have several choices.

A phase can be link-edited to be loaded into and executed from:

- a specific partition's address area
- the shared virtual area
- an absolute address.

A phase can be link-edited as a relocatable phase, a self-relocating phase, or a non-relocatable phase.

The load address you specify in the PHASE statement determines the relocatability status of the link-edited phase:

- For a phase to be relocatable, specify a symbolic address with or without a displacement.
- For a phase to be non-relocatable, specify an absolute address.
- For a phase which you wrote to be self-relocating, specify +0.

Full details on possible load address (also called origin address) specifications are given in *DOS/VSE System Control Statements*.

Link-Editing for Execution at Any Address. If the linkage editor determines that a phase is to be given the relocatable format, it flags the core image directory entry for that phase, and inserts the relocation information behind the text of the phase in the core image library.

When a relocatable phase is link-edited, it is assigned a load address relative to the partition's address area in which the linkage editor was executed. When executing the phase from the same partition, relocation is not required. (This assumes that virtual storage allocations were not changed between link-editing and executing the phase.)

Executing the phase from a different partition requires relocation by DOS/VSE. Loading and relocating a phase takes more processing time than just loading. You should attempt to link-edit phases so as to minimize the relocation overhead.

Assuming that you have a program that is executed from partition F2 most of the time, but occasionally executed in another partition. Link-editing the program from the F2 partition would eliminate the relocation overhead when the program is executed from the F2 partition. If you decide that all link-editing should be done from one partition (for example the BG partition) you may use the linkage editor ACTION control statement to associate phases with other partitions. For example:

```
ACTION F2
```

specifies that the linkage editor should use the current address allocation of the F2 partition in determing the phase's load point.

Link-Editing for Execution in a Specific Partition's Address Space. Unless

otherwise specified in the PHASE statement, a program is link-edited to execute in the same partition in which the linkage editor function occurs. In 370 mode, when the linkage editor is running in real mode, the program is link-edited to execute in that partition's virtual address space.

By using the ACTION statement with one of the partition identifiers (BG, F1, F2, etc.), however, you specify the partition in which the program is to be executed. It is necessary to specify a partition identifier only if the "run" partition differs from the partition in which the linkage editor is being executed.

Use of the ACTION statement with a foreground partition identifier requires that the partition be allocated; if not, the ACTION statement is ignored.

An ACTION statement with a partition identifier is effective only for those phases designated to be loaded at an address relative to the beginning of a partition; that is, for those phases with a load address specification in the form of a symbol with or without a displacement.

An example of the use of the ACTION statement follows. Assume that three partitions are allocated: background, F2, and F1. If you are executing the linkage editor in the background, the statement PHASE PROG1,S causes PROG1 to have its origin at the beginning of the virtual address space of the background partition (plus the BG save area and the BG label area). The sequence

```
ACTION F1
PHASE PROG1,S
```

causes PROG1 to have its origin at the beginning of the address space of the F1 (plus the length of the F1 save area and the F1 label area).

Link-Editing for Inclusion in the Shared Virtual Area. If a relocatable phase is also reenterable, it can be included in the shared virtual area (SVA). Phases resident in the SVA can be shared concurrently by more than one partition. It is advantageous to include frequently-used phases in the SVA because these are then resident when requested for execution (they are not reloaded from the core image library). All phases resident in the SVA must also be cataloged in the system core image library.

To indicate that a phase should reside in the SVA, you must specify the SVA operand in the PHASE statement when cataloging the phase. This operand is ignored if the phase is not relocatable; otherwise, the SVA operand is accepted and the phase is said to be SVA-eligible.

The linkage editor cannot check whether a phase is reenterable; however, a protection check can occur when executing a phase from the SVA that modifies itself and therefore is not reenterable. Because the SDL is sorted prior to the loading of phases into the SVA, the packaging of phases to be executed together should be done using the linkage editor.

Immediately after an SVA-eligible phase is cataloged into the system core image library, it is loaded into the SVA *if* this phase is listed as SVA-eligible in the system directory list (SDL). See the section *Building the SDL and Loading the SVA* earlier in this chapter.

Link-Editing for Execution at an Absolute Address. If you specify an absolute address in the PHASE statement, your program can be loaded only at this address at the time of program execution. Not only must the address you specify be within the address range of your installation's virtual storage, but also the entire program must be included within the boundaries of the area allocated to the partition where you request the program to be executed.

In 370 mode, if you wish to force a phase to be executed in real mode, you may link-edit that phase with the absolute address of a given partition's real address space.

Using Self-Relocating Programs. You should identify self-relocating programs by a PHASE statement with an origin point of +0:

```
PHASE PROGA,+0
```

The linkage editor assumes that the program is loaded at location zero, and computes all addresses accordingly. The job control EXEC function recognizes a zero phase address and adjusts the origin address to compensate for the current partition boundary save area and label area. It then gives control to the updated entry address of the phase.

Building Phases from Object Modules with the INCLUDE Statement

You indicate which object modules or parts of object modules are to be included in a phase by specifying the INCLUDE statement. The format of the INCLUDE statement indicates the location of the modules. The object modules can be either on the card reader, tape unit, disk or diskette device assigned to SYSIPT, or in the relocatable library, or on the disk device assigned to SYSLNK. The modules are extracted in the same order as the INCLUDE statements are issued.

Including Modules from SYSIPT. If the object modules you want to include in a phase are on the SYSIPT file, specify the INCLUDE statement without operands. Job control copies the data from SYSIPT until it encounters end-of-data (/*).

Including Modules from the Relocatable Library. You may want to include in a phase object modules or parts of an object module that are cataloged in the relocatable library. To include an entire module, specify the module name in the INCLUDE statement. To include part of a module, specify the name of the module followed by the names of the control section(s) you wish to be included.

Including Parts of Modules from SYSLNK. You do not need an INCLUDE statement unless you want to change the sequence of control sections or to extract certain control sections from an object module. For either of these cases, specify the names of the control sections in an INCLUDE statement.

Linkage Editor Storage Requirements

The storage requirements for a link-edit run depend on the number of PHASE statements and number of ESD items processed during a link-edit run.

In a minimum size virtual partition of 128K the linkage editor can process for example 10 phases with a total number of 380 unique ESD items.

A unique ESD item is defined as being an occurrence in the control dictionary. All symbols that appear in the MAP are unique occurrences. A symbol that occurs several times in the input stream is normally incorporated into a unique ESD item. However, if the same symbol occurs in different phases (for example, control sections), each resolved occurrence of the symbol within a different phase is a unique ESD item.

You can use the following formula for storage estimates:

$56{,}000 + 40 * x + 20 * y \leq P$

x = number of PHASE statements

y = total number of unique ESD items

P = storage available to the partition.

To execute the linkage editor in real mode requires a 64K allocation of processor storage.

The AUTOLINK Feature

For each phase the automatic library look-up feature (referred to as AUTOLINK) collects any external references and attempts to resolve them. An external reference is an ER item in the control dictionary that has not been matched with an entry point. AUTOLINK searches the private relocatable directory (if assigned) and then the system relocatable directory until a cataloged module with the same name as the external reference is found (or the end of the directory is reached). If found, the module is included in the phase (autolinked). This retrieved module must have an entry point matching the external reference in order to resolve its address.

The following examples show how the AUTOLINK feature works.

Assume that the relocatable library contains the following:

Module Name	Entry Names	External References
A	A, B, C	
D		A
E		B
F		A, C

Examples:

In your linkage editor input stream you specify INCLUDE D. A will be autolinked (included with module D) because the external reference A is also a module name in the relocatable library.

If you specify INCLUDE E, then A will not be autolinked because the external reference B does not relate to a module name. In this case, you

must also specify INCLUDE A, so that the external reference B can be resolved. No autolink is required.

If you specify INCLUDE D and INCLUDE E, then A will be autolinked by module D and the external reference B in module E can then be resolved.

If you specify INCLUDE F, then module A will be autolinked by the reference to A, and the reference to C will also be resolved.

Suppressing the AUTOLINK Feature. You can suppress the AUTOLINK feature in two ways:

- By specifying NOAUTO in a PHASE statement, AUTOLINK is canceled for that phase only.
- By specifying NOAUTO in the ACTION statement, AUTOLINK is canceled for this execution of the linkage editor. By writing a weak external reference (WXTRN), AUTOLINK is canceled for one symbol.

You can do this in assembler language by specifying for example:

```
DC    A(LABEL)
WXTRN    LABEL
```

or

```
DC    V(LABEL)
WXTRN    LABEL
```

For more information, refer to the assembler language publications.

NOAUTO can be used to force a CSECT into a specific phase within an overlay structure. For example, four phases of a program have a V-type address constant called PETE, but in the overlay structure you want the coding for PETE included only in the third phase.

```
PHASE PROGA,*,NOAUTO
PHASE PROGB,*,NOAUTO
PHASE PROGC,*
PHASE PROGD,*,NOAUTO
```

cause PETE to be included in PROGC only.

Reserving Storage for Label Processing

If you operate in an SCP only environment and your program uses standard labeled tape files or nonsequential DASD files (direct access, indexed sequential, or DTFPH with all packs mounted), you must ensure that storage is reserved for the tape and disk labels. These labels are brought into the label save area of the partition containing your program when the file is opened.

You reserve a label save area by specifying the LBLTYP job control statement. The amount of storage you specify to be reserved must be large enough to contain all the labels of the file with the most extents processed by the program. The operand specified in the LBLTYP statement for tape files is different from that for DASD files. For their formats, refer to *DOS/VSE System Control Statements*.

The LBLTYP statement is to be submitted immediately before the // EXEC LNKEDT statement. If your program is self-relocating, however, submit the LBLTYP statement immediately before the // EXEC statement for your program.

The LBLTYP statement is not required if only unlabeled tape files, sequential DASD files, or VSAM files, are to be processed. For more information on file organizations, refer to the *DOS/VSE Data Management Concepts*.

VSE/Advanced Functions dynamically allocates a label processing area, based on DLBL/EXTENT information. Therefore, the LBLTYP statement is not needed. If, however, the LBLTYP statement is included, the linkage editor and loader makes provisions for the requested storage to be allocated in the label save area, but the area will not be used for label processing.

Specifying Linkage Editor Aids for Problem Determination or Prevention

You can specify that the linkage editor aid you in avoiding certain problems in your programs or determining what they are. The actions discussed below are CLEAR, MAP, and CANCEL, which may be specified as operands of the ACTION statement.

Clearing the Unused Portion of the Core Image Library

If you used DS (define storage) statements in your source module, it may be advantageous to fill these areas with binary zeros when the program is link-edited. This eliminates the risk that residual data from a previously linked program be loaded with your program when it is executed. Such irrelevant data might disrupt your program considerably. By specifying CLEAR in the ACTION statement, you request that the unused portion of the core image library is to be set to binary zeros.

Because CLEAR is a time-consuming function, you might want to use DC statements instead of DS statements when designing future programs; but do use ACTION CLEAR when cataloging a supervisor.

Obtaining a Storage Map

You can obtain a linkage editor storage map and a listing of linkage editor error diagnostics, which assist you in determining the reasons for particular errors in your program. If SYSLST is assigned, ACTION MAP is the default. You can specify ACTION NOMAP if you are not interested in this service of the linkage editor.

The storage map contains such information as:

- The lowest and highest addresses that each phase occupies in the partition for which it is link-edited.
- The starting disk address of the phase in the core image library.
- The names of all control sections and entry points, their load addresses and relocation factors.

- The names of all external references that are unresolved.
- An indication whether the phase is relocatable, non-relocatable, self-relocating, or SVA eligible.

The error diagnostics warn you, for example, if:

- The ROOT phase has been overlaid.
- A control section has a length of zero.
- An address constant could not be resolved.

A sample storage map, together with a description of how to interpret it, is included in *DOS/VSE Serviceability Aids and Debugging Procedures.*

Terminating an Erroneous Job

If errors are present in the input to the linkage editor, the output of the linkage editor will most likely also be erroneous. If you specify CANCEL in the ACTION statement, the entire job is terminated when any of the type of errors represented by messages 2100I through 2170I occurs. Refer to these messages in *DOS/VSE Messages.*

Designing an Overlay Program

The nature of virtual storage makes it unnecessary to write programs in an overlay structure, because virtual partitions can be allocated to accommodate very large programs.

Overlay programs consist of control sections organized in an overlay tree structure. An example of an overlay tree structure is shown in Figure 3-29. This structure does not imply the order of execution, although the root phase is normally the first to receive control.

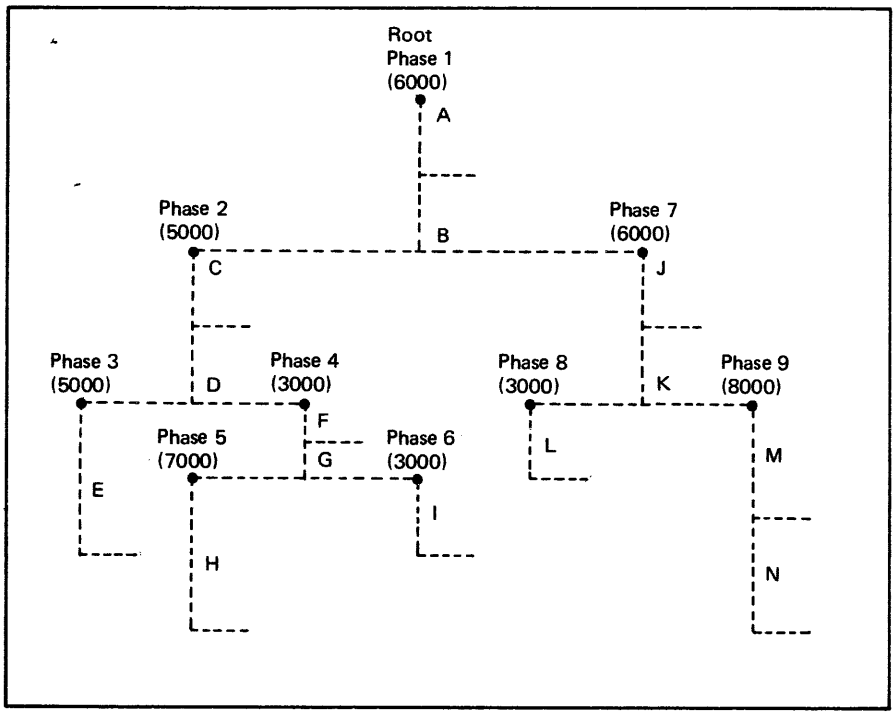
The manner in which control should pass between control sections is discussed in the section *Using FETCH and LOAD Macros.*

Relating Control Sections to Phases

After having organized the control sections of your program into an overlay tree structure, you must prepare a corresponding set of linkage editor control statements.

Link-edit your complete overlay program in a single job step, and conversely, do not include in this job step any phases that are not related to the overlay. Otherwise, the linkage editor may not be able to resolve external references correctly.

The PHASE and INCLUDE statements you prepare are critical to ensure the overlay tree structure you designed. Figure 3-30 is an example of the job stream that ensures the overlay tree structure shown in Figure 3-29.



The letters A through N represent control sections, which are organized to form nine phases in one program. The root phase resides in storage during the entire execution of the program. The remaining phases can overlay each other during execution.

You must guarantee a partition size that is equal to the longest combination of phases that can possibly reside in storage together, namely, phases 1, 2, 4, and 5, which total 21,000 bytes. If the program had not been organized in an overlay structure, it would have required an address space of 46,000 bytes.

Figure 3-29. Overlay Tree Structure

```
// JOB OVERLAY
// OPTION CATAL
    PHASE    PHASE1,ROOT          PHASE1 stays in storage during
    INCLUDE ,(CSECTA,CSECTB)      execution of the entire program.
    PHASE    PHASE2,*             PHASE2 is to be loaded
    INCLUDE ,(CSECTC,CSECTD)      immediately behind PHASE1.
    PHASE    PHASE3,*             Since PHASE3 needs PHASE2, PHASE3
    INCLUDE ,(CSECTE)             is not allowed to overlay PHASE2.
    PHASE    PHASE4,PHASE3        PHASE4 will occupy the same
    INCLUDE ,(CSECTF,CSECTG)      storage locations as PHASE3.
    PHASE    PHASE5,*             PHASE5 will be loaded
    INCLUDE ,(CSECTH)             immediately behind PHASE4.
    PHASE    PHASE6,PHASE5        PHASE6 will be loaded at the
    INCLUDE ,(CSECTI)             same address as PHASE5.
    PHASE    PHASE7,PHASE2        PHASE7 will be loaded at the
    INCLUDE ,(CSECTJ,CSECTK)      end of the root phase.
    PHASE    PHASE8,*             PHASE8 will be loaded at the
    INCLUDE ,(CSECTL)             end of PHASE7.
    PHASE    PHASE9,PHASE8        PHASE9 will overlay
    INCLUDE ,(CSECTM,CSECTN)      PHASE8.
    INCLUDE
       (Object modules containing CSECTs A through N)
/*
// LBLTYP
// EXEC LNKEDT
/&
```

Figure 3-30. Link-Editing an Overlay Program

Using FETCH and LOAD Macros

During execution, an overlay program communicates with the supervisor to request that a subsequent phase be brought into the partition. You include FETCH or LOAD macros within your phases for this purpose.

Use a LOAD macro in a phase that is to remain in control after the requested phase is brought into the partition. A phase loaded by the LOAD macro must be relocatable.

Use a FETCH macro if you want the requested phase to gain control immediately after it is brought into the partition. If a phase loaded by the FETCH macro is relocatable, it will be relocated if necessary. You cannot issue a FETCH macro for a self-relocating phase.

Parameters in FETCH and LOAD allow use of the SDL (system directory list) or the LDL (local directory list), thereby reducing fetching and loading time.

DOS/VSE Macro Reference contains details on the format of the FETCH and LOAD macros.

Examples of Linkage Editor Applications

The linkage editor examples on the following pages illustrate the use of and relation between linkage editor and job control statements. After studying

these examples, you should be able to set up a link-edit job for your own purposes.

Catalog to the System Core Image Library Example

```
   // JOB CATALCIL
   *  LINK EDIT AND CATALOG TO CORE IMAGE LIBRARY
   *  SINGLE PHASE, ELIGIBLE FOR LOADING INTO SHARED
   *  VIRTUAL AREA, MULTIPLE OBJECT MODULES,
   *  MIXTURE OF CATALOGED AND UNCATALOGED OBJECT MODULES
   *  LABELED TAPE FILES AND SEQUENTIAL DASD FILES TO
   *  BE PROCESSED
1  // ASSGN SYSLNK,190
2  // OPTION CATAL
3     PHASE PROGB,*,SVA
4     INCLUDE
      Object deck
   /*

      INCLUDE SUBRX
      INCLUDE SUBRY
      INCLUDE
      Object deck
   /*
5  // LBLTYP TAPE
6  // EXEC LNKEDT
   /&
```

Explanation for Catalog to the System Core Image Library. This example illustrates the cataloging of a single phase composed of multiple object modules. These modules are located in the input stream and the relocatable library. Labeled tape files and sequential DASD files are processed when the phase is executed.

Statement 1: The statement is required, unless SYSLNK is permanently assigned. If the statement is included, it must precede the OPTION statement (Statement 2).

Statement 2: The OPTION CATAL statement sets the LINK switch, as well as the CATAL switch. If SYSLNK is not assigned, the statement is ignored. The linkage editor control statements are not accepted unless the OPTION statement is processed. Link-editing and cataloging to the core image library is requested.

Statement 3: Only one PHASE is produced. It is cataloged to the core image library and may be retrieved by the name PROGB. Because there is only one phase, the origin point * indicates that this phase originates at the starting address of the partition plus the length of the partition save area, the label area (if any), and the COMMON pool (if any). The SVA operand indicates that the phase should be considered SVA-eligible. If the phase name PROGB is already entered as SVA-eligible in the system directory list, PROGB is loaded into the shared virtual area immediately after it is cataloged into the system core image library. (This would not occur if PROGB is link-edited with OPTION LINK.)

Note: *COMMON is used by FORTRAN programs to store data shared by multiple programs.*

Statement 4: Four modules make up this phase. The first and last are not cataloged in the relocatable library; therefore the object decks must be on SYSIPT, and each must be followed by the end-of-data record (/*). SUBRX and SUBRY were cataloged previously to the relocatable library by those names. Job control puts the uncataloged modules on SYSLNK in place of their INCLUDE statements. Job control copies onto SYSLNK the INCLUDE statements for the cataloged modules.

Statement 5: The LBLTYP statement, which is required in an SCP only environment, has the operand TAPE, rather than NSD, because labeled tapes and sequential DASD files are processed when the phase is executed. 80 bytes are reserved ahead of the actual phase for label information. LBLTYP NSD is also satisfactory because it generates a minimum of 104 bytes, and tapes require only 80.

Statement 6: The EXEC LNKEDT statement causes DOS/VSE to bring in the linkage editor program. SYSLNK now becomes input to the linkage editor. It contains:

```
PHASE PROGB,*,SVA
First uncataloged relocatable deck
INCLUDE SUBRX
INCLUDE SUBRY
Second uncataloged relocatable deck
ENTRY
```

The modules are link-edited into one phase so that they occupy contiguous addresses in the sequence in which they appear in the input stream. When the linkage editing is completed, cataloging to the core image libary occurs because of the CATAL option.

In addition, the linkage editor prints a status report that reflects the usage and available space in the core image library. (This does not occur in a LINK situation.)

The example can be modified to illustrate a catalog-and-execute operation by inserting the following statements between the EXEC LNKEDT and / & statements:

- Any job control statements required for execution of PROGB
- A // EXEC statement
- Card reader input for PROGB, if any.

The example does not include an ENTRY statement. Job control, therefore, writes an ENTRY statement on SYSLNK instructing the linkage editor that:

- There is no more input on SYSLNK.
- The entry point defined in the source program should be the entry point of the produced phase.

Catalog to a Private Core Image Library Example

```
   // JOB CATLCIL
   *   LINK EDIT AND CATALOG TO PRIVATE CORE IMAGE LIBRARY
   *   LINKAGE EDITOR EXECUTING IN FOREGROUND
   *   SINGLE PHASE, ALIGNED ON A PAGE BOUNDARY, MULTIPLE
   *   OBJECT MODULES, FOREGROUND PROGRAM
   *   MIXTURE OF CATALOGED AND UNCATALOGED OBJECT MODULES
   *   LABELED TAPE FILES AND SEQUENTIAL DASD FILES TO
   *   BE PROCESSED
1    ASSGN SYSCLB,130
2   // ASSGN SYSLNK,190
3   // OPTION CATAL
4       PHASE PROGB,S,PBDY
5       INCLUDE
        object deck
    /*
        INCLUDE SUBRX
        INCLUDE SUBRY
        INCLUDE
        Object deck
    /*
6   // LBLTYP TAPE
7   // EXEC LNKEDT
    /&
```

Explanation for Catalog to Private Core Image Library. This example illustrates the execution of the linkage editor in a foreground partition; therefore the phase is cataloged to a private core image library. The phase being cataloged is the same as that in the previous example where the linkage editor was executed in the background.

Statement 1: The label information for SYSCLB must be stored in the label information area or, if appropriate, DLBL and EXTENT statements must precede the ASSGN SYSCLB command.

Statements 2 through 7: They are the same as statements 1 through 6 in the preceding example (Catalog to the System Core Image Library).

Just like the preceding example, so can this example be modified to illustrate a catalog-and-execute operation.

Link-Edit and Execute Example

```
   // JOB LINKEXEC
   *  LINK EDIT AND EXECUTE SINGLE PHASE, SINGLE OBJECT
   *  MODULE NOT CATALOGED, BACKGROUND PROGRAM
   *  NONSEQUENTIAL DASD & LABELED TAPE FILES TO
   *  BE PROCESSED
1  // ASSGN SYSLNK,190
2  // OPTION LINK
3       PHASE PROGA,*
4       INCLUDE
       object deck
   /*
5  // LBLTYP NSD(2)
6  // EXEC LNKEDT
7       Any job control statement required for execution
       such as ASSGN or label statements
8  // EXEC
       input data as required
   /*
   /&
```

Explanation for Link-Edit and Execute. This example illustrates the basic concept of link-editing and executing by using a single phase that is constructed from a single object module contained in punched cards. Labeled tape and nonsequential DASD files are to be processed when the phase is executed. No more than two extents are used by any DASD file.

Statement 1: No assignments are necessary because the system units required for link-editing are assumed to be permanently assigned. An ASSGN for SYSLNK is included to illustrate its position relative to the OPTION statement in case an assignment is required.

Statement 2: The statement indicates that a link-edit operation is to be performed. If SYSLNK has not been assigned, the statement is ignored. Linkage editor control statements are not accepted unless the OPTION statement is processed. Because the option is LINK, and not CATAL, only link-editing will be performed.

Statement 3: The PHASE statement is copied on SYSLNK. Job control checks only the first operand; remaining operands are checked by the linkage editor when that program uses SYSLNK as input.

Only one phase is built by the linkage editor because only one PHASE statement is submitted for the entire run. The name of this phase is PROGA, as specified in the first operand. The second operand indicates the origin point for the phase. Because an * has been used, the phase begins in the next storage location available, with forced doubleword alignment. Because this is the first and only phase, it is located at the beginning of the partition plus the length of the save area and label area (reserved by LBLTYP) plus the length of any area assigned to the COMMON pool (as designated by a CM entry in the object module).

A displacement, either plus or minus, may be used with the *, such as *+1024. This causes the origin point of the phase to be set relative to the * by the amount of the displacement.

Statement 4: The INCLUDE statement has no operands so DOS/VSE reads the records from SYSIPT and writes them on SYSLNK until SYSIPT has an end-of-data (/*) record. The data on SYSIPT is expected to be the object module in card image format that is used in this linkage editor operation.

Statement 5: The LBLTYP statement, which is required in an SCP only environment, causes job control to compute the number of bytes needed for label data in the program that is link-edited. The calculation is saved by job control and passed on first to the linkage editor and later to LIOCS.

Statement 6: On encountering the EXEC LNKEDT statement, job control writes an ENTRY statement with no operand on SYSLNK and causes DOS/VSE to bring in the linkage editor program.

Using the data just placed on SYSLNK as input, the linkage editor produces executable code. The output is placed in the next available space of the core image library (immediately after the last cataloged phase). This is true regardless of whether the program is cataloged permanently (OPTION CATAL) or temporarily (OPTION LINK). However, if OPTION LINK is specified, the temporarily cataloged program is overlayed by the next program that is link-edited. A program that is cataloged temporarily must be link-edited each time it is used. No ACTION options are specified. Therefore, in resolving the external references, the system makes use of the AUTOLINK feature. Error diagnostics and a storage map are written on SYSLST, assuming that SYSLST is assigned.

Statement 7: Because the program is not cataloged, it must be executed immediately. Any pertinent job control statements are entered at this point.

Statement 8: An EXEC statement with no program name operand indicates that the phase to be executed was just link-edited. Therefore, no search of the core image directory for linked phases is required, and DOS/VSE brings the program into storage and transfers control to its entry point. Because the automatic ENTRY statement is in effect for this example, the entry point is the address specified in the program.

This example can be modified to illustrate the following:

1. *Catalog and execute.* To cause this phase to be cataloged permanently, change the OPTION statement (2) from LINK to CATAL.
2. *Catalog only.* To catalog only, change the OPTION statement (2) from LINK to CATAL and remove all statements following the EXEC LNKEDT statement (6) up to the / & statement.
3. *Include object module from relocatable library.* The name of the object module in the relocatable library must be supplied by an additional INCLUDE statement. If the name is RELOCA, the statement is INCLUDE RELOCA. This form of the INCLUDE statement is written on SYSLNK when it is read by job control. The linkage editor retrieves the object module when it encounters the INCLUDE statement because it uses SYSLNK for input.

Compile and Execute Example

```
   // JOB COMPEXEC
   *  COMPILE OR ASSEMBLE, LINK EDIT AND EXECUTE
   *  SINGLE PHASE, MULTIPLE OBJECT MODULES, BACKGROUND
   *  PROGRAM SEQUENTIAL DASD FILES TO BE PROCESSED
   *  INPUT TO LINKAGE EDITOR FROM LANGUAGE TRANSLATOR,
   *  RELOCATABLE LIBRARY AND SYSIPT
1  // ASSGN SYSLNK,190
2  // OPTION LINK
3       PHASE PROGA,S
4  // EXEC FCOBOL
       COBOL source statements
   /*
5      INCLUDE SUBRX
       INCLUDE
       object module
   /*
6      ENTRY BEGIN1
   // EXEC LNKEDT
7      Any job control statements required for PROGA
       execution
   // EXEC
       Any input data required for PROGA execution
   /*
   /&
```

Explanation for Compile and Execute. The language translators provide the option of placing their output on SYSLNK. Because the linkage editor uses SYSLNK for input, a program can be assembled or compiled, link-edited and executed, all in one job.

All three sources of object module input to the linkage editor are used: SYSIPT, the relocatable library, and the output from a language translator. It is assumed that the phase is executed in the background partition, and that only sequential DASD files or unlabeled tape files are processed.

Statement 1: The SYSLNK assignment is given to show the position of ASSGN statements relative to the OPTION statement. ASSGN statements are not required if they are permanent assignments.

Statement 2: The statement is required.

Statement 3: The PHASE statement must always precede the relocatable modules to which it applies; it is written on SYSLNK first for later use by the linkage editor. S is the origin point, that is, the phase originates with the first doubleword in the partition plus the length of the partition save area and label area, plus the length of the area assigned to the COMMON pool (if any). This gives the same effect as * gives for a single phase or the first phase of a multiphase link edit run. As with the *, the S may be used with a relocation factor, for example, S+1024.

Statement 4: The appropriate language translator is called (in this case, DOS/VS COBOL). The normal rules for compiling are followed; the source deck must be on the unit assigned to SYSIPT and the /* defines the end of the source data. The output of the language translator is written on SYSLNK.

Statement 5: The INCLUDE SUBRX statement is written on SYSLNK. The linkage editor retrieves the named module from the relocatable library. Because it has no operand, the next INCLUDE statement signifies that the relocatable module is on SYSIPT. The data on SYSIPT is copied on SYSLNK up to the /* statement.

Statement 6: The ENTRY statement is written on SYSLNK as the last linkage editor control statement. The symbol BEGIN1 must be the name of a CSECT or a label definition (which occurs in an ENTRY source statement) defined in the first or only phase. The address of BEGIN1 becomes the transfer address for the first or only phase of the program. The ENTRY is used to provide a specific entry point rather than to use the point specified in the program.

Statement 7: No LBLTYP statement is required because only sequential DASD files are to be processed.

The rest of the statements follow the same pattern as discussed in the Link-Edit and Execute example. The input from SYSLNK to the linkage editor is:

```
PHASE PROGA,S
Relocatable module produced by COBOL compilation
INCLUDE SUBRX
Relocatable module from SYSIPT
ENTRY BEGIN1
```

If certain types of errors are detected during compilation of a source program, the LINK option is suppressed. Under these circumstances the EXEC LNKEDT and EXEC statements are ignored and the message 'STATEMENT OUT OF SEQUENCE' results. This LINK option suppression should be kept in mind if a series of programs is to be compiled and cataloged as a single job. Failure of one job step would cause failure of all succeeding steps.

An OPTION LINK cannot be given if OPTION CATAL is in effect. The message 'STATEMENT OUT OF SEQUENCE' results.

Using the Libraries

After you have planned the size, contents, and location of the libraries (see *Chapter 2, Planning the System*), you need to know how to allocate space to a library, how to create private libraries and how to alter, copy, and inspect the contents of the libraries. All these functions are performed by a group of library processing programs, collectively referred to as the librarian.

Associated with each library is a directory that is located at the beginning of the space allocated to that library. For each element in a library, the corresponding directory contains a unique entry describing the element. A directory entry contains such information as name, disk address, size, load address (core image library only), and version number (relocatable, source statement, and procedure libraries only) of the element. These directory entries are used by the system to locate elements in and retrieve them from a library.

The begin addresses of the individual system library directories are stored in a separate directory, the system directory. At the beginning of each directory is a library descriptor. This entry contains information such as the address of the next available record, the number of active and deleted blocks, and the amount of space allocated to the library. The library descriptor entry comprises the first block of each directory on FBA devices. On CKD devices, the library descriptor information is in the first entry of the core image library directory, and the first five entries of the other library directories.

A core image library normally contains a large number of program phases. Thus, searching for a specific phase can become rather time consuming. To reduce the search time, the core image library directory entries are in alphameric sequence. The second level directory contained in the supervisor assists in locating directory entries. This is discussed in *Second Level Directories for Core Image Libraries* in this manual.

The organization of the directories on SYSRES is shown in Figure 3-31. A more detailed description of the complete SYSRES organization is given in *Appendix A: System Layout on Disk*.

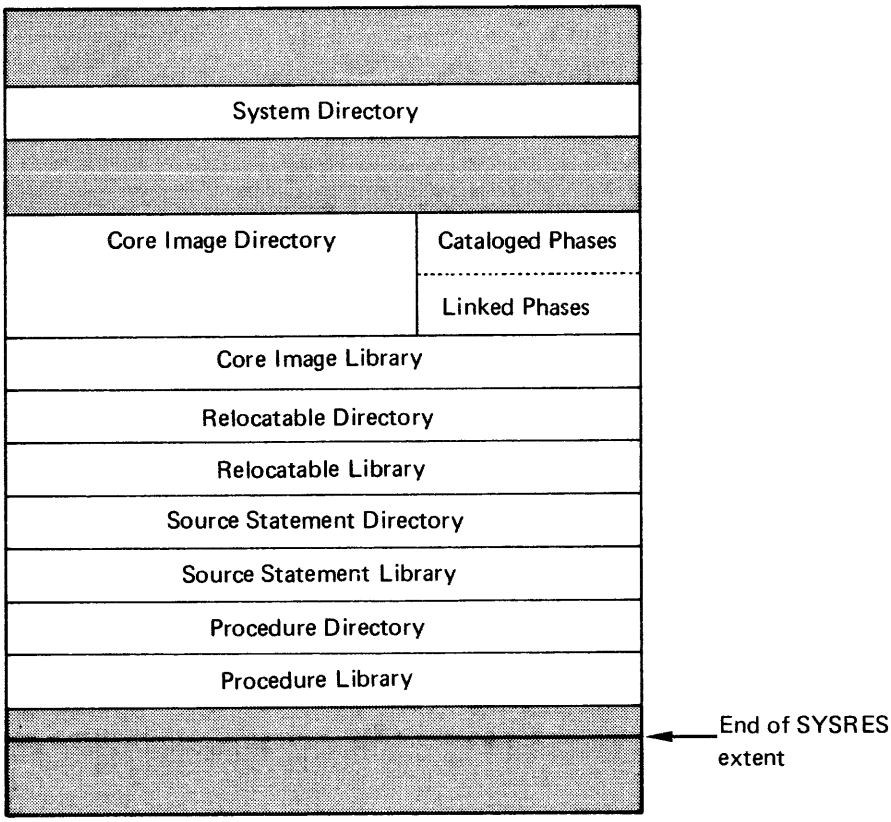


Figure 3-31. Organization of the Directories and Libraries on SYSRES

The Librarian Programs

This section describes how you can manage and control your libraries with the use of the librarian programs. The librarian programs fall into three functional groups: maintenance, organization, and service. The functions are applicable both to the system and private libraries. Figure 3-32 is a summary of the librarian programs, their functions, and the partitions you can use for their execution. The figure also lists the storage requirements for real mode execution: a value (ALLOCR command) to allocate processor storage and a value (SIZE command or SIZE parameter in the EXEC statement) to reserve space for the partition GETVIS area. No special considerations apply to execution in virtual mode; any librarian program will fit into the minimum partition size.

GROUP	PROGRAM NAME	FUNCTIONS	PARTITION	ALLOCR	SIZE
Maintenance	MAINT	Catalog Delete Rename Condense (Note 1) Establish Condense Limit Reallocate (Note 3) Update for Source Statement Library	BG BG BG (Note 2) BG BG BG BG	112K	64K
Organization	CORGZ	Allocate a new SYSRES Create private libraries Transfer elements between any two libraries of the same type	Any (Note 4)	122K	74K
	COPYSERV	Compare library contents and generate input for CORGZ	Any	68K	20K
Service	DSERV	Display the contents of the library directories		68K	20K (Note 5)
	CSERV RSERV SSERV PSERV	Display, punch, or display and punch the contents of the Core image, Relocatable, Source statement, or Procedure library	Any	68K	20K
	ESERV	Convert edited macros to source format. Display and/or punch converted macros		112K	64K

Note 1	Refer to the discussion of the condense function for restrictions related to execution of the CONDS function of the MAINT program.
Note 2	This function may be executed in a foreground partition for a private core image library.
Note 3	Reallocate cannot be used for private libraries.
Note 4	A MERGE to SYSRES function must be run in BG.
Note 5	When requesting sorted DSERV output, an allocation (ALLOCR) of 112K together with a specification of SIZE=64K in the EXEC statement will improve the performance.

Figure 3-32. Summary of Librarian Programs, Their Functions, and Real Mode Requirements

You invoke the individual functions of the librarian programs by means of librarian control statements. The use of these control statements is described and demonstrated by examples in the following section. Their formats are contained in *DOS/VSE System Control Statements.*

If the extended support for the procedure library (SYSFIL) was selected during supervisor generation, the librarian control statements can be cataloged into the procedure library. This excludes maintenance functions for the procedure library itself and reallocation of library sizes.

Results may be unpredictable if librarian programs access a library while this library is being updated in another partition. Therefore, if a private library is assigned to more than one partition, the library should not be updated.

The maintenance functions of the librarian greatly facilitate frequent operations such as:

- Cataloging members to the libraries
- Deleting members from the libraries
- Condensing the libraries
- Establishing limits for condense
- Allocating space to the libraries
- Renaming members of the libraries
- Updating books in the source statement library.

While a procedure is being executed, no maintenance functions can be performed against the procedure library. For this reason, programs whose running time is relatively long (for example, the licensed program VSE/POWER) should not be executed from a procedure.

The maintenance program is invoked by the job control statement:

```
// EXEC MAINT
```

The functions to be performed are specified in librarian control statements which must follow the EXEC MAINT statement on SYSIPT (If SYSIPT is assigned to a tape unit, it must be a single file and a single volume). Any combination of the maintenance functions can be performed in a single run. A sample maintenance job (in skeleton form) is shown below:

```
// JOB ANYMAINT
    .
    .
assignments, if necessary
    .
    .
// EXEC MAINT
    .
    .
librarian control statements
    .
    .
/*
/&
```

When the /* is processed after completion of the maintenance run, DOS/VSE prints (on SYSLST) a status report of the library just updated.

The symbolic unit assignments required for the individual maintenance functions are described in *DOS/VSE System Control Statements.* The examples in this chapter assume that all necessary assignments are established as permanent assignments.

To perform maintenance on a private library, the library must be assigned in the partition in which the maintenance program is being executed. To access the private libraries, you must assign the following symbolic unit names to the device(s) containing the libraries:

```
Private core image library . . . . . SYSCLB
Private relocatable library  . . . . . SYSRLB
Private source statement library . . . SYSSLB
```

To perform maintenance on system libraries, the corresponding private library must be unassigned.

Cataloging Members into the Libraries. The catalog function adds a module to a relocatable library, a book to a source statement library, or a procedure to the procedure library. Phases are cataloged to the core image library by the linkage editor.

The catalog control statements specify the name of the member to be cataloged and, optionally, a change level number. The control statements are:

```
Relocatable library . . . . . CATALR
Source statement library  . . . . . CATALS
Procedure library . . . . . CATALP
```

The catalog function implies a delete for members with the same name. Therefore, you should rename the existing member prior to cataloging the new member that has the same name. Then, when the new member has been successfully tested, the old member may be deleted.

When you add to the contents of a library, watch the status of the system directory, which is printed at the end of the catalog run. If the libraries are becoming full, you may wish to condense them or allocate more space to them. (Condensing and allocating are described later in this section.)

Cataloging to the Relocatable Library. To catalog an object module to the relocatable library, you must submit the object module on SYSIPT immediately behind the CATALR control statement. The following job catalogs two object modules, named MOD1 and MOD2, to the relocatable library; the object modules were produced by language translators in previous jobs:

```
// JOB CATREL
// EXEC MAINT
    CATALR MOD1
    .
    .
object module for MOD1
    .
    .
    CATALR MOD2
    .
    .
object module for MOD2
    .
    .
/*
/&
```

You may compile or assemble a program and catalog the resulting object module in the relocatable library in the same job. In this case, you assign SYSPCH, which receives the output of the language translator, to a disk, diskette or tape and then use the object module on that device as input to the MAINT program. An example using a magnetic tape for SYSPCH is

shown in Figure 3-33. To assign SYSPCH to a disk or diskette, the SYSFIL option must have been specified during supervisor generation, and you must supply the necessary DLBL and EXTENT job control statements.

```
   // JOB CATREL
   // OPTION DECK
1  // ASSGN SYSPCH,180
   // EXEC ASSEMBLY
2           PUNCH   'CATALR MODULE1'
   source module
3
   /*
4  // MTC WTM,SYSPCH,2
5  // MTC REW,SYSPCH
6  // RESET SYSPCH
7  // ASSGN SYSIPT,180
8  // EXEC MAINT
   /&
```

1 A magnetic tape device is assigned to SYSPCH to receive the assembler output.
2 The assembler will punch a CATALR statement on SYSPCH.
3 The assembler processes the source module and writes the object module onto SYSPCH following the CATALR statement.
4 Tapemarks are written on SYSPCH to indicate the end of the object module.
5 The tape is rewound to its load point.
6 The tape is unassigned as SYSPCH.
7 The tape is assigned to SYSIPT to serve as input for the MAINT program.
8 MAINT reads the object module from the tape and catalogs it in the relocatable library.

Figure 3-33. Assembling and Cataloging to the Relocatable Library in the Same Job

All modules in the relocatable library that have the first three characters of the module name in common are considered to belong to one program. This simplifies the control statements to delete, display, punch, merge, and copy an entire program. The names of IBM-supplied modules in the relocatable library begin with the letter I, which should therefore be considered reserved so that you can easily distinguish your modules from IBM's.

Cataloging to the Source Statement Library. To add a book to the source statement library, you use the CATALS statement specifying the name of the book and the sublibrary to which it belongs. A sublibrary is defined by an alphameric character preceding the bookname. For example, the statement

```
CATALS   L.NEWBOOK
```

adds the book NEWBOOK to sublibrary L. Note that the sublibraries in the range from A to I, P, R, and Z are reserved for IBM components.

A -- is the assembler copy sublibrary. It contains books of assembler source code and source macro definitions. See *DOS/VSE System Control Statements* for details.

B -- is the network definition sublibrary for ACF/VTAM.

C -- is the COBOL sublibrary.

D -- is the alternate assembler copy sublibrary. It contains non-edited macros and copy books for programs that are to be executed in a telecommunications network control unit.

E -- is the assembler macro sublibrary. It contains IBM-supplied and user-written macro definitions in an edited (partially processed) format. See *Guide to the DOS/VSE Assembler* for details.

F -- is the alternate assembler macro sublibrary. IBM uses it to distribute edited macros for use by programs that are to be executed in a telecommunications network control unit.

P -- is the PL/I sublibrary.

R -- is the RPG II sublibrary.

Z -- contains sample programs supplied by IBM.

The rest of the reserved characters (G, H, I) will be used by IBM for future additions to the source statement library. You should avoid, wherever possible, cataloging to one of the reserved sublibraries. If you must catalog to a sublibrary that is reserved for IBM components, ensure that you do not use duplicate names. You can obtain a listing of the contents of each sublibrary by means of the SSERV librarian program discussed later in this chapter. You can obtain a listing of the book names within each sublibrary by means of the DSERV librarian program.

Users of previous versions of DOS, who have books in a sublibrary which is reserved under DOS/VSE can easily transfer this sublibrary from the *IBM range* to the *user range* by means of the librarian rename function of the MAINT program.

Edited macro definitions that are to be cataloged in the assembler sublibrary must be preceded by a MACRO statement and followed by a MEND statement. Example:

```
// JOB CATMAC
// EXEC MAINT
    CATALS E.MBOOK
    MACRO
    .
edited macro definition statements
    .
    MEND
/*
/&
```

Books other than macro definitions that are to be cataloged must be preceded and followed by BKEND statements. Example:

```
// JOB CATBOOK
// EXEC MAINT
    CATALS L.SBOOK
    BKEND
    .
    .
source statements
    .
    .
    BKEND
/*
/&
```

The BKEND statement can have optional operands specifying that a sequence check or a card count be performed on the statements to be cataloged, or that the book to be cataloged is in compressed format. If you desire these functions when you catalog a macro definition, BKEND statements can be included in addition to the MACRO and MEND statements.

Cataloging to the Procedure Library. To catalog a procedure in the procedure library you submit a CATALP statement specifying the procedure name. Rules for the naming of procedures are given in *DOS/VSE System Control Statements.*

The control statements to be cataloged follow the CATALP statement; they can be job control or linkage editor control statements or both. The end of the control statements to be cataloged must be indicated by an end-of-procedure delimiter, which is normally a /+.

Each control statement cataloged in the procedure library should have a unique identity. This identity is required if you want to be able to modify the job stream at execution time. Therefore, when cataloging, identify each control statement in columns 73-79 (blanks may be embedded). Refer also to the section *Temporarily Modifying Cataloged Procedures* earlier in this chapter.

The following job catalogs the procedure PROCA in the procedure library:

```
// JOB CATPROC
// EXEC MAINT
    CATALP PROCA
    .
    .
control statements to be cataloged
    .
    .
/+ END OF PROCEDURE
/*
/&
```

If your supervisor was generated with the SYSFIL option, you can also include inline SYSIPT data in the cataloged procedure. The presence of SYSIPT data must be indicated to the MAINT program by the DATA parameter of the CATALP statement. In addition, you must indicate the end of inline data by the /* statement. The following example catalogs a procedure consisting of control statements and SYSIPT data:

```
// JOB CATPROC
// EXEC MAINT
    CATALP PROCA,DATA=YES
    .
    .
control statements
    .
    .
    SYSIPT data
    .
    .
/* END OF SYSIPT DATA
    .
    .
control statements
    .
    .
/+ END OF PROCEDURE
/*
/&
```

The following restrictions apply when you catalog procedures to the procedure library:

1. A cataloged procedure cannot contain control statements or SYSIPT data for more than one job.
2. If the cataloged control statements include the // JOB statement you must not have a // JOB statement when you retrieve the procedure through the EXEC statement.
3. A cataloged procedure must not include either of the following statements:

   ```
   [//] RESET SYS
   [//] RESET ALL
   ```

4. A cataloged procedure with DATA=YES must not include any of the following statements for SYSIN, SYSRDR, or SYSIPT:

   ```
   [//] ASSGN
   [//] CLOSE
   [//] RESET
   /&
   ```

5. A cataloged procedure without inline SYSIPT data must not include any of the following statements for SYSIN or SYSRDR:

   ```
   [//] ASSGN
   [//] CLOSE
   [//] RESET
   /&
   ```

6. Cataloged procedures cannot be nested, that is, a cataloged procedure cannot contain an EXEC statement that invokes another cataloged procedure.
7. When cataloging a procedure that contains an imbedded // JOB statement, in a partition controlled by VSE/POWER, use * $$ JOB and * $$ EOJ statements to define the cataloging job.

Assigning Change Levels. When you catalog a member in one of the libraries, you can assign a change level to the member, which will enable

you to keep track of the current version of your programs. The change level is specified in the catalog control statement by a version and a modification number. The following statement catalogs version 1, modification 3, of module MOD1 in the relocatable library:

```
CATALR   MOD1,1.3
```

Change levels are stored in the directory entry for the member and can be displayed by the librarian service program DSERV. A change level is not used by the system for identification purposes, that is, a change level is *not* sufficient to allow two elements having the same name to coexist in a library.

For the source statement library only, you can request verification of the change level before a book is updated. This can prevent unintentional updating of the wrong version of a book in a particular sublibrary. Specify the character C in the CATALS statement to request change level verification. Example:

```
CATALS   M.BOOK1,1.1,C
```

To update the book you must supply the current change level of the book in the update control statement. This change level is then checked against the change level in the directory entry and, if they match, the book is updated and its change level is increased by one to reflect the new status of the book. If you want to overwrite the version and modification numbers of a book, supply the new change level information in the END statement of the update function. If change level verification is requested for a particular book, the letter C will appear in the column headed LEV CHK (level check) in the DSERV listing.

Deleting Members from the Libraries. You can delete an unwanted member from a library either by cataloging a new member with the same name or by means of the delete function of the librarian, using the following control statements:

```
Core image library . . . . . DELETC
Relocatable library  . . . . . DELETR
Source statement library . . . . . DELETS
Procedure library  . . . . . DELETP
```

To delete individual members from the libraries, you must specify each member name in full in the delete control statement. If a group of members is to be deleted, however, you can simplify the specification of the control statement provided that the recommended naming conventions were used:

- If all the phases of one program in the core image library were named with the same first four characters, you need to specify only these four characters to delete the entire program.
- You can delete all modules in the relocatable library that have the first three characters in common by specifying these three characters in one delete control statement.
- Similarly, you can delete an entire sublibrary from the source statement library by specifying the sublibrary name.

Since no special naming conventions apply to the procedure library, each cataloged procedure to be deleted must be specified individually.

You can also use the delete ALL function to remove all elements of a relocatable library, source statement library, procedure library, or *private* core image library. In this case, the system directory information is updated to show that all blocks of the library in question are available for cataloging programs; no condense operation is required. You cannot delete the entire *system* core image library, but only individual phases or programs.

The following job deletes (1) all phases whose name begin with PHAS from the core image library, (2) modules MOD1 and MOD2 from the relocatable library, (3) sublibrary P from the source statement library, and (4) all the elements of the procedure library:

```
// JOB DELETE
// EXEC MAINT
    DELETC PHAS.ALL
    DELETR MOD1,MOD2
    DELETS P.ALL
    DELETP ALL
/*
/&
```

When you request the deletion of a library member, the name of the member is removed from the corresponding directory entry. The system is then no longer able to recognize the member although it is still physically present in the library. The area taken up by such a member can be referred to as unavailable free space. To make such space available again for cataloging programs, use the condense function of the MAINT program. The delete and condense functions are illustrated in Figure 3-34.

In case an entire component is deleted, the component entry in the system history file should also be deleted using the service program MSHP (Maintain System History Program).

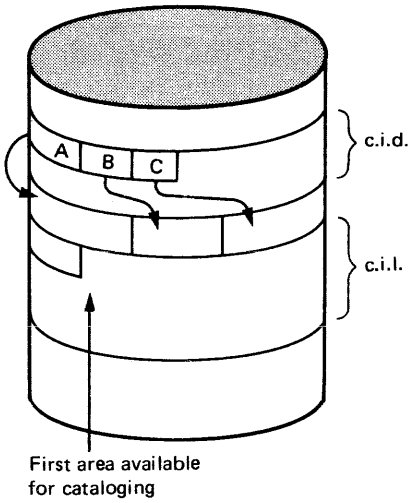
When a phase is deleted from the core image library, it is also flagged as not present in the system directory list (if applicable). The shared virtual area cannot be condensed; it must be recreated. See *Building the SDL and Loading the SVA* under *Starting the System* earlier in this chapter.

Condensing the Libraries. When you delete a member from a library, the space occupied by the 'deleted' member is unavailable for cataloging new members (see Figure 3-34). To make this space available for cataloging, you use the condense function of the MAINT program.

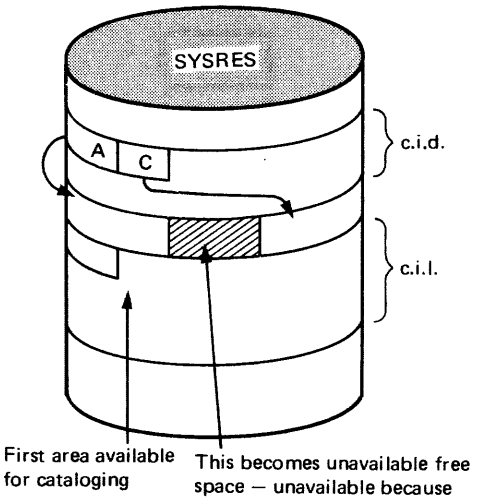
To condense any of the system libraries you use the CONDS control statement specifying which of the libraries is (are) to be condensed. The following job condenses the core image, relocatable, and source statement libraries after the deletion of members from the libraries:

```
// JOB DELCOND
// EXEC MAINT
    DELETC PHAS1,PHAS5,PROGA
    DELETR MOD.ALL
    DELETS P.ALL
    DELETP ALL
    CONDS CL,RL,SL
/*
/&
```

(1) Assume that phases A, B, and C are cataloged in the core image library (c.i.l.). Each core image directory (c.i.d.) entry, which refers to one of these phases, points to the beginning disk address of the phase.



(2) If phase B is no longer desired in the core image library, specify `DELETC B`, which deletes the name B from the directory.



(3) To make full use of the core image library, eliminate the unavailable free spaces by specifying `CONDS CL`.

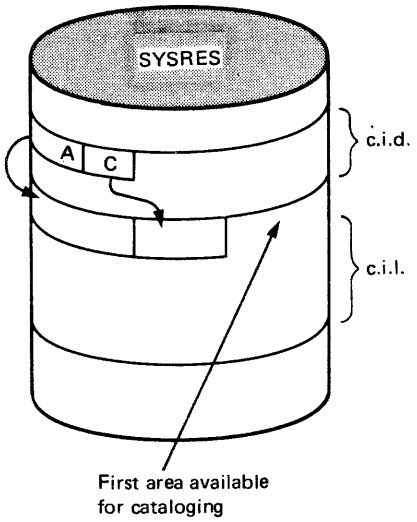


Figure 3-34. Example of Deleting and Condensing

Note that you need not condense a library -- in the above example, the procedure library -- if that library is deleted entirely.

The reallocation function of the MAINT program automatically causes the system libraries to be condensed.

If a condense operation is interrupted by a hardware error or by an operator intervention before the next statement is read, the library being condensed is unusable and must be rebuilt. Note that the condense program shows all the symptoms of a looping program, but should never be canceled by the operator.

There are two methods for condensing libraries that do not use the MAINT program. Both methods involve copying only the undeleted library members to a new volume.

- The utility programs BACKUP and RESTORE can be used if your installation has magnetic tape drives installed. The BACKUP program copies libraries to tape but doesn't copy deleted members. The RESTORE program copies the tape volume to a disk recreating your libraries. For more details see *DOS/VSE System Utilities*.
- The Copy and Reorganize program (CORGZ) copies libraries from one disk extent to a different disk extent. Deleted members are not copied. See the section *Organizing the Libraries* later in this chapter for information on the CORGZ program.

Specifying the Condense Limit. You can specify that a message is to be delivered to the operator whenever the number of available blocks in a library drops below a specified minimum, which is referred to as the condense limit. Through the CONDL statement you specify the library or libraries and the condense limit(s).

Example:

```
// JOB CONDSLMT
// EXEC MAINT
   CONDL CL=10
/*
/&
```

In the above example, the CONDL statement specifies that, whenever the number of available library blocks falls below 10, a message is to be issued.

The condense limit should always be less than the number of blocks allocated to the library; otherwise this message is given after each maintenance function. The MAINT program stores the condense limits in the library descriptor, which can be displayed at the end of each librarian maintenance job. If a library has reached a condense limit, this is indicated in the status report by a note.

When Condense Can Be Performed. While the condense function is being executed, the library directories do not represent the actual status of the library. Thus, if a program in any partition were to attempt to use the library in any way, the results would be unpredictable. For this reason, various controls are provided to minimize the chances of unpredictable results:

- The system core image library and either the system or private relocatable and source statement libraries can only be condensed from the background partition, and then only if there are no active foreground partitions.
- The procedure library may be condensed from the background partition while a foreground partition is active. However, the condense function cannot be performed on the procedure library if it is being accessed from another partition while a procedure is being executed.
- A job stream to condense the procedure library cannot be executed from a cataloged procedure.
- A private core image library may be condensed in any partition, provided it is exclusively assigned to that partition.

The CONDL control statement (which sets the condense limits) can be submitted with the MAINT program at any time in the background partition.

A partition is inactive if it has never been activated with a START or BATCH command or has been deactivated with an UNBATCH command.

Even if a program such as VSE/POWER is not doing any work, if it is resident in a partition, that partition is considered to be active.

Reallocating the Library Sizes. You can use the reallocation function of the MAINT program to

- increase the size of a system library for further additions.
- decrease the size of a system library; for example, to provide space for expanding other libraries.
- eliminate a system library if it is replaced by a private library or is no longer required.
- reestablish a system library after it has been eliminated.

Each library that is reallocated is condensed automatically. You can reallocate any combination of the system libraries on SYSRES within a single run. You *cannot* reallocate private libraries. To change the allocation of a private library, you must create a new private library using the CORGZ program (see *Creating and Working with Private Libraries,* later in this chapter). Tape users may, as an alternative, use the BACKUP/RESTORE utilities. If a private library is assigned and you attempt to reallocate the corresponding system library, a message is issued and the job is canceled.

The reallocation function of the MAINT program must always be executed in the background partition and all foreground partitions must be inactive. This ensures that no program can access any library during reallocation; otherwise, the results would be most unreliable because the final addresses may not have been established and (similar to the condense function) because the directory entries do not reflect the actual status of the libraries until the end of the reallocation step.

You invoke the reallocation function through the ALLOC control statement. In the operand field, you specify (1) the libraries to be

reallocated, (2) the amount of disk space to be allotted to each library, and (3) how much of that allotted disk space is to be reserved for the library directory. The ALLOC statement can be submitted together with any other maintenance control statements.

Changing the Size of the System Libraries. When you increase the size of one library, you must consider the space remaining for the libraries that follow.

Figure 3-35 shows the available disk space by device type. FBA space requirements are in number of FBA blocks, all others are shown in number of cylinders.

Device Type	VTOC	Label information area	Disk space available
CKD:			
2314/2319	1	2	197
3330/3333	1	2	
Model I	1	2	401
Model II			803
3340	1	3	
w/3348 M35	1	3	344
w/3348 M70			692
3350	1	1	554
FBA (see note):			
3310	16	200	125798
3370	16	200	557782

Note: FBA space requirements show the default sizes in FBA blocks; the size of the VTOC may be changed by an Initialize Disk utility run and that of the label information area by a RESTORE utility run. For more information, see *DOS/VSE System Utilities*.

Figure 3-35. Disk Space available for System Libraries

Assume, for example, that the SYSRES library space on a 2314 was allocated during system generation as

```
ALLOC CL=90(5),RL=40(2),SL=60(3),PL=6(5)
```

An attempt to reallocate only the core image library to 120 cylinders would fail, because there is not enough space available for all of the following libraries. To avoid this, you must reduce one or more of these libraries to compensate for the increase. For example, reduce the combined sizes of the relocatable and source statement libraries by 29 cylinders. In this case, the ALLOC statement should read:

```
ALLOC CL=120(7),RL=30(2),SL=41(3),PL=6(5)
```

When you alter the size of the SYSRES file by reallocating libraries, you must define the new SYSRES extent by means of DLBL and EXTENT job control statements. Note that the SYSRES extent begins with relative track 1 for a CKD device or physical block 2 for an FBA device. The EXTENT job control statement must account for the label information area.

The filename specified in the DLBL statement for the SYSRES file must always be IJSYSRS. The new label information for the SYSRES file is stored in the volume table of contents (VTOC) of the SYSRES pack.

The following example shows the job control statements required to reallocate the system libraries as discussed above when the SYSRES device type is 2314/2319:

```
// JOB REORG
// DLBL IJSYSRS,'DOS/VSE SYSTEM RESIDENCE',99/365
// EXTENT SYSRES,111111,1,0,0001,3979
// EXEC MAINT
   ALLOC CL=120(7),RL=30(2),SL=41(3),PL=6(5)
/*
/&
```

For CKD devices, like the 2314 in the above example, allocations are given in cylinders for the libraries. Because the SYSRES file begins at cylinder 0 track 1, the EXTENT statement must take the following into account:

CL = 120 cylinders x 20 tracks	=	2400
RL = 30 cylinders x 20 tracks	=	600
SL = 41 cylinders x 20 tracks	=	820
PL = 6 cylinders x 20 tracks	=	120
		3940
Label information area (2314/19) 2 cylinders x 20		40
		3980
Less cylinder 0, track 0		-1
		3979

This SYSRES file is comprised of 3979 tracks.

No special considerations apply for *reducing* the size of a library except that you must also supply the necessary label information for the new SYSRES extent. Reducing a library does not cause any gaps, that is, the libraries following the one that was reduced are 'moved up' to close the gap. If your allocations are too small for the existing library members, the job is canceled and an appropriate message is displayed. At this point in time, the libraries are still intact.

Eliminating Libraries. If you have created a private relocatable or source statement library containing all the modules or books that you require from the corresponding system library, you can use the reallocation function to eliminate that system library. You do this by setting the space indications in the ALLOC statement to zero. This is only effective, however, if all the directory entries have first been cleared by the DELETS or DELETR control statements.

Similarly, you can eliminate the procedure library if it contains no active members and you are sure that you do not want to use cataloged procedures.

The following job eliminates the system relocatable library -- the example assumes that the libraries were allocated with CL=80(5), RL=40(2), SL=30(3), PL=10(5). The SYSRES device type is assumed to be 2314/2319.

```
// JOB ELIMNT
// DLBL IJSYSRS,'DOS/VSE SYSTEM RESIDENCE',99/365
// EXTENT SYSRES,111111,1,0,0001,3239
// EXEC MAINT
    DELETR ALL
    ALLOC RL=0(0),CL=120(7),SL=30(3),PL=10(5)
/*
/&
```

You cannot eliminate the system core image library because it is required for system operation. If you inadvertently specify a zero allocation for the system core image library, the job is canceled.

Once eliminated, the relocatable, source statement, or procedure library can be added again to the SYSRES file. The same considerations apply to adding a library as to increasing the size of a library. Using the reallocation function to add a library does not include adding the actual members of the library. Once a library exists you can add members either by cataloging or by merging from a private library or another SYSRES. The merge function is described in *Organizing the Libraries*, later in this chapter.

Renaming Members in the Libraries. To change the name of a library member, use the rename function. In a control statement, you supply the existing name and the name to which you want to change it. If the *new* name is identical to a name already cataloged in the library, an error message is issued. You must then select a different name and resubmit the job.

When you name a phase in the system core image library that is also listed in the system directory list, the old phase name in the SDL is invalidated.

After a valid rename operation, the system recognizes only the new name. The version and modification level (change level) is not changed by the rename function.

Each type of library has a unique rename control statement:

```
Core image library . . . . . RENAMC
Relocatable library  . . . . . RENAMR
Source statement library . . . . . RENAMS
Procedure library  . . . . . RENAMP
```

The rename function can be used to establish naming conventions. All phases in the core image library that have the first four characters in common are considered to belong to one program. All modules in the relocatable library that have the first three characters in common are considered to belong to one program. Since the names of IBM-supplied relocatable modules begin with the letter I, it is of advantage to avoid this first character when naming user modules. Similarly, you should avoid the use of the first characters A through I, P, R, and Z when renaming sublibraries in the source statement library. These prefixes are reserved for IBM-supplied components. Names for procedures cataloged in the

procedure library can consist of any combination of alphanumeric characters as long as they adhere to the naming rules for procedure names.

Renaming a member of a library can be advantageous in a testing environment. For instance, after making changes to your source deck, rename the previous version residing in the library and catalog the new source under the original name. This assures you of backup until your new program is in working order, at which time you can delete the old (renamed) version(s).

Updating Books in the Source Statement Library. The update function applies only to a source statement library. This function revises one or more source statements within a particular book. By using update you can make minor changes to a 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:

- resequence statements within a book.
- revise a change level (version and modification) of a book.
- add or remove the requirement for change level verification.
- copy an entire book and rename the old book (for backup purposes).

The UPDATE control statement identifies the update function. This statement may also be followed by one or more of these additional statements as required:

```
ADD   -- To add source statements
DEL   -- To delete source statements
REP   -- To replace source statements.
```

The END statement indicates the end of updates to the particular book specified in the UPDATE control statement.

If the requirement for change level verification was specified in the CATALS control statement when a book was cataloged, the version and modification level must be specified in the UPDATE control statement that refers to this book. This change level must agree with the current change level in the directory entry for that book. (Check the DSERV listing for the current change level and/or requirement for change level verification. For more information on the DSERV program, refer to the section *Displaying the Directories.*) The specification of the version and modification level in the UPDATE statement prevents you from inadvertently making an update based on a book with the wrong version and modification. Regardless of whether or not the requirement is in effect, the version and modification level are incremented by one after each update. If a version and modification level is specified in the END statement, this overrides the current change level.

Organizing the Libraries

The Copy and Reorganize (CORGZ) program and the Copy Service (COPYSERV) program are important tools for establishing and organizing your libraries during system generation or any time thereafter. The

following discusses these programs, their functions, and their application to your library organization requirements.

Copy and Reorganize Program (CORGZ). The functions of the CORGZ program are to:

- Create a new system residence (SYSRES).
- Transfer members between any two existing libraries of the same type, as follows:
  - all members, or
  - some members, or
  - only those members which do not yet exist in the receiving library.
- Create private libraries.

The first two points are described in this section. The creation of private libraries is discussed in *Creating and Working with Private Libraries,* later in this chapter.

The CORGZ program can be executed in any partition, except for the merge function (to copy members) with SYSRES as the destination, which must be executed in the background partition. The program is invoked by the statement

```
// EXEC CORGZ
```

When /* is processed (after completion of the CORGZ program), a status report of the library just updated is printed on SYSLST.

Input and output devices must be of the same disk architecture (CKD or FBA). Given, for instance, a CKD device as input, output cannot be an FBA device.

The functions to be performed by the CORGZ program are specified in a set of librarian control statements, which are discussed below.

Creating a New System Residence. When system generation is completed, you will want a backup SYSRES, which can save you regenerating the system from your distribution medium if the operational pack is inadvertently destroyed. This backup SYSRES is usually kept on tape (from which it can be restored using the RESTORE utility program), but may also be kept on a disk of the same device type as the original SYSRES. If the backup SYSRES is to be on disk, use the CORGZ program with the ALLOC and COPY control statements to define the new SYSRES file and copy the entire contents of the original SYSRES file onto it.

You can also copy the SYSRES file selectively; that is, the new system residence will contain only part of the original SYSRES. This may be useful in an installation that uses certain components only during specific processing periods. For instance, if telecommunication and support for five partitions is required only during the prime shift, a different system configuration (for instance, no telecommunication and three partitions) could be used during the second shift. Therefore, you could copy onto a new SYSRES file only those components required for the second shift and add any additional components needed to that SYSRES. In this case, you must assemble a new supervisor and catalog it into the new SYSRES file.

The effect is a smaller supervisor and smaller libraries on both system residence packs which means faster access to library elements and, thus, improved overall system performance.

When you create a new system residence, SYS002 must be assigned to the device on which the new SYSRES pack resides. The device types of SYS002 and SYSRES must be identical. Note that the IBM 3330-1 and 3330-11 are of the same device type; the same is true for the IBM 3340-35 MB and 3340-70MB. In addition, you must define the extents of the new SYSRES file by means of DLBL and EXTENT job control statements. The filename in the DLBL statement must be IJSYSRS. The lower extent limit must be relative track 1 for a CKD device or block 2 for an FBA device, and the upper extent limit must include the label information area.

The information to be copied from the original to the new SYSRES is specified in one or more of the following COPY control statements:

COPY ALL	to copy the entire system residence file. You can use this form of the COPY statement only if all four system libraries are allocated on the original SYSRES file; otherwise, you must use a combination of the following COPY statements.
COPYC COPYR COPYS COPYP	to copy one or more members, one or more groups of members, or all members of the Core image, Relocatable, Source statement or Procedure library.

If more than one copy control statement is submitted for several libraries, these statements should be grouped per library (for example, first all COPYC statements, then all COPYR statements, and so on). A COPY ALL or COPYx ALL statement must neither be preceded nor followed by any other copy statement for the same library.

Note: *The names of all members copied are printed on SYSLST if you specify // UPSI 10000000.*

The following job creates a backup SYSRES file on a 3330 disk drive. The example assumes that the original SYSRES file does not contain a procedure library:

```
// JOB BACKUP
// ASSGN SYS002,131
// DLBL IJSYSRS,'DOS/VSE SYSRES BACKUP',99/365,SD
// EXTENT SYS002,111111,1,0,0001,2127
// EXEC CORGZ
   ALLOC CL=50(5),RL=30(5),SL=30(5),PL=0(0)
   COPYC ALL
   COPYR ALL
   COPYS ALL
/*
/&
```

Since the 3330 is a CKD device, all space allocations in the ALLOC statement are in number of cylinders. The number of tracks in the EXTENT statement (2127) is the sum of: the library allocations (110 cylinders x 19 trks), minus 1 track (cylinder 0, trk 0); plus the label information area (2 cylinders x 19 trks). For FBA devices the space allocations are given in number of blocks.

For each CORGZ run to create a new SYSRES file, an ALLOC control statement is required, preceding any COPY statements. If you wish to exclude an entire library from being copied, specify a 'zero' allocation (for example, RL=0(0)).

Assume that you have a SYSRES file that contains all four system libraries and you want to create a second SYSRES file containing only selected information from the core image library and the entire relocatable library. The following job creates this new SYSRES file (device type FBA assumed):

```
// JOB SYSRES
// ASSGN SYS002,131
// DLBL IJSYSRS,'DOS/VSE SYSRES II',99/365,SD
// EXTENT SYS002,111111,1,0,0002,12708
// EXEC CORGZ
    ALLOC CL=7500(75),RL=5000(50),SL=0(0),PL=0(0)
    COPYC PHAS.ALL,PROG.ALL,ABCD.ALL
    COPYR ALL
/*
/&
```

The EXTENT statement reflects a SYSRES file beginning at block 2 comprising 12,708 blocks: 12,500 blocks make up the libraries, 200 blocks are allocated as the label information area, and the first 8 blocks are to be reserved for system information.

Phases whose names start with a '$' are automatically copied by the CORGZ program. This provides you with the essential DOS/VSE components listed below:

- IBM supplied supervisor ($$A$SUPn)
- Initial program load (IPL)
- All logical and physical transients
- Job control
- Linkage editor

User created elements can also be copied automatically:

- Phases that you have cataloged with a '$' as the first character (such as a tailored supervisor)
- Partition and system standard labels (cataloged with the PARSTD and STDLABEL options) from the label information area (see Note).

Therefore you may execute the CORGZ program without any COPY statements, and the above items will be copied automatically onto the new SYSRES file.

Note: The CORGZ program does not copy an alternate label information area that you defined through the DLA command.

Transferring Members between Libraries. If you work with more than one system residence pack or private library, you may want to transfer members from one library to another. You can use the CORGZ program with a MERGE statement to transfer the elements. This is especially useful for

system generation when a new version of the system is installed; you can then copy the library elements directly from the old version to the new one.

You use the MERGE control statement to define the characteristics of the libraries to be merged and the direction of transfer between the libraries. The operands of the MERGE control statement are:

RES -- For the system libraries on the system residence file.

NRS -- For the system libraries on a modified or duplicate system residence file that is not currently IPLed.

PRV -- For any private libraries.

For example, the statement MERGE RES,PRV indicates to the CORGZ program that elements are to be transferred from one or more libraries on the system residence file to the corresponding private libraries.

For an SCP only environment, the device types of the input and output devices must be the same. Note that the IBM 3330-1 and 3330-11 are of the same device type; the same is true for the IBM 3340-35MB and 3340-70MB. With VSE/Advanced Functions installed, the device types may be different, within the same disk architecture (CKD or FBA). However, when requesting

```
MERGE RES,NRS   or
MERGE NRS,RES
```

the device types must be the same.

The type of library involved and the elements to be transferred are specified in COPY statements immediately following the MERGE statement. (The COPY statements are the same as those described under *Creating a New System Residence* earlier in this chapter.)

You must define the extents of the libraries involved in a merge operation by DLBL and EXTENT job control statements. The filenames to be used and the necessary symbolic unit assignments are described in detail in *DOS/VSE System Control Statements.*

When the CORGZ program performs a merge operation, it *does not* automatically copy the basic system components as it does when a new system residence is created (see preceding section). You must specify COPYC ALL to transfer the entire core image library or COPY ALL to transfer the entire SYSRES extent.

The job in the following example adds the contents of the core image library on a duplicate SYSRES file (NRS) to the elements in a private core image library (PRV). Any elements with duplicate names (supervisor, job control etc.) are deleted from the receiving library.

```
// ASSGN SYS002,130
// DLBL IJSYSRS,'DOS/VSE SYSRES II',99/365,SD
// EXTENT SYS002,111111,1,0,0001,2519
// DLBL IJSYSCL,'PRIVATE CIL',99/365,SD
// EXTENT SYSCLB,222222,1,0,1600,200
   ASSGN SYSCLB,131
// EXEC CORGZ
   MERGE NRS,PRV
   COPYC ALL
/*
/&
```

Alternatively, for the COPYC, COPYR, COPYS, and COPYP statements, the NEW operand can be used to copy only those members that do not already exist in the receiving library. However, for COPYC NEW:

- supervisor phases are never copied, and
- a number of system phases are always copied.

For a list of phases that are always copied see *DOS/VSE System Control Statements*. In addition, when using the NEW operand, ensure that your receiving library has sufficient space allocated to accommodate the library members that are copied from the other library.

The job in the following example also adds the phases of the core image library on a duplicate SYSRES file (NRS) to the phases in a private core image library (PRV). In this example, only nonduplicate elements are copied.

```
// JOB NRSPRV
// ASSGN SYS002,130
// DLBL IJSYSRS,'DOS/VSE SYSRES II',99/365,SD
// EXTENT SYS002,111111,1,0,0001,2519
// DLBL IJSYSCL,'PRIVATE CIL',99/365,SD
// EXTENT SYSCLB,222222,1,0,1600,200
ASSGN SYSCLB,131
// EXEC CORGZ
   MERGE NRS,PRV
   COPYC NEW
/*
/&
```

Copy Service Program (COPYSERV). This program compares library directories and, on finding differences in contents, produces corresponding COPY statements for use with the CORGZ program. The advantages of COPYSERV over the COPY NEW function of a CORGZ MERGE operation are:

- You may alter a generated COPY statement prior to the actual MERGE.
- The space required for the library members to be copied is calculated and displayed on SYSLST.

You may find COPYSERV particularly useful when installing a new release.

The program allows comparison of both system and private libraries. The libraries you wish to have compared must be defined by the appropriate ASSGN, DLBL, and EXTENT statements; the new (or target) library must be assigned to SYS003, with a filename of IJSYSNR. If private

libraries are involved, it is necessary to provide an additional definition of your compare requirements by means of the UPSI statement.

COPYSERV can be executed in any partition; it is invoked by the statement // EXEC COPYSERV. At the completion of a COPYSERV run, you will receive the following types of statements on SYSPCH which you can include in a CORGZ job stream:

```
// EXEC CORGZ
    MERGE RES,PRV
    COPYC phasename
    .
    .
    .
/*
/&
```

For ease of correcting the output, you get this output sorted by member names.

COPYSERV, in addition, provides a printout with

- A listing of the punched output.
- The number of additional directory entries needed in the new library.
- The number of additional library blocks needed to accomodate the new library.

For a COPYSERV/CORGZ job stream example in the context of a system generation, refer to the *System Generation Procedures* in *DOS/VSE System Generation*.

With the job stream shown below, a comparison between a current and a new private source statement library is executed by COPYSERV.

```
   // JOB COPYSERV
1 { // DLBL IJSYSSL,'OLD.PVT.SOURCE.STMT.LIBRARY'
  { // EXTENT SYSSLB
  { // ASSGN SYSSLB,132
2 { // DLBL IJSYSNR,'NEW.PRV.SOURCE.STMT.LIBRARY'
  { // EXTENT SYS003
  { // ASSGN SYS003,133
3  // UPSI 00100010
   // EXEC COPYSERV
   /&
```

1 Label and assignment statements for the current (or source) library.

2 Label and assignment statements for the new (or target) library.

3 Required UPSI setting for comparing two private source statement libraries.

For more details on the COPYSERV program see *DOS/VSE System Control Statements*.

Using the Service Functions of the Librarian

The service functions of the librarian enable you

- to obtain reports on the contents of your libraries by displaying the directories on SYSLST.

- to print the contents of your libraries on SYSLST, to punch these contents on SYSPCH, or both (in order to transfer the library members to a different location or to correct them).
- to prepare macro definitions in the assembler macro (E) sublibrary for update.

If you use private libraries, the service functions apply only to the assigned private libraries; you must unassign your private libraries for the corresponding system libraries to be accessed by the service programs.

Displaying the Directories. Using the directory service program (DSERV), you can obtain a listing of the following directories:

- Core image directory, or the directory entry of a specific phase or group of phases in the core image library together with their change level, if present
- System directory list (SDL)
- Relocatable directory
- Source statement directory
- Procedure directory
- Status report. Size and level of contents of the assigned private libraries and of the system libraries. (This directory is always listed before any of the directories is printed.)

Depending on the control statement used, the entries of a directory can be displayed in the order as they appear in the directory (DSPLY control statement) or sorted (DSPLYS control statement).

Note: *The entries in the core image directory are always stored in alphameric sequence and therefore displayed in that sequence.*

Within a single job step you can obtain multiple displays of the same directory, either sorted or unsorted, by supplying a separate control statement for each desired display. Similarly, any number of directories can be displayed within one job step, depending on the operands in the control statement. The following job produces a sorted listing of all $-phases and unsorted listings of the relocatable and source statement libraries:

```
// JOB DISPDIR
// EXEC DSERV
   DSPLYS TD
   DSPLY RD,SD
/*
/&
```

If you specify // EXEC DSERV without any control statements, a status report of all libraries present on SYSRES and all private libraries assigned (if any) is printed on SYSLST.

Displaying and Punching the Contents of the Libraries. You can use the library service programs to obtain a listing, a card deck, or a card image copy of the elements in a library. There is a service program for each library:

CSERV -- Core image library
RSERV -- Relocatable library
SSERV -- Source statement library
PSERV -- Procedure library.

You request the library service functions by invoking (with // EXEC) the pertinent service program and one of the following control statements:

DSPLY to print entries of a directory or the members of a library on SYSLST.

PUNCH to punch the members of a library on SYSPCH.

DSPCH to print and punch the members of a library on SYSLST and SYSPCH, respectively.

DSPLYS to produce a sorted listing of the entries of a directory.

Each of these statements can specify one or more individual members, one or more groups of members, or all members of a library to be printed or punched. The following job prints the entire sublibrary P and punches phases PHAS1 and PHAS3 of the core image library:

```
// JOB LIBSERV
// EXEC SSERV
   DSPLY P.ALL
/*
// EXEC CSERV
   PUNCH PHAS1,PHAS3
/*
/&
```

The SYSPCH output (in cards or on tape, diskette, or disk) of any service program can be used as input for recataloging into the type of library from which it was extracted.

With the PUNCH or DSPCH statements the CSERV program produces a Phase statement, naming the output phase, as the first statement on SYSPCH. For the same operations the other service programs produce a CATALR, CATALS, CATALP statement immediately preceding each member on SYSPCH.

CSERV, RSERV and SSERV SYSPCH output is followed by a /*. PSERV SYSPCH output has the end-of-procedure delimiter (default /+) following each procedure and a /* following the last output procedure. Such output can therefore be submitted as is with a // EXEC MAINT statement for recataloging.

The SYSPCH output of the CSERV program is suitable as input to the linkage editor for recataloging to the core image library. The control statement stream would be as follows:

```
// JOB RECATAL
// OPTION CATAL
   INCLUDE

            ----- CSERV output

// EXEC LNKEDT
/&
```

The PHASE statement produced by the CSERV program reflects the status of the phase when it was first cataloged (relocatable, self-relocating, non-relocatable or SVA elegible). If you wish to change the status you must change the PHASE statement prior to re-linking.

Printed output from any of the service programs is useful for debugging purposes. For instance, after determining an error from a dump or source listing, you implement a change to the RSERV object deck by inserting the appropriate REP card(s) directly before the END card and run the MAINT program to recatalog the object module; then to verify that the REP card was correct, execute the RSERV program to obtain a listing. An SSERV listing may be necessary before a single statement update can be performed; after locating the statement in error in the listing, submit an UPDATE maintenance run to implement the change in the source statement library.

Preparing Edited Macros for Update. The assembler uses two sublibraries of the source statement library: the macro sublibrary (sublibrary E) and the copy sublibrary (sublibrary A). All macro definitions in the assembler macro (E) sublibrary have been preprocessed by the assembler; they are said to be edited. An edited macro definition cannot be directly updated; instead, the source macro, either in a card deck or in the copy (A) sublibrary is updated. After the changed macro has been tested and debugged, it must be edited again before it can be recataloged in the macro sublibrary.

If the macro to be updated is not available in source format, you can use the ESERV program to convert the edited macro back to source format: this is called de-editing. If the output of the ESERV program is to be used directly as input to the assembler, you can specify the GENEND control statement to cause the END card and a /* card to be included after the last macro. If the output is to be cataloged directly into the copy (A) sublibrary, you can specify the GENCATALS control statement. This causes a CATALS card to be generated before each macro in the run and a /* card after the last macro. If neither the GENEND nor the GENCATALS control statement is specified after the // EXEC ESERV statement, GENCATALS is assumed.

The remainder of the control statements that you can submit to the ESERV program are the same as for the other librarian service programs: DSPLY, PUNCH, and DSPCH. The following job de-edits the macro named MAC1:

```
// JOB DEEDIT
// EXEC ESERV
    GENEND
    PUNCH E.MAC1
/*
/&
```

The output of the above job is the macro MAC1 in source format on SYSPCH. An END card and a /* card is included after the macro. You can now update the macro, edit it, and catalog it back into the E sublibrary of the source statement library.

You can de-edit and update a macro in a single run by submitting the necessary update control statements. The following job de-edits and updates the macro MAC2. The result will be the updated macro in source format on SYSPCH and a listing of the updated macro on SYSLST:

```
// JOB EDTUPDTE
// EXEC ESERV
    GENCATALS
    DSPCH E.MAC2
    .
    .
    update control statements
    .
    .
/*
/&
```

The update function of the librarian is described in *Updating Books in the Source Statement Library,* earlier in this chapter. Detailed information on editing, de-editing, and updating macro definitions is given in *Guide to the DOS/VSE Assembler.*

Creating and Working with Private Libraries

Private libraries are created and maintained by the system librarian programs. Except for the reallocate (ALLOC) function, all librarian functions are available for private libraries and performed in the same manner as for system libraries. To change the extents of a private library, create a new private library and copy the contents of the old library into it.

The following sections describe how to create private libraries and what you must consider when you use private libraries.

Private Library Creation

You can create private libraries either during system generation or at any time thereafter. Private libraries can reside on the SYSRES pack (outside the SYSRES extent) or on separate disk packs.

Note: If, in an SCP only environment, a private relocatable library or a private source statement library resides on a pack different from the SYSRES pack, that pack must be of the same device type as the SYSRES pack. **This restriction does not apply for VSE/Advanced Functions.**

You can define any number of private core image, relocatable, and source statement libraries; private procedure libraries are not supported.

You create private libraries with the CORGZ librarian program. The creation of an operational private library involves two stages:

1. Defining the extents of the library by means of a NEWVOL (new volume) control statement.
2. Transferring information to the library from an existing library by means of COPY and/or MERGE control statements.

You can execute the two stages either in one job step by one invocation of the CORGZ program or in separate job steps. Exception: creation of a private core image library requires separate job steps.

To define the device on which a private library is to be created and the disk extents occupied by the library, you must supply a set of ASSGN, DLBL, and EXTENT job control statements specifying predetermined symbolic unit names and filenames (see Figure 3-36).

Private Library	Symbolic Unit Name	Filename
Core image	SYS003	IJSYSPC
Relocatable	SYSRLB	IJSYSRL
Source statement	SYSSLB	IJSYSSL

Figure 3-36. Symbolic Unit Names and Filenames Required to Create Private Libraries

You can store the label information submitted by DLBL and EXTENT statements either temporarily (option USRLABEL) or permanently (option PARSTD or STDLABEL). Temporary labels must be resubmitted with every job (or job step, if new labels are submitted in an intermediate job step) that accesses the corresponding library; permanent labels are valid for all subsequent jobs.

Note: *If you catalog additional permanent labels with the STDLABEL or PARSTD option you must also resubmit all existing standard labels; otherwise, they are lost.*

The following example shows the job control and librarian control statements necessary to define the extents of a private relocatable and a private source statement library on CKD devices. The NEWVOL control statement indicates the type of library to be created and the number of cylinders (tracks) to be allocated to each library (directory) and the number of tracks to be allocated to each directory.

```
// JOB DEFINE
// ASSGN SYSRLB,191
// ASSGN SYSSLB,192
// DLBL IJSYSRL,'DOS/VSE PRIVATE RL',99/365,SD
// EXTENT SYSRLB,111111,1,0,20,800
// DLBL IJSYSSL,'DOS/VSE PRIVATE SSL',99/365,SD
// EXTENT SYSSLB,222222,1,0,500,600
// EXEC CORGZ
   NEWVOL RL=40(5),SL=30(5)
/*
/&
```

After you have defined the extents of the private libraries you can either use the merge function of the CORGZ program to transfer members from existing libraries or the catalog function of the MAINT program to store new members.

To create a private library and at the same time copy information into it from the corresponding system library, you submit a COPY statement

following the NEWVOL statement. To transfer information from an existing private library, a MERGE statement must precede the COPY statement. The following job creates a private relocatable library and copies into it the contents of the system relocatable library and of an existing private relocatable library:

```
// JOB CREATE
// ASSGN SYSRLB,191
// ASSGN SYS001,192
// DLBL IJSYSRL,'NEW PRIVATE RL',99/365,SD
// EXTENT SYSRLB,111111,1,0,1700,1200
// DLBL IJSYSPR,'OLD PRIVATE RL',99/365,SD
// EXTENT SYS001,222222,1,0,700,400
// EXEC CORGZ
    NEWVOL RL=60(8)
    COPYR ALL
    MERGE PRV,PRV
    COPYR ALL
/*
/&
```

Note: *To merge from a private relocatable library, you must assign SYS001 to the device containing the library and specify the filename IJSYSPR in the DLBL statement. The logical unit assignments and filenames required for the various merge operations are described in DOS/VSE System Control Statements.*

Private Core Image Library Creation. The organization of a private core image library is the same as that of the system core image library. A private core image library, however, may start on any track. The space requirements must be entered in the NEWVOL statement.

For example, on a 3330 device, the statement NEWVOL CL=20(5) creates a directory of five tracks and a library of 20 cylinders. To create this private core image library starting at relative track number 190, you submit the following control statements:

```
// JOB PCIL
// ASSGN SYS003,191
// DLBL IJSYSPC,'DOS/VSE PRIVATE CL',99/365,SD
// EXTENT SYS003,111111,1,0,0190,380
// EXEC CORGZ
    NEWVOL CL=20(5)
/*
/&
```

In the above example, the core image directory resides on cylinder 10 (tracks 0-4), and the private core image library on cylinders 10-29.

If you desire to start a private core image library on track 1 of cylinder 0 (of a CKD disk) and have it end on a cylinder boundary, the EXTENT statement specifies a number of tracks that is one less than in the corresponding NEWVOL specification. The EXTENT statement in the preceding example then reads:

```
// EXTENT SYS003,111111,1,0,1,379
```

Transferring phases from another core image library would require a second job step.

To access the private libraries, you must assign the following symbolic unit names to the device(s) containing the libraries:

SYSCLB -- Private core image library
SYSRLB -- Private relocatable library
SYSSLB -- Private source statement library

Note that the symbolic unit name required to *create* a private core image library is SYS003; for private relocatable and source statement libraries, the symbolic unit names are the same for creation and subsequent access.

You can assign private relocatable libraries and private source statement libraries either temporarily or permanently by an ASSGN command or statement; you can assign private core image libraries only by an ASSGN command (that is, permanently).

Unless you have cataloged standard labels for your private relocatable and source statement libraries, you must submit label statements with every job that accesses those libraries; the filenames and file identifications in the DLBL statements must be identical to those specified when the libraries were created.

A private library must be unassigned if maintenance and service functions are to be performed on the corresponding system library because the librarian programs assume that the private library is intended whenever assigned. Therefore if, by mistake, your private relocatable library is assigned when you request changes in the system relocatable library, these changes will be performed on the private relocatable library, and you may have to rebuild this library, depending on the nature of the changes. The only system service programs that can access the system libraries when SYSRLB and SYSSLB are assigned are the linkage editor and the CORGZ librarian program.

You can have an unlimited number of private libraries in your system; however, no more than one private core image, one private relocatable, and one private source statement library can be assigned at one time to the same partition. For read access you can also assign a private library to more than one partition, but if you want to update a private library, it must be assigned to one partition only.

If you have more than one private library of the same type, each must be distinguished by a unique file identification in the DLBL statement for the library.

Using Private Core Image Libraries. To *create* a private core image library, the symbolic unit name is SYS003 and, in the DLBL statement, filename IJSYSPC must be specified. To *access* the private core image library, symbolic unit name and filename are SYSCLB and IJSYSCL, respectively.

The assignment of the private core image library must be permanent, via the ASSGN command. Unless you have cataloged partition or system standard labels for your private core image library, you must submit DLBL and EXTENT statements when you assign the library.

Once a private core image library in a system without Access Authorization Checking has been assigned, it remains accessible by subsequent jobs or job steps even if label information was submitted as temporary, that is, with option USRLABEL. The library becomes inaccessible only after an ASSGN command with the UA parameter is given. However, if your DOS/VSE was generated with the Access Authorization Checking service and a program is to be executed while a protected private core image library is assigned, DOS/VSE must open this library to perform the necessary authorization checking. This requires that the labels for that library are available in the label information area when DOS/VSE processes the EXEC statement.

Private core image libraries provide an efficient multiprogramming environment. The linkage editor can be executed not only in the background but also in a foreground partition to which a private core image library is assigned. You can then link-edit a program in any given partition to be executed in the same or in a different partition. If the linkage editor is executed in more than one partition at the same time, you must assign a separate SYSLNK and SYS001 file for each of these partitions.

A separate private core image library can be defined for each partition. Such a private core image library is then said to be *dedicated* to a given partition. Separate versions of the same non-self-relocating program may be link-edited for execution in each partition. This is not necessary, however, for relocatable phases.

If you link-edit primarily relocatable phases, private core image libraries are nevertheless useful to hold special-purpose programs. This allows, for instance, a new version of a program to be tested while the original version remains in working order on the system core image library.

A private core image library should not be assigned to more than one partition at the same time if the linkage editor is being executed in one of these partitions. If you do this, the linkage editor issues a message and terminates abnormally because output from the linkage editor is placed in a private core image library only if that library is uniquely assigned to the partition in which the linkage editor is executed.

When fetching or loading a phase, DOS/VSE first searches the private core image library, if assigned, and if the phase is not found, continues the search in the system core image library. For phases starting with $, DOS/VSE first searches the system core image library and then the assigned private core image library. This library search sequence should be considered when determining names and library residence of programs.

Using System Libraries as Private Libraries. It may be desirable to use the system libraries (excluding the procedure library) as private libraries for certain applications. This is a helpful technique when generating your system; you could, for example, assign system libraries of a follow-on release as private libraries. Note, however, that phases starting with $ (which are fetched from the currently IPLed SYSRES) have to be compatible with phases without a $-prefix (which are fetched from the private core image library).

In order to use any of the three eligible libraries as a private library you must know their begin and end locations on the disk volume. This

information is found in the library status report which you can get by running the DSERV program. You should note that, when using the system core image library as a private library, that library does not begin at the low address of the SYSRES extent. For CKD disk devices, although the SYSRES extent begins at cylinder 0, track 1, the library begins at cylinder 0, track 2. For FBA devices SYSRES begins at block 2, and the library begins at block 10. Figure 3-37 is a sample of a status report produced for a SYSRES file on an FBA device.

```
S T A T U S   R E P O R T                    DATE: 06/15/79 (MM/DD/YY)    TIME: 17.30 (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:          (BLOCKND)   (BLOCKND  BYTE)     (BLOCKND)                                     AVAILABLE COND.LIMIT (%)

SYSRES VOL.SER.SYSRES
CORE IMAGE   DIRECTORY        10          59   452            210         201         50                  2500          831  25
             LIBRARY         211        5565                8009        7799       5378          4     2445            0  69

SYSRES VOL.SER.SYSRES
RELOCATABLE  DIRECTORY      8010        8017   434           8210         201          8                  5239          192   4
             LIBRARY        8211        9959                16009       7799       1748          0     6051            0  22

SYSRES VOL.SER.SYSRES
SOURCE-STMT  DIRECTORY     15010       16011     2          16210         201          2                  5246            0   1
             LIBRARY       16211       16211                24009       7799          0          0     7799            0   0

SYSRES VOL.SER.SYSRES
PROCEDURE    DIRECTORY     24010       24011     2          24210         201          2                  5246            0   1
             LIBRARY       24211       24211                32009       7799          0          0     7799            0   0

NUMBER OF ENTRIES IN SYSTEM DIRECTORY LIST:    18
SHARED VIRTUAL AREA ADDRESSES (HEX)      START:  E22BC     NEXT AVAILABLE LOCATION:  FA317     END:  13FFFF
```

Figure 3-37. Library Status Report for SYSRES on an FBA Device

When accessing a system file as a private library the filename of the DLBL statement should reflect the private library name. The file-ID of the DLBL statement must be the original file-ID of the SYSRES file.

The following job stream would be used to merge from a system residence into a duplicate system residence whose 20 cylinder relocatable library is being used as a private library. (Assume the disk packs are 3330s).

```
// JOB MERGE
// DLBL IJSYSRL,'DOS.SYSRES.FILE'
// EXTENT SYSRLB,SYSRES,1,0,570,380
// ASSGN SYSRLB,161
// EXEC CORGZ
   MERGE RES,PRV
   COPYR M001,M002
/*
/&
```

The DLBL/EXTENT statements refer to the target library. DLBL/EXTENT information describing the IPL SYSRES file is assumed to be in the standard label area.

As another example, you may want to create a backup copy of your system core image library as a private library on magnetic tape. The following job stream illustrates the use of the Backup System utility to achieve that. The system core image library takes up blocks 10 through 8009 of an FBA device (see the Status Report in Figure 3-37).

```
// JOB BACKUP
// ASSGN SYS005,UA
// DLBL IJSYSHF,'DOS.SYSTEM.HISTORY.FILE'
// EXTENT SYSREC,,1,0,5339,57          IBM 3330
// DLBL IJSYSCL,'DOS.SYSRES.FILE'
// EXTENT SYS007,,1,0,10,8000
// ASSGN SYS007,131                     SYSRES FILE ON
// ASSGN SYS006,281,C0                  FBA BACKUP TAPE
// EXEC BACKUP
/*
/&
```

Chapter 4: Using the Facilities and Options of DOS/VSE

This chapter discusses ways and means for monitoring certain activities of the DOS/VSE. This involves the coding of user programs to be used as IPL and job control exit routines and the coding of a job accounting interface routine. In addition, this chapter discusses the checkpointing facility, DASD switching under DOS/VSE, and designing programs for virtual mode execution. The SDAID program which is an effective debugging and measurement tool is discussed in *DOS/VSE Serviceability Aids and Debugging Procedures*.

User-Written Program-Exit Routines

Coding an exit routine to be used when specific conditions arise – conditions that can be determined only by the DOS/VSE supervisor – requires that you issue the STXIT macro in your problem program to establish proper linkage to the pertinent exit routine.

Figure 4-1 is a summary of the supervisor-determined conditions for which an exit routine may be coded and the operand to be coded in the STXIT macro.

The STXIT operands and their use are discussed in *DOS/VSE Macro Reference*.

Condition	Operand of the STXIT Macro
Abnormal termination of the problem program	AB
Interval timer external interrupt	IT
Operator communications interrupt	OC
Program check interrupt	PC
Task timer interrupt	TT

Figure 4-1. Summary of Program Exit Conditions

A partition's task can link to a task timer exit routine only if the TTIME parameter in the FOPT generation macro specified that partition.

Writing an IPL User Exit Routine

The IPL Exit allows you to do some processing at the end of IPL and prior to execution of the job control program. You may want to check about the options of the loaded supervisor, for example whether support for job accounting or access control is included.

Before you start coding your exit routine, take account of any system requirements that should be met at the time the routine is to be executed. The exit routine and any routines that are called by your routine must be present in the system core image library.

Moreover, your routine must adhere to the following conventions:

- Register 15 contains the entry point of the routine.
- Register 14 contains the return address to job control.
- The format of the phase card must be as follows:

  PHASE $SYSOPEN.

After IPL, the job control program executes the exit routine as an overlay phase; an area of 4K has been reserved for the exit routine. While the routine is being executed, the job control program is unable to read any job control statements.

In your exit routine, you may issue SVCs and perform I/O operations to SYSLOG and/or SYSRES. To do so, you may only use the EXCP macro. Any use of LIOCS or of a DTFPH would obstruct proper execution of the job control program. If you code your routine in assembler language, use DC instructions instead of DS instructions.

Phase $SYSOPEN will be executed with a storage protect key of zero. If the phase is abnormally terminated, the job control program will be loaded for execution.

Figure 4-2 illustrates a user-written routine that is executed once each time the IPL procedure is performed.

Immediately after IPL, only a few system units are assigned, the most important ones being SYSLOG and SYSRES. If you want to open a job accounting file, place the necessary ASSGN statements, label information (if not already present in the system standard, the partition standard, or the user label area) and EXEC statement for the pertinent job in your ASI BG JCL procedure, ahead of the statements that activate the foreground partitions. This enables you to use the normal facilities of the system, including LIOCS.

```
*   THIS PROGRAM CHECKS WHETHER THE INSTALLATION INCLUDES
*   JOB ACCOUNTING SUPPORT. IPL OF A SUPERVISOR WITHOUT
*   THIS SUPPORT IS CONSIDERED AS NOT ALLOWED.
*   A MESSAGE INFORMS THE OPERATOR WHY HE/SHE HAS TO
*   REPEAT IPL. THEN A HARD WAIT IS FORCED.

          START     0
          USING     *,R15
BEGIN     ST        R14,RETURN            SAVE RETURN ADDRESS
          COMRG     REG=R2
          TM        56(R2),X'80'          JOB ACCOUNTING SUPPORTED?
          BZR       R14                   YES, RETURN TO JOB CONTROL
          LA        R1,LOGCCB             NO, WRITE MESSAGE TO
          EXCP      (1)                   OPERATOR
          WAIT      (1)
          L         R11,HWCODE            LOAD HARD WAIT CODE
          ST        R11,0                 STORE IT IN LOW CORE
          OI        SVCNPSW+1,X'02'       SET ON WAIT BIT
          SVC       7                     FORCE HARD WAIT
SVCNPSW   EQU       96                    LOCATION OF SVC NEW PSW
LOGCCB    CCB       SYSLOG,LOGCCW
LOGCCW    CCW       X'09',LOGMSG,X'20',L'LOGMSG              (column 72)
LOGMSG    DC        C'JOB ACCOUNTING SUPPORT MISSING, RE-IPL       C
                    CORRECT SUPERVISOR'
RETURN    DC        F'0'
HWCODE    DC        C'NOJA'
R0        EQU       0
R1        EQU       1
R2        EQU       2
R11       EQU       11
R12       EQU       12
R13       EQU       13
R14       EQU       14
R15       EQU       15
          END       BEGIN
```

Figure 4-2. IPL User Exit Example

Writing a Job Control User Exit Routine

It is often desirable to exercise certain control on how a job step is executed, thereby enhancing security, serviceability, and reliability. After a job control statement (or command) has been read, control can be passed to a user exit routine for the purpose of examining and altering the statement (or command) before it is processed by job control.

The DOS/VSE distribution volume contains a dummy phase $JOBEXIT in the system core image library which is automatically loaded into the SVA at IPL. If you do not use the Job-control-exit facility, it has no effect on your system.

In your routine you are free to modify the operands of the job control statement and to add comments. You must not, however, modify the operation field of the statement. For example, // EXEC IBM can be modified to // EXEC USER; the operation field (EXEC) cannot be modified. In your exit routine neither perform any I/O operations nor issue any SVCs nor request the system to cancel the job step.

Link-edit your routine to the system core image library using a PHASE statement as follows:

```
PHASE   $JOBEXIT,S[,NOAUTO],SVA[,PBDY]
```

Your routine must be coded reenterable; it must be SVA eligible, and it must reside in the SVA. The PHASE statement must include the SVA parameter. This ensures that when the phase is cataloged it will also be loaded into the SVA replacing the dummy phase provided by IBM.

Phase $JOBEXIT is executed with a storage protection key of zero. The code is shared between partitions.

When your routine receives control, registers contain control information as follows:

Register Number	Contents of Register
0	System identification characters 'SDOS'.
1	Address of partition communication region.
2	Address of system communication region.
3	Address of job control vector table.
4	Address of buffer that contains the currently processed job control statement.
14	Return address to job control.
15	Entry point to $JOBEXIT; at completion of the routine it contains the return code for job control.

Prior to returning control to job control, your routine must store a return code value into register 15:

a zero value – requests job control to continue processing the current statement.

a non-zero value – requests job control to process the statement as if it were an invalid statement.

The vector table whose layout is given below shows which job control statement is being processed by job control. You must not modify its contents. Use it for comparison only. The size of the buffer into which the job control statement is loaded (left-justified) is 120 bytes, the first 71 bytes of which are printed on the console printer. The full length of 120 bytes is printed on the printer assigned to SYSLST. The / & and End-of-job statements are not displayed.

In the buffer, you may modify the statement up to and including byte 71, except for the operation field. Bytes 72-80 could contain a statement identification, such as for procedure overwrites, and therefore should not be modified. After having set the return code, your routine should pass control back to job control.

Layout of the vector table:

Bytes 0 through 6: Operation field (name of job control statement)

Bytes 7 through 9: Internal control information

Do not attempt to modify the table or modify the operation field in the buffer.

Note: *Make sure your exit routine is free of errors that could cause abnormal termination in a production environment.*

Figure 4-3 illustrates a job control user exit routine.

```
// JOB EXIT ROUTINE
// OPTION CATAL,NODECK
 PHASE $JOBEXIT,S,NOAUTO,SVA,PBDY
// EXEC ASSEMBLY
            EJECT
*********************************************************************************
*            THIS PROGRAM, PHASE $JOBEXIT, EXAMINES ALL EXEC CONTROL STATEMENTS
*            AND EXEC COMMANDS WHETHER THEY WANT TO EXECUTE A PROGRAM NAMED:
*            IBM. THIS PROGRAM IS ASSUMED TO BE RESTRICTED FOR GENERAL USE AND
*            THE STATEMENT:
*      [//]  EXEC IBM
*            IS CHANGED TO:
*      [//]  EXEC USER
*            MESSAGE, 'PROG. IBM RESTRICTED FOR ALL USERS', IS PLACED INTO
*            THE EXEC CARD AND PRINTED ON SYSLOG (IF LOG IS ON) AND SYSLST.
*
*
*            THE PHASE NAMED USER MUST BE CATALOGED IN THE CIL
*
*            $JOBEXIT IS REENTERABLE AND SVA ELIGIBLE AND MUST BE
*            LOADED INTO THE SVA.
*********************************************************************************
            EJECT
JOBEXIT     START  0
            BALR   R12,0                  ESTABLISH
            USING  *,R12                  ADDRESSABILITY
```

Figure 4-3. Job Control User Exit Example (Part 1 of 2)

```
*
*         CHECK FOR EXEC STATEMENT
*         REG.3 POINTS TO JOB CONTROL VECTOR TABLE
*
          CLC       EXECNAM,0(R3)          IS IT AN EXEC STATEMENT?
          BNE       RETURN                 IF NOT RETURN
*
*         EXAMINE THE STATEMENT
*         REG.4 POINTS TO STATEMENT BUFFER
*
          L         R6,=F'1'               INCREMENT VALUE FOR SEARCH LOOP
          L         R7,=F'67'              COUNT MAXIMUM FOR SEARCH LOOP
          SR        R5,R5                  CLEAR R5, USED AS INDEXING REG.
*
*         FIND POSITION OF EXEC STATEMENT
*
SEARCHE   EQU       *
          LA        R8,0(R5,R4)            POINT TO INDEXED POS. IN STMNT. BUF
          CLC       EXECNAM,0(R8)          DETERMINE POSITION OF EXEC
          BE        EXFOUND                FOUND THE STATEMENT
          BXLE      R5,R6,SEARCHE          INCREMENT INDEX AND LOOP
          LA        R15,8                  NO EXEC FOUND, RETURN CODE=8
          BR        R14                    RETURN TO CALLER
EXFOUND   EQU       *
          LA        R5,5(R5)               SKIP OVER EXEC TO PROGNAME
SEARCHP   EQU       *
          LA        R8,0(R5,R4)            POINT TO INDEXED POS. IN STMNT. BUF
          CLC       PROGNAM,0(R8)          LOOK FOR PROGRAM-NAME IBM
          BE        PFOUND                 PROGRAM-NAME FOUND
          BXLE      R5,R6,SEARCHP          INCREMENT INDEX AND LOOP
          B         RETURN                 IF ANY OTHER OR NO PROG.-NAME RETURN
*
*         PROGRAM-NAME-IBM-FOUND PROCESSING
*
PFOUND    EQU       *
          LA        R4,0(R5,R4)            POINT TO PROG.-NAME IN BUFFER
          MVC       0(L'USERTXT,R4),USERTXT MOVE USERTXT TO BUFFER
*
*         PREVIOUS MVC CHANGED PROGRAM-NAME IBM INTO PROGRAM-NAME USER
*         AN ADDITIONAL MESSAGE IS MOVED INTO THE BUFFER
*
RETURN    EQU       *
          SR        R15,R15                RETURNCODE ZERO TO REG.15
          BR        R14                    RETURN TO CALLER
EXECNAM   DC        C'EXEC'
PROGNAM   DC        C'IBM'
USERTXT   DC        C'USER *** PROG. IBM RESTRICTED FOR ALL USERS'
R3        EQU       3
R4        EQU       4
R5        EQU       5
R6        EQU       6
R7        EQU       7
R8        EQU       8
R12       EQU       12
R14       EQU       14
R15       EQU       15
          END       JOBEXIT
/*
// EXEC LNKEDT
/&
```

Figure 4-3. Job Control User Exit Example (Part 2 of 2)

Writing a Job Accounting Interface Routine

A DOS/VSE supervisor generation option provides job accounting interface support for all partitions in the system. At the end of each job step or job, accounting information is accumulated in a table for that partition and can be processed by a user-written routine. This routine can extract data for such purposes as charging system usage and supervising system operation, or for planning new applications or changing the system configuration.

The routine must be relocatable, and it must be SVA eligible. With the distribution volume, IBM provides a dummy phase $JOBACCT as part of the system core image library. If you decide to use the job accounting facility, you must catalog your routine to the system core image library. At IPL, the phase is automatically loaded into the SVA.

When you catalog your routine, the PHASE statement must include the SVA parameter; this causes the phase, after it has been cataloged, to be loaded into the SVA replacing the dummy phase provided by IBM.

Since the processing of the information is an overhead element, the user routine should be efficient and avoid unnecessary reduction or reformatting of data.

If your installation uses VSE/POWER with the job accounting facility included, you do not need such a user routine. For more information about this facility under VSE/POWER, refer to the documentation for this licensed programming support.

Job Accounting Information

When support is generated for *basic job accounting*, DOS/VSE includes for each partition in the system a job accounting table comprising fourteen fields. At the end of each job step and job, information is stored in fields 1 to 14 of the Job Accounting table (see Figure 4-4).

In addition, DOS/VSE may be requested (at the time of system generation) to include the *number of SIO* (Start I/O) *instructions* issued per device for each job step and job. The job accounting table for each partition is then extended to contain the additional fields 15 and 16 shown in Figure 4-4.

SIO accounting is performed for the number of devices specified to be supported by the facility for each partition. The maximum is 255 and has no relation to the number of devices specified for the total DOS/VSE. If more devices are accessed than the number specified, SIOs on the excess devices will not be counted.

Field	Displacement	Byte Length	Contents
1	0 - 7	8	Job name. 8-byte character string taken from JOB statement.
2	8 - 23	16	User Information. 16 characters of information taken from the JOB statement.
3	24 - 25	2	Partition ID, BG, . . . , F2, or F1.
4	26	1	Cancel Code. Refer to *DOS/VSE Serviceability Aids and Debugging Procedures*
5	27	1	Type of Record. S = job step; L = last step of job.
6	28 - 35	8	Date when job step started: mm/dd/yy or dd/mm/yy.
7	36 - 39	4	Job Step Start Time. 0hhmmssF, where h hours, m minutes, s seconds, F is a sign (in packed decimal format).
8	40 - 43	4	Job Step Stop Time (in same format as start time).
9	44 - 47	4	Reserved.
10	48 - 55	8	Phase Name. 8-byte character string taken from the EXEC card.
11	56 - 59	4	Real Mode Processing: Number of fixed pages, multiplied by 2K; equivalent to the partition's allocated processor storage minus the portion of the partition GETVIS area that was not used up by GETVIS requests. Virtual Mode Processing: Number of pages referenced in the partition, multiplied by 2K.
12	60 - 63	4	CPU Time. 4 binary bytes given in 300ths of a second. Time is calculated from exit of the user-written routine called during job control to next entry of the routine. Time used by the user-written output routine is charged to overhead of the next record.
13	64 - 67	4	Overhead Time. 4 binary bytes given in 300th of a second. Includes time taken by functions that cannot be charged readily to one partition (such as attention routine and error recovery). System overhead time is distributed to the partitions in proportion to the used CPU time.
14	68 - 71	4	All Bound Time. 4 binary bytes in 300th of a second. This is the time the system is in the wait state divided by the number of partitions running.
15	72 -		SIO Tables. Variable number of bytes. Six bytes are reserved for each device specified in the JA parameter. First two bytes are X'0cuu', next four are hex count of SIOs for job step. Unused entries contain X'10' followed by five bytes of zeros. Stacker select commands for MICR devices are not counted. Error recovery SIOs are not charged to the JOB Accounting Table. Devices are added to the table as they are used.
16		1	Overflow. Normally X'20'. Set to X'30' if more devices are used than set by the JA parameter at system generation time.

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

Figure 4-4. Job Accounting Table

If physical IOCS is used for printing, you must 'space after' to prevent overwriting of job control statements.

For efficiency, an overlay structure should be avoided and the length of the program should preferably not exceed one core image library block.

If the job accounting program is canceled as the result of an error condition, the current information cannot be retrieved, the job accounting information for the current job step is unreliable. However, provision is made that the job accounting information for any subsequent job steps will be correct, provided the cancellation was not caused by an error in the $JOBACCT routine itself. If there was an error in the $JOBACCT routine, it must be corrected first.

In order to avoid unintentional cancellation of the job accounting program by operator action, the operator should issue the MAP command and check the job name for the running partition. If the job name is 'JOB ACCT', the job accounting routine is active; the CANCEL command should not be issued until the original job name is displayed after another MAP command.

Register Usage. Important data for the user's job accounting routine are passed in the following general registers:

12 Base address for $JOBACCT
15 Address of the job accounting table
11 Length of the job accounting table
13 Address of the user save area
14 Return address to job control

If $JOBACCT uses LIOCS, the contents of general registers 14 and 15 must be saved (also registers 0 and 1 if necessary) because LIOCS uses these registers.

Save Area for the User's Routine. The address of a save area that can be used by the job accounting routine is passed in general register 13. This save area is 16 bytes long unless a greater length (up to 1024 bytes for saving DTF information for LIOCS) was specified at system generation time. However, CCBs and executable CCWs must not be included.

User's Area for LIOCS Label Processing. If your job accounting routine uses LIOCS for processing such items as standard tape labels, DTFDA, or DTFPH with MOUNTED=ALL, then an alternative label area must be specified at system generation. The length of this label area should normally be the number of bytes that would be allocated by a given parameter of the LBLTYP statement. For information on determining the number of bytes, see *DOS/VSE System Control Statements*.

Tailoring the Program

The requirements of the program may be simply to record the accounting information as part of the SYSLST output for each job step or job, or it may be to accumulate information to be used for equitably allocating the costs of a computing center.

If data is to be written out on a disk or tape, the save area can be used for communicating between job steps. Such information as the disk address for the next record or an indication that tape labels have been successfully processed, or even the DTF used to control the output, may be stored in the save area.

Figure 4-5 illustrates a job accounting program that writes records to disk without additional processing.

```
JAACT    CSECT
         USING   *,R12
         USING   JASAVE,R13            JOB ACCT SAVE AREA
         LR      R9,R15                SAVE ADDR OF TBL
         LA      R0,JADTFLNG+L'JABSAVE LENGTH FOR GETVIS
         GETVIS  LENGTH=(0)            GET SPACE IN PARTITION
         LTR     R15,R15               CHECK RETURN CODE
         BNZ     JARET1                NO GETVIS SPACE
         LA      R0,JABROUT            AB' ROUTINE
         STXIT   AB,(0),(1)            SET ABNRML TERM EXIT
         LA      R1,L'JABSAVE(R1)      UPDATE GETVIS POINTER
         TM      JASTATSW,X'C0'        TEST STATUS
         BO      JARET                 DISK AREA FULL
         BM      JAOPEN                SAVE AREA INITIALIZED
* PERFORM LABEL PROCESSING AND INITIALIZE SAVE AREA
         MVC     0(JADTFLNG,R1),JADTF  MOVE DTF TO PARTITION
         OPENR   (R1)                  OPEN FILE (see Note)
         MVC     JACCB,0(R1)           MOVE CCB TO SAVE AREA
         MVC     JASEEK,58(R1)         EXTENT LOWER LIMIT
         MVI     JAR,X'01'             FIRST RECORD
         MVC     JAHIGH,JADTF+54       HIGH EXTENT LIMIT
* RELOCATE CCWS
         MVC     JASKCCW(32),JAMODCCW  PUT MOD CCWS IN SVE AREA
         LA      R10,JASEEK            SEEK ADDRESS
         STCM    R10,7,JASKCCW+1       PUT ADDRESS IN CCW
         LA      R10,JASRCH            SEARCH ADDRESS
         STCM    R10,7,JASRCCW+1       PUT ADDRESS IN CCW
         LA      R10,JASRCCW           SEARCH CCW ADDRESS
         STCM    R10,7,JATIC+1         PUT ADDRESS IN CCW
         LA      R10,JASKCCW           CHANNEL PROGRAM ADDR
         STCM    R10,7,JACCB+9         PUT ADDRESS IN CCB
         MVI     JASTATSW,X'80'        IND SAVE AREA INIT
* WRITE JOB ACCOUNTING TABLE TO DISK
JAOPEN   STCM    R9,7,JADATA+1         PUT ADDR OF TBL IN CCW
         MVC     0(16,R1),JACCB        MOVE CCB TO PARTITION
         EXCP    (1)                   WRITE DATA
         WAIT    (1)                   WAIT FOR COMPLETION
* UPDATE SEEK ADDRESS
         TR      JAR,JARECTAB          RECORD
         CLI     JAR,X'01'             NEW TRACK
         BNE     JARET                 NO
         TR      JAHEAD+1(1),JAHDTAB   HEAD
         CLI     JAHEAD+1,X'00'        NEW CYLINDER
         BNE     JAHTST                NO
         LH      R10,JACYL             CYLINDER ADDRESS
         LA      R10,1(R10)            INCREMENT BY ONE
         STH     10,JACYL              REPLACE IN SEEK ADDR
JAHTST   CLC     JAHIGH,JASRCH         BEYOND UPPER LIMIT
         BH      JARET                 NO
         MVC     0(16,R1),JACCBL       MOVE CONSOLE CCB TO PARTITION
         LA      R2,JAMSG1             ERROR MESSAGE
         STCM    R2,7,9(R1)            PUT ADDRESS IN CCB
         EXCP    (1)                   INFORM OPERATOR
         WAIT    (1)                   WAIT FOR COMPLETION
         OI      JASTATSW,X'40'        INDICATE DISK FULL
JARET    FREEVIS LENGTH=(0)            FREE PARTITION SPACE
         STXIT   AB                    RESET EXIT LINKAGE
JARET1   BR      R14                   RETURN
```

Note: *As this example is self relocating, the self-relocating form of the OPEN macro (OPENR) is used; for a routine that will be linked relocatable, OPEN may be used instead.*

Figure 4-5. Job Accounting Routine Example (Part 1 of 2)

```
JABROUT  LA       R1,L'JABSAVE(R1)      RESTORE ADDR IN GETVIS AREA
         MVC      0(16,R1),JACCBL       MOVE CONSOLE CCB TO PARTITION
         LA       R2,JAMSG2             ERROR MESSAGE
         STCM     R2,7,9(R1)            PUT ADDRESS IN CCB
         EXCP     (1)                   INFORM OPERATOR
         WAIT     (1)                   WAIT FOR COMPLETION
         EOJ
JAMODCCW CCW      X'07',*,X'60',6
         CCW      X'31',*,X'60',5
         CCW      X'08',*,X'00',1
         CCW      X'05',*,X'20',246
JACCBL   CCB      SYSLOG,*
JABSAVE  DS       0CL72
JADTF    DTFPH    TYPEFLE=INPUT,        MEANS CHECK LABELS
                  DEVICE=2314,
                  MOUNTED=SINGLE
JADTFLNG EQU      *-JADTF
         ORG      JADTF
         DC       X'00000B00'           SET CCB OPTION BITS
         ORG
JAMSG1   CCW      X'09',JAERR1,X'20',L'JAERR1
JAMSG2   CCW      X'09',JAERR2,X'20',L'JAERR2
JAERR1   DC       C'JOB ACCOUNTING DISK FULL'
JAERR2   DC       C'JOB ACCOUNTING ROUTINE CANCELED'
JARECTAB DC       X'000203040506070809 0A0B0C0D0E0F101112131401'
JAHDTAB  DC       X'010203040506070809 0A0B0C0D0E0F1011121300'
JASAVE   DSECT
JASEEK   DS       0XL6                  SEEK ADDRESS BBCCHH
JABB     DS       XL2                   BB
JASRCH   DS       0XL5                  SEARCH ADDRESS CCHHR
JACYL    DS       XL2                   CC
JAHEAD   DS       XL2                   HH
JAR      DS       X                     R
JASTATSW DS       X
JACCB    DS       XL16                  COMMAND CONTROL BLOCK
JAHIGH   DS       XL4                   HIGH EXTENT LIMIT
         DS       XL4
JASKCCW  CCW      X'07',JASEEK,X'60',6  SEEK CCW
JASRCCW  CCW      X'31',JASRCH,X'60',5  SEARCH CCW
JATIC    CCW      X'08',JASRCCW,X'00',1 TIC CCW
JADATA   CCW      X'05',*,X'20',246     WRITE DATA ASSUMING 29
*                                       SIO DEVICES TRACED
R0       EQU      0
R1       EQU      1
R2       EQU      2
R9       EQU      9
R10      EQU      10
R11      EQU      11
R12      EQU      12
R13      EQU      13
R14      EQU      14
R15      EQU      15
         END
```

Note: *The DSECT labeled JASAVE through JADATA defines the layout of the job accounting user-save area, which resides within the supervisor. The address of this area is passed, in register 13, to your job accounting phase. When generating your supervisor you must specify the desired length of this save area by substituting a value for s, the first operand of the JALIOCS parameter of the FOPT macro. If the operand is omitted or if JALIOCS=NO is specified the length of the user save area is set to 16 bytes by default.*

Figure 4-5. Job Accounting Routine Example (Part 2 of 2)

Checkpointing Facility

The progress of a program that performs considerable processing in one job step should be protected against destruction in case the program is canceled. DOS/VSE provides support for taking up to 32,767 checkpoint records in a job. Through this facility, information can be preserved at regular intervals and in sufficient quantity to allow restarting a program at an intermediate point.

The CHKPT macro (or the corresponding high-level language statement) causes DOS/VSE to store the checkpoint record on a magnetic tape or disk. For more details about taking checkpoints, refer to *DOS/VSE Macro Reference* if you use assembler language or to the appropriate high-level language manual.

The RSTRT job control statement restarts the program from the last or any specified checkpoint taken before cancelation.

When a checkpointed program is to be restarted, the partition must start at the same location as when the program was checkpointed and its end address must not be lower than at that time unless a lower end address was specified in the CHKPT macro instruction. Unless the user reestablishes all linkages to SVA phases himself, the contents and location of the modules in the SVA when restarting must also be the same as when the program was checkpointed. The SDL must be identical if the restarted program uses a local directory list (for example, one that was generated by the assembler language macro GENL).

If any pages of a virtual mode program were fixed when the checkpoint record was taken, then, in 370 mode, the real address area allocation for the partition must also start at the same or a lower location and its end address must be at least as high as at that time. The pages that were fixed are refixed by the supervisor when the program is restarted.

Restarting a Program from a Checkpoint

To restart a program from a checkpoint the RSTRT job control statement is used. The sequence of job control statements that must be submitted to restart a program is as follows:

1. A JOB statement specifying the jobname used when the checkpoints were taken.
2. ASSGN statements, if necessary, to establish the I/O assignments for the program that is to be restarted.
3. A RSTRT statement specifying
   a) the symbolic name of the tape or disk device on which the checkpoint records are stored.
   b) the sequence number of the checkpoint record to be used for restart.
   c) for checkpoint records on disk the filename (DTF name) of the checkpoint file.
4. An end-of-job (/ &) statement.

Figure 4-6 shows the sequence of job control statements needed to restart a checkpointed program that ended abnormally due to, for example, a power failure. Following are the characteristics of the checkpointed program that must be considered for restart:

- The job name specified in the JOB statement was CHECKP; the same name must be used for restart.
- The checkpoint records were written on magnetic tape; therefore, no filename need be specified in the RSTRT statement.
- The symbolic device name SYS006 is used for the checkpoint file.
- The sequence number of the last checkpoint record written was 0013; this or any previous checkpoint record can be used for restart (the sequence numbers are printed by DOS/VSE on the SYSLOG device).

In reconstructing the job stream note that the // RSTRT statement physically and functionally replaces the // EXEC statement originally used.

Another important consideration is the repositioning of files on magnetic tape or disk. Assembler language users may consult *DOS/VSE Macro Reference*, which discusses the topic in context with using the CHKPT macro. High-level language users should consider printing a file processing status record for each checkpoint that is taken during the execution of a program. This record should indicate the name of the file(s) read or written on magnetic tape or disk when DOS/VSE takes the checkpoint.

```
// JOB CHECKP
// ASSGN SYS006,380      CHKPT TAPE
// ASSGN ...
// ASSGN ...
// RSTRT SYS006,0013
/&
```

Figure 4-6. Example of a RESTART Job

DASD Switching under DOS/VSE

The standard I/O interface between an I/O device and the CPU is a channel and a control unit.

Normally, this interface provides one, and only one, path by which a CPU communicates with an I/O device. However, it may be desirable to access a device, especially a DASD device, by more than one path. For example, a second CPU may be required to back-up the host CPU such that should the host CPU become inoperable, the attached DASD devices may be switched immediately to (made accessible to) the back-up CPU. Multiple CPUs may also need to access the same data base.

A single CPU may require back-up channels and control units, providing alternate paths to the same DASD devices.

In order to do this device sharing, the hardware provides a two-level switching mechanism that allows you to connect one or more DASDs either dynamically or manually to different I/O paths. This mechanism is known as channel switching and string switching.

Channel Switching. Channel switching provides the switching mechanism at the control unit level. The channel switch allows you to connect the control unit to up to four channels, which may belong to the same or different CPUs thus providing up to four distinct I/O paths. A maximum of two channels may connect to one CPU. The connection of any channel can be manually enabled or disabled. When enabled, the switch is dynamically controlled by the hardware.

String Switching. In the case of string switching, the switching mechanism is at the DASD string level. String switching allows you to connect a string of DASDs to two distinct control units, or integrated disk attachments. The two I/O paths may be connected to a single or two different CPUs.

Using DASD Switching. In both types of this hardware-supported switching, a desired I/O path may be selected in one of two ways. In the first case, connection is made dynamically when an I/O command is issued for a device. Provided that the control unit (in channel switching) and the DASD string (in string switching) are free for connection, the target DASD device can be accessed by the requesting CPU. Once a connection is established by one CPU, the other CPU receives device busy status if attempting to access a device on the string.

In the second case, the operator may manually switch the sharable devices to the desired CPU (via the Enable/Disable toggle switches). It should be noted that in this case an entire string of DASD is disconnected from the other CPU.

If, at your installation, a DASD switching feature is being used, it is your responsibility to resolve conflicting CPU references to shared devices (or files) and thus ensure data integrity. Following are two ways of preventing potential conflicts.

First, through scheduling of CPU file referencing, ensure that only one CPU that is updating the file is connected to the shared DASD. The operator needs only to switch the manual control to the updating CPU for that period of time.

Secondly, through scheduling and the use of the operator commands DVCUP and DVCDN (as described below), devices may be reserved for use by one CPU for a particular period of time.

An individual device can be excluded from use by a particular CPU by entering a DVCDN command for that device via the operator console. The other system then has exclusive access to that device. The device can be made available again by issuing a DVCUP command for the device. However, the other system should then issue a DVCDN command for that device. To avoid conflicts, both system operators have to inform each other about the status of the reserved devices. It is therefore recommended that a job, which requires exclusive access to a file or device, notifies the operator when the device has to be reserved, and when it may be released.

Note that the DVCUP/DVCDN commands reserve the DASD at the device level, although the programmer may be interested in reserving only one file on that particular device. It is recommended that DVCUP and DVCDN commands be entered only via the console.

Further hardware details on channel or string switching may be found in the appropriate DASD hardware manuals, and also in the hardware manuals for the IBM 370/115, 370/125.

When using DASD Switching, in order to facilitate the diagnosis of hardware failures, the inclusion of Recovery Management Support (RMS) is required. For System/370 Models 115 or 125, RMS may be included during DOS/VSE system generation by specifying RMS=YES in the SUPVR macro.

Designing Programs for Virtual Mode Execution

This section describes programming techniques that may improve the efficiency of programs that execute in virtual mode. Consider these techniques for new programs to be written and old programs to be revised. The section also contains information on the use of certain macros that are provided especially for virtual storage. Programming conventions for the shared virtual area are also discussed.

Programming Hints for Reducing Page Faults

It is definitely worthwhile to spend some extra programming effort for tuning virtual-mode programs that are used frequently or that require long periods of processing time so that they will cause fewer page faults during execution. Page faults generally occur when the size of the virtual-mode program exceeds the number of page frames available to it during execution. Efforts to reduce the number of page faults occurring in a program generally involve techniques for reducing the size of the *working set* of the program. The term *working set* is one that recurs often in discussions of virtual storage systems.

The working set of a program is comprised of those program pages that contain the most frequently used sequences of instructions for a given period of time. The working set of a program is not a fixed number of pages or instructions of that program; this set changes as the execution of the program proceeds. For example, a program doing an internal sort and writing a formatted table based on the results of this sort would have two completely different basic working sets; one for the sort function and one for the write functions.

What does *execute efficiently* mean? Essentially, this means that a program will not execute appreciably slower than if the entire program were in processor storage during its entire execution.

Although the following section does not tell you how to determine the size of the working set, it does provide techniques for reducing its size.

General Hints for Reducing the Working Set

There are three general rules to keep in mind when working toward a reduction of a program's working set. The first is *locality of reference,* that is, instructions and data used together should be in storage near each other. Second is *minimum processor storage.* In other words, the amount of processor storage necessary for a program to do something should be kept as low as possible. Third is *validity of reference,* that is, references should be made only to data which will actually be used.

The chief means of achieving locality of reference is to make execution sequential whenever possible by avoiding excessive branching.

A program that executes sequentially normally requires a partition larger than the same program when it does not execute sequentially. For example, the functions of a section of code repeat themselves several times throughout the logic of your program. You are tempted to write this code once and branch to it whenever necessary, but branching violates the principle of locality of reference. Branching may cause more page faults than would coding the routine in line each time it is used. Also, it is easier for someone else to follow the logic of a program which is written to execute sequentially.

Locality of reference can be achieved only to a limited extent by programs written in a high-level language.

Elements in arrays in FORTRAN or PL/I can be referred to in the order in which they appear in storage. In FORTRAN, for example, arrays are ordered by columns. The elements of the array DIMENSION (2,2,2) are arranged as follows in contiguous virtual storage locations:

(1,1,1) (2,1,1)
(1,2,1) (2,2,1)
(1,1,2) (2,1,2)
(1,2,2) (2,2,2)

For array structures of other compilers, refer to the appropriate programming language reference manuals.

A routine which processes all the elements of such an array should refer to them in this order. If only certain elements of an array are processed, the elements should be arranged in the order in which they are to be processed. If arranging an array in a certain manner causes it to be processed advantageously one time, but disadvantageously another time, you should consider writing two arrays, even at the cost of additional virtual storage.

In an assembler language program, you should keep frequently used data and constants near each other in storage, and near the instructions which use them. This contrasts with the traditional practice of having one area at the end of the program reserved for all the data areas and constants. By the same token, seldom used data should be separated from the frequently used data and placed with the routines which use it.

Avoid, if possible, using chains which must be searched each time a data item is required. If chains are unavoidable they should be kept in a

compact area of storage. This may result in some wasted (virtual) storage but will be better than searches of large areas of storage.

Another good practice to help reduce paging is to initialize variables just before they are to be used. For example in PL/I instead of the following:

```
DCL A FIXED INIT (10);
.
.
DO B=1 TO 100;
A=A+B;
END;
```

use:

```
DCL A FIXED;
.
.
A=10;
DO B=1 TO 100;
A=A+B;
END;
```

In the first method of coding, PL/I initializes the automatic variable at the beginning of execution. The second method of coding does not require the page containing A to be in processor storage until just before A is used.

An important help in reducing the amount of processor storage needed for execution is to keep coding used for errors or other unusual occurrences in a separate routine. If, for example, the main routine contains code for conditions that occur only 5% of the time, by moving this error code to a separate section of your program, you can reduce the amount of needed processor storage for 95% of the processing.

Frequently-used subroutines should be loaded near each other. Because of their frequent use, these routines tend to be in processor storage almost continuously. If they are scattered over several pages, each of these pages will need to be in processor storage most of the time, thus increasing the size of the working set. By loading these routines near each other, you reduce the number of pages required in processor storage at any one time.

Subroutines should be designed to do as much processing as possible whenever they are called. It is better to duplicate some code from the calling routine in the called routine in order to avoid switching back and forth between routines. One technique for accomplishing this is to have the calling program pass several parameters to the subroutine and make one call, rather than passing one parameter at a time and make several calls.

You should try to keep code that can be modified and code that cannot be modified in separate sections of a large program. This will reduce page traffic by reducing the number of pages that are changed. Also, try to prevent I/O buffers from crossing page boundaries unnecessarily. Check the assembler listing and the linkage editor map to determine where 2K boundaries occur in your programs.

Using Virtual Storage Macros

The macros designed for use by virtual-mode programs, which are discussed below, perform the following services:

- fix pages in processor storage (PFIX macro) and later free the same pages for normal paging (PFREE macro).
- indicate the mode of execution of a program (RUNMODE macro).
- influence the paging mechanism in order to reduce the number of page faults, to minimize the page I/O activity, and to control the page traffic within a specific partition.

In order to use these macros you must be programming in assembler language or, if your program is written in a high-level language, you must write an assembler subroutine to make use of them. Refer to *DOS/VSE Macro Reference* for a detailed description of these macros.

Fixing Pages in Processor Storage

In DOS/VSE, parts of virtual-mode programs must be in processor storage only at certain times. These parts include not only the instructions and data being processed at any one moment, but also data areas for use by channel programs. Instructions and data are always in processor storage when being used. Because of the nature of I/O operations, the data areas for these operations could be paged out during the I/O operation if something were not done to keep them in processor storage during the entire operation. DOS/VSE therefore fixes I/O areas in processor storage for the duration of the I/O operation.

There are other parts of a program, however, which cannot tolerate paging, and these parts are not necessarily kept in storage by DOS/VSE. For instance, programs that control time-dependent I/O operations cannot tolerate paging. A familiar example is a MICR (Magnetic Ink Character Reader) stacker select routine. If a page fault were to occur during the execution of one of these programs, the results would be unpredictable. A page fault in one of these programs can be avoided by fixing the affected pages in processor storage, using the PFIX macro.

The pages that you fix by the PFIX macro are fixed in the processor storage allocated to the partition in which the PFIX request is issued. Only as many pages may be fixed by a program at any one time as there are page frames allocated to the partition. This is done to prevent a loop in one program from fixing all the pages in the system, and to enable other programs to issue a PFIX macro concurrently.

The PFIX macro fixes the pages in processor storage, regardless of whether these pages are stored in contiguous page frames or not. The supervisor keeps a count of the number of times a page has been fixed without being freed. A page that is fixed more than once without having been freed (via the PFREE macro) is not brought in a second time and given another page frame. Instead, the counter for that page is just increased by one and the page remains in the same page frame.

The PFREE macro does not directly free a page for paging out, but each time it is issued, the counter of fixes is reduced by one. As soon as the counter for a page reaches zero, the page can be paged out. At the end of a job step, all pages that have been fixed during the job step are freed.

The PFREE macro should be used as soon as possible to make a maximum possible number of page frames available to all programs running in virtual mode.

Figure 4-7 is a skeleton example using the PFIX and PFREE macros. After the execution of a PFIX macro, a return code is given in register 15. The meanings of the return codes are:

0 - The pages were fixed successfully.

4 - You requested more page frames than the number of PFIXable page frames available to the partition.

8 - Insufficient free page frames were available.

12 - You specified invalid addresses in your macros.

Note in the example how the return code can be used to establish a branch to parts of the program that handle these specific conditions.

```
            .
            .
FIXER       PFIX    ARTN,ARTNEND+2 FIX ARTN IN STORAGE
            B        *+4(15) BRANCH ACCORDING TO RETURN CODE
            B        HERE     CONTINUE IF OK
            B        NOPAGES GO TO CANCEL IF PART TOO SMALL
            B        WAIT     GO TO WAIT UNTIL PAGES FREED
            B        CANCL    GO TO CANCEL IF ADDR INVALID
HERE        BAL     14,ARTN GO TO ARTN
            PFREE   ARTN,ARTNEND+2 FREE ROUTINE AFTER EXECUTION
            ...
ARTN         (time dependent processing which cannot be
            paged out during execution)

ARTNEND     BR       R14       RETURN
...
NOPAGES     LA       R1,OPCCB
            EXCP     (1)       WRITE MESSAGE TO OPERATOR
            WAIT     (1)       WAIT FOR COMPLETION
CANCL       CANCEL ALL
...
WAIT         (routine to free other pages)

END         EOJ
OPCCB       CCB     SYSLOG,OPCCW
OPCCW       CCW     X'09',MSG,X'20',61
MSG         DC      CL32'AM CANCELING PLEASE ENLARGE REAL'
            DC      CL29'ADDR AREA AND RESTART THE JOB'
            .
            .
```

Figure 4-7. PFIX and PFREE Example

Indicating the Execution Mode of a Program

You may have a program that must do different processing depending upon its execution mode. It may be impractical to have two separate programs cataloged in the core image library (one program for real mode and another program for virtual mode). The RUNMODE macro can be issued during the execution of the program to inquire which mode of execution is being used. A return code is issued to the program in register 1.

Influencing the Paging Mechanism

Releasing Pages. With the RELPAG macro, you inform the page management routines that the contents of one or more pages is no longer required and need not be saved on the page data set. Thus, page frames occupied by these released pages can be claimed for use by other pages, and page I/O activity is reduced.

Forcing Page-out. The FCEPGOUT macro is used to inform the page management routines that one or more pages will not be needed until a later stage of processing. The pages are given the highest page-out priority, with the result that other pages, which may be needed immediately, are kept in storage. Except when the RELPAG macro is in operation, the contents of any pages written out are saved.

Page-in in Advance. The PAGEIN macro allows you to request that one or more pages be read into processor storage in advance, in order to avoid page faults when the specified pages are needed in processor storage. If the specified pages are already in processor storage when the macro is issued, they are given the lowest priority for page-out.

Balancing Telecommunication Activity

The use of telecommunication and production processing at the same time may, occasionally, result in long or erratic telecommunication response times. This may be especially true if you have overcommitted processor storage, thus causing excessive paging. The telecommunication application may have to compete so strongly for page frames (because of high processing activity in the other partitions) that response time increases substantially.

Telecommunication balancing improves response time by trading off telecommunication response time against production partition throughput. TP balancing tends to reduce response times, or at least to stabilize them.

After IPL, TP balancing can be activated by the operator issuing the TPBAL command, which specifies the number of partitions that can tolerate delayed processing. These will be the lowest priority partitions. The TPBAL command is also used to change or display the current setting (for more information, see the *DOS/VSE Operating Procedures*). Once activated, the TP balancing function can be invoked by using TPIN/TPOUT macros.

The TPIN macro signals to DOS/VSE that an immediate demand for system resources is being made by the telecommunication application, for instance, when a message has arrived. After processing is completed, TPOUT informs DOS/VSE that the telecommunication application has no further processing to do for the time being, and that the system resources that were exclusively used for telecommunication should be released. Failure to issue the TPOUT macro can cause serious performance degradation in production partition throughput.

The TPIN and TPOUT macros have been made available primarily for use in IBM licensed telecommunication support (for example, ACF/VTAM and CICS/VS). There is no need for these macros to be used in user-written application programs that run under control of IBM supplied telecommunication support.

Coding for the Shared Virtual Area

Besides accommodating the system directory list (SDL), and perhaps the VSE/VSAM phases with their associated GETVIS work area, the shared virtual area (SVA) contains phases that can be used concurrently by more than one partition. The SVA phases must be reenterable and relocatable; code that modifies itself will cause a protection check when executed from the SVA. This section presents some advice on coding phases to use SVA facilities and suggests some standards for base-register usage.

The basic assumptions for coding an SVA phase are:

- The reenterable code must not modify any storage within its own storage area. Therefore, the code must not contain DTFs, CCBs, or other control blocks that are modified during execution.
- The phase can modify registers only if it saves and restores them for each user.
- A user-specified work area (within the calling partition) must be provided for storing registers and for any storage modifications.

Suggested register conventions:

- Use register 12 as the base register in both the main routine and the reenterable code.
- Use register 13 as base for the working storage area. It is the responsibility of the main routine to provide addressability to the work area by loading register 13; the reenterable routine must not modify register 13. The easiest way to address the working storage area in the reenterable code is by a DSECT that defines the fields of the work area and a USING dsectname,13. In this way symbolic addressing can be used.
- Use CALL, SAVE, and RETURN macros. Since register 13 is the base register, SAVE (14,12) and RETURN (14,12) result. Use register notation for CALL, for example, CALL (15) .... Before issuing the CALL, load register 15 with the transfer address. Register 14 will always contain the return address. The standard is thus established of register 15 for calling and register 14 for returning.

- Switches, and other areas that may be modified, can be placed in the working storage area using base register 13.

Figure 4-8 illustrates the suggested conventions: MASTER is the main routine, SLAVE is the SVA phase.

```
MASTER    CSECT
          BALR      12,0
          USING     *,12
          LA        13,SAVE
          LOAD      SLAVE,WORKAREA          CANCELS IF SLAVE NOT IN CIL
*                                           LOADS SLAVE INTO WORKAREA
*                                           IF SLAVE IS NOT IN SVA
          LR        15,1
          CALL      (15),(SWITCH,TECB,FIELDA,FIELDB,WORKAREA)
          .
          .
          EOJ
SAVE      DS        9D
WORKAREA  DS        200D                    SLAVE IS LOADED HERE
*                                           IF NOT IN SVA
SWITCH    DC        XL1'00'
TECB      DS        CL4
FIELDA    DS        CL15
FIELDB    DS        CL11
          END

SLAVE     CSECT                             MUST BE SEPARATE ASSEMBLY
          SAVE      (14,12)
          BALR      12,0
          USING     *,12
          USING     WORKAREA,6
          LM        2,6,0(1)
          MVC       0(15,4),DATA1
          MVC       0(11,5),DATA2
          CLI       0(2),X'FF'
          BE        EXIT
          SETIME    3,(3)                   SETIME ALTERS THE TECB
          WAIT      (3)
          .
          .
EXIT      XI        0(2),X'FF'
          RETURN    (14,12)
DATA1     DC        CL15'THIS IS FIELDA'
DATA2     DC        CL11'THIS IS FIELDB'
          LTORG
WORKAREA  DSECT
FIELDC    DS        3D
FIELDD    DS        3D
          END
```

Figure 4-8. Example of Conventions for SVA Coding

Appendix A: System Layout on Disk

Figures A-1 and A-2 illustrate how the system residence (SYSRES) file is organized. The volume containing the system residence file can be any IBM DASD device supported by DOS/VSE except a 2311 disk.

IPL Records

This area contains the initial program load (IPL) bootstrap records, which cause the IPL retrieval program to be read from SYSRES and loaded into processor storage. For CKD devices the IPL retrieval program is at cylinder 0, track 1, record 5. For FBA devices it is contained within blocks 3 through 9.

System Volume Label

The volume label (VOL1 label) contains the address of the volume table of contents (VTOC) established when the pack was initialized. (The DOS/VSE system utility program Initialize Disk is provided for this purpose). The VTOC must be located outside of the SYSRES extent.

User Volume Label

The user volume label area is provided for any additional standard volume labels (VOL2-VOL8 labels). This area can extend from record 4 through the end of track 0 on CKD devices or from the end of the system volume label to the end of block 1 on FBA devices.

System Directory

The SYSRES file starts with the system directory. This directory contains the starting addresses of the 4 library directories and the address of the label information area.

Library Directories and Libraries

The purpose of these areas of the SYSRES file is discussed in Chapter 3 of this manual.

Label Information Area

The SYSRES file ends with the label information area. The purpose of this area is described in Chapter 2 of this manual.

Component		Starting Disk Address			Number of Tracks (Alloc.)	R = Required O = Optional
		CC	HH	R		
IPL Record	(Phase $$A$IPL1)	00	00	1	1	R
IPL Record		00	00	2		R
System Volume Label		00	00	3		R
User Volume Label		00	00	4		O
System Directory	Record 1	00	01	1	1	R
	Record 2	00	01	2		R
	Record 3	00	01	3		R
	Record 4	00	01	4		R
IPL Records (Phase $$A$PLBK)		00	01	5		R
Core Image Directory	Cataloged Phases	00	02		*	R
	Linked Phase					
Core Image Library Members		X	Y+1	1	*	R
Relocatable Directory		Z+1	00	1	*	O
Relocatable Library Members		X	Y+1	1	*	O
Source Statement Directory		Z+1	00	1	*	O
Source Statement Library Members		X	Y+1	1	*	O
Procedure Directory		Z+1	00	1	*	O
Procedure Library Members		X	Y+1	1	*	O
Label Information Area		Z+1	00	1	Device dependent	R

* Allocation Dependent on User Requirements
X = Ending CC of the Preceding Directory
Y = Ending HH of the Preceding Directory
Z = Ending CC of the Preceding Library
Note: *Track 0 of cylinder 0 is not part of the SYSRES file.*

Figure A-1. System Residence Organization on CKD Devices

Component	Starting Disk Address Block Number	Number of Blocks	R=Required O=Optional
IPL Records (Phase $$A$IPL0)	0	1	R
System Volume Label[1]	1	1	R
System Directory	2	1	R
IPL Retrieval Program (Phase $$A$PLBF)	3	7	R
Core Image Directory	10	*	R
Core Image Library Members	X+1	*	R
Relocatable Directory	Y+1	*	O
Relocatable Library Members	X+1	*	O
Source Statement Directory	Y+1	*	O
Source Statement Library Members	X+1	*	O
Procedure Directory	Y+1	*	O
Procedure Library Members	X+1	*	O
Label Information Area	Y+1	200 [2]	R

* = Allocation dependent on user requirements

X = Last block of preceding directory

Y = Last block of preceding library

[1] Optional user volume labels if written will be in the same block following the system volume label.

[2] Using the Restore program you may allocate a label information area different than the default size of 200 blocks.

Note: *Blocks 0 and 1 are not part of the SYSRES file.*

Figure A-2. System Residence Organization on FBA devices

Glossary

This glossary defines the terms proper to this manual. If you do not find the term you are looking for, refer to the *IBM Data Processing Glossary,* GC20-1699.

This glossary includes definitions developed by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO). This material is reproduced from the American National Dictionary for Information Processing, copyright 1977 by the Computer and Business Equipment Manufacturers Association, copies of which may be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018. American National Standard Definitions are marked with an asterisk (*).

access method: A technique for moving data between virtual storage and input/output devices.

access method services: A multifunction service program that defines VSAM files and allocates space for them, converts indexed-sequential files to key-sequenced files with indexes, modifies file attributes in the catalog, reorganizes files, facilitates data portability between operating systems, creates backup copies of files and indexes, helps make inaccessible files accessible, and lists the records of the files and catalogs.

address: (1) An identification, as represented by a name, label, or number, for a register, location in storage, or any other data source or destination such as the location of a station in a communication network. (2) Loosely, any part of an instruction that specifies the location of an operand for the instruction.

address translation: The process of changing the address of an item of data or an instruction from its virtual address to its real storage address. See also dynamic address translation.

alternate track: One of a number of tracks set aside on a disk pack for use as alternatives to any defective tracks found elsewhere on the disk pack.

application program: A program written by a user that applies to his own work.

assembler language: A source language that includes symbolic machine language statements in which there is a one-to-one correspondence with the instruction formats and data formats of the computer.

asynchronous operator communication A facility which allows the operator to defer the reply to a message that requires an operator's response.

attach: (1) To create a task and present it to the supervisor. (2) A macro instruction that causes the control program to create a new task and indicates the entry point in the program to be given control when the new task becomes active.

auxiliary storage: Data storage other than real storage; for example, storage on magnetic tape or disk. Synonymous with external storage, secondary storage.

blocking: Combining two or more logical records into one block.

blocking factor: The number of logical records combined into one

physical record or block.

book: A group of source statements written in any of the languages supported by DOS/VSE and stored in a source statement library.

buffer: An area of storage that is temporarily reserved for use in performing an input/output operation, into which data is read or from which data is written. Synonymous with I/O area.

byte: A sequence of eight adjacent binary digits that are operated upon as a unit and that constitute the smallest addressable unit of the system.

card punch: A device to record information in cards by punching holes in the cards to represent letters, digits, and special characters.

card reader: A device which senses and translates into machine code the holes in punched cards.

cardless system: A System/370 Model 115/125 configured without a card reader or card punch, but with an IBM 3540 Diskette Input/Output Unit.

catalog: To enter a phase, module, book, or procedure into one of the system or private libraries.

* **central processing unit:** A unit of a computer that includes the circuits controlling the interpretation and execution of instructions. Abbreviated CPU.

channel: (1) * A path along which signals can be sent, for example, data channel, output channel. (2) A hardware device that connects the CPU and real storage with the I/O control units.

channel program translation: In a copy of a channel program, replacement, by software, of virtual addresses with real addresses.

compile: To prepare a machine language program from a computer program written in a high-level language by making use of the overall logic structure of the program, or generating more than one machine instruction for each symbolic statement, or both, as well as performing the function of an assembler.

compiler: A program that translates high-level language statements into machine language instructions.

configuration: The group of machines, devices, etc., which make up a data processing system.

control area: A group of control intervals used as a unit for formatting a file before adding records to it. Also, in a key-sequenced file, the set of control intervals covered by an index record; used by VSAM for distributing free space and for placing a low-level index adjacent to its data.

control interval: (1) A fixed-length area of auxiliary storage space in which VSAM stores records and distributes free space, also, in a key-sequenced file, the set of records pointed to by an entry in the index record. It is the unit of information transmitted to or from auxiliary storage by VSAM, independent of blocksize. (2) For an FBA device, the unit of data transfer between processor storage and the device. It has the same format as a VSAM control interval. In recording data, IOCS maps each control interval over an integral number of FBA blocks.

control program: A program that is designed to schedule and supervise the performance of data processing work by a computing system.

control registers: A set of registers used for operating system control of relocation, priority interruption, program event recording, error recovery, and masking operations.

control section: That part of a program specified by the programmer to be a relocatable unit.

control unit: A device that controls the reading, writing, or display of data at one or more input/output devices.

core image library: A library of phases that have been produced as output from link-editing. The phases in the core image library are in a format that is executable either directly or after processing by the relocating loader in the supervisor.

count-key-data (CKD) device: A disk storage device storing data in the format: count field normally followed by a key field followed by the actual data of a record. The count field contains, among others, the address of the record in the format CCHHR (CC = cylinder number, HH = head or track number, R = record number) and the length of the data; the key area contains the record's key (search argument). See also *fixed block architecture (FBA) device.*

CPU busy time: The amount of time devoted by the central processing unit to the execution of instructions.

data file: A collection of related data records organized in a specific manner. For example, a payroll file (one record for each employee, showing his rate of pay, deductions, etc., or an inventory item, showing the cost, selling price, number in stock, etc.). See also file.

data integrity: See integrity.

data management: A major function of DOS/VSE that involves organizing, storing, locating, retrieving, and maintaining data.

deblocking: The action of making the first and each subsequent logical record of a block available for processing one record at a time.

default value: The choice among exclusive alternatives made by the system when no explicit choice is specified by the user.

deletion of an I/O Device: Removal of the I/O unit from the supervisor configuration tables.

diagnostic routine: A program that facilitates computer maintenance by detection and isolation of malfunctions or mistakes.

dial-up terminal: A terminal on a switched teleprocessing line.

direct access: (1) Retrieval or storage of data by a reference to its location on a volume, other than relative to the previously retrieved or stored data. (2) * Pertaining to the process of obtaining data from, or placing data into, storage where the time required for such access is independent of the location of the data most recently obtained or placed in storage. (3) * Pertaining to a storage device in which the access time is effectively independent of the location of the data. Synonymous with random access.

direct organization: Direct file organization implies that for purposes of storage and retrieval there is a direct relationship between the contents of the records and their addresses on disk storage.

directory: An index that is used by the system control and service programs to locate one or more sequential blocks of program information that are stored on direct access storage.

diskette: A flexible magnetic-oxide coated disk suitable for data storage and retrieval. Data may be stored and retrieved via such devices as the IBM 3740 Data Entry Unit and the IBM 3540 Diskette Input/Output Unit. Diskettes are also used to contain microprograms for some central processing units.

disk pack: A direct access storage volume containing magnetic disks on which data is stored. Disk packs are mounted on a disk storage drive, such as the IBM 3330 Disk Storage Drive.

distributed free space: Space reserved within the control intervals of a key-sequenced file for inserting new records into the file in key sequence; also, whole control intervals reserved in a control area for the same purpose.

dump: (1) To copy the contents of all or part of virtual storage. (2) The data resulting from the process as in (1).

dynamic address translation (DAT): (1) The change of a virtual storage address to an address in real storage during execution of an instruction. (2) A hardware function that performs the translation.

dynamic partition balancing: A DOS/VSE facility which allows the user to specify two or more or all partitions of the system to have their processing priority changed dynamically such that each of these partitions receives approximately the same amount of CPU processing time.

entry sequence: The order in which data records are physically arranged in auxiliary storage, without respect to their contents (contrast with key sequence).

entry-sequenced file: A VSAM file whose records are loaded without respect to their contents, and whose relative byte addresses cannot change. Records are retrieved and stored by addressed access, and new records are added to the end of the file.

error message: The communication that an error has been detected.

error recovery procedures: Procedures designed to help isolate, and, when possible, to recover from errors in equipment. The procedures are often used in conjunction with programs that record the statistics of machine malfunctions.

extent: A continuous space on a direct access storage device, occupied by or reserved for a particular file.

* **file:** A collection of related records treated as a unit. For example, one line of an invoice may form an item, a complete invoice may form a record, the complete set of such records may form a file, the collection of inventory control files may form a library, and the libraries used by an organization are known as its data bank.

FBA: See *fixed block architecture (FBA) device.*

FBA block: A unit of data of fixed length on which the FBA architecture is based.

fixed block architecture (FBA) device: A disk storage device storing data in blocks of fixed size; these blocks are addressed by block number relative to the beginning of the file.

fixed page: A page in processor storage that is not to be paged out.

hard copy: A printed copy of machine output in a visually readable form, for example, printed reports, listings, documents, and summaries.

hard wait state: In general, a wait state is the condition of a CPU when all operations are suspended. System recovery from a hard wait state requires that the user performs a new IPL (initial program load) procedure.

* **hardware:** Physical equipment, as opposed to the computer program or method of use, for example, mechanical, magnetic, electrical, or electronic devices. Contrast with software.

* **idle time:** That part of available time during which the hardware is not being used.

index: (1) * An ordered reference list of the contents of a file or document, together with keys or reference notations for identification or location of those contents. (2) A table used to locate the records of an indexed sequential file.

indexed-sequential organization: The records of an indexed sequential file are arranged in logical sequence by key. Indexes to these keys permit direct access to individual records. All or part of the file can be processed sequentially.

Initial Program Load (IPL): The intialization procedure that causes DOS/VSE to commence operation.

integrity: Preservation of data or programs for their intended purpose.

* **interface:** A shared boundary. An interface might be a hardware component to link two devices or it might be a portion of storage or registers accessed by two or more computer programs.

* **I/O:** An abbreviation for input/output.

ISAM interface program: A set of routines that allow a processing program coded to use ISAM to gain access to a VSAM key-sequenced file with an index.

job: (1) * A specified group of tasks prescribed as a unit of work for a computer. By extension, a job usually includes all necessary computer programs, linkages, files, and instructions to the operating system. (2) A collection of related problem programs, identified in the input stream by a JOB statement followed by one or more EXEC statements.

job accounting interface: A function that accumulates, for each job step, accounting information that can be used for charging usage of the system, planning new applications, and supervising system operation more efficiently.

job control: A program that is called into a partition to prepare each job or job step to be run. Some of its functions are to assign I/O devices to certain symbolic names, set switches for program use, log (or print) job control statements, and fetch the first program phase of each job step.

job (JOB) statement: The job control statement that identifies the beginning of a job. It contains the name of the job.

job step: The execution of a single processing program.

K: 1024.

* **key:** One or more characters associated within an item of data that are used to identify it or control its use.

key sequence: The collating sequence of data records, determined by the value of the key field in each of the data records. May be the same as, or different from, the entry sequence of the records.

key-sequenced file: A file whose records are loaded in key sequence and controlled by an index. Records are retrieved and stored by keyed access or by addressed access, and new records are inserted in the file in key sequence by means of distributed free space. Relative byte addresses of records can change.

label: identification record for a tape, diskette, or disk file.

label information area: Under DOS/VSE, the last portion of the system residence file that stores label information read from job control statements or commands.

language translator: A general term for any assembler, compiler, or other routine that accepts statements in one language and procedures equivalent statements in another language.

leased facility: A circuit of the public telephone network made available for the exclusive use of one subscriber.

librarian: The set of programs that maintains, services, and organizes the system and private libraries.

library: A collection of files or programs, each element of which has a unique name, that are related by some common characteristics. For example, all phases in the core image library have been processed by the linkage editor.

linkage editor: A processing program that prepares the output of language translators for execution. It combines separately produced object modules; resolves symbolic cross references among them, and generates overlay structure on request; and produces executable code (a phase) that is ready to be fetched or loaded into virtual storage.

load: (1) * In programming, to enter instructions or data into storage or working registers. (2) In DOS/VSE, to bring a program phase from a core image library into virtual storage for execution.

main page pool: The set of all page frames in processor storage not assigned to the supervisor or one of the partitions.

message: See error message, operator message.

microprogramming: A method of working of the CPU in which each complete instruction starts the execution of a sequence of instructions, called microinstructions, which are generally at a more elementary level.

multiprogramming system: A system that controls more than one program simultaneously by interleaving their execution.

multitasking: The concurrent execution of one main task and one or more subtasks in the same partition.

object code: Output from a compiler or assembler which is suitable for processing by the linkage editor to produce executable machine code.

* **object module:** A module that is the output of an assembler or compiler and is input to a linkage editor.

object program: A fully compiled or assembled program. Contrast with source program.

* **online:** (1) Pertaining to equipment or devices under control of the central processing unit. (2) Pertaining to a user's ability to interact with a computer.

operand: (1) * That which is operated upon. An operand is usually identified by an address part of an instruction. (2) Information entered with a command name to define the data on which a command processor operates and to control the execution of the command processor.

operator command: A statement to the control program, issued via a console device, which causes the control program to provide requested information, alter normal operations, initiate new operations, or terminate existing operations.

operator message: A message from the operating system or a problem program directing the operator to perform a specific function, such as mounting a tape reel, or informing him of specific conditions within the system, such as an error condition.

* **overflow:** (1) That portion of the result of an operation that exceeds the capacity of the intended unit of storage. (2) Pertaining to the generation of overflow as in (1).

overlay: n. (1) One of the segments, which consists of one or more phases, of a program that is so structured that not all of the segments need be in virtual storage at any one time. v. (2) The process of replacing a previously retrieved program segment in virtual storage by another segment.

page: (1) In DOS/VSE, a 2K block of instructions, data or both. (2) To transfer instructions, data, or both between processor storage and the page data set.

page data set: An extent in auxiliary storage, in which pages are stored.

page fault: A program check interruption that occurs when a page that is marked *not in processor storage* is referred to by an active page. Synonymous with page translation exception.

page fixing: Marking a page as nonpageable so that it remains in processor storage.

page frame: A 2K block of processor storage that can contain a page.

page in: The process of transferring a page from the page data set to processor storage.

page out: The process of transferring a page from processor storage to the page data set.

page pool: The set of all page frames that may contain pages of programs in virtual mode.

paging: The process of transferring pages between processor storage and the page data set.

* **parameter:** A variable that is given a constant value for a specific purpose or process.

partition: In DOS/VSE, a contiguous area of virtual storage available for the execution of programs.

partition balancing: See dynamic partition balancing.

peripheral equipment: A term used to refer to card devices, magnetic tape and disk devices, diskettes, printers, and other equipment bearing a similar relation to the CPU.

phase: The smallest complete unit that can be referred to in the core image library.

POWER: A unit record spooling support available as the IBM licensed program VSE/POWER.

printer: A device that expresses coded characters as hard copy.

priority: A rank assigned to a partition that determines its precedence in receiving CPU time.

private library: A user-owned library that is separate and distinct from the system library.

private second level directory: The private second level directory is a table located in the supervisor containing the highest phase names found on the corresponding directory tracks of the private core image library.

problem determination aid: A program that traces a specified event when it occurs during the operation of a program. Abbreviated PDAID.

problem program: Any program that is executed when the central processing unit is in the problem state; that is, any program that does not contain privileged instructions. This includes IBM-distributed programs, such as language translators and service programs, as well as programs written by a user.

processing program: (1) A general term for any program that is not a control program. (2) Synonymous with problem program.

processor storage: The general purpose storage of a computer. Processor storage can be accessed directly by the operating registers. Synonymous with real storage.

queue: (1) A waiting line or list formed by items in a system waiting for service; for example, tasks to be performed or messages to be transmitted in message switching system. (2) To arrange in, or form, a queue.

random processing: The treatment of data without respect to its location in auxiliary storage, and in an arbitrary sequence governed by the input against which it is to be processed.

real address: The address of a location in real storage.

real address area: In DOS/VSE, the area of virtual storage where virtual addresses are equal to real addresses.

real mode: In DOS/VSE, the mode of a program that cannot be paged.

real storage: The storage of a computing system from which the central processing unit can directly obtain instructions and data, and to which it can directly return results. Synonymous with processor storage.

reenterable: The attribute of a load module that allows the same copy of the load module to be used concurrently by two or more tasks.

relocatable: The attribute of a set of code whose address constants can be modified to compensate for a change in origin.

relocatable library: A library of relocatable object modules and IOCS modules required by various compilers. It allows the user to keep frequently used modules available for combination with other modules without recompilation.

restore: To return a data file created previously by a copy operation from cards, disk or magnetic tape to disk storage.

rotational position sensing (RPS): A standard or optional feature of most IBM disk storage devices. It permits these devices to disconnect from a block multiplexer channel (or its equivalent on Model 3115/3125 CPUs) during rotational positioning operations, thereby allowing the channel to service other devices.

* **routine:** An ordered set of instructions that may have some general or frequent use.

secondary storage: Same as auxiliary storage.

second level directory: A table located in the supervisor containing the highest phase names found on the corresponding directory tracks of the system core image library.

security: Prevention of access to or use of data or programs without authorization.

sequential organization: Records of a sequential file are arranged in the order in which they will be processed.

service program: A program that assists in the use of a computing system, without contributing directly to the control of the system or the production of results.

shared virtual area: An area located in the highest addresses of virtual storage. It can contain a system directory list of highly used phases, resident programs that can be shared between partitions, and an area for system GETVIS support.

software: A set of programs, concerned with the operation of the hardware in a data processing system.

source: The statements written by the programmer in any programming language with the exception of actual machine language.

* **source program:** A computer program written in a source language. Contrast with object program.

source statement library: A collection of books (such as macro definitions) cataloged in the system by the librarian program.

spanned records: Records of varying length that may be longer than the currently used blocksize, and which may therefore be written in one or more continuous blocks. A spanned record may occupy more than 1 track of a disk device.

stand-alone dump: A program that displays the contents of the registers and part of the real address area and that runs independently and is not controlled by DOS/VSE.

standard label: A fixed-format identification record for a tape, diskette, or disk file. Standard labels can be written and processed by DOS/VSE.

storage protection: An arrangement for preventing access to storage.

supervisor: A component of the control program. It consists of routines to control the functions of program loading, machine interruptions, external interruptions, operator communications and physical IOCS requests and interruptions. The supervisor alone operates in the privileged (supervisor) state. It coexists in real storage with problem programs.

switched line: A communication line in which the connection between the computer and a remote station is established by dialing. Synonymous with dial line.

system directory list: A list containing directory entries of highly used phases and of all phases resident in the shared virtual area. This list is contained in the shared virtual area.

system residence device: The direct access device on which the system residence file is located.

system residence volume: The volume on which the basic system and all related supervisor code is located.

task: A unit of work for the central processing unit from the standpoint of the control program.

teleprocessing: The processing of data that is received from or sent to remote locations by way of telecommunication lines.

terminal: (1) * A point in a system or communication network at which data can either enter or leave. (2) Any device capable of sending and receiving information over a communication channel.

throughput: The total volume of work performed by a computing system over a given period of time.

track: The portion of a moving storage medium, such as a tape, diskette, or disk, that is accessible to a given reading head position.

transient area: An area of processor storage used for temporary storage of transient routines.

UCS: Universal character set.

unit record: A card containing one complete record; a punched card.

universal character set: A printer feature that permits the use of a variety of character arrays. Abbreviated UCS.

unrecoverable error: A hardware error which cannot be recovered from by the normal retry procedures.

user label: An identification record for a tape or disk file; the format and contents are defined by the user, who must also write the necessary processing routines.

utility program: A problem program designed to perform a routine task, such as transcribing data from one storage device to another.

virtual address: An address that refers to virtual storage and must, therefore, be translated into a real storage address when it is used.

virtual address area: In DOS/VSE, the area of virtual storage whose addresses are greater than the highest address of the real address area.

virtual mode: In DOS/VSE, the mode of execution of a program which may be paged.

virtual storage: Addressable space that appears to the user as real storage, from which instructions and data are mapped into real storage locations. The size of virtual storage is limited by the addressing scheme of the computing system and by the capacity of the page data set, rather than by the actual number of real storage locations.

virtual storage access method (VSAM): An access method (available as the licensed program product VSE/VSAM) for direct or sequential processing of fixed and variable length records on direct access devices; designed for use in a virtual storage environment.

virtual telecommunications access method (VTAM): A set of IBM programs (available as the licensed program product ACF/VTAM) that control communications between terminals and application programs.

volume: (1) That portion of a single unit of storage media which is accessible to a single read/write mechanism, for example, a diskette, a disk pack, or part of a disk storage module. (2) A recording medium that is mounted and dismounted as a unit, for example, a reel of magnetic tape, a disk pack, or a diskette.

volume table of contents: A table on a direct access volume or diskette that describes each file on the volume. Abbreviated VTOC.

VSAM access method services: A multifunction utility program that defines VSAM files and allocates space for them, converts indexed sequential files to key-sequenced files with indexes, facilitates data portability between operating systems, creates backup copies of files and indexes, helps to make inaccessible files accessible, and lists file and catalog entries.

VSAM catalog: A key-sequenced file, with an index, containing extensive file and volume information that VSAM requires to locate files, to allocate and deallocate storage space, to verify the authorization of a program or operator to gain access to a file, and to accumulate usage statistics for files.

VTOC: See *volume table of contents.*

work file: A file on an auxiliary storage medium reserved for intermediate results during execution of the program.

working set: The set of pages of a user's virtual-mode program that must be in processor storage in order to avoid excessive paging.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Z

DOS/VSE System Management Guide

READER'S COMMENT FORM

GC33-5371-7

This sheet is for comments and suggestions about this manual. We would appreciate *your* views, favorable or unfavorable, in order to aid us in improving *this* publication. This form will be sent directly to the author's department. Please include your name and address if you wish a reply. Contact your IBM branch office for answers to technical questions about the system or when requesting additional publications. Thank you.

Name

Address

What is your occupation?

How did you use this manual?

As a reference source

As a classroom text

As a self-study text

Your comments* and suggestions:

*** We would especially appreciate your comments on any of the following topics:**

Clarity of the text	Accuracy	Index	Illustrations	Appearance	Paper
Organization of the text	Cross-references	Tables	Examples	Printing	Binding

YOUR COMMENTS, PLEASE . . .

This manual is part of a library that serves as a reference source for system analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. 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.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold

Fold

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:

IBM Corporation
1133 Westchester Avenue
White Plains, N.Y. 10604

Attention: Department 813 BP

Fold

Fold

CUT ALONG THIS LINE

IBM

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

GC33-5371-7

DOS/VSE System Management Guide Printed in U.S.A. GC33-5371-7

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