UNIVAC OS/4 JOB CONTROL

PROGRAMMER REFERENCE

SPERRY UNIVAC
COMPUTER SYSTEMS

UP-7793
Rev. 1

H

PAGE STATUS SUMMARY

ISSUE: UP-7793 Rev. 1 Original

Section	Page Number	Update Level
Cover/Disclaimer		
PSS	1	
Contents	1 thru 4	
1	1, 2	
2	1 thru 14	
3	1 thru 26	
4	1 thru 35	
5	1 thru 12	
Appendix A	1 thru 3	
Appendix B	1 thru 4	
Appendix C	1 thru 10	
Appendix D	1	
Index	1 thru 13	
User Comment Sheets		

Section	Page Number	Update Level

Section	Page Number	Update Level

CONTENTS

1. INTRODUCTION

1.1. GENERAL

In the UNIVAC OS/4 Operating System (OS/4), job control is the nonresident component that manages the system resources and prepares jobs for execution in either a tape or disc operating environment. A job represents a task or unit of work to be performed; each job can be divided into job steps to be executed serially; job steps are made up of problem programs. Job control services are performed prior to the execution of the initial step of a job, during the transition between job steps, and at the conclusion of the job. OS/4 job control refers only to disc operating environments. The tape operating environment described in this manual applies only to the UNIVAC 9400 System Tape Operating System (TOS). For additional information concerning OS/4, see *UNIVAC OS/4 Operating System Guide Programmer Reference, UP-7934* (current version).

The services of job control are directed by the user through statements known as the job control language. These control statements define the system resources required for proper execution of a job and facilitate the efficient management of those resources before, during, and after job step execution. The flexibility provided by these statements enables the user to define the requirements for a variety of facilities and to be independent of many limitations previously imposed by system configurations.

Some of the services performed by job control are:

- Storage allocation
- Device assignment
- Volume label and file label storage
- Constructing a job file of control streams on the resident direct access storage device for subsequent retrieval and execution
- Restarting a program from a checkpoint

This manual describes the manner in which these services are performed by job control in either a disc or tape operating environment. It also describes how the resources of the system are allocated to a job and defines the control statements and their use.

A knowledge of *UNIVAC 9400 System Assembler/Central Processor Unit Programmer Reference, UP-7600* (current version); *UNIVAC 9400 System Supervisor Programmer Reference, UP-7689* (current version); *UNIVAC 9400 System Data Management System Programmer Reference, UP-7629* (current version); and *UNIVAC 9400 System Description, UP-7566* (current version) is helpful in the use of this manual.

1.2. CONTROL STREAM CONCEPTS

A control stream is a group of sequenced control statements, written by the user in job control language, which define a job and direct the execution of that job. The control stream acts as an interface between the user job and job control. The control statements contain information regarding:

- Job Identification
- Device Assignment
- File Label Information
- Storage Allocation
- Extent Requests
- Tape Control
- Job Termination and Deletion

A control stream may also contain data required during the execution of the job or it may include the source code in an assemble-link-and-go type of operation.

Each job executed in the system must have a unique name which is used by the system to identify and to locate the job's control stream. Devices are assigned for each input or output file required by the job by means of control statements grouped in sets. These sets contain the required information associated with that device. Device assignment sets may also include volume, extent request, and file label information. In addition, the control stream may contain control statements requesting that certain features, facilities, and options be made available during job execution, that a job step accept parameters at execution time, or that further control of the magnetic tape units is required.

In a tape-oriented system, the control stream is read in from the system input device (RDR) and processed one statement at a time. In a disc-oriented system, control streams are read in from the system input device and stored in a job file on disc for subsequent retrieval and execution.

In either a tape-oriented or disc-oriented system, job control reads a control statement in the control stream by issuing a GETCS macro instruction in the program. The GETCS macro instruction is a function of the supervisor. Thus, the user is prevented from reading a job control statement directly from the control stream. The user is also prevented from executing two job control functions at the same time. If a second job control function is requested by the user, the first job control function is completed before the second one can be executed. Job control functions include: FILE, DELETE, TERMINATION, etc.

1.3. MULTIJOB CAPABILITY

Multijobbing is the concurrent execution of several jobs residing simultaneously within the system. These jobs must be executable in a disc-oriented system. OS/4 job control has the capability of executing up to five jobs concurrently. Therefore, there can be any number of control streams filed in the job file but only five may be active at a given time. The control stream link addresses for each job within the job file are totally independent. Although several jobs may be executing concurrently, only one job control function can be operating in the system at a given time. The job control function for the job with the highest priority is performed; a request for another job control function must wait for the completion of the current function.

2. JOB MANAGEMENT

2.1. GENERAL

Job control is activated when a job is to enter the system or when the various functions required for job execution must be coordinated. The user specifies these functions by means of statements written in job control language (Section 4). These statements are submitted to the system through an input control stream. A control stream consists of a series of punched cards that are read and either immediately executed or are stored on auxiliary storage for subsequent retrieval and execution. An explanation of the functions of job control is given in the following subsections.

2.2. JOB ENTRY

The job entry function of job control permits the user to submit control streams either for immediate retrieval and execution or for temporary storage on tape or disc for subsequent retrieval and execution. When control streams are written on auxiliary storage, it is referred to as control stream buffering. Tape buffering may be used to put job control streams out onto tape, which can be later filed from tape to disc. Tape buffering works only with prepped tapes (tapes containing a VOL1 record and a HDR1 record), as it checks the volume serial number and the Julian date in the HDR1 record for expiration. It updates the Julian date to the current date in the system when the tape is initially created.

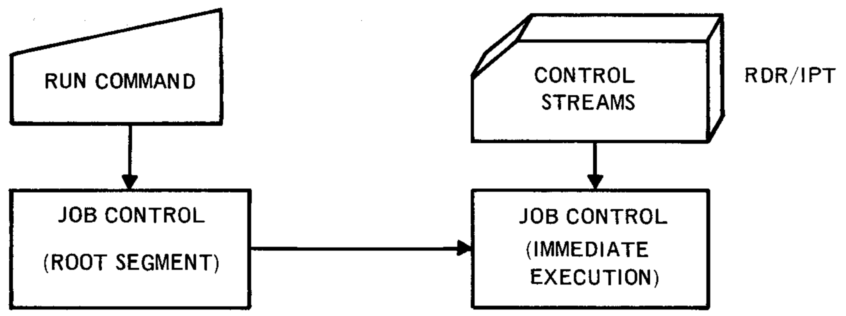
The operation of the job entry function in tape- and disc-oriented systems is explained in the following paragraphs.

- Tape Systems (TOS)

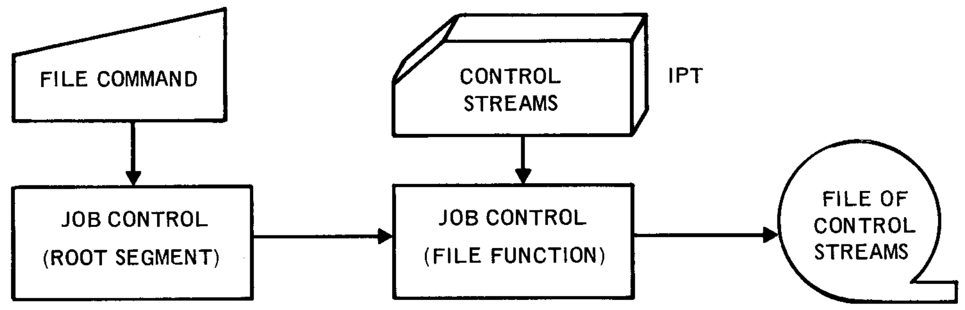
  In a tape-oriented system, control streams are introduced through a punched card reading device for immediate execution or are written on magnetic tape for subsequent execution. When control streams are submitted to the system through a punched card reading device for immediate execution, that device must be defined as both the system reader (RDR) and system input device (IPT). When control streams are written on tape for subsequent execution, the punched card reading device must be specified as IPT, and the magnetic tape device on which the control stream is to be written must be specified as RDR and referenced on the file command by its volume serial number.

  A FILE operator command is used to activate the control stream buffering function and to identify the tape volume on which the control stream is to be written. A RUN operator command or control statement is used to retrieve a control stream for execution. (When control streams are submitted through a punched card reading device for immediate execution, only the RUN operator command or control statement need be used.) As each control statement is retrieved, job control provides verification checks to be reasonably certain that the control stream has been constructed in a logical manner. If an erroneous statement is detected, job control displays the type of error at the system console and aborts the job by reading and bypassing all remaining statements in the control stream, up to and including the end-of-job (/&) statement. Following normal or abnormal terminations, the system reader is positioned to the next control stream. Figure 2–1 illustrates the job entry functions for a tape operating system.

WITHOUT CONTROL STREAM BUFFERING:



WITH CONTROL STREAM BUFFERING, USING FILE OPERATOR COMMAND:



WITH CONTROL STREAM BUFFERING, USING FILE CONTROL STATEMENT:

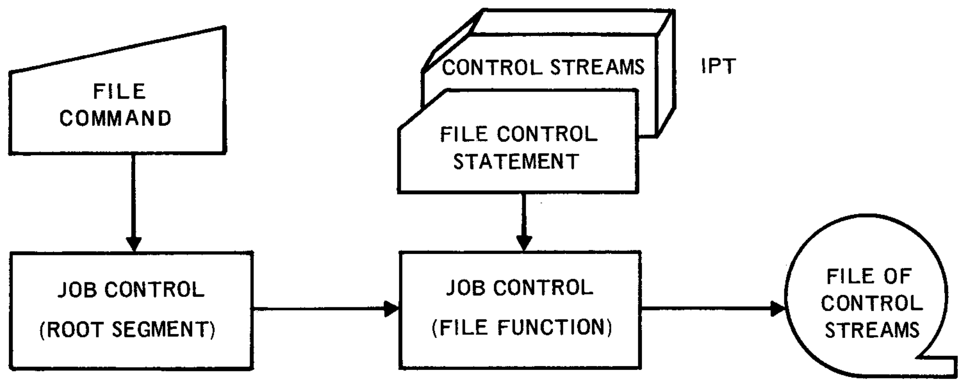


Figure 2–1. Job Entry in a Tape Operating System

When control streams are written on tape, each card image retrieved from the system input device occupies one physical block on the tape volume. Control streams are separated by tape marks, and messages are displayed at the system console indicating the job names and their respective file numbers so that the tape volume can be subsequently repositioned to allow the execution of a job in other than a sequential manner. Thus, the first control stream written on a particular magnetic tape volume can be executed by positioning the tape to the first file. This is normally accomplished by issuing an MTC operator command designating that the tape volume is to be positioned immediately following the first tape mark.

- Disc Systems (DOS)

  In a disc-oriented system, control streams are introduced through a punched card reading device and written in the job file on the system resident disc pack for subsequent retrieval and execution. The card reading device must be defined as IPT, and the system resident disc device must be defined as RDR. A function of job control, called the JPROC procedure processor, which can be specified when the superisor is generated, has been provided to the user for generating repetitive sequences of statements in a job stream upon the specification of a single procedure call statement. This function is stored in the module JPROC in the system procedure library. The use of JPROC procedures saves time and effort and reduces the risk of error in job stream preparation. The use of JPROC increases the FILE memory requirement by 8192 bytes.

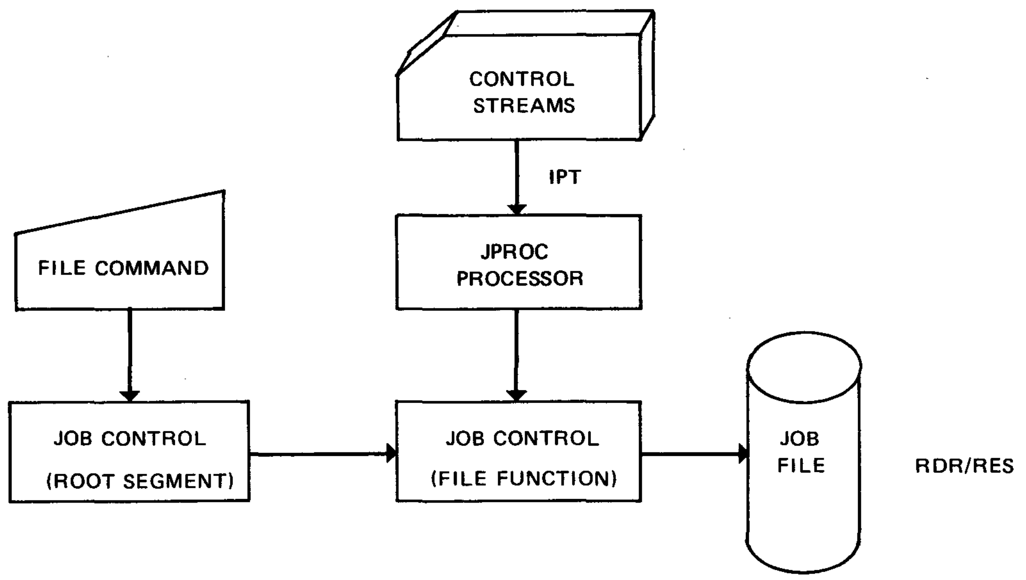
  The JPROC procedure processor expands the procedure call statement, according to the procedure definition, into a series of lines, which then becomes a portion of the control stream. The JPROC function also provides the ability to insert and/or change parameters in the generated control statements (such as volume serial numbers, file names, program names, etc.). Section 5 describes the use of job control procedures in detail.

  Disc storage space to contain the control streams is dynamically acquired by job control from the system temporary storage pool one cylinder at a time. Space for the job file index is permanently reserved and restricted to one cylinder. Each entry in the job file index consists of an 8-byte key and a 24-byte data record. The control stream is stored in fixed-length blocks of 446 bytes. A maximum of four control statements or five data statements is contained in each block. A minimum of one track is required for each control stream.

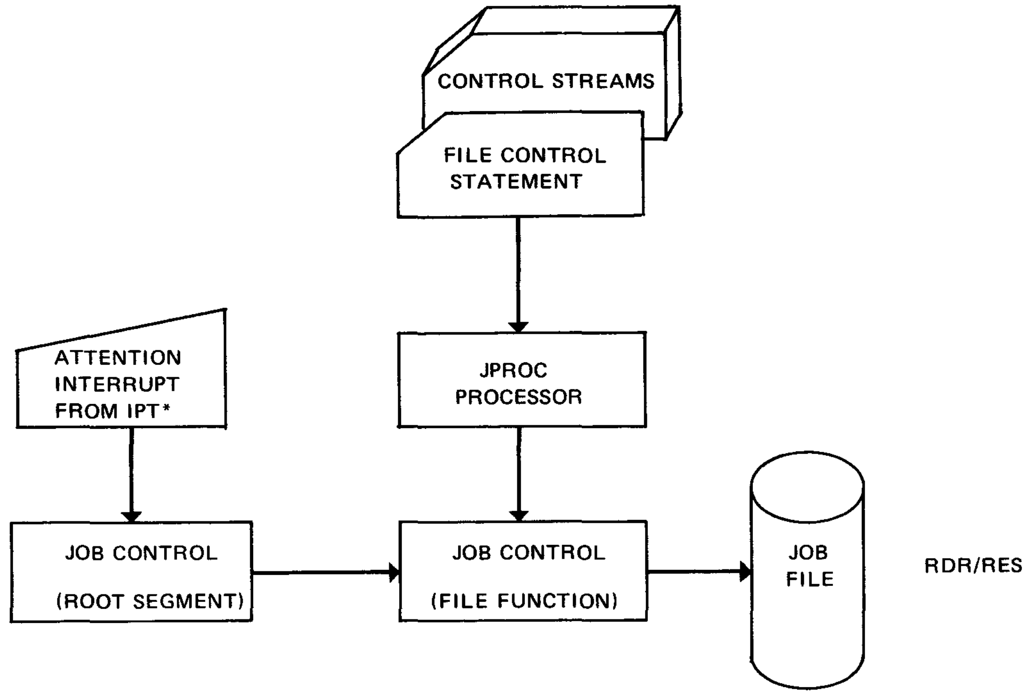
  A FILE operator command or control statement is used to activate the file function or control stream buffering function. Before each control stream is filed, job control scans the job file index to determine if there is another control stream with an identical name. If a matching name is found and the associated job is currently running in the system, all processing of the newly submitted control stream ceases. However, if a matching name is found and the associated job is found and the associated job is not currently running in the system, the previously filed stream is deleted and the newly submitted control stream is filed. If no matching name is found, the newly submitted control stream is filed on the disc.

  Immediately after each control stream is filed, it is re-read and verified to determine if the stream is constructed in a logical manner. If no errors are detected, an entry is written in the job file index identifying the filed control stream, and a message is displayed at the system console indicating that the control stream has been both filed and verified. If an error is detected in the control stream, the index entry is not written on the job file; messages are displayed at the system console to indicate the type of error and that the control stream has not been filed. (Since there is no index entry for the erroneous control stream, that stream cannot be retrieved.) Figure 2–2 illustrates the job entry functions for a disc operating system.

WITH CONTROL STREAM BUFFERING, USING FILE OPERATOR COMMAND:



WITH CONTROL STREAM BUFFERING, USING FILE CONTROL STATEMENT:


CONTROL STREAMS
FILE CONTROL STATEMENT
ATTENTION INTERRUPT FROM IPT*
JPROC PROCESSOR
JOB CONTROL (ROOT SEGMENT)
JOB CONTROL (FILE FUNCTION)
JOB FILE
RDR/RES

*Caused by operator intervention at the peripheral device.

Figure 2–2. Job Entry in a Disc Operating System

2.2.1. Assignment of Sequence Numbers to Statements

Sequence numbers are assigned to job control statements during the file function. The purpose is to facilitate locating incorrect statements which are indicated by an error message printout on the console (JF04). The following examples explain how sequence numbers are assigned.

Example 1:

When control streams are submitted unpacked (only one statement per card) and without sequence numbers, the file function assigns sequence numbers in increments of 100.

The submitted control stream

```
   LABEL      ᵬ OPERATION ᵬ              OPERAND                  ᵬ
1               10          16
// JOB jobname
// DVC n
// LFD PRNTR
```

is filed as follows:

```
// JOB jobname                                         000100
// DVC n                                               000200
// LFD PRNTR                                           000300
```

Example 2:

When control streams are submitted and more than one job control statement appears on the card without sequence numbers, the file function will assign sequence numbers in increments of 100 for a card and in increments of 10 for each statement on the card.

The submitted control stream

```
// JOB jobname  // DVC n  // LFD PRNTR
// DVC n    // VOL volno  // LFD TAPE
```

is filed as follows:

```
 LABEL      ƀ OPERATION ƀ          OPERAND                                     
1              10       16                                        72      80
// JOB jobname                                                        000110
// DVC n                                                              000120
// LFD PRNTR                                                          000130
// DVC n                                                              000210
// VOL volno                                                          000220
// LFD TAPE                                                           000230
```

Each card in the submitted control stream has three statements; when they are filed, these statements are broken down into separate card images before being filed to disc. The sequence numbers on the card images that are filed could be defined in the following manner: the hundreths position designates the physical card and the tenths position designates the statement on that card. For example, take the first card. When it was filed, it was assigned 000110. This tells us that this image would appear as the first statement on the first card submitted, so 000120 would be the second statement on card one, etc.

Example 3:

When control streams are submitted that are only one statement per card and they contain sequence numbers, these statements are filed with the sequence number that was submitted. These should be sequenced in increments of 1000, so that if there is a need to insert a card, it can be inserted with the sequence of the previous card incremented by 100.

```
// JOB jobname                                                        001000
// DVC n                                                              002000
// LFD PRNTR                                                          003000
```

If there is a need to insert a card between the JOB statement and the DVC statement, the number given this card should be 001100. This number would define this card as being the first insert card after statement number one.

Example 4:

When control streams are submitted that contain more than one statement per card and are sequenced, again they should be sequenced by 1000.

The submitted control stream

```
// JOB jobname   // DVC n   // LFD PRNTR                              001000
// DVC n  //LFD TAPE                                                  002000
```

is filed as follows:

```
LABEL      ␢ OPERATION ␢           OPERAND                          
1            10      16                                    72        80
// JOB jobname                                               001010
// DVC n                                                     001020
// LFD PRNTR                                                 001030
// DVC n                                                     002010
// LFD TAPE                                                  002020
```

The scheme for assigning sequence numbers for these cards would be the same as multistatement cards that are unsequenced, with the exception that the sequence number that appears on the submitted cards will be retained and the individual statements will be incremented by 10.

Example 5:

The same applies for inserting cards between cards that are already sequenced as explained in example 2. The following can also be done for inserting cards between sequenced cards.

The submitted control stream

```
// JOB jobname                                               001000
// DVC n                                                     002000
// LFD PRNTR                                                 003000
```

If the need arises to insert a card after the JOB statement, it can be inserted without a sequence number and it is filed as follows:

```
// JOB jobname                                               001000
// inserted card                                             001100
// DVC n                                                     002000
// LFD PRNTR                                                 003000
```

An example for sequenced statements that have more than one statement per card is:

```
// JOB jobname  // DVC n  // LFD PRNTR                       001000
// DVC n  // VOL volno  // LFD TAPE                          002000
```

If the need arises to insert a card that has more than one statement between the preceding two cards, they are filed as follows:

LABEL 1	b OPERATION b 10	OPERAND 16	72 80
// JOB j	obname		001000
// DVC n			001020
// LFD P	RNTR		001030
// 1st s	tatement	from inserted card	001110
// 2nd s	tatement	from inserted card	001120
// 3rd s	tatement	from inserted card	001130
// DVC n			002010
// VOL v	ol no		002020
// LFD T	APE		002030

Example 6:

When control streams are submitted with data in the stream, the sequencing is as given in examples 1 and 2 except that the /* is not sequenced.

The submitted stream

```
// JOB jobname  // DVC n  // LFD PRNTR
// DVC n  // VOL volno  // LFD TAPE
// EXEC CARDTAPE
// ALTER
// PARAM
/$
    :
    data
    :
/*
/&
```

is filed as follows:

LABEL 1	♭ OPERATION ♭ 10	OPERAND 16	72 80
// JOB	jobname		000110
// DVC	n		000120
// LFD	PRNTR		000130
// DVC	n		000210
// VOL	volno		000220
// LFD	TAPE		000230
// EXEC	CARDTAPE		000300
// ALTER			000400
// PARAM			000500
/$			000600

Data statements and/* are filed as is, without sequence numbers. In other words, the /* is filed as data.

/&			000700

Example 7:

If JF04 console ressages are printed out at verification time, the following example illustrates assignment of sequence numbers that will help in finding the invalid statement:

```
JF04        S  000920        S  000930
```

This example indicates that a sequence error had occurred on card 9, statements 2 and 3.

2.3. JOB INITIATION AND CONTINUATION

The job initiation and continuation functions of job control establish the operating environment for a job and provide the processing control required between job steps.

2.3.1. Job Initiation

Jobs are initiated by a RUN operator command or a RUN control statement. Job control retrieves and examines the control stream to determine the required hardware and software resources. If sufficient system resources are available, job control allocates the required resources, and establishes and identifies the job in the system. If sufficient system resources are not available, a message is displayed at the system console indicating the reason, and the job initiation process is immediately aborted.

During a job initiation, the priority level of the job is established. All jobs are executed at user priority level 3 unless the priority is specified on the JOB statement, RUN statement, or RUN operator command. If the priority is specified in the RUN statement or operator command, this priority takes precedence over the priority specified in the JOB statement. Job control enters the priority at the proper level in the program switch list. See *UNIVAC 9400 System Supervisor Programmer Reference, UP-7689* (current version) for a detailed explanation of the program switch list.

The operation of the job initiation function in tape- and disc-oriented systems is explained in the following paragraphs.

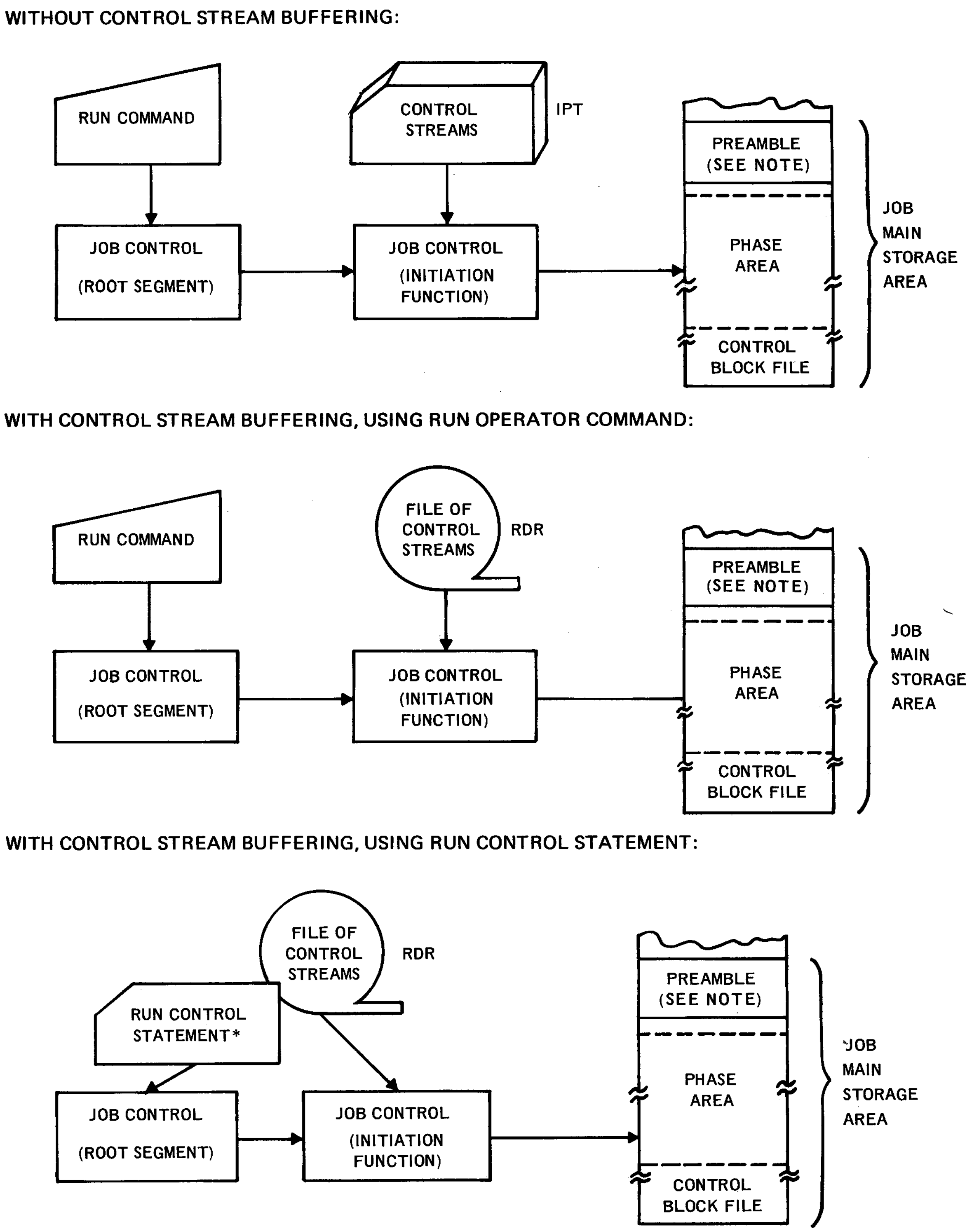
- Tape Systems (TOS)

  In a tape-oriented system, multiprogramming is permitted only if the program mix consists of one main program and one or more symbionts. Since control streams are not prescanned, it is impossible for job control to determine the main storage requirements for multistep jobs. If a problem job consists of more than a single job step, the main storage requirement for the job is determined to be the requirement for the first job step. For example, if the main storage requirement for job-step 2 exceeds that of job-step 1, it may be impossible to obtain the results of job-step 1, and this fact is not known until execution time for job-step 2. When this occurs, all remaining job steps are aborted.

  Additional main storage is required by job control in which the file control blocks, volume serial number list blocks, and extent request blocks are stored and retrievable until no longer required by the user program. Job control automatically allocates main storage space for this purpose beginning with the highest address of the job main storage area. Since the number of these blocks can be determined by the user from the structure of the control stream, it is important to consider this main storage requirement when approximating the total requirement for the job. Figure 2–3 illustrates the job initiation function for a tape operating system.

- Disc Systems (DOS)

  In a disc-oriented system, from one to five problem jobs can be executed concurrently depending on the options selected by the user at system generation time. In this operating environment, symbiont programs are considered as problem jobs, where a single symbiont job may control one or more symbiont functions. Since control streams are always filed on the system resident disc pack prior to their execution, job control scans the entire control stream at initiation time to determine the size of the main storage area suitable for running all job steps in the control stream. Therefore, prior to the execution of job-step 1, job control determines that there is sufficient main storage for the execution of all job steps. File control blocks, volume serial number list blocks, and extent request blocks are created and stored on the system resident disc pack. For this reason, it is not necessary to reserve main storage space for these items. Figure 2–4 illustrates the job initiation function in a disc operating system.


WITHOUT CONTROL STREAM BUFFERING:
RUN COMMAND
CONTROL STREAMS
IPT
JOB CONTROL (ROOT SEGMENT)
JOB CONTROL (INITIATION FUNCTION)
PREAMBLE (SEE NOTE)
PHASE AREA
CONTROL BLOCK FILE
JOB MAIN STORAGE AREA
WITH CONTROL STREAM BUFFERING, USING RUN OPERATOR COMMAND:
RUN COMMAND
FILE OF CONTROL STREAMS
RDR
JOB CONTROL (ROOT SEGMENT)
JOB CONTROL (INITIATION FUNCTION)
PREAMBLE (SEE NOTE)
PHASE AREA
CONTROL BLOCK FILE
JOB MAIN STORAGE AREA
WITH CONTROL STREAM BUFFERING, USING RUN CONTROL STATEMENT:
FILE OF CONTROL STREAMS
RDR
RUN CONTROL STATEMENT*
JOB CONTROL (ROOT SEGMENT)
JOB CONTROL (INITIATION FUNCTION)
PREAMBLE (SEE NOTE)
PHASE AREA
CONTROL BLOCK FILE
JOB MAIN STORAGE AREA

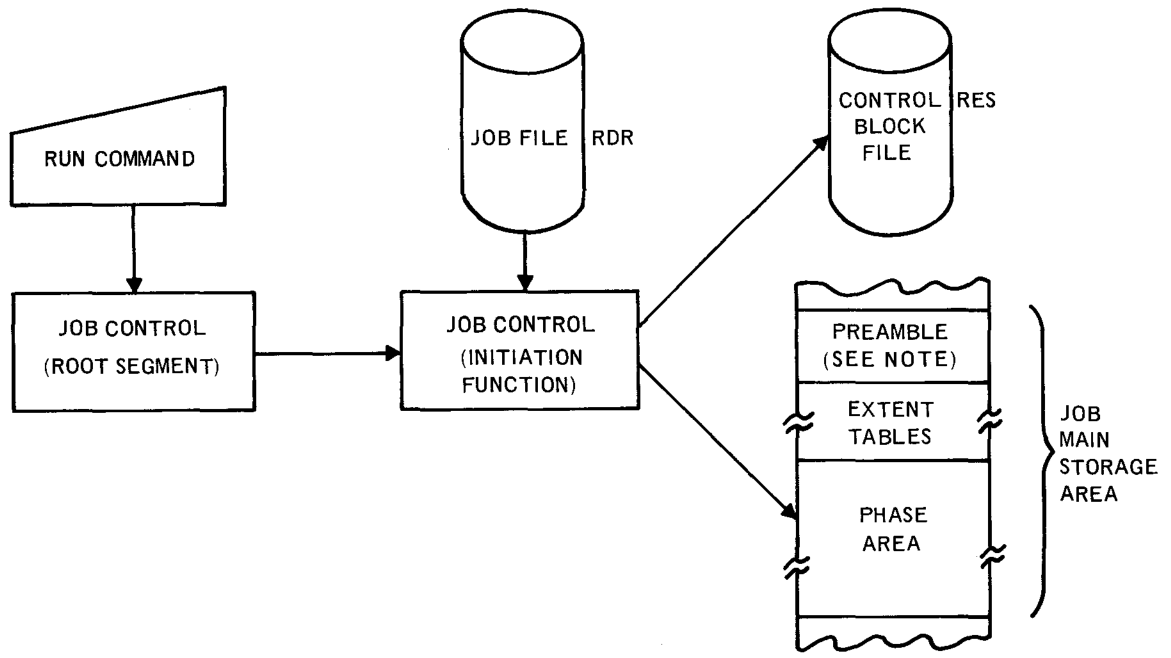
*Encountered in the control stream of the currently active job causing termination of that job and initiation of the specified job.

NOTE:

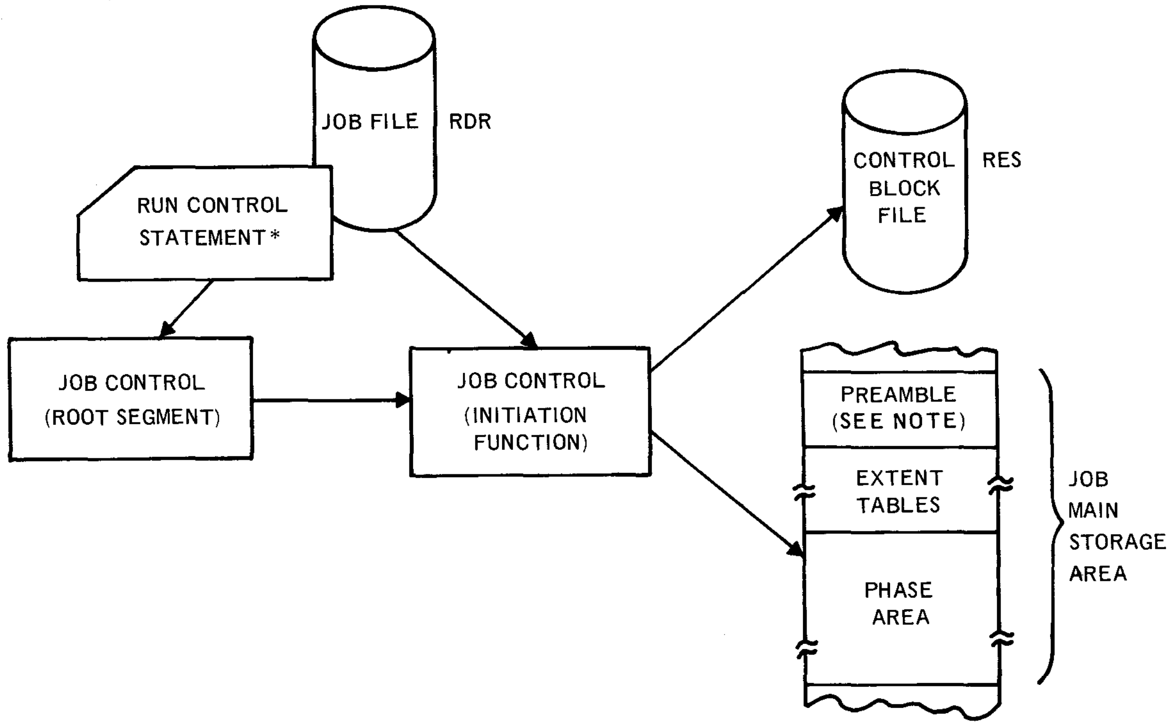
In systems up to 131K memory, preamble size = 512 bytes.
In systems over 131K memory, preamble size = 1024 bytes.

Figure 2–3. Job Initiation in a Tape Operating System

WITH CONTROL STREAM BUFFERING, USING RUN OPERATOR COMMAND:



WITH CONTROL STREAM BUFFERING, USING RUN CONTROL STATEMENT:



*Encountered in the control stream of the currently active job causing termination of that job and initiation of the specified job.

NOTE:

In systems up to 131K memory, preamble size = 512 bytes.

In systems over 131K memory, preamble size = 1024 bytes.

Figure 2–4. Job Initiation in a Disc Operating System

2.3.2. Job Continuation

The job continuation function of job control is automatically called at the termination of each job step to provide any required inter-job-step processing.

In both tape- and disc-oriented systems, normal job step termination results from the execution of either an EOJ or a DUMP macro instruction. When these macro instructions are issued by the problem program, system transient routines are called which, in turn, call and load job control into the minimum main storage partition of that job's main storage area. Peripheral devices which have been assigned for the duration of the job remain allocated to the job; those devices which have been assigned only for the duration of the job step are automatically released to the system for possible reassignment to remaining job steps or other jobs. Job control then continues to process the control stream as specified by the user until the next job step is ready for execution. If all required peripheral devices are available, job control executes a FETCH macro instruction which results in the loading and execution of the next job step.

During the time of inter-job-step processing, job control preserves the contents of all user-related storage fields within the problem job preamble. For example, the contents of the date field and the user communication region are preserved from one job step to the next. If a problem program terminates a job step without completely processing its data file (stored in the job file), the remaining data statements are bypassed by job control ~~in order to retrieve the next valid control statement for processing~~. Because job control operates within the job main storage area and is loaded by systems transient routines, no permanent resident main storage is required for its functions.

2.4. JOB CONTROL PROCEDURES

A function of job control, called the JPROC procedure processor, has been provided to the user for generating repetitive sequences of statements in a job stream upon the specification of a single procedure call statement. This function is stored in the module JPROC in the system procedure library. The use of JPROC procedures saves time and effort and reduces the risk of error in job stream preparation.

The JPROC procedure processor expands the procedure call statement, according to the procedure definition, into a series of job control statements, which then become a portion of the control stream. The JPROC function also provides the ability to insert and/or change parameters in the generated control statements (such as volume serial numbers, file names, program names, etc.). Section 5 describes the use of job control procedures in detail.

2.5. JOB TERMINATION

The job termination function of job control is called as a result of either normal or abnormal termination. These are explained in the following paragraphs. All terminations result in the deallocation of all system resources (such as peripheral devices, main storage, and disc scratch area) previously allocated to the job. Any remaining data images or control statements in the control stream are bypassed.

- Normal Terminations

  Normal job terminations are those which are initiated by the program, control stream, or system operator. These terminations are generally caused by the execution of either an end-of-job (/&) statement or a RUN control statement. The end-of-job statement causes the termination of a job, whereas the RUN control statement causes the termination of the job and the immediate initiation of another job. Jobs can also be terminated by a CANCEL or STOP operator command, CANCEL macro instruction, or CANCEL control statement. Jobs terminated by a CANCEL macro instruction, control statement, or operator command causes immediate cessation of the problem program and the termination of the job. The STOP operator command is used to terminate a job between job steps. (In order to be effective, the operator must precede the STOP command with a PAUSE command. The purpose of the PAUSE command is to indicate to the operating system that the job step currently in execution should be allowed to terminate normally and that execution of the next job step is to be postponed until the operator responds with either a READY or STOP operator command.) ?

When the STOP operator command is issued, job control terminates the job before executing any control statements associated with the next job step in the control stream. This permits an orderly restart of the job without requiring a program checkpoint procedure. Jobs terminated by the STOP operator command are normally referred to as suspended jobs.

- Abnormal Terminations

  Abnormal terminations are those caused by program errors, control stream errors, or the expiration of the estimated processing time for the job. When an abnormal termination occurs, a main storage printout is provided, which includes page and field headings, to assist the programmer in determining the causes of that termination. No main storage printout is provided when control stream errors occur.

3. RESOURCE MANAGEMENT

3.1. GENERAL

The resource management functions of job control provide the allocation of system facilities to the one or more requesting jobs and deallocation of these facilities to the system for subsequent re-use. Resources managed by these functions of job control include main storage, auxiliary storage, and the peripheral devices.

This section is provided to assist the user in determining the total amount of storage space required by the job, what devices must be online or available, the priority of the jobs to be run, what optional software features are needed, and in extending the definition of the files and their labels. This section also contains a detailed description of the control block file and its contents.

3.2. STORAGE ALLOCATION

The allocation of main storage to a job is a function of job control. The amount of main storage space which remains after the supervisor has been loaded is available for user jobs* or job control functions. As jobs are loaded and executed, the size of this available space varies and becomes fragmented. (The starting address of the first available space is stored in the system information block (SIB); a link address is maintained in each of the remaining unallocated spaces.) Job control assigns a portion of the main storage area to the job as it is initiated. As job execution concludes, the area occupied by the job is released back to the system and becomes available for a subsequent job.

NOTE:

For UNIVAC OS/4 Operating System (OS/4) used with the UNIVAC 9700 System, all storage allocation is on 2048-byte boundaries.

The portion of main storage assigned to a job consists of the minimum main storage partition plus any additional storage required as calculated by Job Limits Processing (3.2.6). The minimum main storage partition is that amount of storage necessary for job control functions plus the job preamble. This size is 8192 bytes in systems with up to 131K main storage capacity and 8704 bytes in systems with greater than 131K main storage capacity.

When a job is initiated, the first portion of the job control transient routine is loaded and executed. This routine analyzes the amount of available space, selects the largest space adequate to contain the job, and allocates this space to the job. If the available space is not adequate, the job is aborted and the operator is notified of the fact. Any space not required by the job is considered to be available for another job. The remainder of job control is loaded into the allocated space after a test has been made to ensure that 8K or 9K bytes of contiguous space (depending upon system main storage size) are available. Job control continues to process the control stream until it transfers control to the job. The job is read in, overlaying job control, and then is executed. For a multistep job, the steps are sequentially loaded and executed.

****User jobs also include such programs as assemblers, compilers, and language processors.***

Job control performs separate and different analyses of the storage allocation requirements of tape-oriented systems and disc-oriented systems. The fundamental differences are explained in the following paragraphs.

■ Tape Systems (TOS)

In a tape-oriented system where only one main job is executed at a time, all of main storage is available except that which is required for the supervisor and the control block file (3.4). However, if one or more symbionts are included in the program mix (multiprogramming), the amount of available storage decreases by the amount of space required by the symbionts. Similarly, in a tape-oriented system that also supports disc data files, the amount of storage decreases by the amount of space required by the extent tables and control block file.

The control stream is read from the system input device and may or may not be filed. As a result, the control stream cannot be prescanned to determine storage requirements nor can several control streams be accessible during the same time period as can be done in a disc-oriented system. Thus, a tape-oriented system is unsuitable for multijobbing as only one control stream can be active at a time. When symbiont processing is required, the symbionts are loaded prior to the main job and they may not attempt to access their control stream during main job processing. If symbiont processing is completed before main job processing is completed, main storage space occupied by the symbiont is not returned to the system until main job processing is completed.

NOTE:

For OS/4 used with the UNIVAC 9700 System, refer only to the disc operating system information. The tape operating system does not apply to OS/4.

■ Disc Systems (DOS)

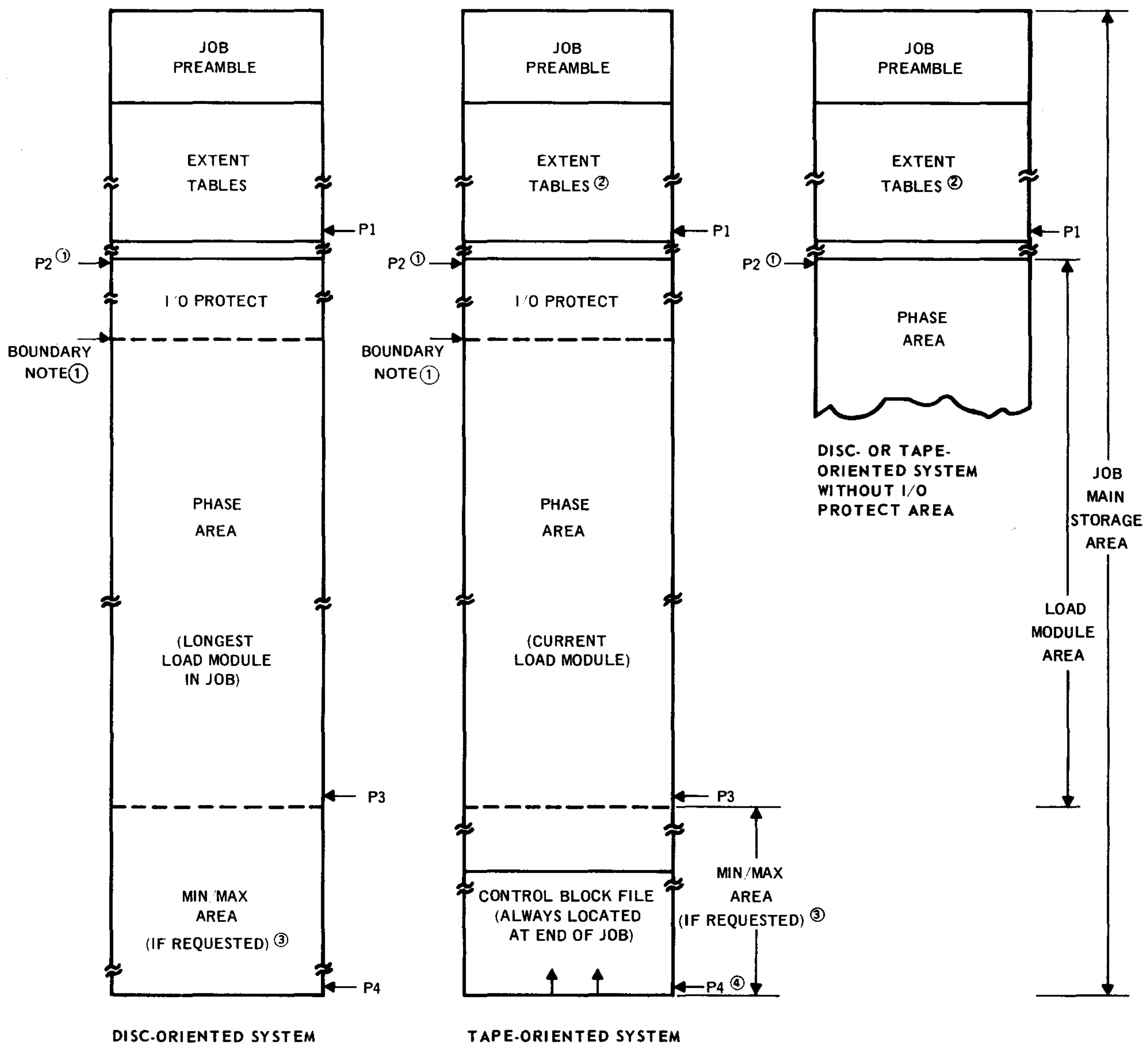
In a disc-oriented system, the minimum storage requirement of 8K bytes must include the job preamble and the extent tables, in addition to the load modules produced by the linkage editor or the assembler. In a multijob environment, each job is assigned individual storage based on the size of the largest job step and its associated disc extent requirements.

Job control scans the control stream for the specified job. During this scan, each available program within the job is located, and the length of the longest first load module is determined. If extent requirements are specified, the area required for the extent tables is calculated. The sum of these requirements, plus any additional space requests, are compared with the amount of available space. If there is insufficient space in which to execute the job, the user must either wait until the job can be executed or terminate other jobs to make sufficient storage space available.

When a job is terminated, the next job to be executed may or may not receive this space; this depends on the manner in which the other jobs in the multijob environment have been allocated storage and the size of each job.

The minimum amount of storage that can be allocated to a job is 8K bytes. Although a job submitted by a user may be less than 8K bytes in length, job control allocates 8K bytes of storage space to that job. This is due to the fact that job control operates in the same area as the user job after a control statement has requested a job control function.

During the process of allocating main storage to a job, job control assigns the job boundaries and sets the necessary addresses in the job preamble. These addresses are as follows: P1, the end of the extent table area; P2, the beginning of either the I/O protect area (if present) or the phase area; P3, the end of the phase area; and P4, the last main storage location the job can reference. Diagrams of the division of main storage allocated to a job in a tape-oriented system and in a disc-oriented system are shown in Figure 3–1. A description of each of these divisions is given in the following subsections.



NOTES:

① In systems up to 131K memory, adjusted from a 512-byte boundry. In systems over 131K memory, adjusted from a 1024-byte boundry.

② Required in a tape-oriented system using disc data files. If not required, then P1 contains the ending address of the preamble.

③ If not requested, then P3=P4.

④ In a tape-oriented system, P4 is the highest main storage address.

Figure 3–1. Storage Allocation for a Job

3.2.1. Job Preamble

A job preamble is generated by job control for each job to be executed in the system. A job preamble requires 512 bytes in the low order area of the job main storage area and contains environmental and control information about the job. The 512-byte area of the preamble must be included when estimating the size of the total storage requirement for the job. The contents of a preamble are discussed in detail in *UNIVAC 9400 System Supervisor Programmer Reference, UP-7689* (current version).

At job initiation, a job control transient routine constructs the preamble for the job and then loads the root segment of job control into the available space following the preamble. Job control analyzes the control stream and stores additional information pertinent to the job in the preamble. Thus, when the first step of the job is loaded, the information in the preamble reflects the conditions required by this job step.

When the execution of a job step is completed, job control is recalled to process subsequent control statements in the control stream. Job control determines whether the next job step to be executed requires any changes to the preamble. If so, such changes are made before the execution of the next job step.

NOTE:

The job preamble requirement given throughout this section does not apply for OS/4 used with the UNIVAC 9700 System, in which a job preamble requires 2048 bytes.

3.2.2. Extent Table Area

Files are defined on direct access storage devices in terms of extents. An extent is space on the volume reserved for a file and is made up of contiguous tracks. Therefore, files may consist of from one extent up to a maximum of 16 extents for each volume in a file. Files may be stored in any available area on the disc pack.

Job control generates the extent entries and determines the amount of main storage space required to contain these extent entries. This amount of space is included in the minimum storage allocation for the job. Data management maintains the index for each extent. This index resides in main storage during the execution of the job.

Since the amount of storage varies widely between jobs, job control dynamically allocates the space based on parameters in the statements in the control stream for the job. The user must specify the total number of extents required for the file for which main storage must be allocated. (See the LFD statement in Section 4 for an explanation of the required parameters.) The formulas for computing the amount of space required for the extent table area are:

Sequential Files

$$10n = S$$

Direct Access Files

$$16 + 10n + 10v = S$$

Direct Access Files (Relative Addressing)

$$48 + 12n + 10v = S$$

Indexed Sequential Files

$$12n = S$$

where:

n	=	number of extents in the file.
v	=	number of online volumes.
S	=	number of bytes required for the extent table area, rounded to the next highest double word.

Job control places the address of the last byte of the extent table area in the field defined by JP$PAD in the preamble. If there are no extent tables, JP$PAD is set to the address of the last byte in the preamble. The address of the first entry in the extent table for each file is stored in the field defined by JF$EBA in the file control block for that file.

3.2.3. I/O Protect Area

An optional hardware feature provides storage protection. When this feature is installed and is selected at supervisor generation time, the problem program is prevented from writing in all parts of main storage except the phase area and the min/max area allocated to the job being executed. The linkage editor then places in this protected area of storage certain critical portions of DTF macro instructions specified within a load module. For a multiphase load module, the longest I/O protect area for a phase determines the length of the area.

The I/O protect area starts immediately following the extent table area, or it follows the job preamble if there are no extents. The starting address of the I/O protect area is stored in the field defined by JP$PAD+4 in the preamble. The linkage editor establishes this address so that the area ends at a 512-byte boundary. Although this area is protected, it is considered to be a part of the phase area.

3.2.4. Phase Area

The phase area contains the load modules as constructed by the linkage editor or the assembler. Job control determines the required length of this area. In a tape-oriented system, the length of the phase area is equal to the length of the current phase. In a disc-oriented system, the length of the phase area is equal to the length of the longest load module in the job. If the user requests both a minimum and a maximum number of bytes to be allocated to a job, the end of the job main storage area is considered to be the end of the phase area.

The phase area starts and ends on 512-byte boundaries. The ending address is stored in the field defined by JP$PAD+8 in the preamble.

3.2.5. Min/Max Area

The user may change the size of a job main storage area by specifying through the control stream the minimum and/or maximum number of bytes required for the job. Job control then dynamically allocates the requested space at job initiation time in multiples of 512 bytes. If the space is available, job control allocates the maximum number of bytes. Otherwise, job control allocates the minimum number of bytes or any amount that is available between the minimum and maximum requirements. If the user specifies a minimum number of bytes that is in excess of the amount of space available, the job is aborted. The minimum number of bytes which can be allocated to a job by job control is 8K bytes.

The address of the last byte in the min/max area is placed in a field defined by JP$PAD+12 in the preamable.

NOTE:

The minimum storage requirement for a job given throughout this section does not apply for OS/4 used with the UNIVAC 9700 System, in which the minimum job partition is 10,000 bytes.

3.2.6. Job Limits Processing

Job limits processing has been implemented to provide the automatic computation of main storage requirements at job initiation time. A description of job limits processing follows:

RUN command –	The initial main storage partition for a job is allocated by the RUN command transient before job control is initiated on behalf of the job.
JOB statement –	The determination of main storage requirements is made when the JOB statement is processed. The job stream will be scanned regardless of presence or lack of main storage parameters on the JOB statement. The alternate load libraries and module complex library (MCL) will be bypassed when the stream is scanned. However, when executing from libraries stated above, to ensure having sufficient main storage to run a MIN parameter large enough to contain the program, preamble, and extent tables, rounded up to the next module boundary is necessary. If MAX is specified, additional available main storage is added to the defined or computed MIN valve until the MAX value is satisfied.
ALL cases –	Note that main storage is not released by job limits processing. If a partition larger than the MIN/MAX parameters is allocated as a result of RUN command (statement) request, the larger partition will remain allocated for the entire job. All specification rules still apply.

3.2.6.1. Main Storage Allocation Using Job Control Statements

The RUN statement or RUN operator command and the JOB statement are used for allocation of main storage. The following compares main storage allocation on the RUN statement with main storage allocation on the JOB statement:

■ Main Storage Not Specified on RUN Command/Statement and Not Specified on JOB Statement

  If a RUN command/statement is encountered without parameters for allocation of main storage, a minimum main storage partition is allocated on a temporary basis to the job. Then the control stream is scanned. The libraries specified on the EXEC statements are queried for program names specified, and the largest phase length is saved until the entire stream has been scanned. Upon completion of the scan, additional main storage, up to the length of the largest phase is allocated, if required. However, because of the size requirement of job control, if the job does not require a minimum main storage partition, the minimum partition allocated originally remains allocated for the job.

■ Main Storage Specified on RUN Command/Statement and Not Specified on JOB Statement

  If a RUN command/statement is issued with parameters for allocation of main storage, that amount of main storage is allocated on a temporary basis to the job. Then, the control stream is scanned. The libraries specified on the EXEC statements are queried for program names specified and the largest phase length is saved until the entire stream has been scanned. Upon completion of the scan, additional main storage, up to the length of the largest phase, is allocated if required. However, if the job does not require more main storage than that specified on the RUN command/statement, the job is run with the amount of main storage originally allocated on the RUN statement.

■ Main Storage Specified on RUN Command/Statement and Specified on JOB Statement

  If a RUN command/statement is encountered with parameters for allocation of main storage, that amount of main storage is allocated on a temporary basis to the job. Then if the JOB statement contains a storage allocation parameter, a check is made to see if the main storage request on the JOB statement is greater than the request on the RUN command/statement. If the JOB statement request is greater, additional main storage will be allocated to fill the request. If it is not greater, the amount allocated on the RUN command remains allocated for the duration of the job. In addition, the job stream is scanned and the largest of the following three values, RUN main storage, MIN main storage, or computed main storage is allocated.

- Main Storage Not Allocated on RUN Command and Allocated on JOB Statement

  If a RUN command is issued without parameters for allocation of main storage, a minimum main storage partition is allocated on a temporary basis to the job. Then, if the JOB statement contains a main storage parameter, a check is made to see if the JOB request is greater than the minimum partition. The stream is then scanned and the larger of the two is saved. If the request is greater than the minimum partition, the additional main storage is allocated. If it is not greater, the minimum partition remains allocated for the duration of the job.

3.2.6.2. Attempting to Execute From Library

When attempting to execute a program from the module complex library or from libraries that do not contain programs at the beginning of the job, a large enough MIN parameter must be specified to accommodate any program that does not exist in the library at scan time. Examples of such a situation would include programs executed from MCL, programs moved into libraries, programs loaded from alternate devices, or programs executed from the absolute execution area.

3.2.6.3. Job Limits Processing Examples

In the following examples all sizes are hexidecimal notation.

RUN JOB,,GO,A000

// JOB NAME,,,5800

Example – When the RUN command is processed, A000 bytes are allocated. When the JOB statement is processed and the parameter is checked, if the amount of main storage allocated is already large enough to contain the amount requested on the JOB statement, the job stream is then scanned to further check if sufficient main storage has been allocated. If additional space is not required, A000 bytes remain the main storage allocation for that job.

RUN JOB,,GO,5800

// JOB NAME,,,A000

Example – When the RUN command is processed, 5800 bytes are allocated. The JOB statement is then read and checked for a main storage parameter. The parameter found is checked against main storage that has been allocated. It sees that an additional 4800 bytes are needed and goes to allocate them. The stream is then scanned to see if additional space is required. If not, A000 is allocated for the duration of the job.

RUN JOB,,GO

// JOB NAME,,,5800

Example – When the RUN command is processed and found not to contain a main storage parameter a minimum main storage partition is allocated. The JOB statement is then read and scanned for a main storage parameter. On finding the parameter, it is checked against main storage allocated. An additional 3400 bytes are requested and it goes to allocate same. The stream is then scanned to see if additional space is required. If not, 5800 is allocated for the duration of the job.

```
RUN JOB,,GO

// JOB NAME
```

Example – Since neither the RUN command nor JOB statement contains a main storage parameter, at the time the RUN command is processed, a minimum main storage partition is allocated. The job stream is then scanned through its entirety and the libraries and program phase headers checked for the largest storage allocation stated in the load module length. If there is only one EXEC statement, the storage allocated is taken from that phase header and checked against main storage that is already allocated and if it is sufficient, then that amount is allocated for the JOB's duration. If not, the additional requirement is allocated.

```
RUN JOB,,GO,A000

// JOB NAME
```

Example – When the RUN command is processed, A000 bytes are allocated. For example, after the JOB statement was scanned and no parameter was found, the job stream was scanned and the largest phase that was found had only a size of 5000 bytes. Job limits would then check this size against main storage already allocated, being that A000 was allocated by way of the RUN command and 5000 was the size found in the phase header. A000 will remain allocated for the duration of the job.

```
RUN JOB,,GO,5000

// JOB NAME
```

Example – When the RUN command is processed, 5000 bytes are allocated to the job. Upon scanning the JOB statement and finding no main storage parameter, it scans the job stream and for example, the size it found in the phase header was A000, it checks this against main storage that is already allocated, finding it needs an additional 5000 bytes and then goes to allocate same. Therefore, A000 bytes will be allocated for the duration of the job.

The following examples are of special cases:

```
  RUN JOB,,GO,A000

// JOB NAME
               .
               .
               .
// EXEC DASM,LOAD$LIB,,REL
               .
               .
               .
// EXEC DLINK,LOAD$LIB,,REL
               .
               .
               .
// EXEC PGM,MCL,,REL
/&
```

This this particular type of stream is introduced without min, max parameters on the JOB statement, the size A000 will be allocated unless DASM4 or DLINK require more than A000. MCL is not queried and is bypassed.

```
   RUN JOB,,GO,A000

// JOB NAME
              .
              .
              .
// EXEC DASM
              .
              .
              .
// EXEC DLINK
              .
              .
              .
// EXEC LIBUPS
              .
              .
              .
// EXEC PGM, USR$LIB,,REL
```

In this case, in the event that the program that is going to be executed from USR$LIB is not present in the USR$LIB when the job stream is scanned, the same would occur as stated in the previous example.

```
   RUN JOB,,GO,A000

// JOB NAME
              .
              .
              .
// EXEC PGM,,LOAD,REL
```

In this case job limits will not abort the job. The load module size will be considered as the minimum partition size for main storage computation. When the program is executed, load errors may result if the program requires more storage than was specified on the RUN command or JOB statement.

3.3. DEVICE ASSIGNMENT

The job control language permits the user greater flexibility in assigning the devices required to execute a job. The various peripheral devices available at an installation are assigned to a job based on information specified in the control stream entered at job execution time. The user may request a device to be assigned to a job by specifying the logical unit name or number, a particular device type, or the device identification. In addition to specifying which device is to be assigned, the user may specify the duration of the assignment, and the qualifying factors of the assignment.

The user must supply some or all of the following information in the control stream:

- Logical Unit Designator – The user may designate either a logical unit name or number at system generation time to assign a device.
- Volume Serial Number (VSN) – The user may request a device by specifying a particular volume. If the volume is mounted, the device is assigned to the job; if the volume is not mounted, job control determines whether the device can be assigned.
- Device Type and Identification – At system generation time, device type codes are associated with logical unit numbers; this association may be temporarily changed at job execution time. A unique 3-character device identification is assigned to each device at system generation time and may be referenced at job execution time for special device assignment.
- Duration of Device Assignment – At job execution time, the user may specify whether a device is to be assigned for the duration of the entire job or on a job step basis.
- System Devices – At system generation time or through operator commands at the system console, the user may assign certain devices that are available to all jobs executed in the system.
- Shareable Devices/Volumes – Devices or volumes may be assigned to more than one job at a time if the user so specifies.
- Alternate Devices – The user may assign two devices of the same type to a file within a job even though only one device is online at a time.
- Device Type Substitution – The user may specify the various devices which can be used as substitutions for an unavailable basic device.
- Optional Devices – The user may specify that the assignment of a particular device is optional to the execution of a job.
- File Characteristics – The user may extend the file definition and specify the type of file organization at job execution time.

Job control assigns devices as specified by the user through the DVC, VOL, EXT, LBL, and LFD statements. Due to timing considerations, these statements should be grouped in the order shown into device assignment sets where the number of these sets varies according to the job. At least one DVC and one LFD statement are required to assign a device.

A brief explanation of the logic to be used by the user in specifying the assignment of devices through parameters in the control stream is given in the following paragraphs. The basic logic used by job control in assigning devices to a job is shown in Figure 3–2. The detailed logic is given in Appendix A.

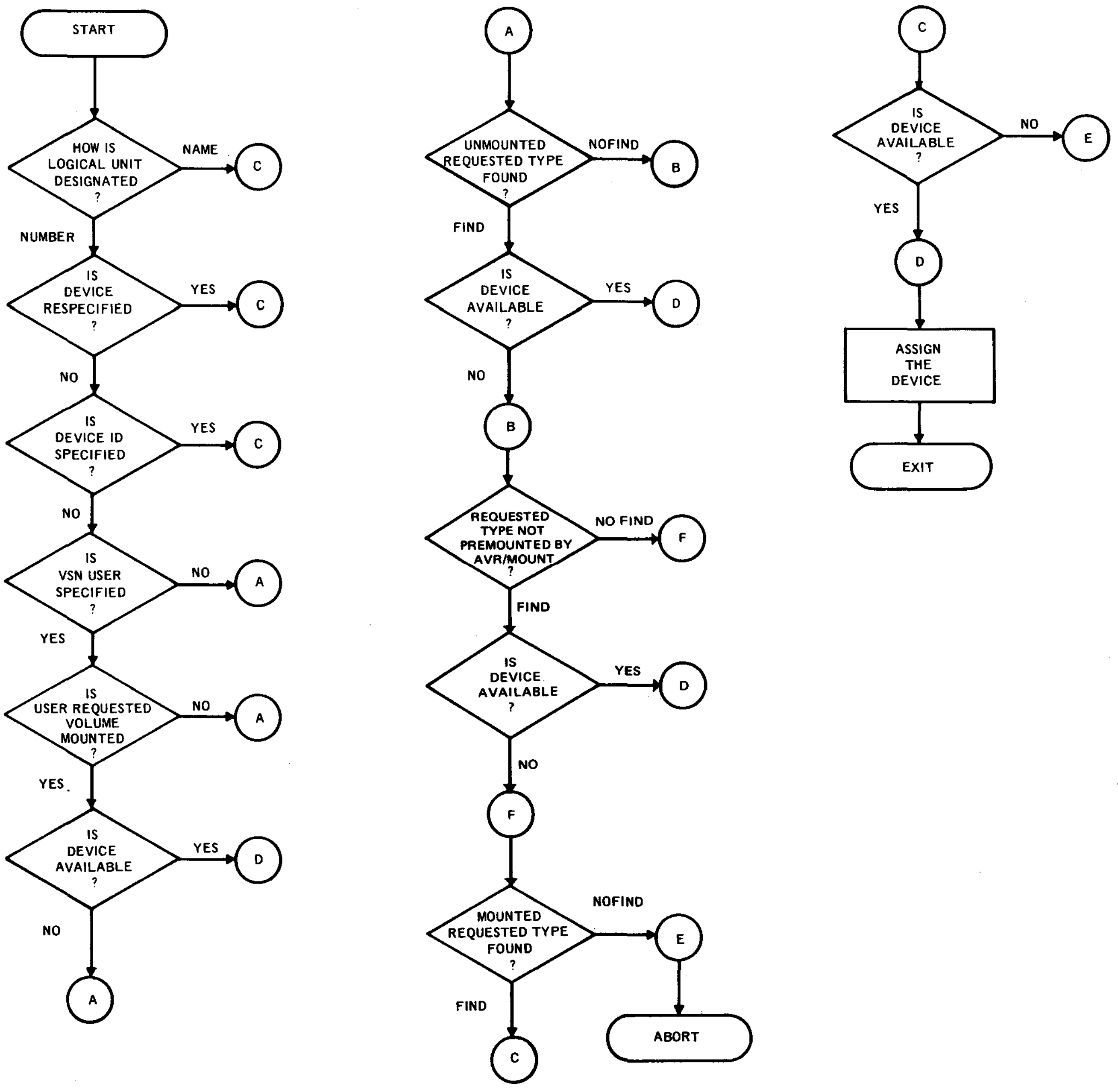


Figure 3–2. Basic Logic for Device Assignment

3.3.1. Logical Unit Designator

A device may be designated by either a logical unit number or a logical unit name. Logical unit numbers are decimal numbers in the range 000 through 255. The logical unit names may be IPT, LOG, LST, PCH, RDR, or RES.

At system generation time the user equates the logical unit numbers to given device types according to the user installation. If a different device assignment is required, the user can temporarily change the logical unit number. When assigning logical unit numbers, the user should keep the range of numbers as compact as possible in order that the number of blocks in the logical unit table be kept to a minimum, thus conserving space in the supervisor and adding to the efficiency of job control. The logical unit number is used to access an entry in the job logical unit table. The device type code is retrieved from this entry and is used to assign the requested device. (Device type codes are listed under the explanation of the EQU statement in Section 4.)

Logical unit names may not be equated to a given device type. They are specified at system generation time or through the system console. Each name represents a unique entry in the SIB and each entry contains the PUB address of the device. These device names are:

- IPT – identifies the peripheral device as the system primary input device for reading control streams.
- LOG – identifies the peripheral device as the system logging device. This is usually the system console.
- LST – identifies the peripheral device as the system listing device. This can be a line printer, magnetic tape unit, or disc drive.
- PCH – identifies the peripheral device as the system card punch.
- RDR – identifies the peripheral device as the system reader. This can be a card reader, magnetic tape unit, or disc drive.
- RES – identifies the peripheral device as the system resident device. This can be either a magnetic tape unit or disc drive.

It is possible to have more than one file name associated with a device by specifying the same logical unit number in more than one device assignment set, thus specifying a multifile volume.

A device can be respecified by repeating the logical unit number in two or more steps of the control stream and changing either the volume serial number and/or the file name on the LFD statement. If the volume serial number is not subsequently respecified, the original volume serial number remains in force. A new volume serial number may not be specified if the device and volume are shared with another job.

3.3.2. Volume Serial Number

All disc packs and magnetic tape reels, including scratch tapes, should have an assigned volume serial number. Each volume serial number must be six alphanumeric characters in length. If fewer than six characters are specified by the user, the characters are right-justified in the field, and leading zeros are added by job control.

In single volume files or in direct access files where only one volume is mounted on a single device at a time, the volume serial number of the mounted volume is stored in the PUB and in the FCB. In multivolume online direct access files, the volume serial numbers are stored in the PUB. No volume serial number list block (VSNLB) is constructed. In multivolume sequential files, the first volume serial number is stored in the PUB; all volume serial numbers, including the first, are then stored in the VSNLB. The search key to the VSNLB is contained in the file control block.

If an extent request block (ERB) is associated with the volume serial number, the sequence number of the block is appended to the volume serial number as a seventh character when job control places the volume serial number in the control block.

Volume serial numbers (VSN) are submitted to the system as parameters on the VOL statement (Section 4). Duplicate disc VSNs are not allowed except when SCRTCH is specified for the VSN.

3.3.3. Device Type and Identification

The device type code is usually specified at system generation time. However, the device type code can be temporarily changed at job execution time through the use of the EQU statement. This selection is in effect only during the execution of a particular job, or it may be reset to the original value during job execution by the RESET statement. The available device type codes are listed under the EQU statement in Section 4.

Device identification may be specified at job execution time by a parameter in the DVC statement. The device specified must not be an alternate device and the logical unit number must be associated with the correct device type.

The device identification feature is included in job control primarily for online maintenance routines and is not intended for general use by programmers.

See Section 4 for a description of the DVC, EQU, and RESET statements.

3.3.4. Duration of Device Assignment

The job control language allows the user to assign a device for the duration of a job or on a job step basis only and also to release the device upon completion of the job or job step. Devices are considered to be assigned for the duration of the job unless the user specifies through a parameter supplied at job execution time that the device is to be assigned on a job step basis only.

Devices assigned on a job basis are released at job termination unless the user supplies a statement in the control stream to release the device during job execution.

Devices assigned on a job step basis are released automatically through the job continuation function of job control or during job step execution when the user issues a macro instruction in the problem program. See *UNIVAC 9400 System Supervisor Programmer Reference, UP-7689* (current version) for an explanation of the FREE macro instruction.

3.3.5. System Devices

The devices that are to be assigned to the system are specified at system generation time or through operator commands at the system console. These devices may be the system resident device, the system reader, the system primary input device, the system logging device, the system card punch, and the system listing device.

System devices are considered to be shareable and assigned to all jobs. When a system device is to be assigned by logical unit number, the user must specify a volume serial number that is the same as the volume serial number in the physical unit block (PUB). Otherwise, the device is not assigned. If a device is allocated by logical unit name and no volume serial number is specified, the device is assigned without regard to volume serial number. If a blank volume serial number is in the PUB, then a blank volume serial number must be used, as in the case of the console or the card reader.

3.3.6. Shareable Devices/Volumes

Shareable devices or volumes are those which may be assigned to more than one job at a time. In order for a device to be designated as shared, the operator must key in a SET IO operator command at the system console. The most commonly used shareable device is the UNIVAC 8411 or 8414 Disc Subsystem. Before a volume can be shared, the user must identify it by volume serial number in the control stream and must specify that the device is shareable.

Any device may be designated as shared. A shareable device may not be specified as an alternate device. The volume serial number in the PUB must be the same for both jobs.

3.3.7. Alternate Devices

The user may wish to assign another device of the same type which can be used as an alternate during job execution. Assigning alternate devices gives the user the opportunity of having two devices assigned to a file within a job even though only one device can be online at a time. This facility is most frequently used during the processing and handling of sequential files. Alternate devices are specified by the user through parameters supplied at job execution time. The device assigned as an alternate is the same type as the basic device and cannot be a shareable device.

If the basic device is not assigned, the alternate device cannot be assigned. If the basic device is assigned but the alternate device cannot be assigned, the user receives an indication that the alternate device was not assigned.

If both the basic device and the alternate device are assigned, the address of the PUB for the alternate device is placed in the field defined by IP$ALT in the PUB for the basic device, while the address of the PUB for the basic device is placed in the field defined by IP$ALT in the PUB for the alternate device.

The alternate device is placed online when the user issues a SWAP macro instruction in the problem program. See *UNIVAC 9400 System Supervisor Programmer Reference, UP-7689* (current version) for an explanation of the SWAP macro instruction.

3.3.8. Device Type Substitution

Logical unit numbers are associated with specific device types. At times this may be too restrictive for the user whose installation has a mixture of similar types of devices. The user may wish to assign either a tape or a disc, but the specific type is not important to the execution of the job. At system generation time the user can establish within the resident supervisor a table of device type substitutions which job control can check if the basic device cannot be assigned. The table is located directly following the logical unit table (LUT).

Device type substitution is indicated by positional parameter 2 of the DVC statement. This parameter is an alphabetic character (A through Z) which has a corresponding entry in the table. Up to 26 device type substitution entries can be submitted by the user. The alphabetic character identifying the entry is used by job control to assign a device as a substitution for an unavailable basic device. In addition to the identifier, each entry in the table contains, in hexadecimal, the one or more device type substitutions as determined and specified by the user. The table entry is terminated by a device type of 00 (hexadecimal). The table is terminated by an entry of FF (hexadecimal). One of the device type substitutions listed in the table is selected if the active device type cannot be found in the job logical unit table.

An example of the relationship between positional parameter 2 of a DVC statement and the corresponding entry in the device type substitution table is given in Appendix D.

3.3.9. Optional Devices

Requested devices are considered to be prerequisite to the execution of the job unless specified by the user as optional in the control stream. If the device is specified as being optional and is available, the device is assigned. If the device is specified as being optional to the running of a job and the device is not available, an indication is given in the FCB that the request is not filled. The field defined by JF$PUB (the address of the associated PUB) is set to binary 0's.

3.3.10. File Characteristics

The user must specify the type of file organization and whether the file is a new disc file. A file may be organized for direct access, direct access relative, sequential, or indexed sequential processing.

When a new disc file is required by a problem program, the user specifies a parameter in an LFD statement in the control stream at job execution time. A VOL statement and at least one EXT statement must also be included in the same device assignment set (Section 4).

If a problem program requires an ASCII file or if mismatch errors on a printer file are to be ignored, the appropriate parameters must be specified in an LFD statement at job execution time.

3.4. CONTROL BLOCK FILE

During job initiation, job control establishes a file which contains the various control blocks for the job. This file may be updated during the execution of the job as well as between job steps. The control block file consists of the following:

- Job Logical Unit Table – contains a copy of the master logical unit table adjusted to represent the current control stream.
- File Control Blocks (FCB) – contain the device assignments, tape or disc label information, and the search key for the volume serial number list block.
- Volume Serial Number List Blocks (VSNLB) – contain the volume serial numbers associated with the file.
- Extent Request Blocks (ERB) – contain the disc space information associated with a volume.

In a tape-oriented system, the logical unit table is located in the system information block (SIB) and is therefore not a part of the actual control block file; all other blocks are stored in high order main storage allocated to the job main storage area. In a disc-oriented system, the master logical unit table is in the SIB; the job logical unit table and the other control blocks pertaining to a particular job are written in the job control block file area of the system resident disc pack.

Each block in the control block file, except the logical unit table, is linked to every other block. An entry in the job preamble contains the starting address of the control block file. Search keys for the volume serial number list block and the extent request block are stored in the file control block. The starting address of the file control block and the extent request block sequence number are stored in the volume serial number list block. The starting address of the file control block and the associated volume serial number are stored in the extent request block. An example of this linkage is illustrated in Figure 3–3.

The various blocks forming the control block file and the contents of the fields within the blocks are described in the following paragraphs.

SEQUENTIAL FILES (Tape or Disc)

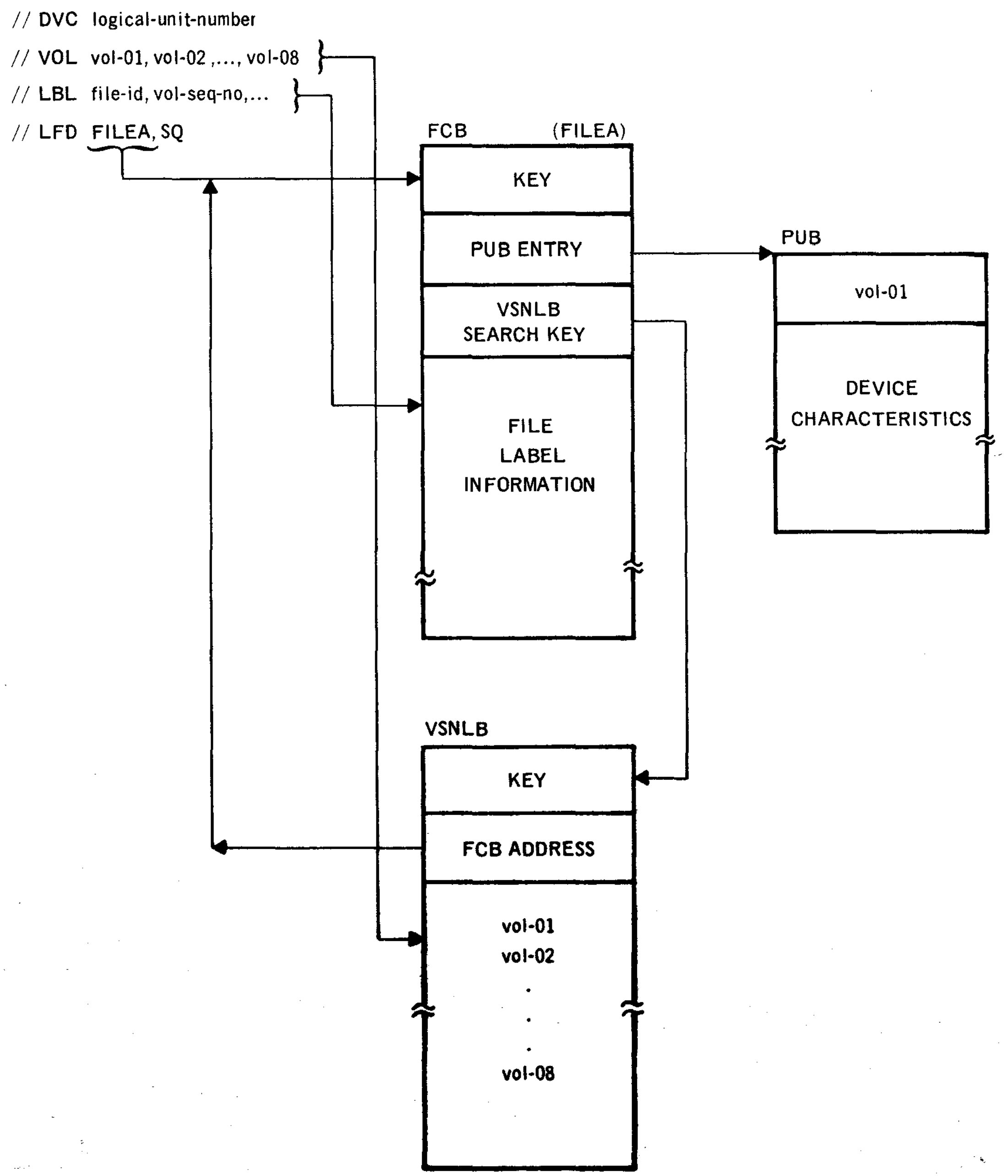


Figure 3–3. Examples of Control Block File Linkage (Part 1 of 2)

DIRECT ACCESS FILES (Multivolume online)

```
// DVC  logical-unit-number
// VOL  vol-04
// EXT  C, , CYL, 20
// DVC  logical-unit-number
// VOL  vol-05
// EXT  C, , CYL, 20
// LBL  file-id, vol-seq-no,...
// LFD  FILEB, DA,2,NEW
```

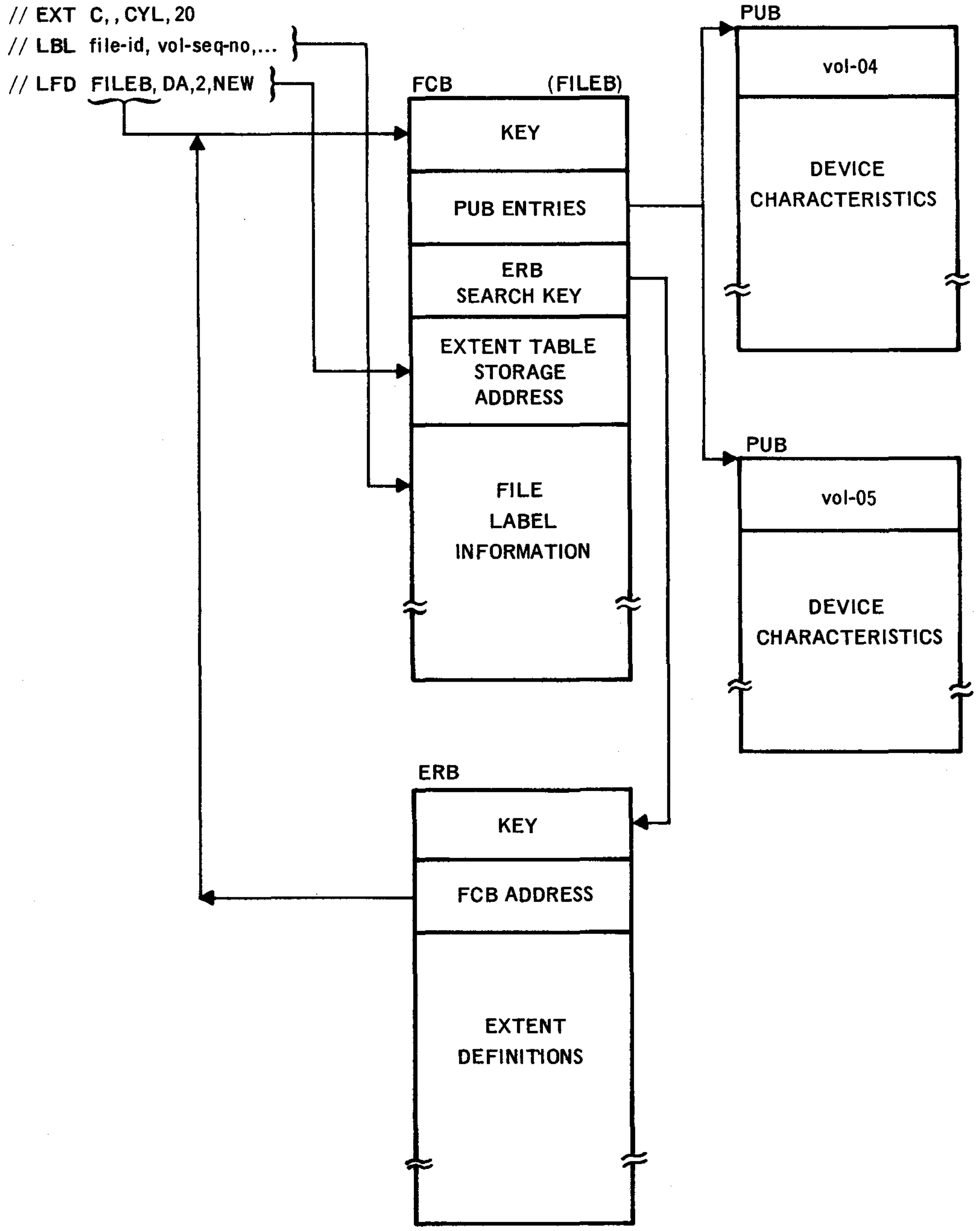


Figure 3–3. Examples of Control Block File Linkage (Part 2 of 2)

3.4.1. Logical Unit Table

The logical unit table is specified by the user and is generated at system generation time. The master logical unit table is a part of the SIB. In a tape-oriented system, this is the only logical unit table necessary. In a disc-oriented system with multijobs, a logical unit table is required for each job in the system in addition to a master logical unit table. At the beginning of a job, the job logical unit table is copied from the master logical unit table and is destroyed at the end of that job. A job logical unit table may be changed or updated between job steps. The master logical unit table cannot be altered.

The job logical unit table in a disc-oriented system consists of an 8-byte search key and up to 30 full-word entries. Each entry in the table corresponds to a logical unit number associated with a device of a specific type. The first byte in the entry contains the active device type code. The second byte contains the standard device type code which is assigned at system generation time. The third and fourth bytes of the entry contain the PUB address for the device assigned on a job basis to this logical unit number. No PUB address is present for a device assigned on a job step basis.

In a disc-oriented system, each entry in the master logical unit table consists of one byte which contains the device type code for each logical unit required by the system.

An explanation of the method of assigning logical unit designators is given in 3.3.1.

Job control must assign a device for each input or output file. The logical unit number associated with a given device is specified by the user as a parameter in the DVC statement. Job control uses the logical unit number as a displacement value to obtain the associated device type entry in the job logical unit table. The device type entry is used as a search key to locate in the PUB table the first unassigned device of the same type. Job control then assigns this device to the appropriate file and places the PUB address in the job logical unit table, as well as in the file control block.

The assignment of a device type to a logical unit number may be changed by means of the EQU statement. Any device type that has been changed by an EQU statement may be reset to the original (system generation) value by means of the RESET statement.

When assigning a device to a job and if the PUB address entry in the job logical unit table is blank, job control searches the PUB table for a volume serial number of a device that is the same type as that specified in the job logical unit table. The device may then be assigned, and the PUB address of that volume serial number is placed in the job logical unit table.

Fields within each block are identified by standard system labels. These labels are a maximum of eight characters and are expressed in the form JU$xxxxx, where the characters JU$ identify logical unit table labels and the characters xxxxx identify fields within the block. Field labels and a brief description of their contents are provided in Table 3–1.

Table 3–1. Logical Unit Table Standard Labels

Classification	Label	Boundary Alignment	Field Length (Bytes)	Label Description
Key		Full word	8	Search key in the form JU$LTx, where x is the block number (0 through 8).
Counter	JU$NE	Full word	4	Number of entries in this block (1 through 30).
	JU$NE+4		4	Reserved.
Device type	JU$FE	Full word	120	First entry in this block.
	JU$FE+0		1	Active device type code.
	JU$FE+1		1	Reset device type code assigned at system generation time.
	JU$FE+2		2	PUB address for the device assigned to this logical unit number.
	JU$FE+4		4	Second entry in this block in the format previously described.
	JU$FE+8		4	Third entry in this block in the format previously described.
	. . .			. . .
	JU$FE+116		4	Thirtieth entry in this block in the format previously described.

3.4.2. File Control Blocks

The file control blocks contain file and device information compiled by job control when the control stream is evaluated. In a tape-oriented system, the file control blocks are stored in high order main storage and are built from the end of the assigned job main storage area backwards. In a disc-oriented system, job control acquires space in the system temporary storage pool and constructs the file control block in this area.

In either system, a file control block is constructed for each file of the job. The block has a 1- to 8-byte search key which is the file name as specified in the LFD statement. Unless otherwise specified, addresses in the file control block are expressed in binary and are absolute.

Fields within a file control block are identified by standard system labels. These labels are a maximum of eight characters and are expressed in the form JF$xxxxx, while the characters JF$ identify file control block labels and the characters xxxxx identify fields within the block. Field labels and a brief description of their contents are provided in Table 3–2.

Table 3–2. File Control Block Standard Labels (Part 1 of 3)

Classification	Label	Boundary Alignment	Field Length (Bytes)	Label Description
PUB addresses and volume serial number	JF$PUB	Full word	2	PUB entry 1
	JF$PUB+2		2	PUB entry 2
	JF$PUB+4		2	PUB entry 3
	JF$PUB+6		2	PUB entry 4
	Direct Access, Direct Access Relative, and Indexed Sequential Files Only			
	JF$PUB+8		2	PUB entry 5
	JF$PUB+10		2	PUB entry 6
	JF$PUB+12		2	PUB entry 7
	JF$PUB+14		2	PUB entry 8
	Sequential Files Only			
	JF$VSN	Full word	8	If one VSN exists per file, a 6-byte VSN is stored left-justified in the 8-byte field. (Remaining two bytes are reserved.) If more than one VSN exists per file, an 8-byte VSNLB search key (JV$snnnn) is stored in the field. The field is subdivided as follows:
	JF$VSN+0	Full word	3	The constant JV$.
	JF$VSN+3		1	A sequence number (0 through 9) identifying the block within the file.
	JF$VSN+4		4	A random file number in EBCDIC (nnnn) identifying the block within the job.
Volume counters and flags	JF$VCK	Full word	1	Number of VSNs, or maximum number of lines per page when CO-OP forms loop is specified.
	JF$VCF+1		1	Scratch volume indicators: Bit 0, 1 = scratch volume in PUB 1 Bit 1, 1 = scratch volume in PUB 2 Bit 2, 1 = scratch volume in PUB 3 Bit 3, 1 = scratch volume in PUB 4 Bit 4, 1 = scratch volume in PUB 5 Bit 5, 1 = scratch volume in PUB 6 Bit 6, 1 = scratch volume in PUB 7 Bit 7, 1 = scratch volume in PUB 8

*For printer files only. Indicates mismatch errors should be ignored by data management.

Table 3 2. Table 3–2. File Control Block Standard Labels (Part 2 of 3)

Classification	Label	Boundary Alignment	Field Length (Bytes)	Label Description
	JF$VCF+2		1	Flags: Bit 0, 1 = block count option specified Bit 1, 1 = optional device is not found Bit 2, 1 = file is new* Bit 3, 1 = ASCII file Bit 4, 1 = direct access relative file organization Bit 5, 1 = index sequential file organization Bit 6, 1 = sequential file organization Bit 7, 1 = direct access file organization
	JF$VCF+3		1	Extent flags: Bit 0, 1 = extent request block for PUB 1 Bit 1, 1 = extent request block for PUB 2 Bit 2, 1 = extent request block for PUB 3 Bit 3, 1 = extent request block for PUB 4 Bit 4, 1 = extent request block for PUB 5 Bit 5, 1 = extent request block for PUB 6 Bit 6, 1 = extent request block for PUB 7 Bit 7, 1 = extent request block for PUB 8
Extent block descriptor	JF$EBA	Full word	1	Number of double words of extent storage in binary.
	JF$EBA+1		3	Storage address in binary containing the starting address of extent storage.
Tape file header information	JF$LBL	None assumed	17	Tape file identifier left-justified and space filled.
	JF$TLB		65	Tape file information. This information conforms to the tape header label format.
	JF$TVS		6	Tape volume serial number in EBCDIC right-justified and zero filled.
	JF$TVQ		4	Tape volume sequence number in EBCDIC right-justified and zero filled.
	JF$TFQ		4	Tape file sequence number in EBCDIC right-justified and zero filled
	JF$TGV		4	Tape generation number in EBCEIC right-justified and zero filled.
	JF$TGV+4		2	Tape version number in EBCDIC right-justified and zero filled.
	JF$TCD		6	Tape creation date consisting of one space and five EBCDIC characters.

*For printer files only. Indicates mismatch errors should be ignored by data management.

Table 3–2. File Control Block Standard Labels (Part 3 of 3)

Classification	Label	Boundary Alignment	Field Length (Bytes)	Label Description
Tape file header information (cont)	JF$TED		6	Tape expiration data which has the same format as the creation data.
	JF$TSB		1	Tape security
	JF$TSB+1		6	Tape block count
	JF$TSC		13	Tape system code
Disc label information	JF$LBL	None assumed	44	Disc identifier left-justified and space filled.
	JF$DLB		65	Disc file information. This information conforms to the disc format 1 label format.
	JF$DFS		6	Disc volume serial number in EBCDIC right-justified and zero filled.
	JF$DVQ		2	Disc file sequence number in binary.
	JF$DCD		3	Disc file creation date in discontinuous binary.
	JF$DED		3	Disc file expiration date which is in the same format as the creation date.
	JF$DED+3		3	Reserved
	JF$DSC		13	Disc file system code
	JF$DSC+13		18	Reserved
	JF$DMF		1	Reserved
	JF$DMF+1		6	Reserved
	JF$EBK		9	Disc file extent request block key (xx$snnnn) is stored in the field. The field is subdivided as follows:
	JF$EBK+0		3	The constant xx$
	JF$EBK+3		1	A sequence number (0 through 9)
	JF$EBK+4		4	Blanks
	JF$EBK+8		1	Unused

*For printer files only. Indicates mismatch errors should be ignored by data management.

3.4.3. Volume Serial Number List Blocks

When more than one volume is associated with a sequential file, job control compiles a volume serial number list block at job initiation time. The first volume serial number in the file is placed in the PUB, as well as in the volume serial number list block. The file control block then contains the 8-byte search key of the volume serial number list block.

The volume serial number list block contains the starting address of the file control block, a maximum of 14 volume serial number entries, and an 8-byte search key of the next volume serial number list block. If an extent request block is associated with the volume, the sequence number of the request is stored in the entry for the associated volume serial number. Additional blocks are compiled as required.

If the first volume of a multivolume file has a volume serial number, every other volume of the file should also have a volume serial number.

Fields within a volume serial number list block are identified by standard system labels. These labels are a maximum of eight characters and are expressed in the form JV$xxxxx, where the characters JV$ identify volume serial number list block labels and the characters xxxxx identify fields within the block. Field labels and a brief description of their contents are provided in Table 3–3.

3.4.4. Extent Request Blocks

When an EXT statement is encountered in the control stream, job control constructs an extent request block which supplies the disc space information associated with one volume. Each extent request block contains the address of the associated file control block, the associated volume serial number, the number of requests appearing in the block, and up to 16 extent requests.

Fields within extent request blocks are identified by standard system labels. These labels are a maximum of eight characters and are expressed in the form JX$xxxxx, where the characters JX$ identify extent request block labels and the characters xxxxx identify fields within the block. Field labels and a brief description of their contents are provided in Table 3–4.

Table 3–3. Volume Serial Number List Block Standard Labels (Part 1 of 2)

Classification	Label	Boundary Alignment	Field Length (Bytes)	Label Description
FCB address	JV$FCB	Double word	8	Address of the file control block*
First volume serial number entry	JV$VSN	Full word	6	First volume serial number in the block in EBCDIC right-justified and zero filled
	JV$VSN+6		1	Extent request block sequence number in EBCDIC associated with first volume serial number (if required)
	JV$VSN+7		1	Unused

*DOS – has the form 00CCHHR0
TOS – first word – memory address of FCB;
second word – binary zeros

Table 3–3. Volume Serial Number List Block Standard Labels (Part 2 of 2)

Classification	Label	Boundary Alignment	Field Length (Bytes)	Label Description
Subsequent volume serial number entries	JV$VSN+8	Double word	8	Second volume serial number entry (in the format previously described)
	JV$VSN+16		8	Third volume serial number entry (in the format previously described)
	JV$VSN$24		8	Fourth volume serial number entry (in the format previously described)
	JV$VSN+32		8	Fifth volume serial number entry (in the format previously described)
	JV$VSN+40		8	Sixth volume serial number entry (in the format previously described)
	JV$VSN+48		8	Seveneth volume serial number entry (in the fomat previously described)
	JV$VSN+56		8	Eighth volume serial number entry (in the format previously described)
	JV$VSN+64		8	Ninth volume serial number entry (in the format previously described)
	JV$VSN+72		8	Tenth volume serial number entry (in the format previously described)
	JV$VSN+80		8	Eleventh volume serial number entry (in the format previously described)
	JV$VSN+88		8	Twelfth volume serial number entry (in the format previously described)
	JV$VSN+96		8	Thirteenth volume serial number entry (in the format previously described)
	JV$VSN+104		8	Fourteenth volume serial number entry (in the format previously described)
List block linkage	JV$LNK	Double word	8	Search key of next volume serial number list block, in EBCDIC

*DOS – has the form 00CCHHR0
TOS – first word – memory address of FCB;
second word – binary zeros

Table 3–4. Extent Request Block Standard Labels (Part 1 of 2)

Classification	Label	Boundary Alignment	Field Length (Bytes)	Label Description
Number of extent requests	JX$NER	Half word	2	Number of extent requests, in binary
FCB address	JX$FCB	Half word	8	Address of file control block*
Volume serial number	JX$VSN	Half word	8	Volume serial number associated with this extent request block
First extent definition	JX$EXT	Half word	1	Extent identification: Bit 0, 1 = file is contiguous Bits 1, 2: 00 = track request 10 = cylinder request 11 = absolute address request Bits 3–7: unused
	JX$EXT+1		1	Flag bits: Bits 0–3: unused Bits 4–7: data management ISAM flags
	JX$EXT+2		2	Starting address in binary of an absolute request in the form: ccc 1 byte hh 1 byte
	JX$EXT+4		2	Number of tracks requested
Subsequent extent definitions	JX$EXT+6	Half word	6	Second extent definition (in the format previously described)
	JX$EXT+12		6	Third extent definition (in the format previously described)
	JX$EXT+18		6	Fourth extent definition (in the format previously described)
	JX$EXT+24		6	Fifth extent definition (in the format previously described)
	JX$EXT+30		6	Sixth extent definition (in the format previously described)
	JX$EXT+36		6	Seventh extent definition (in the format previously described)
	JX$EXT+42		6	Eighth extent definition (in the format previously described)

*DOS – has the form 00CCHHR0
TOS – first word – memory address of FCB;
second word – binary zeros

Table 3–4. Extent Request Block Standard Labels (2 of 2)

Classification	Label	Boundary Alignment	Field Length (Bytes)	Length Description
Subsequent extent definitions (cont.)	JX$EXT+48	Half word	6	Ninth extent definition (in the format previously described)
	JX$EXT+54		6	Tenth extent definition (in the format previously described)
	JX$EXT+60		6	Eleventh extent definition (in the format previously described)
	JX$EXT+66		6	Twelfth extent definition (in the format previously described)
	JX$EXT+72		6	Thirteenth extent definition (in the format previously described)
	JX$EXT+78		6	Fourteenth extent definition (in the format previously described)
	JX$EXT+84		6	Fifteenth extent definition (in the format previously described)
	JX$EXT+90		6	Sixteenth extent definition (in the format previously described)

*DOS – has the form 00CCHHR0
TOS – first word – memory address of FCB;
second word – binary zeros

4. CONTROL LANGUAGE

4.1. GENERAL

Jobs are defined by the user to the system in the job control language. This language consists primarily of a sequence of control statements that are used to define the particular aspects of a job to the system and direct the execution of the job.

The control statements allow the user to:

- identify the job and the programs that make up the job;
- specify the storage requirements and the devices necessary to execute the job;
- establish the linkage between the physical devices and data management;
- terminate a job;
- restart a job from a checkpoint.

4.2. CONTROL STATEMENT FORMAT CONVENTIONS

All control statements provided in the UNIVAC OS/4 Operating System (OS/4) Job Control language are essentially freeform. Information starts in character position 1 and is not permitted to extend beyond 71 positions. Control statements begin with either one or two slashes. In control statements beginning with one slash, no space is permitted between the slash and the symbol, and the slash must appear in column 1. However, one space must appear between the symbol and the operand field. In control statements beginning with two slashes, at least one space must appear between the last slash and the operation field, and also between the operation and operand fields. An exception to this rule is the continuation statement (//n).

More than one control statement of the type beginning with two slashes may be written on a card, but must not extend beyond column 71 and must not be continued on a second card. At least one space precedes the slashes, denoting the beginning of the second statement on the card. Control statements written in this manner are referred to as packed statements.

Control statements must not be packed if the tape supervisor was generated with the MINS parameter specified on the system proc. If the statements are packed, multiple error conditions will result. Also, control statements must not be packed if a tape-only system is being run in 32K bytes of storage.

The conventions used to illustrate control statements in this manual are:

- Capital letters and punctuation marks (except braces, brackets, and ellipses) are information that must be coded exactly as shown.
- Lowercase letters and terms represent information that must be supplied by the programmer.
- Information contained within braces represents necessary entries of which one must be chosen.
- Information contained within brackets represents optional entries that are included or omitted depending on program requirements. Braces within brackets signify that one of the entries must be chosen if that positional parameter is included.
- An ellipsis (a series of three periods) indicates the presence of a variable number of entries.
- Commas are required when positional parameters are omitted except for trailing parameters.

4.3. CONTROL STATEMENT SOFTWARE CONVENTIONS

The following rules and conventions apply for the use of control statements in job control:

- Control statements and data statements cannot appear on the same punched card.
- Data statements are assumed to be 80 characters.
- Punched cards which contain control statements must begin with one of the following in columns 1, 2, and 3: //ƀ, /*ƀ, /$ƀ, /&ƀ. An exception to this rule is the continuation statement (//nƀ in columns 1, 2, 3 and 4).
- On packed cards a statement is assumed to terminate with a blank column immediately followed by a slash.
- The slash ampersand (/&), slash dollar (/$), and slash asterisk (/*) control statements must appear in columns 1 and 2 (may not appear on packed cards).
- Comments are permitted on job control statements so long as they do not contain a slash which is immediately preceded by a blank column.
- Cards containing multiple control statements are processed from column 1 to column 71 inclusive.
- Columns 72 through 80 are not scanned, but their contents are returned with each GETCS macro instruction. (For explanation of assignment of sequence numbers see 2.2.1.)

4.4. CONTROL STATEMENTS

The control statements provided in the job control language are explained in the remainder of this section. The statements are listed in alphabetical order by operation code, followed by the data delimiter statements. Examples of the proper sequence of statements within various types of control streams are shown in Appendix B.

4.4.1. ALTER Statement

Function:

The ALTER statement is used to introduce alterations to a load module at execution time. The user may change portions of the program status word or establish a new base address for a program. The effective main storage address is calculated by the alter routine as follows:

E = I + A + O if R* address or address

E = A + O if A* address

E = P + A + O if P* address

where:

E	is the effective address.
I	is the phase area base address.
P	is the preamble address.
A	is the address from positional parameter 2.
O	is the previously specified ORG address.

Job control verifies that the main storage address is within the assigned area. If the address is not within the assigned area, the alter function is not performed. ALTER statements must be placed between the EXEC statement and the PARAM statements for the phase of the program to be altered. In order for the alter to be effected, // OPTION ALTER must be specified for the job step in which the alter is to take place.

Format:

```
          ┌ ┌ PM         ┐ ┐
          │ │ RST        │ │ ┌  ┌ program-mask ┐ ┐ ┌  ┌ OPR   ┐ ┐
          │ │ ORG        │ │ │  │ rst-address  │ │ │ ,┤ RESET ├ │
// ALTER  │ ┤ address    ├ │ │ ,┤ org-address  ├ │ └  └ LAST  ┘ ┘
          │ │ A*address  │ │ └  └ change       ┘ ┘
          │ │ P*address  │ │
          └ └ R*address  ┘ ┘
```

Positional Parameter 1:

PM	indicates that bits 2 through 5 of the byte specified in positional parameter 2 (program-mask) replace the condition code and the program mask portion of the job's program status word, bits 34 through 37. (These bits are located in the job control block at JB$PSW+4.) Bits 0, 1, 6, and 7 are ignored.
RST	indicates that the address specified in positional parameter 2 (rst-address) replaces the program restart address in the job's program status word, bits 47 through 63. (These bits are located in the job control block at JB$PSW+5 through JB$PSW+7.)

ORG	indicates that the address specified in positional parameter 2 (org-address) is to be added to all addresses specified in positional parameter 1 on succeeding, consecutive ALTER statements. This address is used as the base address in the calculation of the effective address for subsequent ALTER statements. This base address remains effective until another ALTER statement specifying the ORG parameter is encountered, or until LAST is specified as positional parameter 3.
address	the 1- through 5-digit hexadecimal address of the first byte of the area into which the byte or bytes specified by positional parameter 2 (change) are to be stored. If the address is not prefixed with A* or P*, then R* is assumed.
A* address	the prefix A* is used to indicate an absolute address.
P* address	the prefix P* is used to indicate that the address is relative to the first byte of the job preamble.
R* address	the prefix R* is used to indicate that the address is relative to the code image area of the problem program. Relative patches must be computed by means of the linker map and/or assembler listings.

Positional Parameter 2:

program mask	two hexadecimal digits used to replace the condition code and program mask portion of the program status word, bits 34 through 37, for the job. The bit layout of the program mask byte is:

0	1	2	3	4	5	6	7
0	0	new condition code		new program mask		0	0

NOTE:

For OS/4 used with the UNIVAC 9700 System, use the following replacement.

program mask	two hexadecimal digits used to place the condition code and program mask portion of the program status word (bits 34 through 39) for the job. The bit layout of the program mask byte is:

0	1	2	3	4	5	6	7
0	0	new condition code		new program mask			

rst-address	the hexadecimal address, relative to the first byte of the code image area of the job, at which the job is to resume control. This value plus the code base address is stored in the restart PSW in the job control block. The effective address (rst-address + l) is verified so that it is within the phase area of the problem program.
org-address	specifies the base address which is to be added to subsequent main storage addresses in the computation of effective addresses. The base address may be from one through five hexadecimal digits.

change — specifies the contents of the byte or bytes to be stored in main storage beginning at the address specified by positional parameter 1 and extending for as many bytes as required by this parameter. Change characters are not stored if both the lowest and highest effective addresses for the change are not valid. The change may be specified by either hexadecimal or EBCDIC characters. The formats are:

X'cccccccc...'
or
cccccccc...

specifies that the change characters are in hexadecimal. The number of characters must be even, and the maximum number of hexadecimal characters allowed is 16 (eight bytes).

C'cccccccc...'

specifies that the change characters are in EBCDIC. The number of characters in the string indicates the number of bytes to be changed in main storage.

Positional Parameter 3:

LAST — used to indicate the last of a series of alterations initiated by an ALTER statement or operator command and causes the termination of the alter function. If positional parameters 1 and 2 are omitted (,,LAST), control is returned to the problem program. (This is considered to be a null ALTER statement. The alter mode indicator is not reset and control remains in the proper program overlay segment.)

RESET — resets the alter mode indicator for the job after this statement is processed. RESET implies the LAST option.

OPR — causes changes to be solicited from the system console until a LAST or RESET parameter is encountered in an ALTER command.

if omitted — causes control statements to be read until an ALTER statement with a LAST or RESET parameter is encountered or until a control statement other than an ALTER statement is read.

Examples:

LABEL 1	OPERATION 10	OPERAND 16	REMARKS
// ALTER		,,LAST	No alterations for the first program segment
// ALTER		PM,3C	New condition code = 3, program mask = 3
// ALTER		RST,144	Restart at relative location 144
// ALTER		P*8,00000001	R0 will be 1 when restarted
// ALTER		P*D4,12345678	Set first word of communication region
// ALTER		R*1000,15	Modify relative location 1000
// ALTER		1000,15	Same as previous statement
// ALTER		ORG,186	Add X'186' to following addresses
// ALTER		52,12345678	Modify X'186' plus X'52' relative to start of code area
// ALTER		ORG,0,LAST	Terminate the addition of previous ORG value to addresses. This is the last ALTER statement for this program segment and job step.
// ALTER		10243,C'ABCDXYZ',RESET	Modify 7 bytes at relative location 10243

4.4.2. CANCEL Statement

Function:

The CANCEL statement is used to terminate the processing of a control stream. When a CANCEL statement is encountered in the control stream in either a tape- or disc-oriented system, job control terminates the job at that point. In a tape-oriented system, the rest of the control stream is bypassed; in a disc-oriented system, the rest of the control stream is ignored and the termination is considered to be abnormal.

No positional parameters are required for the CANCEL statement.

Format:

// CANCEL

Example:

LABEL ƀ	OPERATION ƀ	OPERAND ƀ
1	10	16
// CANCEL		

4.4.3. DELETE Statement

Function:

The DELETE statement in the control stream of a job is used to indicate that this job is to be deleted from the job file. If the job has terminated abnormally or has aborted, the control stream is not deleted from the job file. The DELETE statement may occur at any time in the control stream and the control stream is then deleted at the normal termination of that job.

When a control stream is deleted from a job file, the associated direct access storage area is deallocated and made available for subsequent allocation.

The DELETE statement is only applicable to a disc-oriented system (DOS).

No positional parameters are required for the DELETE statement.

Format:

// DELETE

Example:

LABEL ƀ	OPERATION ƀ	OPERAND ƀ
1	10	16
// DELETE		

4.4.4. DVC Statement

Function:

The DVC statement is used to request the assignment of a peripheral device to the job. An alternate device or an optional device of the same type can also be requested. Devices can be assigned for the duration of the job or for the duration of a job step. Specific devices can also be requested and assigned by this statement.

Format:

$$// \text{ DVC logical-unit} \left[, \left\{ \begin{array}{l} \text{ALT} \\ \text{a} \\ \text{SYM} \\ \text{ASYM} \end{array} \right\} \right] [\text{,STEP}][\text{,uuu}][\text{,OP}]$$

Positional Parameter 1:

logical-unit	a logical unit number or name identifying the type of device requested. Valid numbers are in the range 0 through 255; the names are IPT, LOG, LST, PCH, RDR, or RES. When a logical unit number is specified, a specific device is requested by indicating the device identification in positional parameter 4.

Positional Parameter 2:

ALT	used to request the assignment of an alternate device of the same type (ignored if positional parameter 4 is specified).
a	any alphabetic letter (A through Z) that identifies an entry in the table listing the device type substitutions which can be used in assigning devices.
SYM	specifies that the punch or printer output may be routed to the cooperative if a punch or printer is not available for assignment. This parameter is not effective if the cooperative feature is not included in the resident supervisor.
ASYM	specifies that the punch or printer output must be routed to the cooperative. If the cooperative feature is not included in the resident supervisor, the allocation request will be rejected and the job terminated.
if blank or omitted	must be blank if positional parameter 1 is a logical unit name rather than a logical unit number.

Positional Parameter 3:

STEP	specifies that the device and its alternate (if any) are to be assigned to the job only for the duration of the job step.
if omitted	the device is assigned for the duration of the job.

Positional Parameter 4:

uuu	a 3-character device identification code assigned when the system is installed. This positional parameter is used to request the assignment of a specific device. When a specific device is requested, positional parameter 2 must be blank. If the specific device cannot be assigned, the DVC statement is not processed. This parameter is intended primarily for the convenience of field engineers.
if omitted	the device is allocated on the basis of device type only, as specified by positional parameter 1.

Positional Parameter 5:

OP	indicates that the requested device is not essential to the running of the job. The device and its alternate (if requested) are assigned if available. If they are not available, the DVC statement is ignored.
if omitted	indicates that the requested devices are essential to the running of the job.

See 3.3.1. for additional information regarding the use of the positional parameters for this statement.

Examples:

LABEL 1	ƀ OPERATION ƀ 10	16 OPERAND ƀ
// DVC	IPT	
// DVC	250	
// DVC	6, ALT	,STEP
// DVC	7, ALT	,,,OP

4.4.5. EQU Statement

Function:

The EQU statement is used to equate logical unit numbers to the specific code for a device type. This statement affects only the logical unit table associated with the job; the master logical unit table is not altered by the EQU statement. The logical unit number must be equated to a device type before that logical unit number can be used within a job. If the logical unit number already has a device associated with it, the job is canceled. The EQU statement provides the ability to temporarily change the logical unit number associated with a given device type from that specified at system generation time, and permits the user to run on a system other than the one the control stream was designed to run on.

EQU statements are effective until a RESET statement is encountered in the control stream.

Format:

```
// EQU lun-1,tt[,lun-2,tt,...,lun-5,tt]
```

Positional Parameters 1, 3, 5, 7, and 9:

lun-1, lun-2,...,lun-5	the logical unit numbers to be equated to device type codes. The only valid logical unit numbers are 0 through 255. Up to five logical unit numbers can be assigned in one EQU statement.

Positional Parameters 2, 4, 6, 8, and 10:

tt	a 2-character code identifying the type of device. The codes are specified in Table 4–1.

Examples:

```
LABEL      ƀ OPERATION ƀ            OPERAND                    ƀ
1            10         16
// EQU       3,40,4,40,5,6F
// EQU       1,0F,2,10
```

Table 4–1. Device Type Codes (Part 1 of 2)

Device Type	Type Code	Device
Console typewriter	0F	Model KSR 35 Type Keyboard Incremental Printer
Card reader	10	UNIVAC 0711–05 Card Reader Subsystem (80-column, 600-cpm)
	10	UNIVAC 0716–99 Card Reader Subsystem (80-column, 1000-cpm)
	12	UNIVAC 0711–05 Card Reader Subsystem (80-column, 600-cpm) with hardware features
	12	UNIVAC 0716–99 Card Reader Subsystem (80-column, 1000-cpm) with hardware features
	14	UNIVAC 1004/1005 Card Processor Systems Card Reader
	19	UNIVAC 0711–00 Card Reader Subsystem (UNIVAC 9200 System, 80-column, 400-cpm)
	19	UNIVAC 0711–02 Card Reader Subsystem (UNIVAC 9300 System, 80-column, 600-cpm)
Card punch	20	UNIVAC 0604–99 Card Punch Subsystem (80-column, 250-cpm row punch)
	21	UNIVAC 0604–99 Card Punch Subsystem (80-column, 250-cpm row punch) with read/punch feature F08705–00
	24	UNIVAC 1004/1005 Card Processor Systems Card Punch
	29	UNIVAC 0603–04 Card Punch Subsystem (UNIVAC 9200/9300 Systems, 80-column, 75/200-cpm column punch)
	29	UNIVAC 0604–00 Card Punch Subsystem (UNIVAC 9300 System, 80-column, 200-cpm row punch)
	29	UNIVAC 0604–99 Card Punch Subsystem (UNIVAC 9300 System, 80-column, 250-cpm row punch)
Printer	30	UNIVAC 0768–00 Printer Subsystem (900/1100-lpm drum printer)
	31	UNIVAC 0768–99 Printer Subsystem (1200/1600-lpm drum printer)
	32	UNIVAC 0768–02 Printer Subsystem (840/1000/2000-lpm drum printer)
	34	UNIVAC 1004/1005 Card Processor Systems Printer
	39	UNIVAC 9200/9300 Systems Bar Printer

Table 4–1. Device Type Codes (Part 2 of 2)

Device Type	Type Code	Device
Magnetic tape	40	UNISERVO V1–C Magnetic Tape Subsystem (7-track)
	41	UNISERVO V1-C Magnetic Tape Subsystem (9-track NRZI)
	50	UNISERVO 12 Magnetic Tape Subsystem (9-track phase encoded)
	51	UNISERVO 12 Magnetic Tape Subsystem (9-track NRZI)
	52	UNISERVO 12 Magnetic Tape Subsystem (9-track) with due! density feature F0935–00
	53	UNISERVO 12 Magnetic Tape Subsystem (7-track NRZI)
	58	UNISERVO 16 Magnetic Tape Subsystem (9-track phase encoded)
	59	UNISERVO 16 Magnetic Tape Subsystem (9-track NRZI)
	5A	UNISERVO 16 Magnetic Tape Subsystem (9-track) with dual density feature F0937–00
	5B	UNISERVO 16 Magnetic Tape Subsystem (7-track NRZI)
Disc	60	UNIVAC 8411 Disc Subsystem
	61	UNIVAC 8414 Disc Subsystem
Paper tape	81	UNIVAC 0920 Paper Tape Subsystem with read only feature F1033–02
	82	UNIVAC 0920 Paper Tape Subsystem with punch only feature F1032–02
ODR	90	UNIVAC 2703 Optical Document Reader (ODR)
1004/1005	F4	First UNIVAC 1004/1005 Card Processor Systems Channel Adapter
	F6	Second UNIVAC 1004/1005 Card Processor Systems Channel Adapter
9200/9300	F9	UNIVAC 9200/9300 Systems Channel Adapter

4.4.6. EXEC Statement

Function:

The EXEC (execute) statement is a job step delimiter and is the last statement processed by job control before the execution of the program named in the statement. All control statements preceding the EXEC statement in the control stream are effective during the execution of the program named in that EXEC statement. Any statements following the EXEC statement are not processed until the program terminates.

Format:

$$// \textbf{ EXEC} \quad \text{program-name,}\left[\begin{Bmatrix}\text{library-name}\\ \textbf{EX}\\ \textbf{MCL}\end{Bmatrix}\right][\text{,filename}][\textbf{,REL}]$$

Positional Parameter 1:

program-name	up to an 8-character name identifying the program to be executed in this job step. The program name may be in the form nnnnnnpp (nnnnnn is the name of the program and pp is the phase number). The program is retrieved from the program library identified by positional parameters 2 and 3 and is relocated to the main storage area allocated to the job. In a multistep job where the program does not exist at the beginning of the job, the MIN parameter must be specified on the JOB statement.

Positional Parameter 2:

library-name	1 to 44 characters identifying the program library on disc within which the program identified by positional parameter 1 will be found. Embedded blanks are not permitted in the library name. This specification is not used in a tape-oriented system. In a multistep job where the library does not exist at the beginning of the job, the MIN parameter must be specified on the JOB statement.
EX	indicates that the program resides in the absolute execution area in absolute form. Before this parameter can be used, the referenced program must have been copied from a tape or disc load library and formatted in absolute form by the tape/disc mapping utility program.
MCL	specifies to load the program identified by positional parameter 1 from the module complex library. However, the program only remains available in the module complex library for the duration of the job. The normal use of this parameter is to do an assembly, or compilation, linkage editor, and execute from MCL.

NOTE:

When loading from EX and MCL, the JOB statement (4.4.10) must specify positional parameter 4 (minimum main storage).

Positional Parameter 3:

filename	the 1- to 8-character file name identical to the file name specified in a previously defined LFD statement. This file contains the program library identified by positional parameter 2. This parameter enables the user to load a program from a device other than the system resident volume. When loading a program from a device other than the system resident volume, the JOB statement (4.4.10) must specify positional parameter 4 (minimum main storage). (See 3.2.6.)

Positional Parameter 4:

REL	indicates that all program overlays are to be retrieved from the program library and relocated each time a program overlay is called.
if omitted	indicates that program phases are made absolute. If this feature is to be used, programs must be mapped into the absolute execution area (see *UNIVAC 9400 System Disc Mapping Program Programmer Reference, UP-7833* (current version)), and positional parameter 2 must be specified as EX.

Examples:

	LABEL 1	OPERATION 10	OPERAND 16
1.	// EXEC	UTPREP	00
	// EXEC	UTPREP	,EX
	// EXEC	UTPREP	,LOADLIBRARY,,REL
2	// EXEC	UTPREP	,,LOAD
3	// EXEC	PAYROL	02,PAYMAST,LOADFILE,REL

1. For tape-oriented system only. Program will be loaded from tape SYSRES.

2. Program will be loaded from an alternate tape file.

3. Program will be loaded from an alternate disc file.

4.4.7. EXT Statement

Function:

The EXT (extent) statement is used to provide information to establish new files or extend existing files on direct access storage devices. This statement is not required for unit record files or tape files.

All extent information for a given volume must be contained on one EXT statement and its associated continuation statements, thus creating a single extent request block.

A maximum of two extent specifications can be written in a single EXT statement. To specify any additional extents within a given volume, the EXT statement must contain a nonblank character in column 72 and is followed by one or more continuation statements (//n). Up to 16 extent specifications are permitted for each volume of a file. Each EXT statement in the control stream causes the creation of an extent request block which is written in the control block file.

Format:

$$// \text{ EXT } \left[\begin{Bmatrix} \text{C} \\ \text{N} \end{Bmatrix}\right] \text{[,ff],} \begin{Bmatrix} \text{addr} \\ \text{CYL} \\ \text{TRK} \end{Bmatrix} \text{,qty,...}$$

Positional Parameters 1 and 5:

C or omitted	indicates that cylinders or tracks must be assigned to a contiguous area.
N	a noncontiguous area is assigned.

Positional Parameters 2 and 6:

ff	indicates index sequential access method (ISAM) specifications. These are: 00 not ISAM 01 prime data area 02 overflow data area 04 index
if omitted	00 is assumed

Positional Parameters 3 and 7:

addr	indicates that the absolute disc address at which the file area is to begin is expressed in decimal in the form ccchh, where ccc indicates the cylinder and hh indicates the head. Leading zeros may be omitted from the cylinder address. The quantity of space specified by the next positional parameter is in units of tracks.
CYL	indicates that the quantity of space specified by the next positional parameter is expressed in units of complete cylinders.
TRK	indicates that the quantity of space specified by the next positional parameter is in units of tracks.

NOTE:

When CYL or TRK is specified, the area allocated to the file is determined by the space allocation routine. When addr is specified and the requested space beginning at that track is not available, the EXT statement cannot be processed.

Positional Parameters 4 and 8:

qty	the number of tracks or cylinders required to satisfy the request. If the preceding parameter is CYL, this quantity is in terms of cylinders; otherwise, the quantity of space is in terms of tracks.

Examples:

LABEL 1	OPERATION 10	OPERAND 16	COMMENTS	72	80
// EXT		,,1009,425,,,6109,1	8411 DISC		
// EXT		,,119,4,,,19919,1	8414 DISC		
// EXT		N,01,TRK,10,N,02,TRK,2			
// EXT		C,0,CYL,1			
// EXT		C,,CYL,1,N,,CYL,2	AN EXTENT REQUEST	X	
//1		N,,CYL,2,N,,CYL,3	REQUIRING CONTIN-	X	
//2		N,,CYL,1	UATION STATEMENTS		

4.4.8. FILE Statement

Function:

The FILE statement is used to store control streams on the resident direct access storage device. This statement must be immediately followed by a JOB statement.

The file function of job control is loaded and controlled by the control routine of job control. Therefore, in order for the file function to run, there must be adequate main storage space to contain the control routine. If there is not enough space, a message is printed at the system console indicating the reason why the file function cannot be run. If the control routine is running in response to another operator command, or as the result of an EOJ, DUMP, or CANCEL macro instruction executed in a problem program, the function in process is allowed to terminate normally; then, the file function is started.

The file function of job control can also be called by the FILE operator command.

Format:

// **FILE** [n]

Positional Parameter 1:

n	indicates the number of control streams following the FILE statement. Each control stream is identified by a JOB statement and is terminated by an end-of-job (/&) statement.
if omitted	the filing process continues until all submitted control stream are filed. The filing process is terminated when five blank cards are detected following an end-of-job (/&) statement.

Example:

LABEL 1	ƀ OPERATION ƀ 10	16 OPERAND	ƀ
// FILE	6		

4.4.9. FREE Statement

Function:

The FREE statement is used to release peripheral devices. This is performed on a job basis. Devices, including related alternate devices, released by the FREE statement are returned to the pool of unallocated system resources and are available for subsequent assignment to other jobs.

Format:

// FREE dev-1 [,dev-2,dev-3,...,dev-10]

Positional Parameters 1 to 10:

dev-1, dev-2,..., dev-10	identifies the devices to be released. The logical unit number is in the range of 0 through 255 and is used as an index into the logical unit table. The logical unit names may be IPT, LOG, LST, PCH, RDR, or RES and are used as entries into the SIB.

Example:

LABEL 1	ƀ OPERATION ƀ 10	OPERAND 16	ƀ
//	FREE	5,6,IPT,RDR	

4.4.10. JOB Statement

Function:

The JOB statement is a job delimiter and indicates the beginning of control information for a job. Included in the JOB statement is a user-specified job name that identifies the control stream for later retrieval or deletion from the job file. In addition, the JOB statement can be used to designate the priority level at which the job is to be executed, the minimum and maximum main storage limits required for efficient execution of the job, an estimated maximum amount of central processor time required to complete the job, and the storage requirements for the extent tables for this job. Note that the job name ALL is not acceptable to the verify routine; however, job names beginning with ALL are allowed:

Format:

$$\text{// JOB jobname[,priority] [,time] [,min] [,max] [,nn]}\left[\text{,JOBLOG=}\left\{\begin{array}{l}\text{filename}\\\text{blank}\end{array}\right\}\right]$$

Positional Parameter 1:

jobname	one to eight alphanumeric characters identifying the job. This name must be used to reference the control stream once the control stream has been filed.

Positional Parameter 2:

priority	identifies the priority at which the job is to be executed. See 2.3.1 for further explanation regarding the use of priorities. The three priority levels are:
	1—Message Control Program (extension of supervisor)
	2—Jobs with high input/output utilization
	3—Jobs with low input/output utilization
if omitted	the priority of 3 is assumed.

Positional Parameter 3:

time	the estimated maximum amount of central processor time (in minutes) required to complete the job. If the job uses the central processor to the extent of this time limit, the operator is notified, and the job is allotted more time or is aborted. This time limit does not include I/O time.
if omitted	standard amount of time is used as specified at system generation time.

Positional Parameter 4:

min	a hexadecimal number indicating the minimum number of main storage bytes required to execute all job steps of the job. The preamble and the extent table requirements must be included. In a DOS system, the control stream is scanned to determine whether the value is large enough to contain the longest first load module of the programs in libraries specified on the EXEC statements. If not, the larger value overrides that supplied in positional parameter 4.
if omitted	In a DOS system, the minimum amount of main storage is determined at job initiation time by scanning the control stream to find the largest first load module of the programs in libraries specified on the EXEC statements.

NOTE:

The only libraries which are searched for load modules are those which are contained on the system resident device (SYSRES). Those libraries which are on alternate load devices are not searched.

Positional Parameter 5:

max	a hexadecimal number indicating the maximum number of main storage bytes requested, but not required, to execute all job steps of the job. If the maximum amount is not available, an amount not less than that specified in positional parameter 4 is allocated.
if omitted	the maximum amount of main storage is determined at job initiation time.

Positional Parameter 6:

nn	a decimal number indicating the number of 512- or 1024-byte increments needed in main storage to contain the extent tables for this job. This storage requirement must also be included in the minimum amount of storage specified in positional parameter 4. *NOTE:* *For OS/4 used with the UNIVAC 9700 System, storage for extent tables is in 2048-byte increments.*
if omitted	no extent tables are required.

Positional Parameter 7:

JOBLOG=blank	specifies that job control messages be routed to the cooperative output device as well as the console if the cooperative feature is included in the resident supervisor.
JOBLOG=filename	specifies that job control messages be routed to the cooperative output device and that the JOBLOG file be associated with the specified print file if the cooperative feature is included in the resident supervisor.
if omitted	the cooperative is not included in the resident supervisor.

Examples:

```
    LABEL      ƀ OPERATION ƀ              OPERAND                      ƀ
1                10         16
// JOB         PAYTAX,2,,,,,JOBLOG=
// JOB         UPDATE,2,3,9000,B000,11
```

4.4.11. LBL Statement

Function:

The LBL statement is used to supply label information for files on disc and tape volumes for use by data management. The LBL statement normally follows a VOL or EXT statement, or follows a DVC statement when the VOL or EXT statements are not specified. If more than one LBL statement is necessary, it must contain a nonblank character in column 72 and must be followed by one or more continuation statements (//n). The LBL statements are required in a disc-oriented system and must immediately precede the LFD statement (4.4.12).

For additional information regarding the use of the LBL statement, see *UNIVAC 9400 System Data Management System Programmer Reference, UP-7629* (current version).

Format:

$$\text{// LBL}\quad \left\{\begin{array}{l}\textbf{file-identifier}\\ \textbf{'file-identifier'}\end{array}\right\} \left[,\left\{\begin{array}{l}\textbf{file-serial-number}\\ \textbf{VCHECK}\end{array}\right\}\right] \textbf{[,volume-sequence-number]}$$

[,expiration-date] [,creation-date] [,file-sequence-number]

[,generation-number] [,version-number]

Positional Parameter 1:

file-identifier	identifies the file by name. For tape files, up to 17 bytes may be used for the file identifier. For disc files, up to 44 bytes may be used for the file identifier. This corresponds with the file label.
'file-identifier'	identifies a file with a name that contains embedded blanks.

Positional Parameter 2:

file-serial-number	one to six alphanumeric characters identical to the volume serial number of the first volume of the file.
VCHECK	indicates to data management that the volume-serial-number/file-serial-number relationship is to be checked on input or created on output.
if omitted	tape does not contain VOL1 label, or the volume-serial-number/file-serial-number relationship does not exist.

Positional Parameter 3:

volume-sequence-number	indicates the position of the volume relative to the first volume in which the file is written.
if omitted	for a tape and disc file, the system software routines assume a value of 1.

Positional Parameter 4:

expiration-date	the expiration date of the file in the form yyddd, where yy is the year and ddd is the day of the year. May be written in the form Rdddd, where R indicates retention cycle and dddd is the number of days (1-9999) that the output file is to be saved beyond the creation date.
if omitted	the field is ignored.

Positional Parameter 5:

creation-date	the creation date of the file in the form yyddd, where yy is the year and ddd is the day of the year.
if omitted	for a tape output file, the date stored in either the job preamble or the SIB is assumed. For a disc output file, the date stored in the job preamble is assumed; for an input file, the field is ignored.

Positional Parameter 6:

file-sequence-number	the position of this file with respect to the first file within a multifile set. This applies only to tape files.
if omitted	0001 is assumed by system software routines.

Positional Parameter 7:

generation-number	uniquely identifies the edition of a file; this applies only to tape files.
if omitted	0001 is assumed by system software routines.

Positional Parameter 8:

version-number	indicates the version of a generation of a file; this applies only to tape files.
if omitted	01 is assumed by system software routines.

Examples:

```
 LABEL      ƀ OPERATION ƀ          OPERAND                              ƀ          CC
1             10         16
// LBL      DISCTAXCOMP,DISC02,0010,R365,69221
// LBL      TAPEFILE,,1122,71365,69221,2001,2010,05
// LBL      ABCD,VCHECK
```

4.4.12. LFD Statement

Function:

The LFD statement is used to link the file definition with the file information in the control stream. Each time an LFD statement is encountered in the control stream, a file control block is generated. (This block contains information determined as a result of processing DVC, VOL, EXT, LBL, and LFD statements.) At least one DVC statement and one LFD statement are required for each file to be accessed in the problem program. The VOL, EXT, and LBL statements are optional, depending on the requirements of the file definition. When only one DVC and one LFD statement are needed to complete a file definition, the LFD statement must appear immediately following the DVC statement.

Once the file control block is generated, it is written on the disc for later retrieval by the RDFCB macro instruction provided by the supervisor. The details of the RDFCB macro instruction are included in *UNIVAC 9400 System Supervisor Programmer Reference, UP-7689* (current version). When data management is used, the RDFCB macro instruction is executed during the open file function (OPEN macro instruction) and the close file function (CLOSE macro instruction).

Format:

$$\text{// LFD} \quad \begin{Bmatrix}\text{filename}\\ \text{*filename}\end{Bmatrix} \left[, \begin{Bmatrix}\text{SQ}\\ \text{DA}\\ \text{IS}\\ \text{DR}\end{Bmatrix} \right] [,n] \left[, \begin{Bmatrix}\text{NEW}\\ \text{MISM}\end{Bmatrix} \right] [,\text{ASC}]$$

Positional Parameter 1:

filename	one to eight characters identifying the logical file name of the file control block as specified by the DTF macro instruction. Data management requires a maximum of only seven characters for the file name.
*filename	an asterisk preceding the file name indicates an input only file. The operator should verify that the write enable ring has been removed from the tape reel or that the file protect ON/OFF switch has been pressed to the ON position in the disc pack. The system resident volume cannot be an input only file.

Positional Parameter 2:

SQ	indicates that the file organization is sequential.
DA	indicates that the file organization is direct access.
IS	indicates that the file organization is indexed sequential.
DR	indicates that the file organization is direct access relative.
if omitted	SQ is assumed.

Positional Parameter 3:

n	specifies the total number of extents required for that file for which main storage must be allocated. The maximum value is 128 (16 extents per volume, 8 volumes per file). The address of the first byte of the area is written in the file control block. Space acquired in this way increases the total storage requirement for a job and must be taken into consideration when allocating main storage.
if omitted	no main storage is reserved for the extent tables.

Positional Parameter 4:

NEW	indicates a new direct access file. The control stream must also include a VOL and an EXT statement in the device assignment set for disc file.
if omitted	the file is assumed to be an existing file.
MISM	specifies that mismatch (MISM) errors on the specified print file be ignored by data management.

Positional Parameter 5:

ASC	specifies that the file is an ASCII file.

Examples:

```
 LABEL      ƀ OPERATION ƀ          OPERAND                     ƀ
1             10        16
// LFD        TAPEPAY,SQ
// LFD        *INPTFILE,DR,7
// LFD        DISCFILE,DA,64
// LFD        TAPFILE,,,,ASC
```

4.4.13. MTC Statement

Function:

The MTC (magnetic tape control) statement is used to position tape volumes. It must be specified after the device assignment set for that device. The MTC statement can be used to position a data file or to pre-position a multifile tape volume. It performs the following functions: Space the tape volume forward or backward a specified number of tape marks or blocks, rewind the tape volume, or write a tape mark. Only one MTC function can be performed on a specific logical unit during any one job step and is executed at job control volume mounting time.

Format:

$$\text{// MTC logical-unit-number,}\left\{\begin{array}{l}\text{FM,nn}\\ \text{FB,nn}\\ \text{BM,nn}\\ \text{BB,nn}\\ \text{WM,nn}\\ \text{RL}\\ \text{RU}\end{array}\right\}$$

Positional Parameter 1:

logical-unit-number	the logical unit number that identifies the magnetic tape device to be positioned. Only tape units that have been allocated and on which tape volumes have been mounted can be referenced by MTC statements.

Positional Parameter 2:

FM	space the volume forward a specified number of tape marks.
FB	space the volume forward a specified number of blocks.
BM	space the volume backward a specified number of tape marks.
BB	space the volume backward a specified number of blocks.
WM	write the specified number of tape marks.
RL	rewind the volume to load point.
RU	rewind the volume and unload.

Positional Parameter 3:

nn	specifies the number of blocks or tape marks that the volume is to be spaced forward or backward or the number of tape marks that are to be written.

Examples:

LABEL ƀ (1)	OPERATION ƀ (10)	OPERAND (16) ƀ
// MTC	7,FB,5	
// MTC	5,RU	

4.4.14. n Statement (Continuation)

Function:

The n statement is used to continue the information from the preceding control statement. The only control statements which may be followed by continuation statements are the LBL, VOL, or EXT statements and JPROC call. Up to nine continuation statements can follow a single control statement. A nonblank character must appear in column 72 of each continuation statement except the last.

Format:

```
//n  p1,...,px
```

where:

n	is a decimal number from 1 through 9. The numbers do not need to be consecutive; however, each successive number should be greater than the preceding number.
$p_1,...,p_x$	are the positional parameters required to continue the immediately preceding VOL, EXT, JPOC call, or LBL statement.

Examples:

```
LABEL    OPERATION   OPERAND                    COMMENTS                         72      80
1         10     16
//5 C,0,CYL,1,,,TRK,50          CONTINUED EXT STATEMENTS                   X
//6 ,,TRK,125


//2 ESP111,ESP112,ESP113        A CONTINUED VOL STATEMENT
```

4.4.15. OPR Statement

Function:

The OPR statement is used to display a message at the console and, optionally, to cause a delay between two job steps of a job. This statement can appear anywhere in the control stream and is effective when encountered. A READY operator command is used to reactivate a job that has been made inactive by an OPR statement.

Format:

// OPR [*] comment-line

Positional Parameter 1:

*	used to cause a suspension in control stream processing until the operator responds with a READY operator command. When used, the asterisk must appear as the first nonblank character following OPR. Blank characters between the asterisk and the first message character cause spacing on the console printer. These messages are displayed at the system console prefixed with the letter R.
no asterisk	processing continues immediately following the display of the message.
comment-line	up to 60 characters to be printed at the system console when the OPR statement is encountered in the control stream. If no asterisk is used, 61 characters may be printed.

Example:

```
// OPR *MOUNT INVOICE FORMS ON HIGH SPEED PRINTER
```

4.4.16. OPTION Statement

Function:

The OPTION statement allows the user to specify the optional software features ALTER, ALTER-NO, SYSDUMP, SYMBIONT, NODUMP, MULTIFIL, NOREADY, RETRY, MAYIDUMP, SCR, MCL, and NOVOL. It also allows the user to specify the operating environment of this program by means of the DOF, BOF, and NOWP options.

Up to eight options may be specified on one OPTION statement. Options are only effective in the job step in which they are specified.

Format:

```
// OPTION p1,...,p8
```

where:

$p_1,...,p_8$	specifies the options to be used. The appropriate parameters as required by the job may be written in any order and must be separated by commas, with no intervening spaces. The options are explained in the following paragraphs.

Options:

ALTER	specifies that the alter function is to be called when the problem program is loaded. The ALTER option cannot be specified if the program is to be loaded from the absolute program area.
ALTER-NO	specifies that the alter function is to be called when the problem program is loaded but the ALTER statements are not to be logged.
SYMBIONT	designates that this job is a symbiont. If specified, this statement must be present in the control stream before any device assignment statements.
SYSDUMP	specifies that a system dump is to be taken instead of a job dump, if a dump is initiated in behalf of this job.
BOF	specifies that the problem program is to be given control with binary overflow enabled.
DOF	specifies that the problem program is to be given control with decimal overflow enabled.
NOWP	specifies that the problem program is to be given control with write protection disabled, if the supervisor was generated to allow this option.
NOVOL	specifies that the VOL1 headers are not to be read and verified on the disc or tape volumes. SCRTCH volumes are not to be read nor are their numbers displayed on the system console or written in the PUB's. *NOTE:* *For OS/4 used with the UNIVAC 9700 System, no volume serial number checking is possible for IPL tapes created by the tape librarian. Any job which is processing IPL tapes must have this option.*
NODUMP	specifies that all output from the SNAP, DUMP, CANCEL, CDIAG, and DMEAR functions initiated on behalf of a job is to be supressed. However, the error PSW and the status or error code are printed on the console even though NODUMP is specified.
MULTIFIL	must be specified for all job steps that may route multiple output files within a step to the cooperative, including printer/printer, printer/punch, and punch/punch file combinations. The option alerts the cooperative to differentiate files on the basis of their PIOCBs. If the option is not used, all files within a step will be processed as a single file.
NOREADY	specifies that the job control volume mounting message (JC02/JC03) sequence not be displayed unless volume assignments have changed. Volume checking will be performed unless OPTION NOVOL is specified.

RETRY	specifies that supervisor I/O error messages will be displayed on the console for problem programs which accept unrecoverable I/O errors if the supervisor is generated to allow this option.
MAYIDUMP	specifies that any systems routine other than the cooperative that routes output to the systems LST device will query the operator to determine if the output should be initiated. If OPTION NODUMP is specified for the same jobstep it will override the MAYIDUMP option. The operator query will take place on all requests by the SNAP, DUMP, CDIAG, DMEAR, and CANCEL macros and on all main storage dumps initiated by the system. OPTION MAYIDUMP will not be effective if the DUMP, ANALYZE, or CANCEL function is initiated from the console.
SCR,n	specifies that n is the multiplicative factor for the SCRATCH file allocation in SYSPOOL used by the disc processors (FORTRAN, COBOL, RPG). For further information on this option, refer to Section 5 of *UNIVAC 9400 System Software Conventions, UP-7945* (current version).
MCL,n	specifies that n is the multiplicative factor for the MCL file allocation in SYSPOOL used by the disc processors (FORTRAN, COBOL, assembler, RPG) and the disc linkage editor. For further information on this option, refer to Section 5 of *UNIVAC 9400 System Software Conventions, UP-7945* (current version).

Example:

```
   LABEL      ␢ OPERATION ␢              OPERAND                     ␢
1               10       16
// OPTION     ALTER,SYMBIONT,BOF,DOF
```

4.4.17. PARAM Statement

Function:

The PARAM statements are used to introduce parameters required by the problem program at job execution time. PARAM statements are written in the form of control statements and are included in the control stream for the job. However, job control does not process these statements; it is the responsibility of the user to supply the correct operands. The PARAM statement can contain up to 62 characters of user-supplied information. User-submitted characters beyond column 71 are ignored. Normally, the PARAM statement or statements are grouped immediately following the EXEC statement for the job step. There is no limit to the number of PARAM statements allowed, providing the maximum size of the control stream is not exceeded.

The job step can retrieve PARAM statements from the control stream by executing the GETCS macro instruction provided by the supervisor. The details of the GETCS macro instruction are included in *UNIVAC 9400 System Supervisor Programmer Reference, UP-7689* (current version).

Format:

// **PARAM** ccc...c

where:

ccc...c	up to 62 characters in any format as required by the job step.

Examples:

LABEL	␢ OPERATION ␢	OPERAND	␢ COMMENTS
//	PARAM	X,06A00,06A04	DISC DUMP ROUTINE
//	PARAM	75,20,3,ASSEMBLY	TAPE PREPARATION ROUTINE
//	PARAM	RESUME=(CYCLE,5,798)	SORT SUBROUTINE
//	PARAM	NOCKSM=6	SORT SUBROUTINE

4.4.18. RESET Statement

Function:

The RESET statement is used only in conjunction with the EQU statement. It resets the values in the logical unit table for device types to the original active values as found in the master logical unit table. If any devices are allocated, it releases these devices back to the system on a job basis.

The use of the RESET statement in conjunction with the EQU statement allows the user to run on a system other than the one the control stream was designed to run on. No positional parameters are required for the RESET statement.

Format:

// **RESET**

Example:

//	RESET		

4.4.19. RSTRT Statement

Function:

The RSTRT statement is used to restart a program from a checkpoint. It is a function of the supervisor to establish checkpoint records in a program by means of the CHKPT macro instruction. The details of the CHKPT macro instruction are included in *UNIVAC 9400 System Supervisor Programmer Reference, UP-7689* (current version).

The RSTRT statement must appear in the control stream at a point prior to the control statements required to specify the job properly. Most of the information required to restart a program from a checkpoint is provided by checkpoint records. The control stream permits a flexibility not otherwise practical, such as the respecification of peripheral devices utilized by the job.

Format:

// **RSTRT** **logical-unit-number,serial-number [,filename]**

Positional Parameter 1:

logical-unit-number	the logical unit number of the device on which the checkpoint records are stored. This device must be assigned to the job by a DVC statement.

Positional Parameter 2:

serial-number	the checkpoint serial number printed at the system console or alternate printing device at the time the CHKPT macro instruction was executed. This number identifies which checkpoint records are to be used to restart the job.

Positional Parameter 3:

filename	specifies the name of the disc checkpoint file.
if omitted	the checkpoint file is on tape.

Example:

LABEL 1	ƀ	OPERATION ƀ 10	16	OPERAND	ƀ
// RSTRT		8,23			

4.4.20. RUN Statement

Function:

The RUN statement is a job delimiter and is used to cause the normal termination of the currently active job and the initiation of the specified job. This statement accomplishes the functions of both a RUN operator command and an end-of-job (/&) statement.

Control streams can be selected from the job file for processing by a RUN operator command or a RUN statement. The RUN operator command is introduced at the system console and can be issued at any time, providing job control is not processing another job initiation, termination, or interjob step request. The RUN statement can appear at any point prior to the end-of-job (/&) statement in the control stream.

Format:

// RUN jobname[,priority] [,GO] [,partition-length] [,preamble-address]

Positional Parameter 1:

jobname	a 1- to 8-character alphanumeric name identifying the control stream to be initiated. This name must be identical to the name submitted in the JOB statement in the control stream.

Positional Parameter 2:

priority	specifies the user priority level at which the job is to be run. This priority code overrides the one appearing in the JOB statement. There are three levels of priority: 1–Message Control Program (extension of supervisor) 2–Jobs with high input/output utilization 3–Jobs with low input/output utilization
if omitted	the prioirty level appearing in the JOB statement is used.

Positional Parameter 3:

GO	causes the initiated job to be started immediately following the job preparation and loading sequence.
if omitted	processing does not begin until a GO operator command is entered at the system console.

Positional Parameter 4:

partition-length	specifies, in hexadecimal, the partition length (in bytes) to be allocated to the job.
if omitted	see 3.2.6.

Positional Parameter 5:

preamble-address	specifies, in hexadecimal, the location in main storage at which the job is to be run. Used for restarting a program.
if omitted	starting location is assigned by the supervisor.

Examples:

```
  LABEL      ƀ OPERATION ƀ              OPERAND                    ƀ
1              10         16
// RUN         PAYROL
// RUN         PAYTAX,,GO,A000,D000
```

4.4.21. SET Statement

Function:

The SET statement is used to set up or modify the date field, user program switch indicator, and communication region in the job preamble. The SET statement cannot be used to alter fields in the SIB; this must be done by the SET operator command.

Format:

$$
\text{// SET}\left\{\begin{array}{l}\textbf{DATE,xx/xx/xx[,yyddd] [,yyddd]}\\ \textbf{UPSI,switch-setting}\\ \textbf{COMREG,character-string [,ASC]}\end{array}\right\}
$$

Positional Parameter 1:

DATE	indicates that positional parameter 2 and optionally, parameters 3 and/or 4, are to be stored in the date fields of the job preamble.
UPSI	indicates that the user program switch indicator in the job preamble is to be set according to the pattern of positional parameter 2. (The UPSI byte is the last byte of the 12-byte communication region in the job preamble.)
COMREG	indicates that positional parameter 2 is to be stored in the communication region of the job preamble.

Positional Parameter 2:

xx/xx/xx	calender date: month, day, and year, in any order.
switch-setting	one to eight characters in length. The only allowable characters are:
	0 – Set bit to off.
	1 – Set bit to on.
	X – Bit is to remain unchanged.
	Unspecified rightmost positions are assumed to be X.
character-string	1 to 24 hexadecimal characters or 1 to 12 EBCDIC characters are to be stored in the communication region of the job preamble. Unspecified rightmost characters remain unchanged. Hexadecimal characters are designated X'ccccc...'; EBCDIC or ASCII characters are designated C'ccccc...'.

NOTE:

The last byte (12th) of the communication region is the user program switch indicator.

Positional Parameter 3:

yyddd	1- to 5-character date for a tape file where yy designates the year and ddd designates the day of the year. This date is right-justified in a 6-character field in the job preamble. The leftmost character is set to an EBCDIC blank. If six characters are submitted, all six characters are stored in the job preamble. This allows the user to indicate the quarters of a year.
if omitted	the date is set from the tape date stored in the SIB.
ASC	indicates that the character string stored in the communication region is in ASCII.

Positional Parameter 4:

yyddd	1- to 5-character date for a disc file, where yy designates the year and ddd designates the day of the year. The date is converted to discontinuous binary in the form 0ydd.
if omitted	the date specified in positional parameter 3 is used. If positional parameter 3 is not specified, the date is set from the disc date stored in the SIB. If both positional parameters 3 and 4 are not specified when the job is initialized, the date fields are set to the date in the SIB unless there is a SET DATE statement in the control stream on which parameters 3 and 4 are specified. If UPSI and COMREG are not specified, the fields are cleared to binary 0's.

Examples:

```
    LABEL       ␢ OPERATION ␢                 OPERAND                         ␢
1                 10          16
// SET           DATE,11/20/69,,69324
// SET           DATE,10/31/69,69304,69304
// SET           UPSI,0011XX10
// SET           COMREG,X'FF00001124'
// SET           COMREG,C'ABC123',ASC
```

4.4.22. SKIP Statement

Function:

The SKIP statement is used to bypass, conditionally or unconditionally, a stated number of control statements, advance to the first control statement beyond the next EXEC statement, or to advance to the first control statement beyond a specified EXEC statement. The skip function terminates following the successful advance of the control stream, or upon the detection of end-of-job (/&) statement, whichever occurs first. PARAM, ALTER, /&, data, and /* statements are not counted by the SKIP statement processing. When advancing to a specified job step, the PARAM statements and data files that are associated with the specified EXEC statement are bypassed.

Format:

$$// \text{ SKIP } \left[\begin{Bmatrix}\text{program-name}\\ \text{n}\end{Bmatrix}\right] [\text{,mask}]$$

Positional Parameter 1:

program-name	an 8-character program name in the form nnnnnnpp which is the same as the program name in an EXEC statement in the control stream and serves to identify the EXEC statement. The control stream is advanced to the first statement following the EXEC statement so identified, and processing of the control stream continues at that point.
n	indicates the number of control statements to be skipped. This may be any number up to the maximum number of statements within a given control stream.

Positional Parameter 2:

if omitted	the control stream is advanced to the first statement following the next EXEC statement encountered in the control stream, and processing of the control stream continues at that point.
mask	one to eight characters identifying the bits of the user program switch indicator (UPSI) to be tested. This parameter makes the SKIP statement conditional. Effectively, a test-under-mask instruction is executed. If any of the specified bits are set, the skip condition is satisfied and the SKIP statement is processed. Otherwise, the SKIP statement is ignored and the processing continues with the next statement in the control stream. If fewer than eight characters are specified, the unspecified rightmost positions are assumed to be 0. The only allowable characters in the mask are: 0 – ignore the bit. 1 – test the bit to determine if it is set.
if omitted	the SKIP statement is unconditional.

Examples:

LABEL 1	␢ OPERATION ␢ 10	16 OPERAND ␢
// SKIP	PAYROL02,01100011	
// SKIP	20	
// SKIP		

4.4.23. VOL Statement

Function:

Two formats are available for the VOL statement.

The first VOL statement format supplies the volume serial numbers for data and program volumes to be accessed by the job. They are used to uniquely identify tape reels and disc packs. For tape volumes, the user can indicate whether or not standard block counts are to be checked on input or are to be written on output. Other characteristics of the volume, such as tape density or mode, are also specified on the VOL statement. For additional information regarding the use of volume serial numbers, see *UNIVAC 9400 System Data Management System Programmer Reference, UP–7629* (current version).

If used, the VOL statement must immediately follow the DVC statement. For a disc system, the first VOL statement must precede any EXT statement for that volume. If more than eight volume serial numbers are to be listed, a nonblank character must appear in column 72 of the VOL statement and one or more continuation statements (//n) follow.

A second VOL statement format is provided to specify parameters which define a forms loop for a printer file that is to be processed by the cooperative.

Format 1:

```
        ⎧ C      ⎫
        ⎪ Mcc    ⎪ ⎡ ⎧volno-1 or 2⎫⎤
// VOL  ⎨ CMcc   ⎬ ⎢,⎨            ⎬⎥ [,volno-2 or 3 through 7 or 8]
        ⎪ volno-1⎪ ⎣ ⎩SCRTCH      ⎭⎦
        ⎩ SCRTCH ⎭
```

Positional Parameter 1:

C	only used for volumes in a tape system. If it is an input volume, block counts are expected in each block and are checked as the volume is advanced. If it is an output volume, all blocks are sequentially numbered.
if C is not used	the block option is not effective. However, blade numbers may be present if specified for data management use.
Mcc	specifies the mode setting for tape devices and certain card reader features. The characters, cc, are hexadecimal characters equivalent to the appropriate mode setting command. Table 4–2 lists the mode settings for UNISERVO VI-C, 12, and 16 tape units, and card reader features.
if M is not used	mode settings specified at system generation time are used.

The allowable values for cc are given in Table 4–2.

Table 4–2. Mode Settings (Part 1 of 2)

A. UNISERVO VI-C Magnetic Tape Subsystems

Track Type	Mode Setting	Bytes Per Inch	Parity	Convert Feature
7-track	10	200	odd	on
	20	200	even	off
	30	200	odd	off
	50	556	odd	on
	60	556	even	off
	70	556	odd	off
	90	800	odd	on
	A0	800	even	off
	B0	800	odd	off
9-track	80	800	odd	off

Table 4–2. Mode Settings (Part 2 of 2)

b. UNIVERVO 12 and 16 Magnetic Tape Subsystem

Track Type	Mode Setting	Bytes Per Inch	Parity	Translate	Convert Feature
7-track	10	200	odd	off	on
	20	200	even	off	off
	28	200	even	on	off
	30	200	odd	off	off
	38	200	odd	on	off
	50	556	odd	off	on
	60	556	even	off	off
	68	556	even	on	off
	70	556	odd	off	off
	78	556	odd	on	off
	90	800	odd	off	on
	A0	800	even	off	off
	A8	800	even	on	off
	B0	800	odd	off	off
	B8	800	odd	on	off
9-track	C8	800	odd	off	off
	C0	1600	odd	off	off

NOTE:

The mode must always be specified for tape devices with phase encoded capability.

c. Card Reader Features

Card Reader Type	Mode Setting	Feature
0716	04 08 10	1000 cpm Alternate stacker select or primary stacker full Dual translate feature (ASCII hardware translate)
0711/0716	20 40 80	66-column stub card feature 51-column stub card feature Validity check feature

NOTE:

The mode setting is an extension of device type for descriptive use, and is used by job control in assigning an appropriate device. Any combination may be utilized with 40 and 20 mutually exclusive. For example, 20 and 08 would be specified as 28.

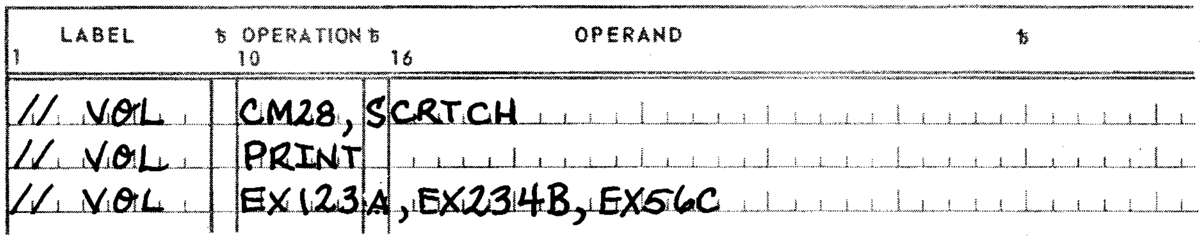
Positional Parameters 1 or 2 through 8 or 9:

volno-1,...,volno-8	each volume serial number can be from one to six alphanumeric characters in length. If fewer than six characters are specified by the user, the characters are right-justified in the field and leading zeros are added by job control. The volume serial numbers must appear in the same sequence as required for mounting the volume.
SCRTCH	a scratch volume is to be used and its volume serial number is not known.
if omitted	the volume serial number option is not effective.

NOTE:

If the block count and mode options are not specified as positional parameter 1, the first volume serial number or SCRTCH can appear as positional parameter 1.

Examples:



```
LABEL      ƀ OPERATION ƀ            OPERAND                         ƀ
1              10      16
// VOL       CM28,SCRTCH
// VOL       PRINT
// VOL       EX123A,EX234B,EX56C
```

Format 2:

// VOL FORM,name,mmmlineno=h,lineno=e,lineno =ss,[,lineno=ss,...,lineno=ss]

Positional Parameter 1:

FORM	specifies that all following parameters define a forms loop for a printer file.

Positional Parameter 2:

name	specifies a 1- to 6-character forms loop name for printer files. Specifies a 1- to 6-character card type for punch files.

Positional Parameter 3:

mmm	specifies the maximum number of lines that can be printed on the form. Required for print files only

Positional Parameter 4:

lineno=h	specifies line number and home paper code. Required for print files only.

Positional Parameter 5:

lineno=e	specifies the line number and end-of-form code (e). The end-of-form code number must be 9. Required for print files only.

Positional Parameters 6 through 10:

lineno=ss,...,lineno=ss	specifies the line number and skip codes (ss) to be used in a printer forms loop (print files only). The number specified cannot exceed the number of skip positions specified for the cooperative at supervisor generation time. Skip codes must be specified in ascending line number order. Additional parameters, if required, may be specified via a continuation statement.

Examples:

```
LABEL    ␢ OPERATION ␢              OPERAND
1          10     16                                           72       80
// VOL     FORM, SAMPLE,88,2=14,87=9,40=4                       X
//  42=5,  60=6, 75=7
```

4.4.24. Start-of-Data (/$) Statement

Function:

The start-of-data (/$) statement is used to indicate the beginning of data that is to be filed for subsequent retrieval by the job. All statements following the start-of-data (/$) statement up to and including the first end-of-data (/*) are filed in the job file. The user can access the filed data statements using the GETCS macro instruction.

The start-of-data statement must start in column 1 and must be the only statement on the card.

No positional parameters are required.

Format:

/$

Example:

```
/$         START OF DATA
```

4.4.25. End-of-Data (/*) Statement

Function:

The end-of-data (/*) statement is used to indicate the end of the data being introduced with the control stream. This statement is used in conjunction with the start-of-data (/$) statement. When the end-of-data statement is encountered by job control, the data file is closed and control stream processing is resumed. The end-of-data (/*) statement is filed with the data.

The end-of-data statement must start in column 1 and must be the only statement on the card.

No positional parameters are required.

Format:

/*

Example:

```
  LABEL      ␢ OPERATION ␢              OPERAND                  ␢
1              10         16
/*               END OF     DATA
```

4.4.26. End-of-Job (/&) Statement

Function:

The end-of-job (/&) statement is a job delimiter and is used to indicate the end of a control stream. The end-of-job statement must start in column 1 and must be the only statement on the card.

No positional parameters are required.

Format:

/&

Example:

```
/&               END OF     JOB
```

5. JOB CONTROL PROCEDURES

5.1. GENERAL

The power of job control is greatly increased by use of special directives and conditional JPROC directives which allow the programmer to specify and generate repetitive sequences of statements. To save the time and effort required to write a series of job control statements repeatedly, and to eliminate possible errors in transcription, the series can be written once in a procedure definition.

A JPROC procedure (proc) is a series of job control statements and directives starting with a PROC directive, followed by one NAME directive, and ending with an END directive. The PROC directive is used to signal the beginning of the procedure, the NAME directive declares a label by which the procedure can be called, and the END directive signals the end of the procedure. Each time the statements are needed, a procedure call statement is written. Job control inserts 0 or more statements at the point of reference.

The JPROC procedure specifies to job control the coding and statements for a particular operation, and the procedure call statement specifies the values of the variable parameters to be used when the call is executed. Job control then replaces the parameters to produce a specific section of output statements.

During the FILE function of job control, the JPROC function is called by recognizing a statement which is other than a specific job control statement such as DVC, VOL, etc. The JPROC library ($Y$JPROC) is then searched for the JPROC procedure. When the procedure is retrieved, additional statements are generated as specified by the directives and job control statements in the JPROC procedure. The generated statements are FILE'd in the normal manner. Sequence numbers are assigned to the generated job control statements by the FILE function after the JPROC routine passes the statement.

5.1.1. Statement Conventions

The JPROC procedures are entered into the JPROC procedure library via the disc librarian (*UNIVAC 9400 System Disc Librarian Programmer Reference, UP-7745* (current version). The JPROC library is created in the same manner as an ordinary proc library, via the disc librarian. This library must reside on the system resident disc and must have a file-identifier of $Y$JPROC. In order for the JPROC procedures to be entered into the library, the first slash (/) on the job control statements in the procedure must be replaced with an ampersand (&). When the JPROC procedure is expanded, and the job control statements are generated and passed to the file routine, the JPROC function reinserts the slash (/) in place of the ampersand (&). Also, if any disc librarian directives are included within data in the JPROC procedure, they must contain ampersand-blank (&b̸) in columns 1 and 2, and the JPROC must be informed to do parameter replacement within data via the REPL directive. This is necessary so that the librarian does not recognize these directives at the time the JPROC procedures are being inserted into the library. When the JPROC procedure is expanded, the ampersand in column 1 is treated as a null character.

The job control statements within the body of the JPROC procedure follow standard control statement conventions with respect to multistatement cards and continuation. If multistatement images are present in the JPROC procedure, they are expanded and passed to the file function one statement at a time. If job control statements with continuation appear in the body of the JPROC procedure, they are expanded and passed to the file function in the same form (that is, with continuation). The JPROC function does not attempt to verify whether or not continuation is allowed on the particular control statement.

The JPROC function does only a limited amount of control stream logic verification; this function is performed by the file routine on the generated statements.

5.1.2. Character Set

The character set used in writing statements and directives in the JPROC procedures consists of:

Letters	A B C through Z
Special letters	? $ # @
Digits	0 1 2 through 9
Special characters	+ – * / , = ' blank () . > < & ! : ;¬

5.1.3. Terms, Expressions, and Operators

The operand field of a statement or directive within a JPROC procedure consists of one or two operands, which must comprise either one symbol or one symbol and one basic expression.

5.1.3.1. TERMS

Terms are representations of values. Two classes of terms are recognized by the JPROC processor:

- Character String

  A character string can represent up to 44 valid characters, all of which are printable. Character strings containing embedded blanks or commas must be enclosed in apostrophes or parentheses. The enclosing apostrophes and parentheses are considered part of the character string.

  A null character string on JPROC directives is represented by two consecutive apostrophes.

  All parameter values are evaluated as character strings by the JPROC processor.

- Symbols

  A symbol is a group of up to eight alphanumeric characters used for parameter identification and as labels. The first, or leftmost, character must be alphabetic. Special characters or blanks may not be contained within a symbol, except for the # which must be the first character for positional parameters. The following are examples of valid symbols:

V	CARDAREA
GS279	R$INTRN
BOB	#13
#1	

For a symbol to be recognized by the JPROC processor as a parameter identification, it must be immediately preceded by an ampersand.

In order for parameter replacement to be effected by the JPROC processor, the parameter references (symbol preceded by an ampersand) must be preceded and followed by either a blank or a comma.

The following are not valid symbols for the reasons stated:

READ ONE	embedded blank
SPEC'L	special character
6AGN	first character not alphabetic

A symbol may be more than eight characters long; however, only the first eight characters are analyzed by the JPROC processor. If the first eight characters of any two symbols are identical, they are considered to be identical symbols regardless of following characters. The label on a NAME directive cannot exceed eight characters. The number of positional parameters supplied on the call may be obtained by referencing the symbol #0.

5.1.3.2. Basic Expressions

A basic expression comprises two terms, one of which must be a symbol, connected by an operator. Basic expressions are used to specify information for certain directives. Allowable terms of a basic expression are symbols and character strings.

5.1.3.3. Operators

The valid operators are equal (=), not equal ($\neg$), greater than (>), and less than (<). The equal operator is used to compare the value of two terms. If equal, the basic expression is true. The not equal operator is used to compare the value of two terms. If unequal, the basic expression is true. The greater than operator makes a comparison between two terms or expressions. If the value of the first (left) term is greater than the value of the second (right) term, the basic expression is true. The less than operator compares the value of the first (left) term with the second (right) term. If the value of the first term is less than the value of the second one, the basic expression is true; otherwise, it is false.

Operators must be preceded and followed by at least one blank.

5.1.4. JPROC Directive Formats

Statements and directives written for JPROC procedures are written on the UNIVAC 9000 Series coding form. Statements, directives, and comments are generally written in columns 1 through 71. Column 72 is used to indicate continuation. Columns 73 through 80 may contain program identification and sequencing information.

Although JPROC procedure directives are written in free form, it is recommended that directives be written with the first character of the operation code in column 10 and the first character of the operand field in column 16. Tabulating the directives in this fashion creates a program listing which is neater in appearance and easier to read.

The format of each field of a JPROC procedure directive is described as follows:

- Label Field

  The label field must begin in column 1 of the coding form and it is terminated by a blank column. There may not be any embedded blanks. The field may either be blank or it may contain a symbol. Labels are meaningful only on NAME directives and LABEL directives.

- Operation Field

  The operation field begins with the first nonblank after the label field and is terminated by a blank. There may not be any embedded blanks. The operation field contains a JPROC procedure directive.

- Operand Field

  The operand field begins with the first nonblank column after the operation field and it is terminated by a blank. This field may contain data and/or information used by the JPROC directives.

- Comments

  Comments may not be written on a directive or job control statement. A line may consist entirely of comments from columns 2 through 71 if column 1 contains an asterisk(*).

5.2. JPROC PROCEDURE DIRECTIVES

The JPROC procedure directives are described in the paragraphs that follow.

5.2.1. PROC – Procedure Definition

Function:

The procedure is introduced into job control by the PROC directive. This directive is used to signal the beginning of a procedure. No label may appear on the PROC directive.

Format:

LABEL	␢ OPERATION ␢	OPERAND
unused	PROC	$n\begin{bmatrix},k \\ ,k\ldots\end{bmatrix}$

where:

n	is a decimal number that represents the maximum of positional parameters that are found in the proc call line. If there are no positional parameters, 0 must be specified.
k	represents keyword parameters.

Examples:

LABEL 1	ƀ OPERATION ƀ 10	OPERAND 16	ƀ
	PROC	4,TDAM,TSPQ,A	
	PROC	0,OPT,IPT	
	PROC	1,2	

5.2.2. NAME – Call Label

Function:

The NAME directive specifies a name by which the procedure is referenced. The NAME directive must immediately follow the PROC directive.

Format:

LABEL	ƀ OPERATION ƀ	OPERAND
symbol	NAME	unused

Example:

	PROC	1
CDSN	NAME	

5.2.3. GOIF – Change Processor Source Sequence

Function:

The GOIF directive is used to conditionally alter the sequence in which the procedure directives and job control statements are processed by the JPROC processor.

Format:

LABEL	ƀ OPERATION ƀ	OPERAND
unused	GOIF	[,b]

where:

symbol	specifies the LABEL directive at which the JPROC processor should resume processing.
b	is a basic expression, which is evaluated to determine if it is true or false. If the expression is true, the LABEL directive named by symbol is the next directive to be processed by the JPROC processor. If the expression is false, then the next sequential statement or directive is processed. If the expression is omitted, the LABEL directive named by symbol is the next directive to be processed by the JPROC processor.

Rules:

1. The symbol used must be identical to the symbol in the label field of the LABEL directive:
2. A GOIF directive within a procedure may not specify a destination within another procedure.
3. A GOIF directive must specify a destination point forward in the body of the JPROC.
4. The GOIF directive must not be used between the &$ and the &* directives since everything within these boundaries is recognized as data.

Examples:

LABEL 1	ƀ OPERATION ƀ 10	OPERAND 16 ƀ
	GOIF	KAK
	⋮	
KAK	LABEL	

5.2.4. LABEL – Procedure Destination

Function:

The LABEL directive is used to identify a destination point for the GOIF directive only.

Format:

LABEL	ƀ OPERATION ƀ	OPERAND
symbol	LABEL	unused

The symbols specified in the label field of the LABEL directive are the only valid destination points for the GOIF directive.

5.2.5. DATA – Data Include

Function:

The DATA directive is used to inform the JPROC processor that a data set (/$ through /*) from the input reader stream, if present, is to be included at this point within the generated stream. The data set may be preceded by //PARAM and //ALTER statements.

Format:

LABEL	ƀ OPERATION ƀ	OPERAND
unused	DATA	unused

Rules:

1. The DATA directive normally follows the &/ EXEC control statement and, if present, includes the following from the input reader stream:

   // ALTER

   // PARAM

   /$ through /*

2. Data included from the input reader stream is not searched for parameter references.

3. Each DATA directive causes one complete group of data (preceded by a /$ and followed by a /* card) to be included from the input stream.

5.2.6. REPL – Replace

Function:

The REPL directive instructs the JPROC processor to do parameter replacement to a data set within the JPROC

Format:

LABEL	ƀ OPERATION ƀ	OPERAND
unused	REPL	unused

Rules:

1. The REPL directive normally follows the &/ EXEC control statement, or its associated &/PARAM and &/ALTER statements, and must precede the &$ control statement. The data set following the REPL directive is searched for parameter references.

2. In order for parameter references to be recognized within the data set they must be preceded and followed by a blank or comma. Exceptions to this are when the reference either starts in column 1 or ends in column 80.

5.2.7. END – Proc Defintion End

Function:

The END directive is used to signal the end of a proc.

Format:

LABEL	ƀ OPERATION ƀ	OPERAND
unused	DATA	unused

Rules:

1. The operand field should be blank.

2. The statements between PROC and END directives are defined as the body of a proc or, in other words, a proc.

Example:

LABEL (1)	OPERATION (10)	OPERAND (16)
	PROC	1
CDSN	NAME	
	⋮	
	END	

5.3. CODING PARAMETERS

In order to activate a proc, certain information must be given to it at the time it is referenced or called. Each item of information is known as a parameter, is coded in the operand field for a proc reference, and must be separated from the other parameters by commas.

5.3.1. Positional and Keyword Parameters

The basic distinction between positional parameters and keyword parameters is in the way they are identified. Positional parameters are identified by their position within the operand field of the procedure call statement. Keyword parameters are identified by the symbols which are assigned to them in the procedure call statement.

A method is allowed whereby the programmer can preset the value of a keyword. This preset value is automatically used if the particular keyword is not specified; however, the preset value can be changed or overridden by specifying a new value for the keyword on the call statement.

Positional parameters are defined in the body of the proc, and the number of positional parameters allowed in the proc is written in the definition directive. To define a positional parameter, the user must code &#n in the positional parameters of the desired job control statements, where n = the numerical position of the positional parameter.

5.3.1.1. Positional Parameters

In any call line the positional parameters must be specified before the keyword parameters. The order of the expressions in the operand determines the order of the parameters specified. Positional parameter specifications are separated by commas. When a nontrailing positional parameter specification is omitted, the comma must be retained to indicate the omission. Thus if a proc call line has four positional parameters

p_1,p_2,p_3,p_4

and the second one is not specified, the operand would appear as follows:

$p_1,,p_3,p_4$

If the third and fourth parameters are not specified, the operand is written as follows:

p_1,p_2

If only the last parameter is specified, the operand is written as follows:

$,,,p_4$

Thus, preceding or intervening parameters which are missing must be indicated by a comma. Trailing parameters which are missing need not be so indicated.

5.3.1.2. KEYWORD PARAMETERS

Keyword parameters when used with positional parameters must follow the positional parameters, if any, on the call line. Keyword parameters need not appear in any specific order. Each keyword is equated to a symbol, value, or character string. Keywords are coded on the call line as follows:

$k_1=v_1,k_2=v_2,...,k_n=v_n$

where k represents a symbol or name which is used to identify the parameter and v represents the parameter value. Keyword parameter specifications must be separated by commas and can appear in any order. Because it is identified by name and not by position, an omitted parameter does not require a comma to indicate the omission. A comma must separate the last positional parameter from the first keyword parameter when a combination of both is used.

If a PROC directive specifies three keyword parameters in the operand field

k_1,k_2,k_3

and if the call line specifies only two of the three with the second keyword parameter missing, the format is as follows:

$k_1=v_1,k_3=v_3$ or $k_3,=v_3,,k_1=v_1$

or if the call line specifies only one of the parameters, the format is:

$k_3=v_3$

If the value of the missing keyword parameters has been preset in the PROC directive, then the preset values will be used in the called procedure. If values have not been preset, then the missing keyword parameters are set to a null character string. A preset value can be changed to a null value by specifying k =, followed by at least one additional keyword specification on the call statement.

Values for keyword parameters may contain an equal sign (=) for use in supplying operands to //PARAM statements within the JPROC procedure.

Example:

JPROC call line:

```
LABEL     OPERATION     OPERAND                     COM
1         10            20        30        40      50
// EXAM   KEY1=IN=PROG/FILE
```

JPROC procedure:

```
   PROC   0,KEY1
```

EXAM name:

```
&/ EXEC    DASM,LOAD$LIB,,REL
&/ PARAM   &KEY1
   END
```

Output of JPROC:

```
// EXEC    DASM,LOAD$LIB,,REL
// PARM    IN=PROG/FILE
```

5.3.1.3. Combined Parameters

Both positional and keyword parameters can be specified in the proc call line. The following rules apply to the proc call line:

- In any call statement, the positional parameters must be specified before keyword parameters.
- In all cases all parameters are separated by commas.
- Positional parameters can be specified without keyword parameters, or keyword parameters can be specified without positional parameters, or a combination of both can be used.
- Preceding or intervening positional parameters which are missing must be indicated by a comma; trailing positional parameters which are missing need not be indicated.
- Omitted keyword parameters do not need a comma to indicate the omission.
- Keyword parameters can be specified in any order.

- An omitted keyword parameter that has been assigned a preset value receives the preset value in the procedure coding.
- An omitted keyword parameter without a preset value receives a value of a null character string.

5.3.2. Special System Defined Parameters

Special system defined parameters which allow the user to incorporate job control statements and librarian directives in the body of the proc are available for use. These parameters are:

&/	used to indicate job control statements within the proc body.
&/n	used to indicate continuation from previous card.
&$	used to indicate a start-of-data statement within the proc body.
&*	used to indicate an end-of-data statement within the proc body.
&&	used to indicate an end-of-job statement within the proc body.
&	used to indicate librarian control statements within the proc body.

When the proc is expanded, these special parameters are replaced in the resulting stream with the following values:

&/	replaced by //
&$	replaced by /$
&*	replaced by /*
&	blank or null (only when REPL directive precedes the data set)

5.4. JPROC CALL STATEMENT

Function:

The JPROC call statement provides the user with the ability to call control stream procedures previously entered in the system JPROC procedure library. Positional and keyword parameters specified by the user in this statement permit any preset values in those procedures to be overridden.

Format:

$$\text{//procname[.n]}\left[\begin{Bmatrix}\text{.L}\\\text{.O}\end{Bmatrix}\right]\quad [p_1,p_2,\ldots,p_n,k_1=v_1,\ldots,k_1=v_1,k_2=v_2,\ldots,k_n=v_n]$$

Operation Code:

procname	specifies the name of the procedure. Must be the same name as specified in the label field of the NAME directive.
n	indicates the proc group from which the procedure is to be obtained.
if omitted	proc group 1 is used.

L	indicates that the JPROC processor is to list all the generated control statements and JPROC procedure directive statements within the proc.
O	indicates that the JPROC processor is to list the call statement and the generated control statements only.

Positional Parameterers:

$p_1,p_2,\ldots,p_n$	positional parameters are specified as described in 5.3.1 and 5.3.1.1.

Keyword Parameters:

$k_1=v_1,\ldots,k_n=v_n$	keyword parameters are specified as described in 5.3.1 and 5.3.1.2.

Rules:

1. Specification of combined positional and keyword parameters are described in 5.3.1.3.

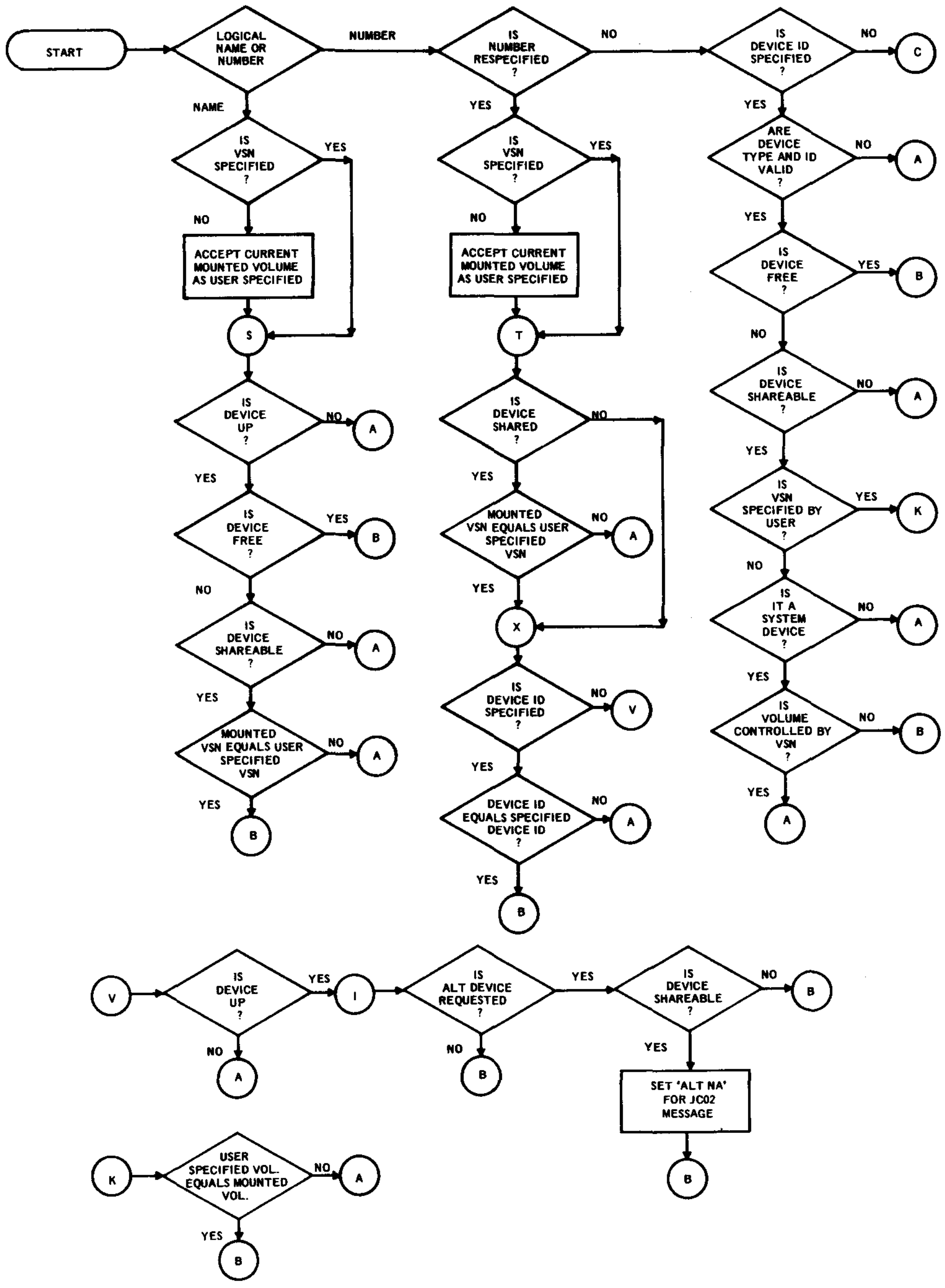
2. The JPROC call statement must be the only statement on the card, and if continuation is required, column 72 must be nonblank in every card but the last continuation card. A maximum of nine continuation cards are allowed for the JPROC call statement. Normal job control conventions concerning continuation cards must be adhered to (//n, n=1 to 9). The last parameter on a statement to be continued must be followed by a comma.

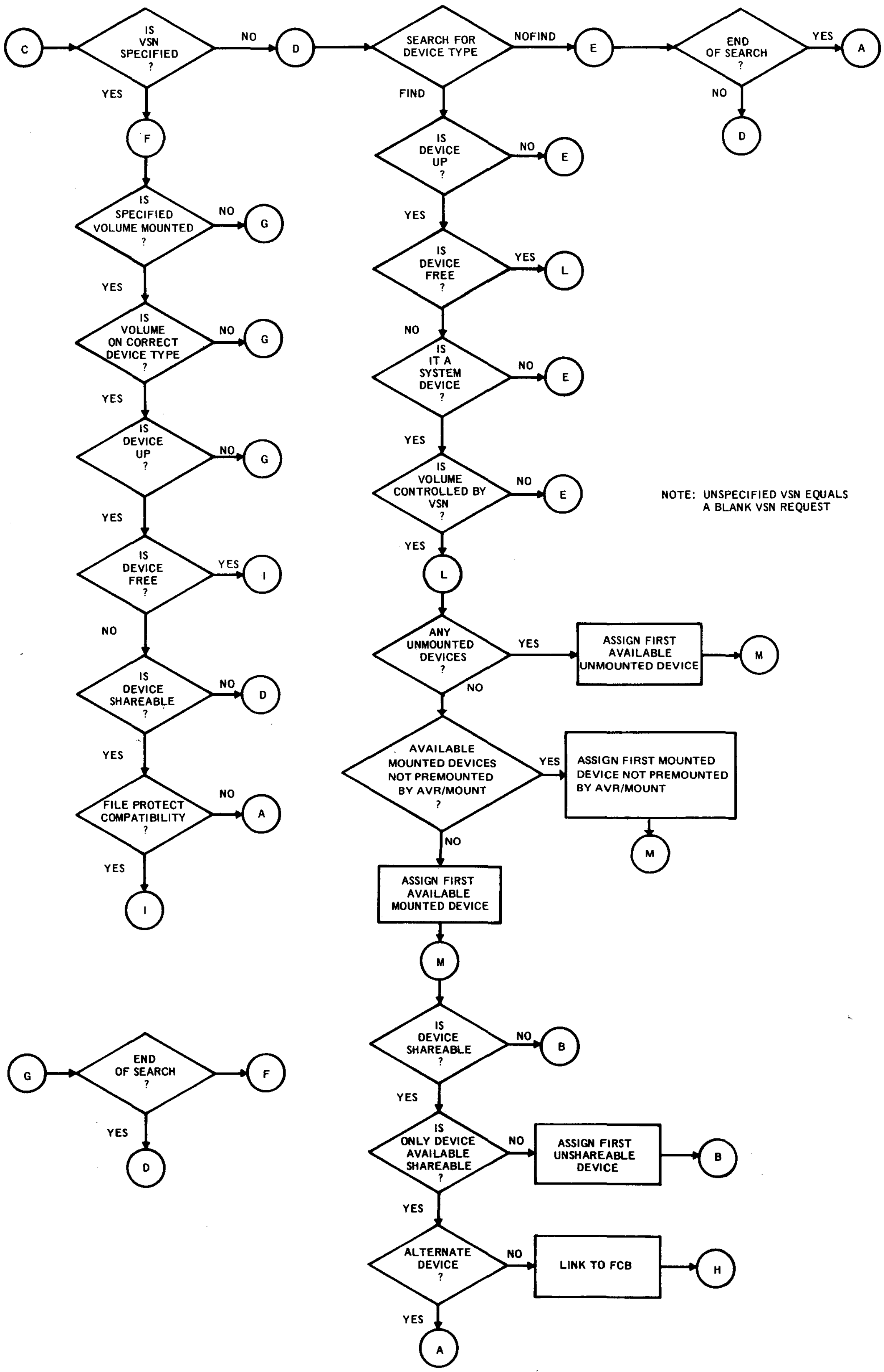
3. When the L (list) option is specified on the JPROC call statement, the call statement, all directives within the JPROC procedure, and all generated statements are listed on the system list (LST) device.

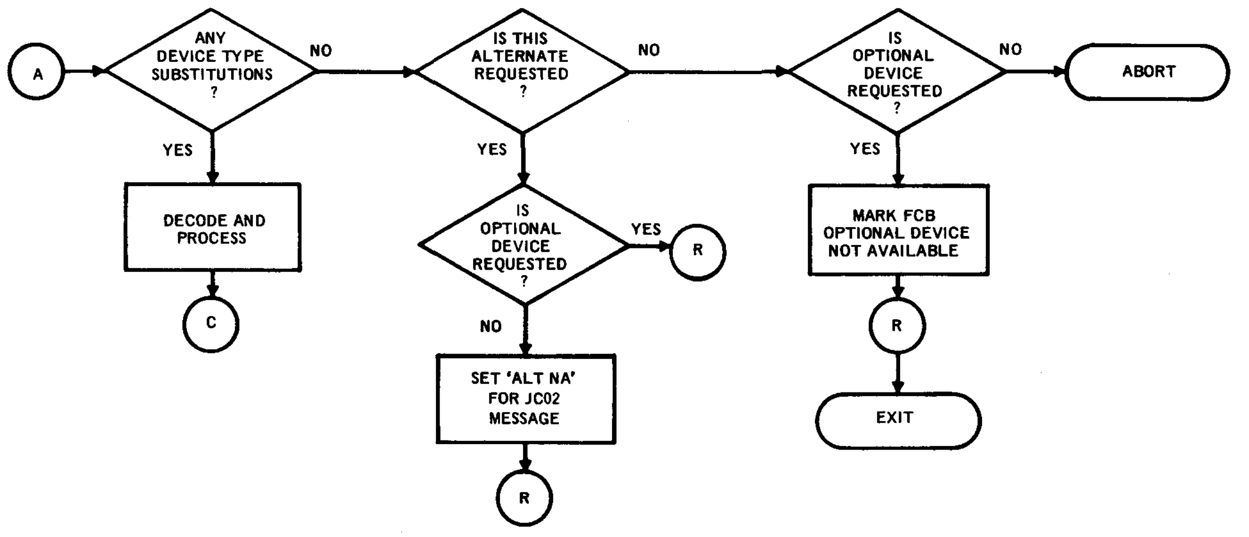
4. When the O (output) option is specified on the JPROC call statement, the call statement and all generated statements are listed on the system list (LST) device.

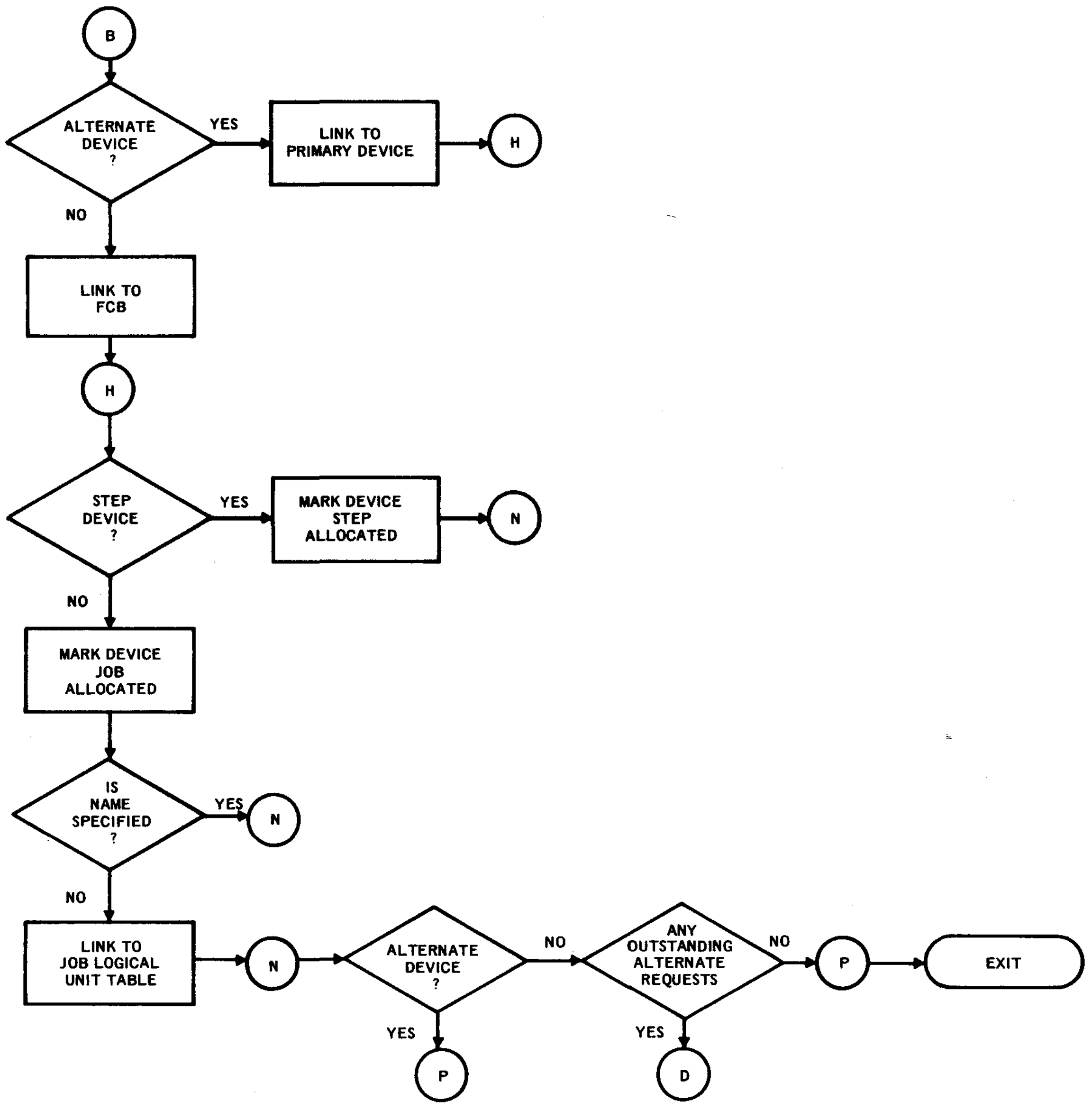
APPENDIX A. DEVICE ASSIGNMENT LOGIC

This appendix is the flow diagram of the logic used by job control in assigning devices to a job.


START
LOGICAL NAME OR NUMBER
NUMBER
NAME
IS NUMBER RESPECIFIED ?
IS DEVICE ID SPECIFIED ?
IS VSN SPECIFIED ?
ACCEPT CURRENT MOUNTED VOLUME AS USER SPECIFIED
ARE DEVICE TYPE AND ID VALID ?
IS DEVICE FREE ?
IS DEVICE UP ?
IS DEVICE SHARED ?
IS DEVICE SHAREABLE ?
MOUNTED VSN EQUALS USER SPECIFIED VSN
IS VSN SPECIFIED BY USER ?
IS IT A SYSTEM DEVICE ?
IS VOLUME CONTROLLED BY VSN ?
DEVICE ID EQUALS SPECIFIED DEVICE ID ?
IS ALT DEVICE REQUESTED ?
SET 'ALT NA' FOR JC02 MESSAGE
USER SPECIFIED VOL. EQUALS MOUNTED VOL.
YES
NO
A
B
C
K
S
T
V
X
I


C
IS VSN SPECIFIED ?
NO
YES
D
F
IS SPECIFIED VOLUME MOUNTED ?
G
IS VOLUME ON CORRECT DEVICE TYPE ?
IS DEVICE UP ?
IS DEVICE FREE ?
I
IS DEVICE SHAREABLE ?
FILE PROTECT COMPATIBILITY ?
A
SEARCH FOR DEVICE TYPE
NOFIND
FIND
E
END OF SEARCH ?
IS IT A SYSTEM DEVICE ?
IS VOLUME CONTROLLED BY VSN ?
L
ANY UNMOUNTED DEVICES ?
ASSIGN FIRST AVAILABLE UNMOUNTED DEVICE
M
AVAILABLE MOUNTED DEVICES NOT PREMOUNTED BY AVR/MOUNT ?
ASSIGN FIRST MOUNTED DEVICE NOT PREMOUNTED BY AVR/MOUNT
ASSIGN FIRST AVAILABLE MOUNTED DEVICE
B
IS ONLY DEVICE AVAILABLE SHAREABLE ?
ASSIGN FIRST UNSHAREABLE DEVICE
ALTERNATE DEVICE ?
LINK TO FCB
H
NOTE: UNSPECIFIED VSN EQUALS A BLANK VSN REQUEST


A
ANY DEVICE TYPE SUBSTITUTIONS ?
NO
YES
DECODE AND PROCESS
C
IS THIS ALTERNATE REQUESTED ?
NO
YES
IS OPTIONAL DEVICE REQUESTED ?
YES
R
NO
SET 'ALT NA' FOR JC02 MESSAGE
R
IS OPTIONAL DEVICE REQUESTED ?
NO
ABORT
YES
MARK FCB OPTIONAL DEVICE NOT AVAILABLE
R
EXIT


B
ALTERNATE DEVICE ?
YES
LINK TO PRIMARY DEVICE
H
NO
LINK TO FCB
H
STEP DEVICE ?
YES
MARK DEVICE STEP ALLOCATED
N
NO
MARK DEVICE JOB ALLOCATED
IS NAME SPECIFIED ?
YES
N
NO
LINK TO JOB LOGICAL UNIT TABLE
N
ALTERNATE DEVICE ?
NO
YES
P
ANY OUTSTANDING ALTERNATE REQUESTS
NO
YES
D
P
EXIT

APPENDIX B. CONTROL STREAM EXAMPLES

B.1. PREP OF SYSTEM POOL

The following example preps the system pool of temporary storage (SYSPOOL) area on a UNIVAC 8414 Disc Subsystem. SYSPOOL is an area on the system resident volume containing the job file, the system indexes, and information of a temporary nature. A disc is prepped in phases.

```
LABEL     b OPERATION b   OPERAND                          b        COMMENTS
1           10     16                                                          72        80
// JOB      PREP1481
// OPR PREP FOR 8414 SYSPOOL DISC, EXPD06
// OPTION NOVOL  // DVC 3  // LFD PRNTR
// DVC 8    // VOL EXPD02  // LFD SYSRES
// DVC 9    // VOL EXPD06  // LFD DISCIN
// EXEC     PREP14,LOAD$LIB,,REL
// PARAM    PHASE 0 41234 0000C7\3EXPD06SYN
// PARAM    PHASE 1 B
// PARAM    PHASE 2 E50
// PARAM    PHASE 3
// PARAM    PHASE 4 NL
/&
```

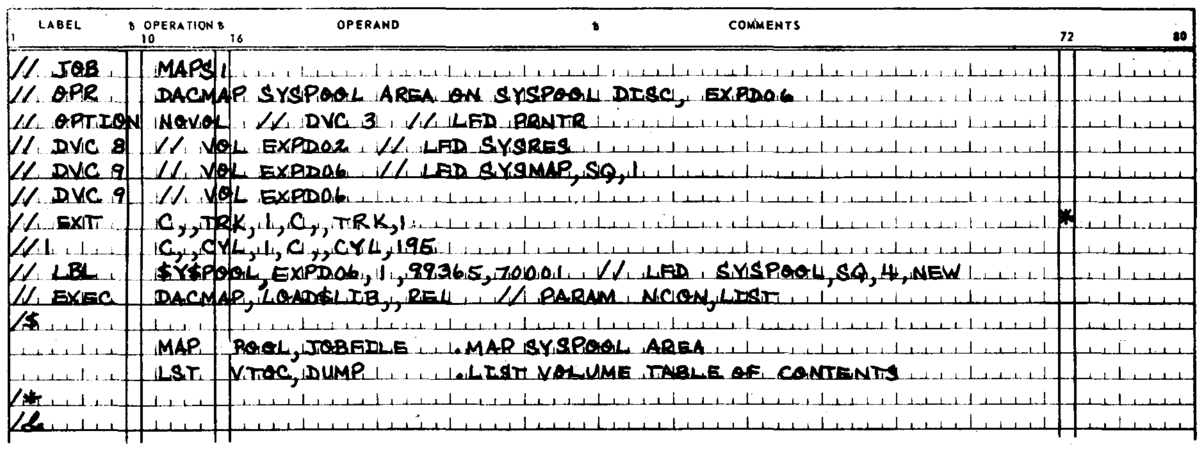
This example shows four phases, in addition to the required phase 0, which perform the following functions:

Phase 0	–	initializes the program and controls the phases to be run.
Phase 1	–	writes home address and track descriptor records on all selected tracks and then verifies them.
Phase 2	–	writes a test pattern in the R0 data field to the end of the track. It then reads back and compares the data.
Phase 3	–	rewrites home address and track descriptor records in accordance with the track condition tables. Defective tracks are exchanged for alternate tracks. All data written is verified.
Phase 4	–	prints out the condition of the tracks.

This example also illustrates the use of the OPR and PARAM control statements and the fact that more than one control statement can be written on a card.

B.2. MAPPING SYSPOOL

The following example maps the SYSPOOL area on the system resident volume. This is the same area that was prepped in Example 1.



```
LABEL      OPERATION    OPERAND                         COMMENTS
1          10     16                                                        72      80
// JOB     MAPS1
// OPR     DACMAP SYSPOOL AREA ON SYSPOOL DISC, EXPD06
// OPTION  NOVOL // DVC 3 // LFD PRNTR
// DVC 8   // VOL EXPD02 // LFD SYSRES
// DVC 9   // VOL EXPD06 // LFD SYSMAP,SQ,1
// DVC 9   // VOL EXPD06
// EXIT    C,,TRK,1,C,,TRK,1                                                *
//1        C,,CYL,1,C,,CYL,195
// LBL     $Y$POOL,EXPD06,1,99365,70001 // LFD SYSPOOL,SQ,4,NEW
// EXEC    DACMAP,LOAD$LIB,,REL // PARAM NCON,LIST
/$
           MAP    POOL,JOBFILE    .MAP SYSPOOL AREA
           LST    VTOC,DUMP       .LIST VOLUME TABLE OF CONTENTS
/*
/&
```

This example also illustrates packed control statements but points up the fact that the EXT statement must not be packed if it is followed by a continuatiốn statement. The device assignment set

//DVC 9//VOL EXPD06//LFD SYSMAP,SQ,1

is required by the disc mapping program in order to access the volume table of contents. The second device assignment set for the device with the logical unit number of 9 is used to access the SYSPOOL area on the system resident volume.

Note that the /$ (start-of-data), /* (end-of-data), and /& (end-of-job) statements start in column 1 and are not packed. The statements between the /$ and the /* statements direct the disc mapping program.

B.3. ASSEMBLE-LINK-AND-GO CONTROL STREAM

The following example illustrates a control stream containing five job steps which are to be executed consecutively. This is an assemble-link-and-go type of control stream.

```
  LABEL     OPERATION       OPERAND                                  COMMENTS
 1          10      16                                                          72      80
 // JOB      TEST01
 // DVC 3    // LFD PRNTR
 // DVC 21   // VOL DSP212  // DVC 22  // VOL DSP213  // LFD SYSPOOL
 // DVC 21   // VOL DSP212  // LBL PROC$LIB,DSP212   // LFD PROC$
 // OPTION   NOVOL
1// EXEC     DASM,LOAD$LIB,,REL
 // PARAM    LST=(O,P,B)
 /$
      .
      .      SOURCE CODE OF PROGRAM TO BE ASSEMBLED, LINKED, AND EXECUTED
      .
 /*
 // DVC 22   // VOL DSP213  // LBL RESV$LIB,DSP213   // LFD RESV$
 // OPTION   NOVOL
2// EXEC     DLINK,LOAD$LIB,,REL
 /$
    LOADM      SPCALL,X'5400'
    INCLUDE    STR,*
 /*
 // DVC 22   // VOL DSP213  // LBL USERLIB,DSP213   // LFD LIB1
 // OPTION   NOVAL
3// EXEC     LIBUPS,LOAD$LIB,,REL
 // PARAM    LIN=(1,NALT1,NALT2,NLTB)
 /$
    FILL     LIB,I,1
    ADDL     SPCALL(MCL)
 /*
 // DVC 1    // LFD CARD  // LFD PUNCH  // DVC 3  // LFD PRNTR
 // DVC 6    // LFD TAPEA
 // DVC 23   // VOL DSP177  // LBL DISCFILEONE,DSP177,1,71366,70021
 // LFD      DISC,SQ,1
 // OPTION   NOVOL
4// EXEC     SPCALL,USERLIB,LIB1,REL
 // DVC 3    // LFD PRNTR  // DVC 6  // TAPEIN
 // OPTION   NOVOL
5// EXEC     UTTPPR00,LOAD$LIB,,REL
 // PARAM    B,0,50
 /&
```

The first job step in the control stream assembles the source code of a test program called STR. The source code deck is placed between the /$ and /* statements. Because this program is being assembled for disc, at least two disc packs are required to be online. The disc assembler designates one of them as the module complex library (MCL) which is within the SYSPOOL area on the system resident volume. This assembly also requires a proc library.

The object module produced by the assembler is then processed by the linkage editor which produces the load module SPCALL. This is the second job step in the control stream. The load module is output on the MCL. As the third job step in the control stream, the disc librarian transfers the load module SPCALL from the MCL to a library USERLIB in the disc file defined as LIB1 from which it is then executed.

The fourth job step in the control stream executes the program SPCALL. Input data for the program is on cards. The output data generated by the program is punched, printed, and written on both tape and disc.

The last job step in the control stream prints from the tape file the output data generated in job step 4 on the high speed printer.

APPENDIX C. JPROC PROCEDURRE EXAMPLES

C.1. FILE ALLOCATION

The following example allocates the file $Y$JPROC, then, by using the disc librarian, places the JPROC control stream into the file $Y$JPROC.

```
LABEL       OPERATION     OPERAND
1           10      16
// JOB      CREATE
// DVC      lun     // LFD PRNTR
// DVC      lun     // VOL SYSRES
// EXT      C,,TRK,1,C,,CYL,15
// LBL      $Y$JPROC,SYSRES
// LFD      LIB1,,2,NEW
// EXEC     LIBUPS,LOAD$LIB,,REL
// PARAM    LIN=(1,NMCL,NALT1,NALT2,NLIB)
/$
   FILP     LIB1,I
   FILP1    LIB1,I,2/14
   ADDP1    jprocname-1
    :
   proc source cards
    :
   ENDCARD
   ADDP1    jprocname-n
    :
   proc source cards
    :
   ENDCARD
   DISP1    DIR
/*
/&
```

C.2. WRITING AND CALLING A PROCEDURE

Example 1:

The following three coding examples illustrate the method of writing a procedure, calling the procedure, and the control stream that results from the call.

- Writing the Procedure

	LABEL 1	ƀ OPERATION ƀ 10	OPERAND 16 ƀ
1		PROC	3,POOL=ZAP1
2	FORTC	NAME	
3	&/ DVC	20	
4	&/ LBL	NAM21	,,10
5	&/ LFD	PROC3,	&#2
6	&/ DVC	22	
7	&/ VOL	&POOL	
8	&/ LBL	NAM3,	,&#1
9	&/ LFD	POOL,	&#2
10	END		

1. The PROC definition directive shows that this proc has a maximum of three positional parameters associated with it and that the keyword parameter POOL is preset to the value ZAP1.

2. FORTC is defined to the JPROC processor as the name of the proc.

3, 4. Job control statements

5. In this job control statement, the value for positional parameter 2 (&#2) must be specified in positional parameter 2 of the JPROC call statement.

6. Another job control statement

7. In this statement, the keyword parameter name POOL is referenced preceded by an ampersand. In the expanded control stream, the value ZAP1 is inserted into the operand of this VOL statement.

8. In this statement, the value for positional parameter 3 must be specified in positional parameter 1 of the JPROC call statement.

9. Same as for line 5.

10. Declares the end of the proc.

- Calling the Procedure

```
LABEL       b OPERATION b        OPERAND                    b
1             10        16
// JOB        FOR
// FORTC      3333,DR
// EXEC
/$
:
source code
:
/*
/&
```

The procedure call line (2) specifies that the procedure named FORTC is to be used. 3333 and DR are positional parameters.

- Resultant Control Stream

```
// JOB FOR
// DVC 20
// LBL NAM21,,10
// LFD PROC3,DR
// DVC 22
// VOL ZAP1
// LBL NAM3,,3333
// LFD POOL,DR
// EXEC
/$
:
Source code
:
/*
/&
```

Example 2:

The following three coding examples illustrate the method of writing a procedure, calling the procedure, and the control stream that results from the call.

- Writing the Procedure

```
   LABEL     OPERATION   OPERAND                                        COMMEN
   1         10     16
 1           PROC   3, LIBIN,LIBOUT=SCRTCH,OBJFIL,ALTLIB
 2 LIB       NAME
 3           GOIF   AA,&LIBIN = ''
 4 &/ DVC     50,F    &/ VOL &LIBIN &/ LFD LIBIN
 5 AA        LABEL
 6           GOIF   BB,&ALTLIB = ''
 7 &/ DVC     51,F    &/ VOL &ALTLIB &/ LFD ALTLIB
 8 BB        LABEL
 9           GOIF   CC,&OBJFIL = ''
10 &/ DVC     52,F    &/ VOL &OBJFIL &/ LFD OBJFIL
11 CC        LABEL
12 &/ DVC     53,F    &/ VOL &LIBOUT &/ LFD LIBOUT
13 &/ DVC     20,SYM &/ LFD PRNTR &/ OPTION NOVOL
14 &/ EXEC   LIBS,LOAD$LIB,,REL
15    REPL
16 &$
17 &  LIB    IPL,&#1,&#2
18 &  CORS   &#3
19    INS    1
20    STDEQU
21 &  ENDCARD
22 &*
23    END
```

1. The user has specified that there are three positional parameters in the proc, and that the keyword parameter LIBOUT is preset to the value SCRTCH. Keyword parameters LIBIN, OBJFIL, and ALTLIB are not preset.

2. LIB is defined to the JPROC processor as the name of the proc.

3. Illustrates the use of the GOIF directive. AA is the label to which control is transferred if the outcome of the evaluation of the basic expression (&LIBIN = '') is true. Otherwise, control proceeds to the next sequential statement.

4. Job control statements. Note the use of the special system defined parameters (&/) which are used on the job control statements. Note also the parameter replacement specified for the volume serial number.

5. Destination point for GOIF directive in line 3.

6. Similar to the GOIF directive in line 3 except that the basic expression (&ALTLIB =") is evaluated.

7. Similar to line 4.

8. Destination point for GOIF directive in line 6.

9. Similar to the GOIF directive in line 3 except that the basic expression (&OBJFIL =") is evaluated.

10. Similar to line 4.

11. Destination point for GOIF directive in line 9.

12 through 14. Job control statements.

15. Illustrates the use of the REPL directive. This signals the JPROC processor to scan the data set that follows for parameter replacement until an end-of-data statement is encountered.

16. Start-of-data job control statement. Note the special system defined parameter (&$).

17. Illustrates the use of another system defined parameter (&) which is used on librarian directives.

18 through 21. Data statements.

22. End-of-data statement. Note the use of the system defined directive (&*).

23. Declares the end of the proc.

■ Calling the Procedure

	LABEL	␢	OPERATION ␢	OPERAND	␢	COMMENTS
	1		10	16		
1	// JOB		DOLIB			
2	// LIB.2	.	L NAL	T,NOBJ,PROGAB,LIBIN=SP3278,LIBOUT=SP0032		
3	/&					

The procedure call line (2) specifies that the proc named LIB stored in JPROC group number 2 is to be used and that the expansion of the procedure is to be listed on the SYSLST device. NALT, NOBJ, and PROGAB are positional parameters. Keyword parameter LIBIN is set to SP3278 and LIBOUT is set to SP0032.

■ Resultant Control Stream

```
  LABEL     ƀ OPERATION ƀ          OPERAND                    ƀ
1           10         16
// JOB DOLIB
// DVC 50,F
// VOL SP3278
// LFD LIBIN
// DVC 53,F
// VOL SP0032
// LFD LIBOUT
// DVC 20,SYM
// LFD PRNTR
// OPTION NOVOL
// EXEC LIBS,LOAD$LIB,,REL
/$
   LIB IPL,NALT,NOBJ
   CORS PROGAB
   INS 1
   STDEQU
   ENDCARD
/*
/&
```

Example 3:

The following coding examples illustrate the method of writing the procedure, calling the procedure with no parameter variation and the resultant stream, and calling the procedure with parameter variation and the resultant stream.

- Writing the Procedure

	LABEL 1	OPERATION 10	OPERAND 16	COMMENTS
1		PROC	1,DPAC,POOL1=DPKY,POOL2=SYPK3,PACK5=DPK4	
2	DASP1	NAME		
3	&/ DVC	20 &/	VOL &POOL1	
4	&/ DVC	21 &/	VOL &POOL2	
5	&/ LFD	SYSPOO	L	
6		GOIF	ABA, &PACK5 ¬ &POOL1	
7	&/ DVC	20 &/	VOL &PACK5 &/ LFD PROC$	
8		GOIF	A2	
9	ABA	LABEL		
10		GOIF	T6, &PACK5 ¬ &POOL2	
11	&/ DVC	21 &/	VOL &PACK5 &/ LFD PROC$	
12		GOIF	A2	
13	T6	LABEL		
14	&/ DVC	23 &/	VOL &PACK5 &/ LFD PROC$	
15	A2	LABEL		
16	&/ EXEC	DASM,L	OAD$LIB,,REL	
17		DATA		
18		GOIF	END,&#1 = ''	
19		GOIF	END,&#1 = 1	
20	&/ EXEC	DASM,L	OAD$LIB,,REL	
21		DATA		
22	END	LABEL		
23		END		

1. The user has specified that there is one positional parameter in the proc. In addition, POOL1, POOL2, and PACK5 are keyword parameters that are preset to the values DPK4, SYPK3, and DPK4, respectively. The keyword parameter DPAC is not preset.

2. DASP1 is defined to the JPROC processor as the name of the proc.

3. 4, and 5. Job control statements.

6. Illustrates the use of the GOIF directive. ABA is the label to which control is transferred if the outcome of the evaluation of the basic expression &PACK5 ¬ &POOL1 is true; otherwise, control proceeds to the next sequential statement.

7. A job control statement.

8. This GOIF statement specifies an unconditional transfer to the statement named by the symbol A2.

9. Destination point for the GOIF directive in line 6.

10. Similar to the GOIF in line 6 except that one of the keyword parameters is POOL2.

11. A job control statement.

12. After including the job control statment in line 11 in the user's expanded control stream, control is unconditionally transferred to line 15, which is the destination point for this GOIF statement.

13. Destination point for the GOIF directive in line 10.

14. A job control statement.

15. Destination point for the GOIF directives in lines 8 and 12.

16. A job control statement.

17. Includes the data from the user's control stream into the expanded stream at this point.

18. Tests the presence of positional parameter 1. If omitted, control is transferred to line 22.

19. Tests positional parameter 1 equal to 1.

20, 21. Same as lines 16 and 17.

22. Destination point for the GOIF directives in lines 18 and 19.

23. Declares the end of the proc.

- Calling the Procedure Without Parameter Variation

```
  LABEL      ␢ OPERATION ␢            OPERAND                          ␢
1                10        16
1 // JOB      ASSMBL
2 // DASP1
3 /$
4 :
5 source coding
6 :
7 /*
8 /&
```

Line 2 calls the proc named DASP1. The parameters are unchanged by the user; therefore, the preset values of the keyword parameters remain the same.

The resultant control stream is:

```
LABEL       ƀ OPERATION ƀ    OPERAND                          ƀ
1             10        16
// JOB ASSMBL
// DVC 20
// VOL DPK4
// DVC 21
// VOL SYPK3
// LFD SYSPOOL
// DVC 20
// VOL DPK4
// LFD PROC$
// EXEC DASM,LOAD$LIB,,REL
/$
:
Source code
:
/*
/&
```

Lines 3 through 5, 11, and 16 of the PROC are added to the user's original control stream.

- Calling the Procedure With Parameter Variation

```
// JOB      ASMYB
// DASP1    2,PACK5=AB123
/$
:
source code
:
/*
/&
:
Source code
:
/*
/&
```

This example illustrates the use of positional and keyword parameter variation. Two assemblies are requested. Upon encountering the proc call DASP1, PACK5 is changed to AB123 in the proc. Lines 3 to 5 of the proc are added to the user's control stream. In the proc, control passes to the label ABA because PACK5 is not equal to POOL1. Another test is made in the proc where control passes to T6 because PACK5 is not equal to POOL2. Lines 14 and 16 are added to the user's control stream. Line 17 indicates that the user's first source code is to be included. Line 18 is then included as well as the second source code.

The resultant control steam is:

```
LABEL      ƀ OPERATION ƀ            OPERAND                        ƀ
1            10         16
// JOB ASMYB
// DVC 20
// VOL DPK4
// DVC 21
// VOL SYPK3
// LFD SYSPOOL
// DVC 23
// VOL AB123
// LFD PROC$
// EXEC DASM,LOAD$LIB,,REL
/$
:
source code
:
/*
// EXEC DASM,LOAD$LIB,,REL
/$
:
source code
:
/*
/&
```

APPENDIX D. DEVICE TYPE SUBSTITUTION TABLE

This appendix gives an example of the relationship between positional parameter 2 of a DVC statement in a control stream and the corresponding entry in the device type substitution table. This table is established within the resident supervisor by the user at system generation time. The table is located directly following the logical unit table (LUT).

Control Stream	**Device Type Substitution Table**	
//JOB ASSM		
//DVC 3	DC C'A'	entry A identifier
	DC X'41'	
// LFD PRNTR	DC X'52'	
// DVC 6,A →	DC X'40'	
	DC X'00'	entry A terminator
// LFD SCR1	.	
	.	
	.	
.	DC C'M'	entry M identifier
.	DC X'61'	
.	DC X'00'	entry M terminator
	DC X'FF'	table terminator

In entry A, if the UNISERVO VI-C Magnetic Tape Unit (41) is in use and cannot be assigned, a device type 52 is assigned if available. If it is not available, a device type 40 is assigned. There may be up to 26 entries in the table. There is no limit to the number of device type specifications allowed in one entry; however, each entry must be terminated by X'00'.

INDEX

CUT

Comments concerning this manual may be made in the space provided below. Please fill in the requested information.

System: ______________________________

Manual Title: ______________________________

UP No: ____________ Revision No: ____________ Update: ____________

Name of User: ______________________________

Address of User: ______________________________

Comments:

FOLD

FIRST CLASS
PERMIT NO. 21
BLUE BELL, PA.

BUSINESS REPLY MAIL NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

UNIVAC

P.O. BOX 500

BLUE BELL, PA. 19422

ATTN: SYSTEMS PUBLICATIONS DEPT.

FOLD

CUT