IBM Data Processing Techniques

IBM System/360 Operating System User Libraries

The purpose of this document is to illustrate, by means of a program testing application, the creation, use, and maintenance of user program libraries operating under the IBM System/360 Operating System. The program testing application was chosen for illustrative purposes only and should not be construed as a workable system as it stands. The information in this text is based on information and components available at the time of the initial release of Operating System/360. The use of Assembler E, COBOL E, FORTRAN E, and Linkage Editor E is assumed. The user should therefore refer to the following texts and their most recent technical newsletters for the most complete, accurate, and up-to-date information:

Programming

IBM System/360 Operating System: Utilities (C28-6586)

IBM System/360 Operating System: System Generation (C28-6554)

IBM System/360 Operating System: Linkage Editor (C28-6538)

IBM System/360 Operating System: Job Control Language (C28-6539)

IBM System/360 Operating System: System Programmer's Guide (C28-6550)

" System control Blocks C28-6628

This text is a major revision of, and obsoletes IBM Operating System/360 User Libraries, Preliminary Edition (Y20-0008). The major changes are on pages 12, 14, 22, 32, 34, and 37.

Copies of this and other IBM publications can be obtained through IBM branch offices. Address comments concerning the contents of this publication to IBM, Technical Publications Department, 112 East Post Road, White Plains, N.Y. 10601

IBM **Technical Newsletter**

Re: Form No. C20-1663-0

This Newsletter No. N20-1010-0

Date February 15, 1967

Previous Newsletter Nos. None

IBM SYSTEM/360 OPERATING SYSTEM USER LIBRARIES

This technical newsletter amends the publication IBM System/360 Operating System User Libraries (C20-1663-0). Make the corrections and additions listed below in existing copies of the publication.

Cover and Preface Pages

Add the following to the list of reference texts on both pages:

IBM System/360 Operating System: System Control Blocks (C28-6628)

Page 12

In the drawing at top of page, change the word above the volume containing the SYS1.PROCLIB from "TESTVL" to "SYSRES".

In the program listing below this drawing, add a comma at the end of the second line of item 4: "DISP=(NEW, KEEP),". Similarly, add a comma at the end of the second line of item 5. The commas indicate continuation of the data definitions.

Page 32

In the Note at bottom of page, change the next to last line from:

require an additional parameter, that is VOLUME=SER-TESTVL.

to the following:

require two additional parameters, that is, VOLUME=SER=TESTVL and UNIT-2311.

Page 37

Add "(see C28-6628)" to the end of the next to last paragraph.

Note: File this newsletter at the back of the publication. It will provide a reference to changes, a method of determining that all amendments have been received, and a check that the publication contains the proper pages.

IBM Corporation, Technical Publications Dept., 112 E. Post Road, White Plains, N.Y. 10601

Printed in U.S.A.

PREFACE

The purpose of this document is to illustrate the creation, use, and maintenance of user program libraries operating under OS/360. To facilitate the explanation of interplay and control among the various components of OS/360, the application of program testing has been chosen as a vehicle to demonstrate:

Library creation and use
Cataloged procedure creation and use
Library maintenance
Backup procedures

The program testing application, hereafter referred to as TESTS, was chosen for illustrative purposes only and should not be construed as a workable system as it stands. The information in this manual is based on the information and components available at the time of the initial release of OS/360. The use of Assembler E, COBOL E, FORTRAN E, and Linkage Editor E is assumed.

The user should therefore refer to the following texts and their most recent technical newsletters for the most complete, accurate, and up-to-date information:

IBM System/360 Operating System Utilities (C28-6586)
IBM System/360 Operating System System Generation (C28-6554)
IBM System/360 Operating System Linkage Editor (C28-6538)
IBM System/360 Operating System Job Control Language (C28-6539)
IBM System/360 Operating System System Programmer's Guide (C28-6550)

CONTENTS

LIST OF ILLUSTRATIONS

DEFINITION OF A LIBRARY

A 'library' is a partitioned data set (PDS), which is a data set with one or more sequentially organized members, residing on and not exceeding in space one direct access volume. OS/360 libraries may be categorized as follows:

1. Libraries required by OS/360 for its operation, and residing on either the system residence volume or some other direct access volume.

2. Libraries required when using certain processors or features of OS/360 (for example, the COBOL and FORTRAN libraries), but not required for the system to function.

3. Libraries defined, organized, and named by the user to best accommodate the installation's requirements.

The libraries falling in category 1 and referred to in this document are:

SYS1.LINKLIB (Link library)
SYS1.PROCLIB (Procedure library)

Those falling in category 2 and referred to here are:

SYS1.MACLIB (Macro library)
SYS1.COBLIB (COBOL library)
SYS1.FORTLIB (FORTRAN library)

Full descriptions of these two categories may be found in C28-6554.

Libraries in category 3 may be given simple or qualified names. Since a library is nothing more than a data set, it may be created during execution of any job step by defining the library name, allocating space on a volume, etc. (see "Library Naming and Creation"). While libraries in categories 1 and 2 are neither created nor named by the user, they may be accessed and used as in category 3.

User programs may be located in the Link library. To execute such a program, the user merely specifies the name of this program in the EXEC statement.

User programs may also be located in a library created by the user (category 3). To execute a program in a user-created library, the user must define this library by inserting a data definition (DD) statement (with a ddname of JOBLIB) prior to the EXEC statement or statements requiring the use of this library. This DD statement causes OS/360 to search the identified user's library for the program to be executed before searching the Link library.

OVERALL EXAMPLE — PROGRAM TESTING — "TESTS"

In many installations the development and testing of applications consumes a great deal of effort and time. While testing systems vary from installation to installation, certain library maintenance methods should be followed to take full advantage of the computing system.

The Test Cycle

The sample testing system example, TESTS, is based on the concept of using a separate disk-pack exclusively for all program testing within an installation. The basic reason for this approach is to isolate undebugged programs and to ensure that they do not contaminate space on other packs or the system residence volume.

TESTS is further thought of as a "stacked testing" procedure in which it is desirable to perform testing at convenient intervals during a shift and thus stack all tests to be performed, mount the TESTS pack, and perform the tests required. This approach will probably be valid for most OS/360 installations, at least in the early phases of their development.

From an operations point of view, TESTS is thought of as an application in which the user may specify certain standard procedures to be performed in this test environment.

From a programmer's point of view the tests performed on his program are done on a remote basis. The programmer must request the type of test he wishes.

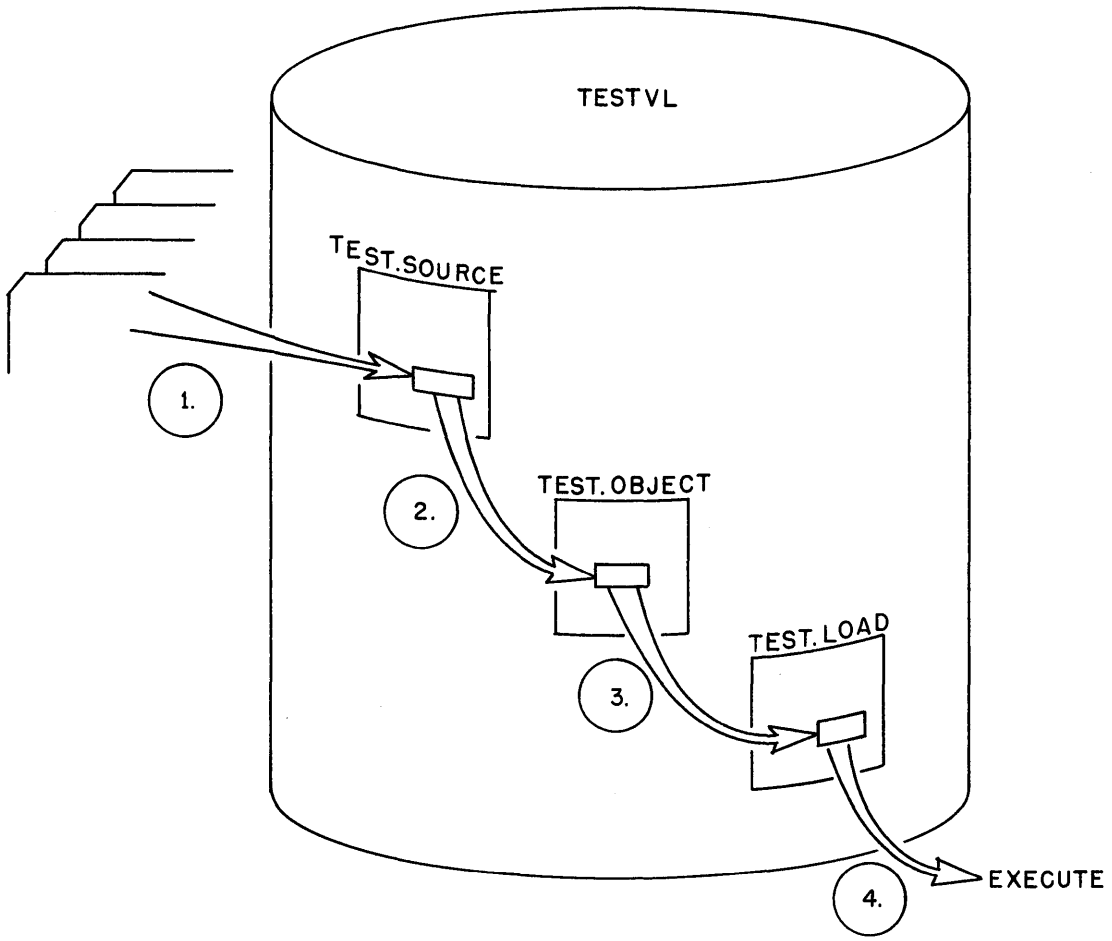
The TESTS system consists of three libraries, or PDS's. As shown in Figure 1, a library is available for each of the following:

Source modules — programs in source language
Object modules — compiled programs or subroutines
Load modules — Linkage Editor output (executable programs)

The programmer's first action is to request that his source module be entered into the source library. Once this is done, he may request one or more of the following:

1. Compilation
2. Modification of his source program
3. Linkage editing
4. Execution
5. Linkage editing to combine additional object or load modules
6. Compile — linkage edit — execute

Each of the modules (source, object, load) is retained in the appropriate library until the test cycle has been completed and the programmer wishes to remove it.



1. Source code and modifications entered
2. Compilations
3. Link-edit runs
4. Program executions

Figure 1. Libraries in TESTS system

Thus the programmer debugs his program using computer output. He then updates and retests his program by requesting the appropriate phases mentioned above, without having to continually maintain a source deck throughout the complete compilation and testing cycle.

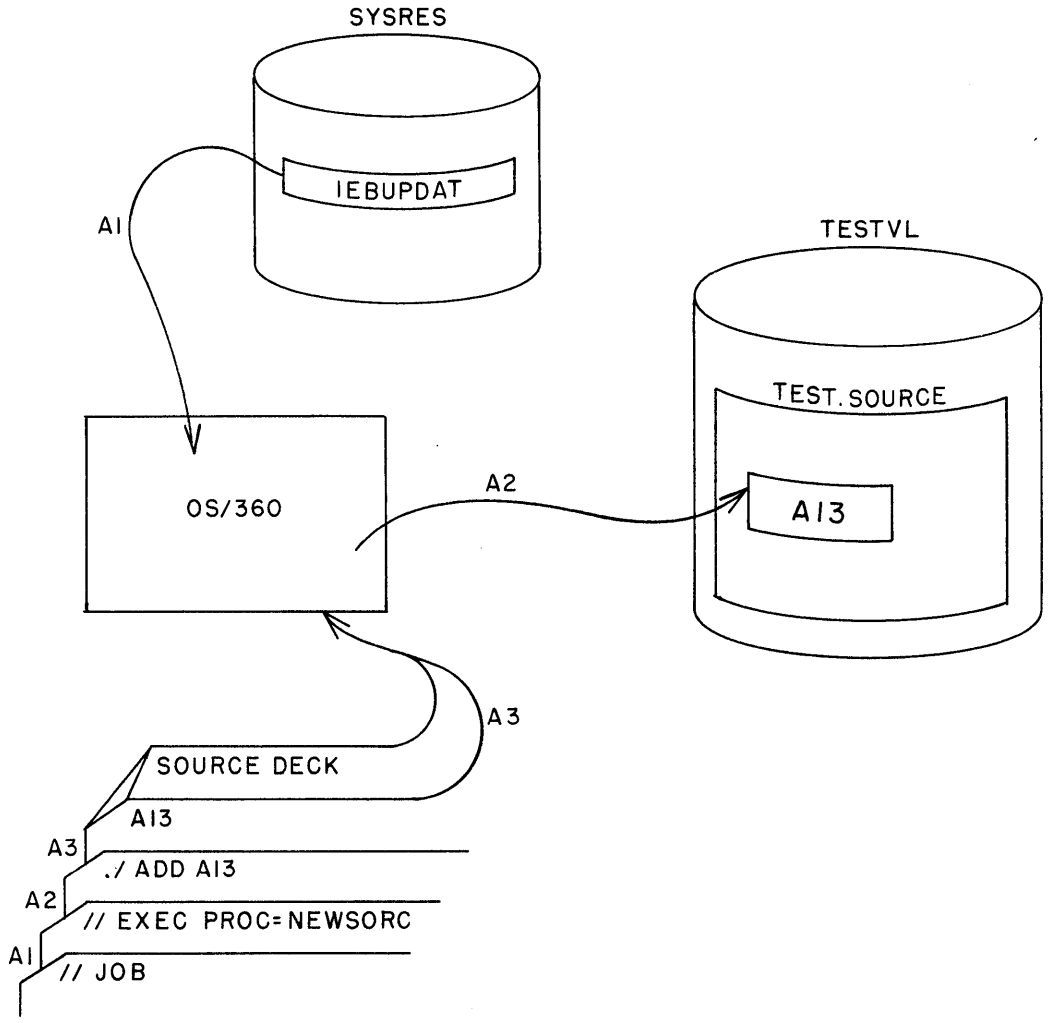
Once his program has been thoroughly tested, he may request (among other things) the updated source deck, a listing, and the deletion of his program from the three TEST libraries.

Overall Flow

To illustrate the flow of operations that occur in the TESTS application environment, two examples have been given. Both consist of a diagrammatic representation of the job stream, the processing to occur, and the libraries used, as well as a description of the function of each statement in the job stream. The statements in the job stream are illustrative rather than actual. The actual job control language and control statements are specified in the detailed illustrations of each phase of the TEST application.

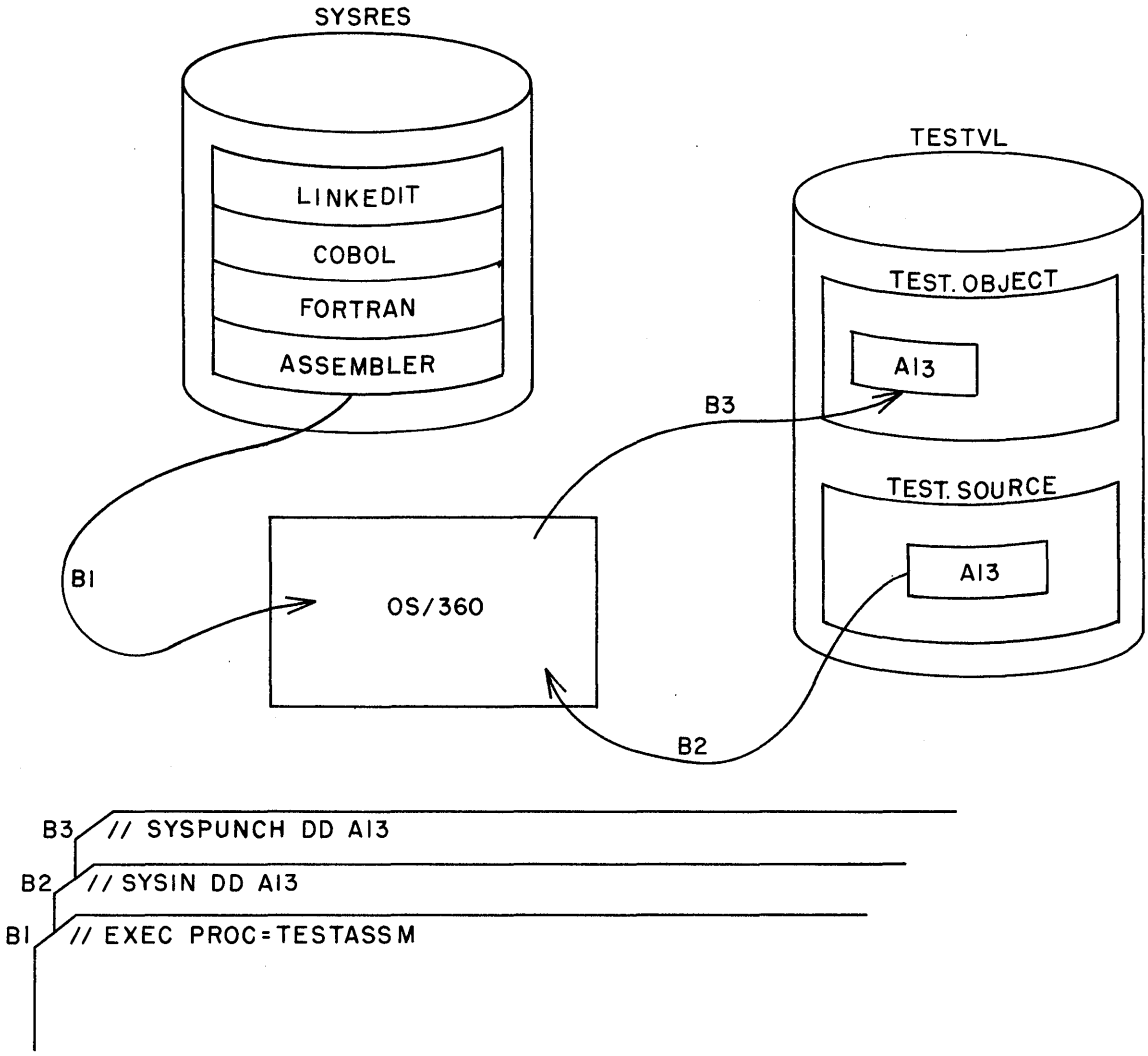
The first example (Figures 2-5) illustrates the processing of A13, a source program that is to be placed in the source module library (TEST.SOURCE), assembled into an object module library (TEST.OBJECT), link-edited into a load module library (TEST.LOAD) and executed from TEST.LOAD.

The second example (see "Multiple Job Flow in TESTS") illustrates the flow of multiple programs operating in the TESTS environment and is more meaningful as a summary of the contents of this document.



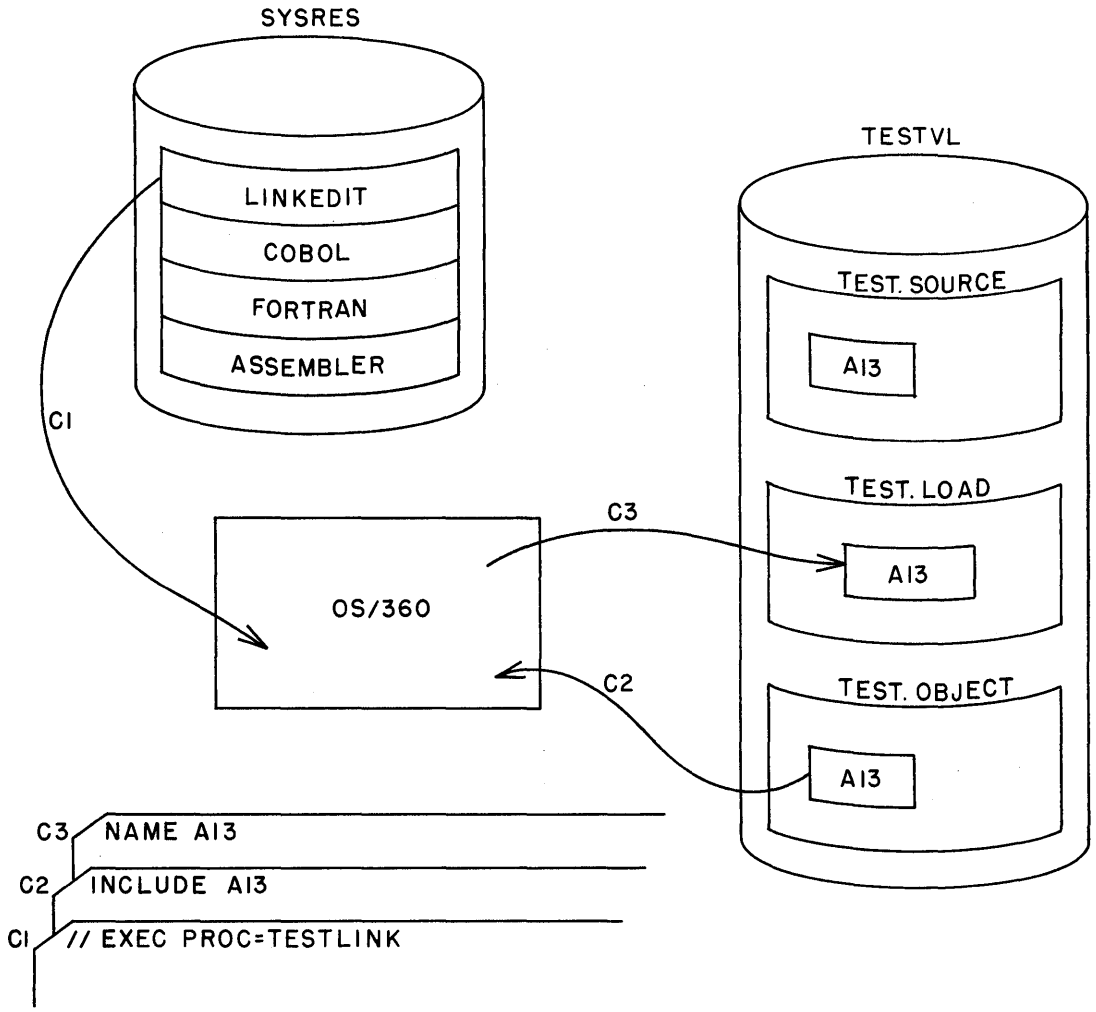
Job Stream	Processing Incurred
A1. EXEC PROC=NEWSORC	Invokes cataloged procedure to place source program on source library (see Figure 9).
A2. ADD A13	Indicates the name of the source program, A13.
A3. Source Deck	The source statements that are entered into TEST.SOURCE as A13.

Figure 2. Job flow of one program from source to execution: source program to source library



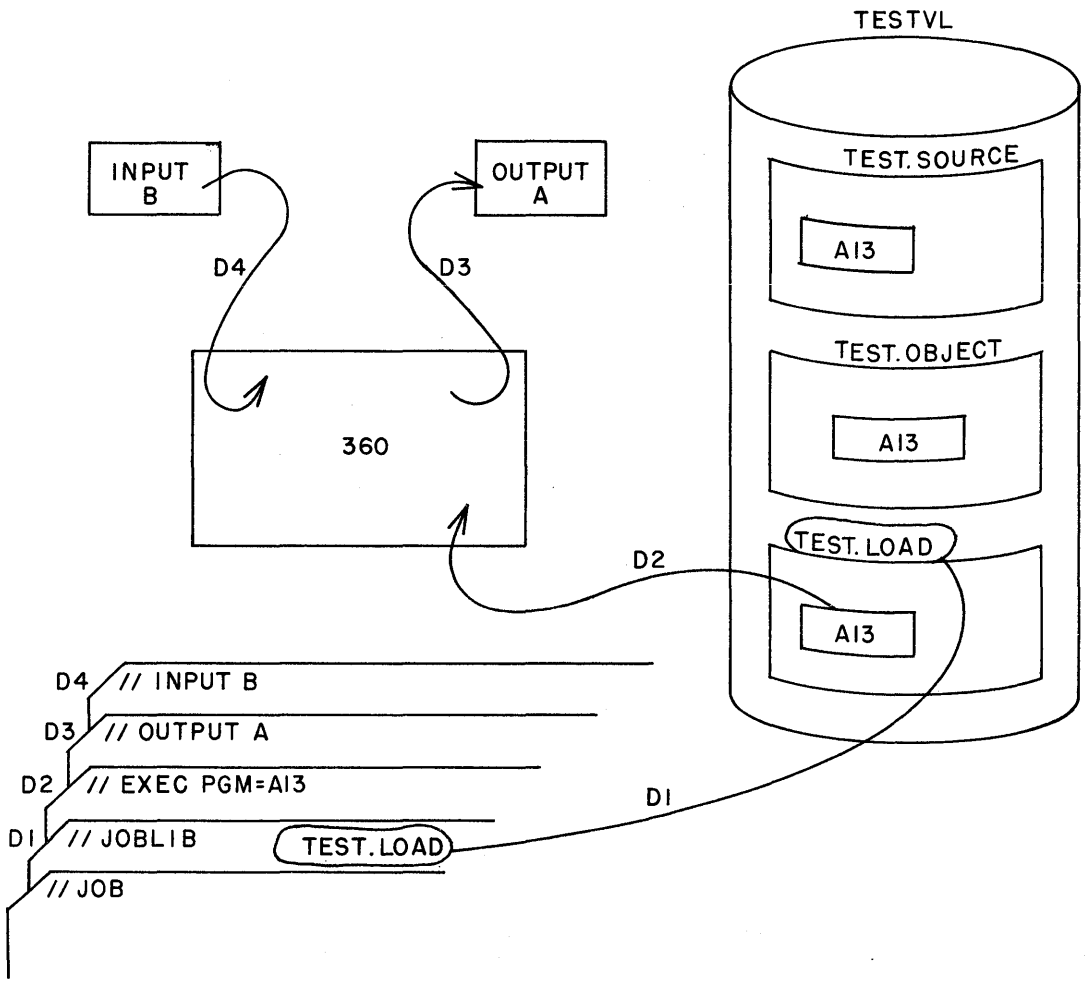
Job Stream	Processing Incurred
B1. EXEC PROC=TESTASSEM	Invokes the cataloged procedure to assemble the source module A13 (see Figure 13).
B2. SYSIN DD A13 (Input)	Represents the DD statement indicating the name of the source module to be assembled.
B3. SYSPUNCH DD A13 (Output)	Represents the DD statement indicating that the member name A13 be assigned to the output object module to go into TEST.OBJECT.

Figure 3. Job flow of one program from source to execution: source program in source library assembled into object library



Job Stream	Processing Incurred
C1. EXEC PROC=TESTLINK	Invokes the cataloged procedure to link-edit the object module (see Figure 16).
C2. INCLUDE A13	Represents the Linkage Editor statement that specifies the name of the object module to be link-edited, A13.
C3. NAME A13	Represents the Linkage Editor statement that specifies the name to be assigned to the output load module going into the load library.

Figure 4. Job flow of one program from source to execution: object module is link-edited and becomes a load module in the load library (TEST. LOAD)



Job Stream	Processing Incurred
D1. JOBLIB TEST.LOAD	Specifies to the control program that the program to be executed resides in the library named TEST.LOAD.
D2. EXEC PGM=A13	Specifies the member name, A13, of the program in the library to be executed.
*D3. OUTPUT A13	Symbolically represents one or more DD statements required to specify the devices and data sets required for the output results of A13.
*D4. INPUT A13	Symbolically represents one or more DD statements required to specify the sources of input data for A13.

*Must be specified by the programmer or set up as standard DD statements for input and output at the user's descretion. Note: It is not the intent of this document to stress test data set manipulation and control.

Figure 5. Job flow of one program from source to execution: execution of program A13

CREATION OF TESTS LIBRARIES

Two phases are required before TESTS may be put into operation: (1) initialization and (2) library naming and creation.

Volume Initialization

In this phase the Independent Utility DASDI (see C28-6586) is used to create a volume label and to allocate space for the Volume Table of Contents (VTOC). Figure 6 shows the control statements required for this phase. The volume serial number to be placed in the volume label is TESTVL.

Library Naming and Creation

Three libraries (partitioned data sets) are required in the TESTS environment (see note at end of this section). They will be named:

TEST.SOURCE — for source modules
TEST.OBJECT — for object modules
TEST.LOAD — for load modules

Although the three libraries in the TESTS example are not cataloged, each library is assigned a two-element name. The reason for this is that someone else may wish to refer to a data set called SOURCE. To avoid duplication, the second person could call his library MY.SOURCE instead of TEST.SOURCE.

It is apparent that all users of a test volume could have unique names for their libraries. However, if this were the case, each user would have to develop his own procedure or use the TESTS procedures and override certain DD cards.

The approach taken in the TESTS example is a more standardized one permitting more accurate control, easier procedure specification, and more convenient maintenance of the TESTS volume. This standardization, however, requires the programmer to name his program according to some convention.

A member (module) or program may have a name as large as eight characters. Without some type of naming convention, two independent programmers could name their programs by the same name — say, MATRIX. This would be intolerable, especially if both programs were expected to be in TESTS at the same time. Therefore, a naming convention must be established. For our example we will assign a two-digit code to individuals or departments. Thus department 23 will submit a program named MATRIX23 to TESTS. It must retain this name at least for the life of this program within TESTS.

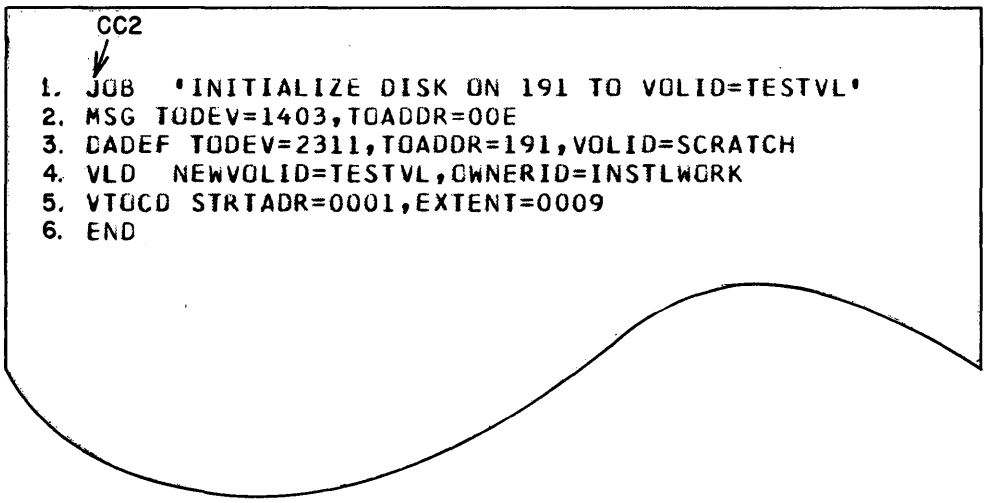
Figure 7 shows the job control statements required to allocate space for and create the three libraries as well as to enter a standard procedure, described in the following paragraphs, into SYS1.PROCLIB. It is desirable to develop standard procedures for the testing environment such as those for updating source modules, linkage editing, compiling, etc., and to place these procedures in the procedure library (SYS1.PROCLIB). This document describes several of the procedures used in TESTS.

Since the entering of a procedure into the SYS1.PROCLIB is a procedure itself, it will also be convenient to place that procedure in SYS1.PROCLIB. This procedure (called ADDPROCS) was initially put into the SYS1.PROCLIB via the UPDATE utility, as seen in Figure 7. The use of ADDPROCS for adding a procedure to SYS1.PROCLIB is illustrated in Figure 8.

It is possible to add to or delete from a procedure in SYS1.PROCLIB through the UPDATE utility program. However, since a TESTS procedure would represent relatively few cards, a change to the procedure could also be accomplished by updating the original card deck. The updated procedure deck would then be added to SYS1.PROCLIB using the ADDPROCS procedure (Figure 8). Although the original procedure has the same name as the new one, the ADDPROCS will remove the pointers in SYS1.PROCLIB to the old one and point to the new procedure.

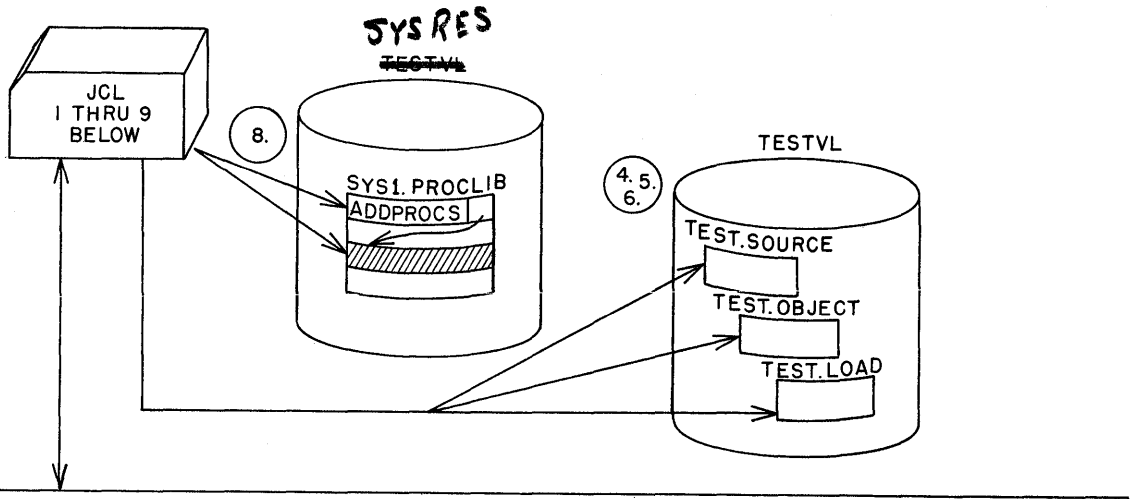
Although not illustrated, all procedures for TESTS have been entered into SYS1.PROCLIB. Further references to these TESTS procedures will assume their residence in SYS1.PROCLIB.

Note: Although not implemented in the TESTS examples, it would be advantageous to preallocate space for all utility (work) data sets — for example, SYSUT1 and SYSUT2 — at the same time that the TESTS libraries are created (see Figure 7). If this were done, space allocation for the utility (work) data sets would be avoided in subsequent procedures.



Card	Narrative
1.	JOB with comments.
2.	Messages will be printed on the printer.
3.	The specific pack to be initialized.
4.	The volume will be called TESTVL and owner is INSTLWORK.
5.	The VTOC will span nine tracks starting in cylinder 0, track 1.
6.	END card indicating end of JOB to the DASDI utility.

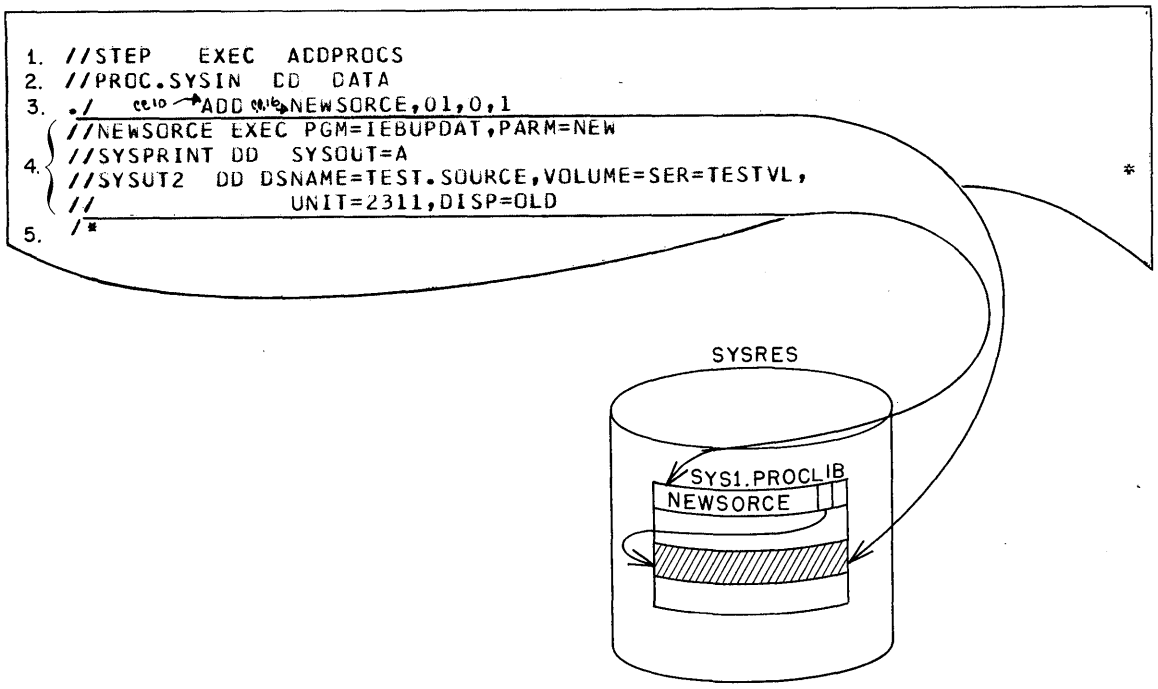
Figure 6. To initialize the TESTS volume




```
   //INITAL       JOB    007,INSTLWORK,MSGLEVEL=1
1. //        EXEC  PGM=IEBUPDAT,PARM=NEW
2. //SYSUT2 DD    DSNAME=SYS1.PROCLIB,DISP=OLD
3. //SYSPRINT DD  SYSOUT=A
4. //DD1     DD    DSNAME=TEST.SOURCE,VOLUME=SER=TESTVL,UNIT=2311,          *
   //              SPACE=(TRK,(50,10,10)),DISP=(NEW,KEEP),                  *
   //              DCB=(,RECFM=F,BLKSIZE=80)
5. //DD2     DD    DSNAME=TEST.OBJECT,VOLUME=SER=TESTVL,UNIT=2311,          *
   //              SPACE=(TRK,(400,20,10)),DISP=(NEW,KEEP),                 *
   //              DCB=(,RECFM=F,BLKSIZE=80)
6. //DD3     DD    DSNAME=TEST.LOAD,VOLUME=SER=TESTVL,UNIT=2311,            *
   //              SPACE=(TRK,400,20,10)),DISP=(NEW,KEEP)
   //SYSIN   DD  DATA
7. ./        ADD   ADDPROCS,01,0,1
   //PROC    EXEC  PGM=IEBUPDAT,PARM=NEW
8. //SYSUT2    DD  DSNAME=SYS1.PROCLIB,DISP=MOD
   //SYSPRINT DD   SYSOUT=A
9. /*
```

1. Execute the utility UPDATE program (see C28-6586) in order to enter into SYS1.PROCLIB a standard procedure for entering procedures into SYS1. PROCLIB.
2. The data to follow will be put into SYS1.PROCLIB (SYSIN DD DATA).
3. Required by the utility.
4,5,6. DD cards which allocate space on TESTVL. Note that each library directory will handle 10x(4 to 7) members. Therefore, at any one time, a library directory can handle 10x6 (on the average) = 60 members. Note also that the actual number of modules that can be stored in a library depends on the size of the modules and the total space allocated to the data set.
7. Control statement for IEBUPDAT. It names the member to be added. In this case it will be named ADDPROCS.
8. The job control language for the procedure called ADDPROCS.
9. Required by the utility and the control program.

Figure 7. To create the TESTS libraries and add a procedure that will add procedures



1. Executes the ADDPROCS procedure (see Figure 7) for entering a procedure into the SYS1.PROCLIB.

2. DD * statement for the ADDPROCS procedure — that is, the member(s) follows. Note: the DD name must be qualified with the step name of the procedure. The step name is PROC (see Figure 7).

3. Required by the utility UPDATE (see C28-6586). The NEWSORCE name will be the name of the member (procedure) that will be added.

4. The job control cards that will be entered as a procedure.

5. Required by the utility and the control program.

Note: Any procedure may be added in this manner. If multiple procedures are to be added with one EXEC ADDPROCS, the ADD cards (with the procedure names and associated job control statements to be entered) must be in binary collating sequence.

Figure 8. To add a procedure to SYS1.PROCLIB with ADDPROCS

UTILIZATION OF TESTS

Source Library

The library called TEST.SOURCE contains source modules. Each module is in source code (Assembler, FORTRAN, COBOL, PL/I). The original source code is entered into the TEST.SOURCE PDS once. After errors are detected via the debugging cycle, the programmer requests changes to his source code. This method eliminates voluminous card and tape handling since the source code is always on disk.

SOURCE MODULE CREATION

By executing the NEWSORCE procedure, any set of symbolic coding may be entered into the TEST.SOURCE library. This source coding then becomes a member of the source library, with a program name specified by the programmer. Figure 9 illustrates this method. Note that NEWSORCE assigns a sequence number to each source statement. This sequence number can be referenced by the programmer when making changes to the source module.

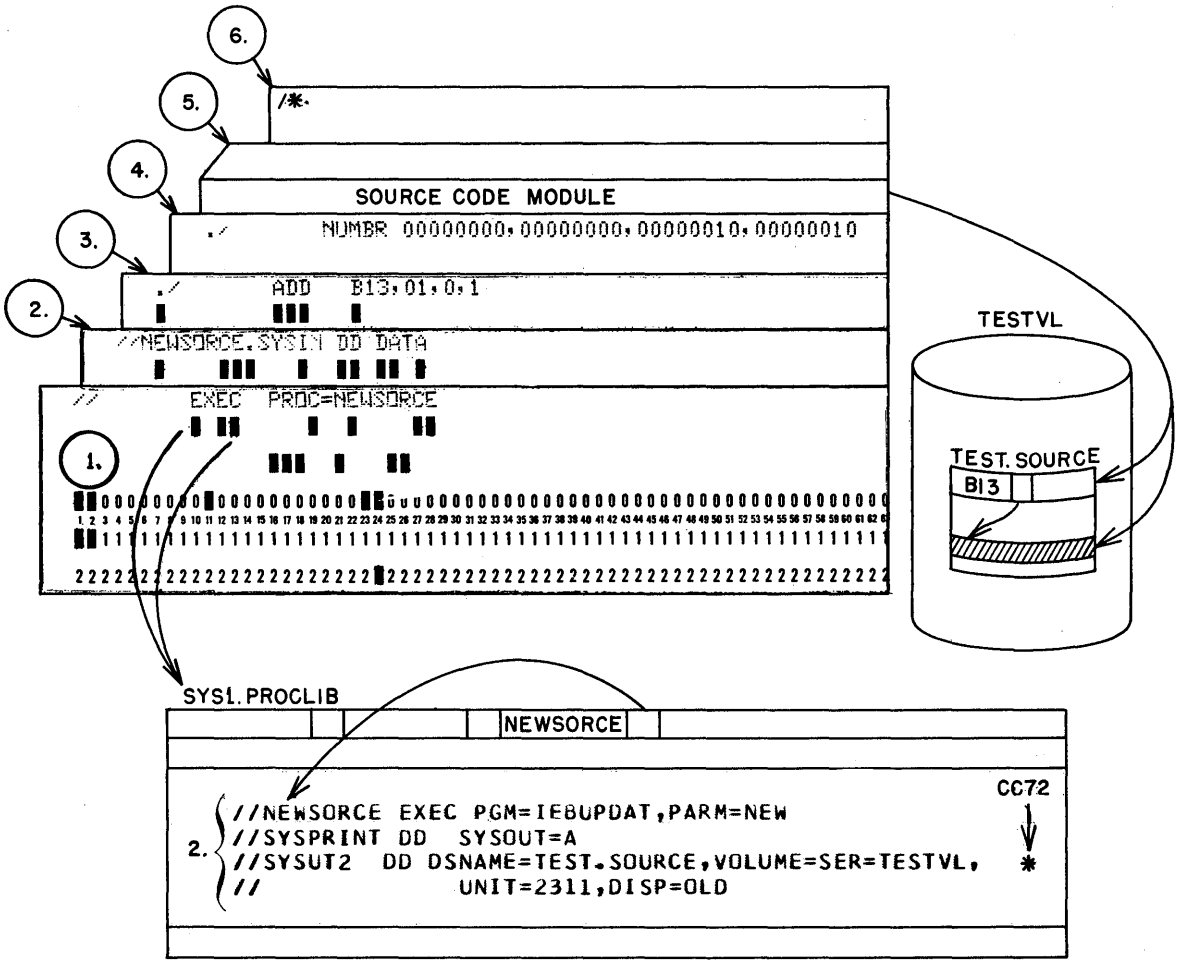
SOURCE MODULE CORRECTION

After a debugging run, changes to the original source code may be needed. The procedure CHGSORCE allows the programmer to specify which original source statement(s) he desires to have deleted (if any) and whether he wishes new source statements added to the original source.

A standard TESTS form is illustrated to allow the subsequent additions and deletions of source code (see Figure 10). The implementation of these changes is shown in Figure 11, and the listed results indicating what took place is shown in Figure 12.

Object Module Creation

Once the source modules have been entered on TEST.SOURCE, they are processed by one of the language translators (Assembler, FORTRAN, PL/I, or COBOL). The output of a language translator is defined as an object module, which in this application becomes a member in the object library (TEST.OBJECT).



1. Execute the NEWSORCE procedure.
2. NEWSORCE procedure in SYS1.PROCLIB.
3. Name the added member B13.
4. Sequence-number the source code, starting at 10 and incrementing by 10.
5. Symbolic code to be entered into TEST.SOURCE.
6. Required by IEBUPDAT (C28-6586) and reader-interpreter. (Note: Source code for multiple members may be entered with one EXEC NEWSORCE; however, the ADD card with the member name and associated source code must be in collating sequence by member name.)

Figure 9. To enter symbolic (source) module into the source library

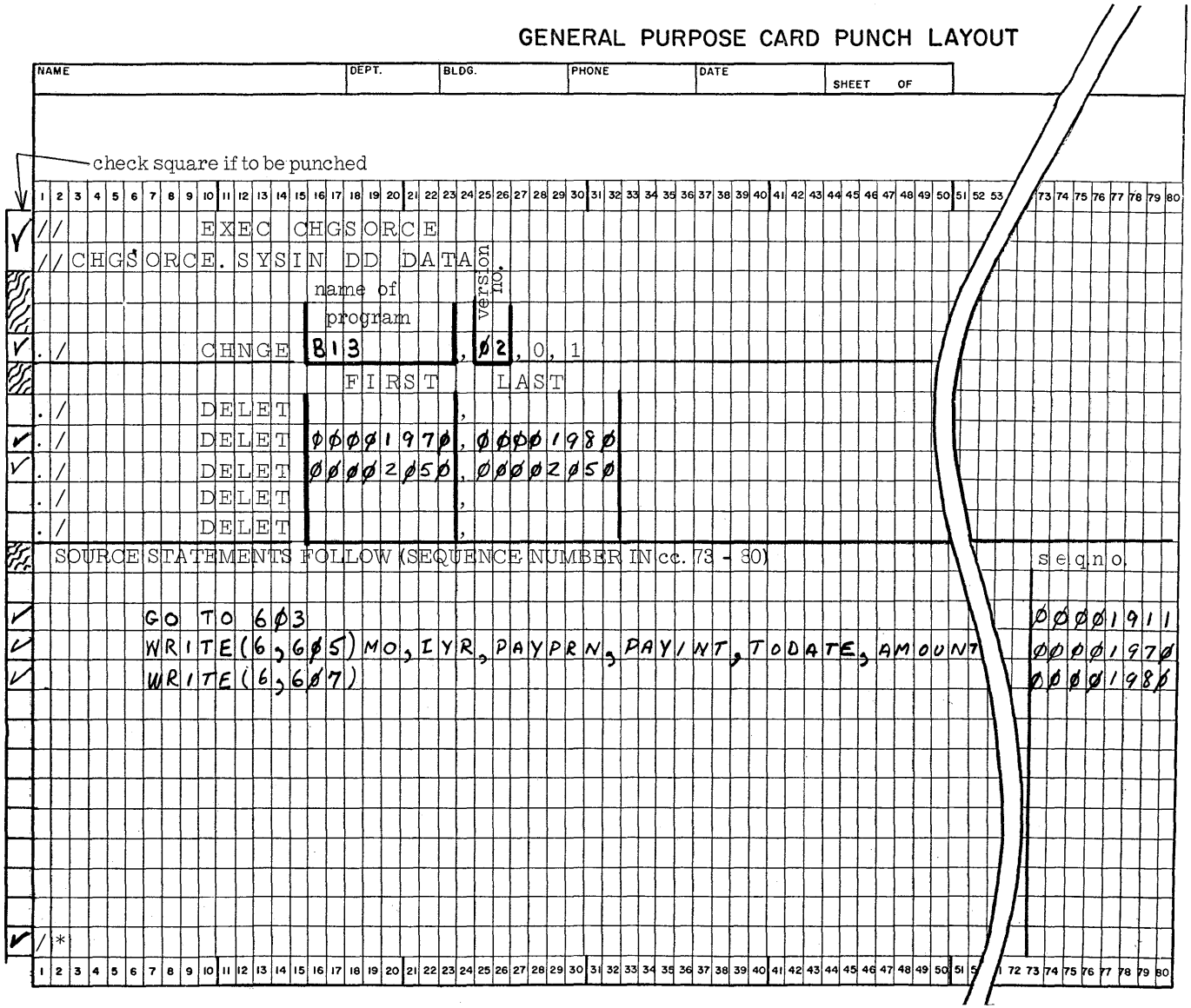
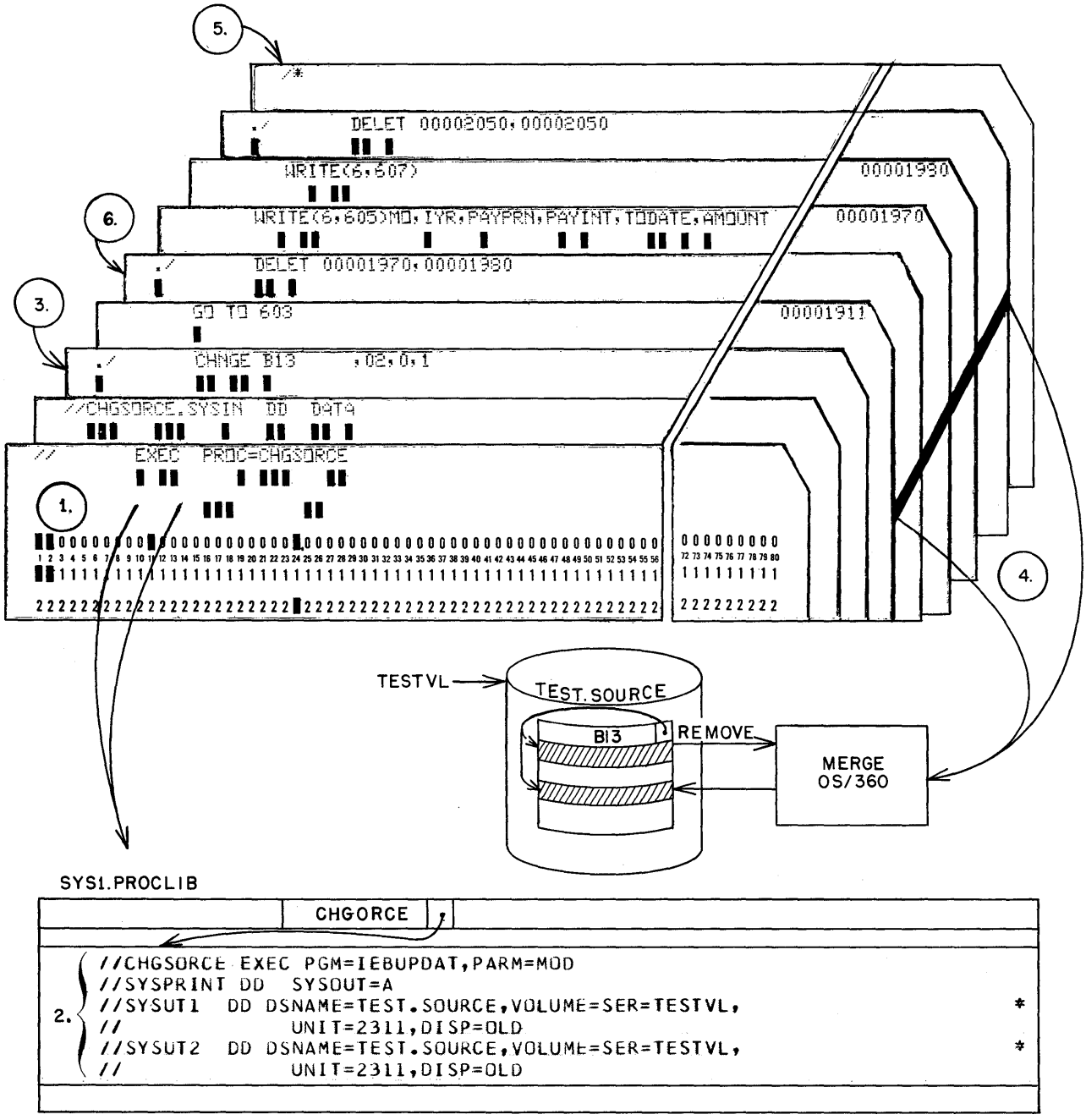
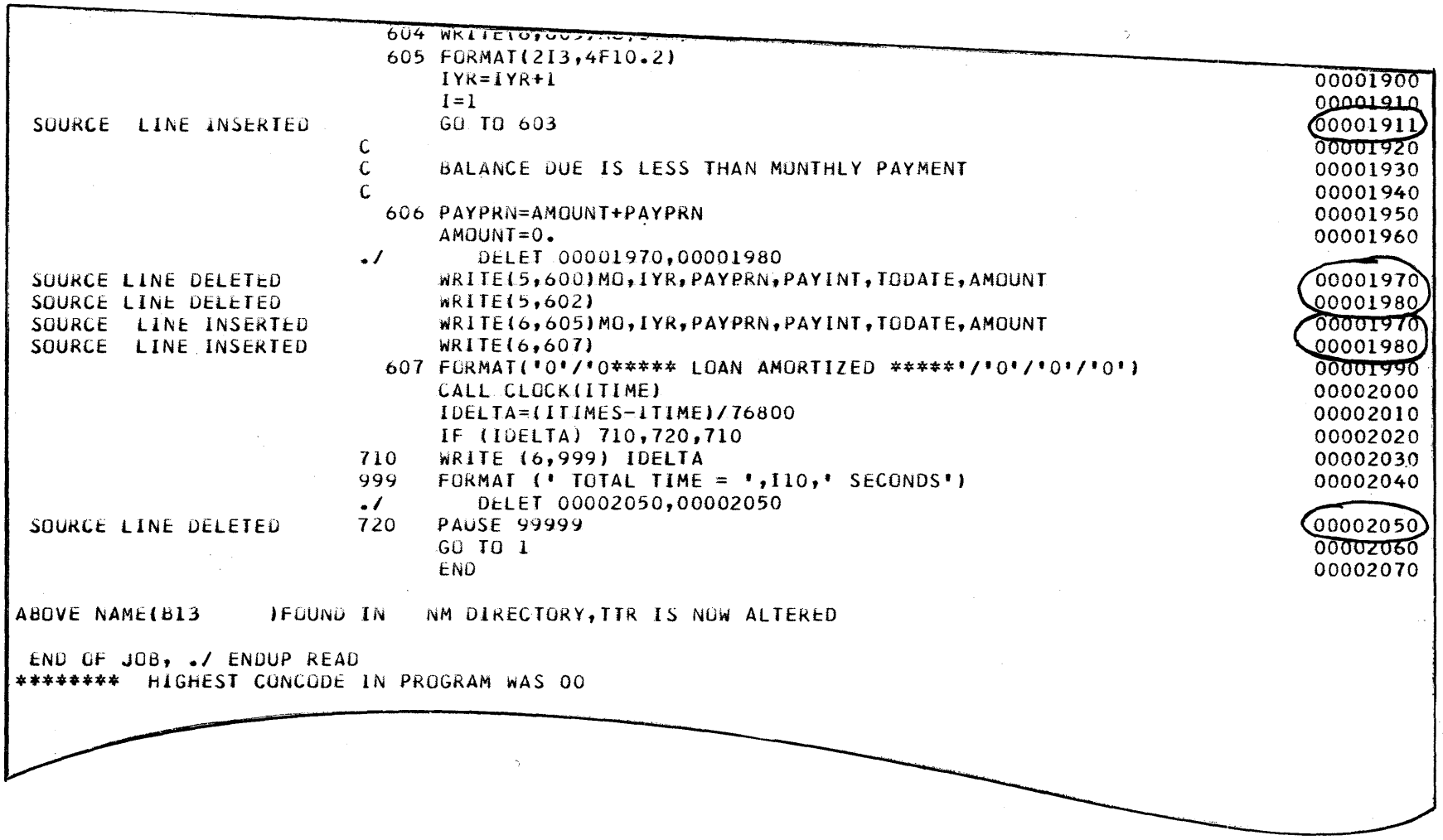


Figure 10. Form for entering changes into the source library



1. Invoke the CHGSORCE procedure.
2. CHGSORCE JCL in SYS1.PROCLIB.
3. Name of the program (member) to be changed — in this case B13.
4. Specified deletions of 80-character records and source code to be added.
5. Required by IEBUPDAT (C28-6586) and the Interpreter.
6. This DELET card is not required since both the old 1970 and 1980 would be automatically deleted and replaced by the new 1970 and 1980 source statements.

Figure 11. To make corrections to the source library



```
                          604 WRITE(6,609)...
                          605 FORMAT(2I3,4F10.2)
                              IYR=IYR+1                                          00001900
                              I=1                                                00001910
 SOURCE  LINE INSERTED        GO TO 603                                          00001911
                     C                                                         00001920
                     C    BALANCE DUE IS LESS THAN MONTHLY PAYMENT              00001930
                     C                                                         00001940
                          606 PAYPRN=AMOUNT+PAYPRN                             00001950
                              AMOUNT=0.                                        00001960
                     ./       DELET 00001970,00001980
 SOURCE LINE DELETED          WRITE(5,600)MO,IYR,PAYPRN,PAYINT,TODATE,AMOUNT   00001970
 SOURCE LINE DELETED          WRITE(5,602)                                     00001980
 SOURCE  LINE INSERTED        WRITE(6,605)MO,IYR,PAYPRN,PAYINT,TODATE,AMOUNT   00001970
 SOURCE  LINE INSERTED        WRITE(6,607)                                     00001980
                          607 FORMAT('0'/'0***** LOAN AMORTIZED *****'/'0'/'0'/'0')  00001990
                              CALL CLOCK(ITIME)                                00002000
                              IDELTA=(ITIMES-ITIME)/76800                      00002010
                              IF (IDELTA) 710,720,710                          00002020
                     710      WRITE (6,999) IDELTA                             00002030
                     999      FORMAT (' TOTAL TIME = ',I10,' SECONDS')         00002040
                     ./       DELET 00002050,00002050
 SOURCE LINE DELETED 720      PAUSE 99999                                      00002050
                              GO TO 1                                          00002060
                              END                                              00002070

ABOVE NAME(B13      )FOUND IN    NM DIRECTORY,TTR IS NOW ALTERED

 END OF JOB, ./ ENDUP READ
********  HIGHEST CONCODE IN PROGRAM WAS 00
```

Figure 12. Results of source correction

A cataloged procedure named TESTASSM, using the assembler as the language translator, compiles a source module from TEST.SOURCE into an object module. The name of each of the input source modules from the source library, and the names of each of the output modules to be entered in the object library (TEST.OBJECT), are specified in the job stream for each language translator job step. This procedure and the required DD statements specifying input and output are illustrated in Figure 13. Similar procedures may be executed for FORTRAN and COBOL (see Figures 14 and 15).

Load Module Creation

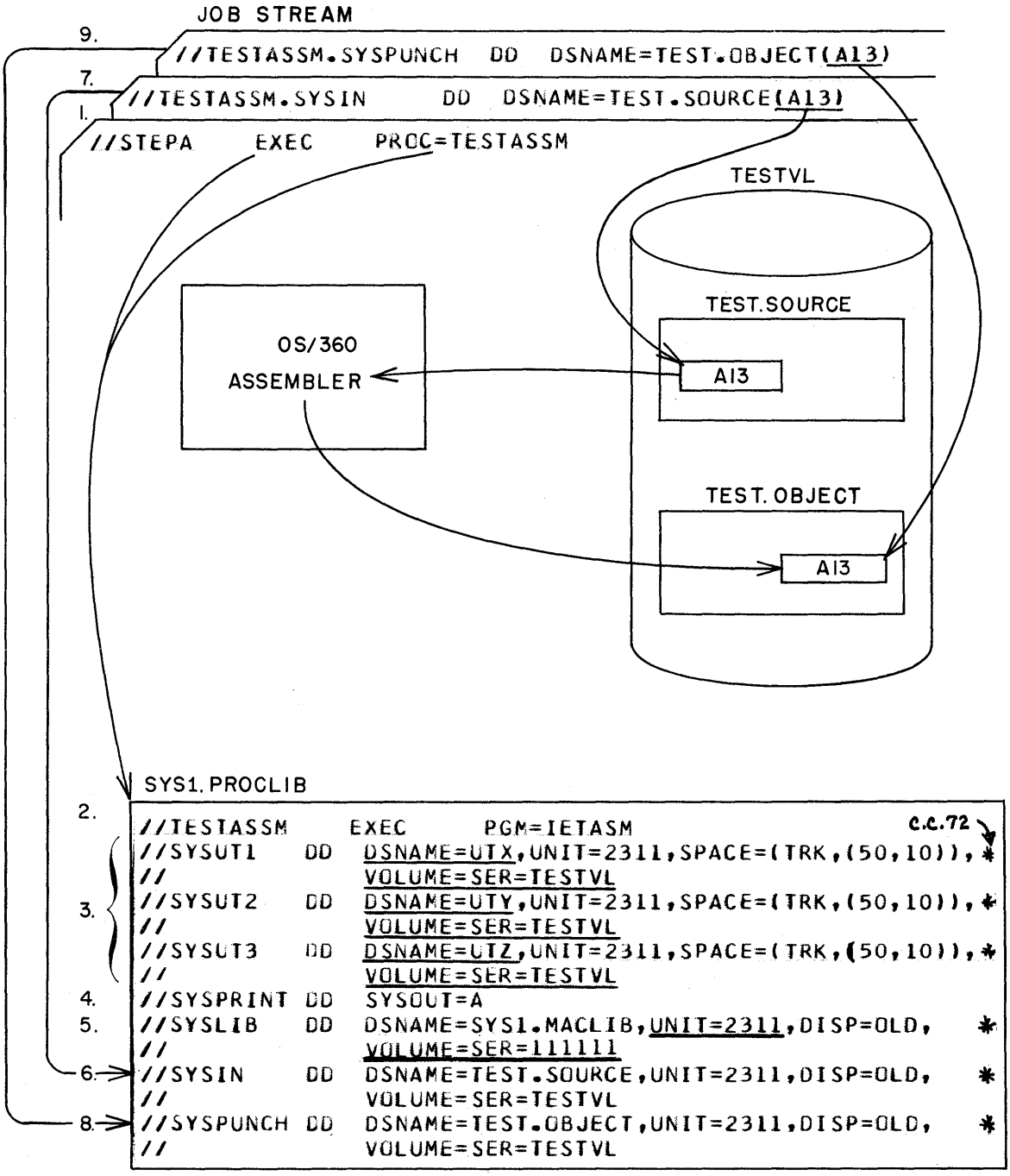
Object module output from language translators is in relocatable, but not executable, format. Therefore, before execution, the object modules must be processed by Linkage Editor so that they may become executable load modules. In addition, adhering to OS/360's basic concept of modularity, modules that have been separately tested may be combined by the Linkage Editor. Also, any editing or overlay structuring of existing object or load modules is done at this time. Because in this application all object modules are in TEST.OBJECT and all load modules are in TEST.LOAD, each has access to the others in the TESTS environment, easing considerably the difficulties in locating modules.

While the linkage editing can be done on a compile-linkage edit-execute basis for each program to be tested within the TEST environment (see Figure 18), the procedure to be discussed here addresses itself to a single Linkage Editor run during which multiple load modules are created, thereby reducing the number of times the processor is brought into core storage.

Once the programs to be tested are in the TEST.OBJECT library as object modules, they will be link-edited via the cataloged procedure TESTLINK, onto the load module library (TEST.LOAD). This, then, allows the programmer to reference these libraries for any additional modules he may require by use of the Linkage Editor INCLUDE statement.

As illustrated below, the job stream for the Linkage Editor run, contains (1) an EXEC statement calling for execution of the cataloged procedure TESTLINK, (2) a DD * statement named TESTLINK.SYSLIN, which indicates that the input specifications to Linkage Editor will follow in the job stream, and (3) a set of Linkage Editor control statements specifying the names of the input and output modules of each load module to be created.

```
// EXEC PROC=TESTLINK
// TESTLINK.SYSLIN DD *
   INCLUDE OBJPDS (object module name 1)
   NAME load module name 1 (R)
   INCLUDE OBJPDS (object module name 2)
   NAME load module name 2 (R)
```



Underlined parameters are not necessary.

Figure 13. To compile or assemble a source module from the source library into the object library

1. This EXEC statement in the job stream invokes the cataloged procedure TESTASSM.

2. This EXEC statement invokes the assembler IETASM.

3. Three DD statements defining the space and volume (TESTVL) on which the three utility data sets required by the assembler should be allocated.

4. SYSPRINT specifies that the assembly listing should be printed.

5. This SYSLIB DD statement specifies that SYS1.MACLIB, which is required for the assembler, resides on the system residence volume, 111111.

6. SYSIN specifies the name of the library (TEST.SOURCE) containing the input source modules, which will be used as input to the assembler or compiler, and indicates that this library resides on TESTVL.

7. TESTASSM.SYSIN specifies the name of the source module (A13) to be assembled from the library and overrides the parameter in 6.

8. SYSPUNCH specifies that the library named (TEST.OBJECT) residing on TESTVL is the library in which the object modules are to be placed.

9. TESTASSM.SYSPUNCH specifies that the name of the object module to be placed in TEST.OBJECT is A13 and overrides the DSNAME parameter in 8.

Figure 13 (continued).

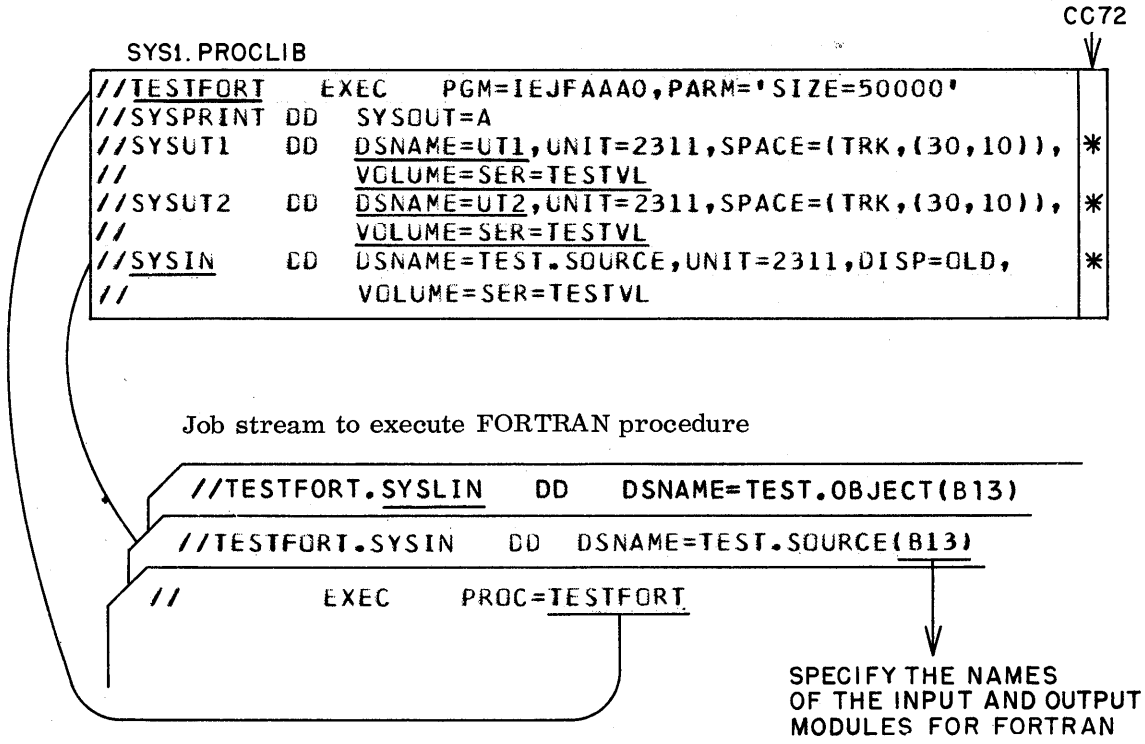
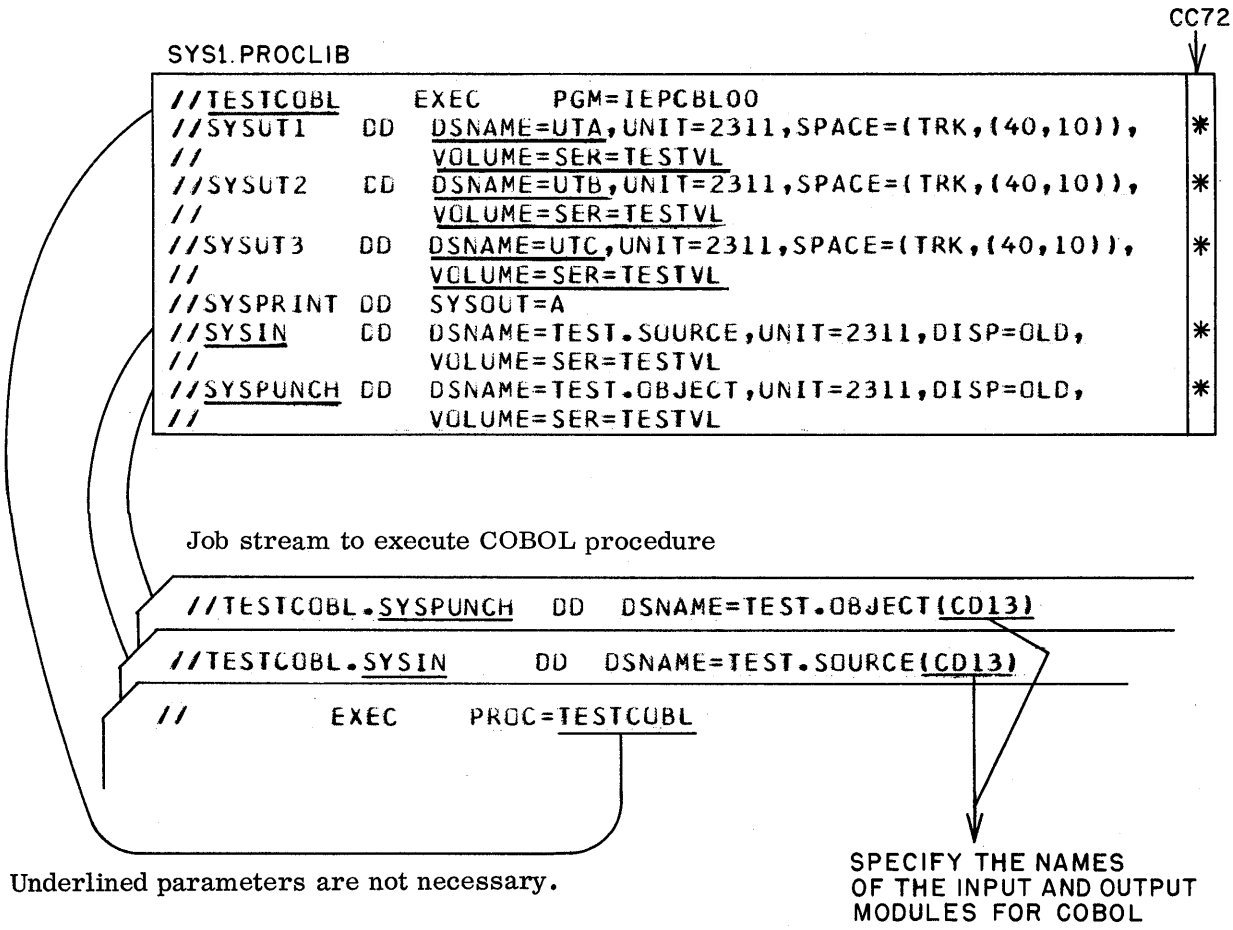


Figure 14. To create object modules using TESTS cataloged procedure for FORTRAN



Underlined parameters are not necessary.

Figure 15. To create object modules using TESTS cataloged procedure for COBOL

To combine additional object modules in a load module, their names may be specified in one INCLUDE statement (see item 1 below), or additional INCLUDE statements may be inserted (see items 2 and 3 below).

1. INCLUDE OBJPDS (name 1, name 2, name 3)

2. INCLUDE OBJPDS (name 2)

3. INCLUDE OBJPDS (name 3)

Load modules from the load library may be combined with other modules as follows:

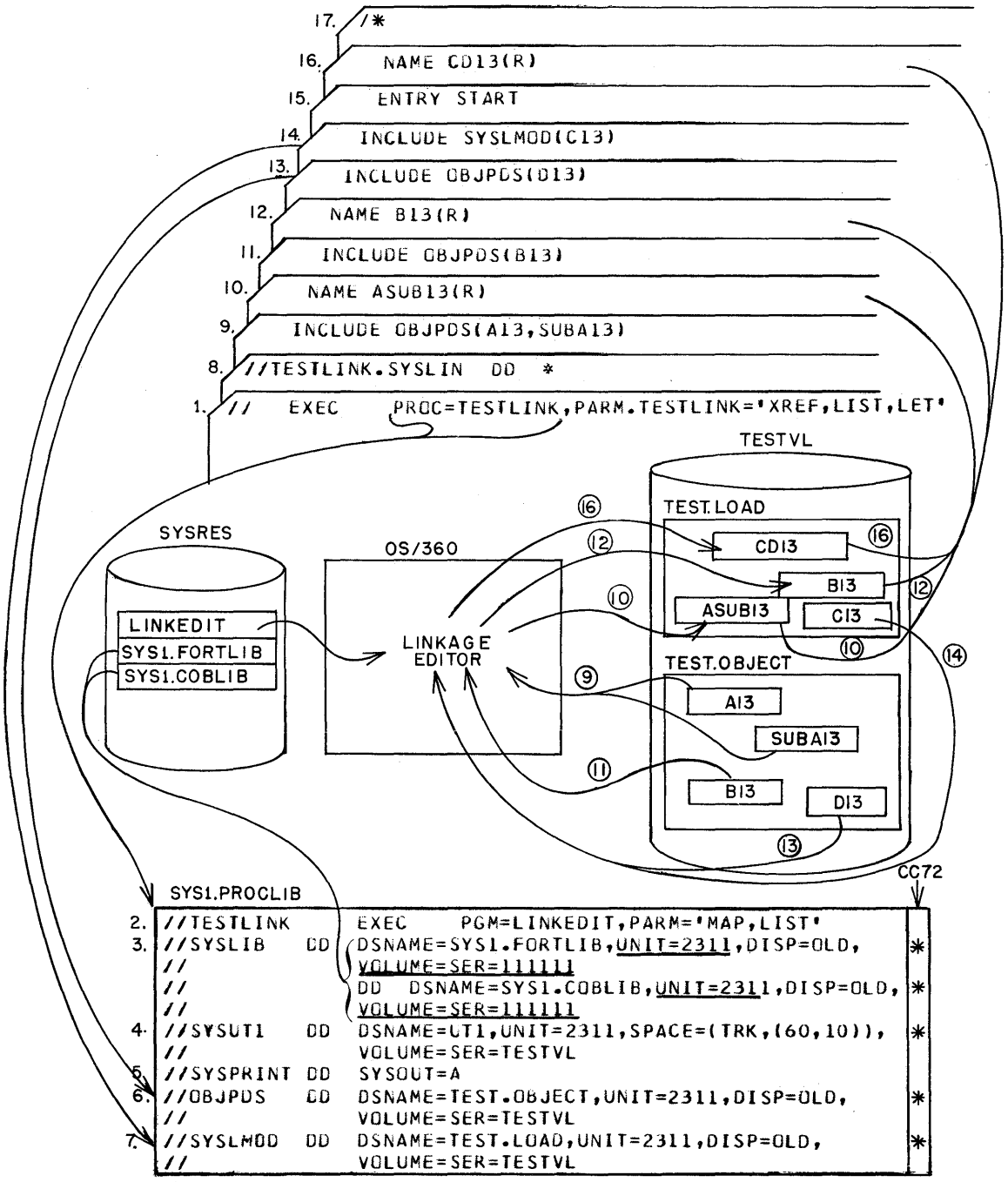
INCLUDE SYSLMOD (load module name or names)

Additional specifications for each load module may be inserted between the INCLUDE and NAME statements. If more than one module is to comprise the load module, an ENTRY statement specifying the entry point to be assigned to the load module should immediately precede the NAME statement.

Any Linkage Editor control statements to create an overlay structure or to edit the modules should be placed in the job stream preceding the NAME statement as specified in the Linkage Editor manual (C28-6538).

The Linkage Editor procedure in Figure 16 (TESTLINK) produces a module map and a list of all Linkage Editor control statements. If additional or different processing options are desired, all parameters required should be specified in the EXEC card, as shown in Figure 16.

FORTRAN and COBOL object modules require that SYS1.FORTLIB and SYS1.COBLIB respectively be specified as the Linkage Editor automatic call library (SYSLIB) (see C28-6538). Therefore, they have been concatenated in the TESTLINK procedure.



1. The EXEC statement in the job stream invokes the cataloged procedure TESTLINK. PARM. TESTLINK='XREF, LIST, LET' overrides the PARM field in the EXEC statement of the cataloged procedure and will cause a cross-reference listing to be produced instead of a memory map and put into effect the processing option LET.

2. The EXEC statement invokes Linkage Editor and specifies processing options MAP and LIST (in this example they were overridden).

Figure 16. To linkage-edit multiple load modules in one Linkage Editor run

3. SYSLIB defines the automatic call library to Linkage Editor and concatenates SYS1.FORTLIB and SYS1.COBLIB. This allows any object modules to be processed, whether compiled by COBOL or FORTRAN.

4. SYSUT1 specifies that the Linkage Editor's utility (work) data set be allocated space on the volume TESTVL.

5. SYSPRINT specifies that the diagnostic messages, memory map, and a list of Linkage Editor control statements processed should be written on the printer.

6. This DD statement indicates that any reference to OBJPDS in Linkage Editor control statements will refer to the object library (TEST.OBJECT), which resides on volume TESTVL.

7. The SYSLMOD DD statement specifies that all load modules created by Linkage Editor in this run will be placed in the load library (TEST.LOAD), which resides on the volume TESTVL.

8. This specifies that the primary input data (SYSLIN) follows immediately in the job stream.

9. This Linkage Editor control statement specifies that there are two members, A13 and SUBA13, in the library specified by the DD statement named OBJPDS that will be the input to this load module.

10. This control statement specifies that the name of the first load module to be placed in TEST.LOAD is ASUB13.

11. This control statement specifies that the input to the second load module is the member named B13 on TEST.OBJECT.

12. This control statement specifies that the name of the second load module to be entered in the load library is B13.

13. This statement specifies that member D13 on TEST.OBJECT will be part of the third load module.

14. This specifies that C13, previously link-edited and on TEST.LOAD (indicated by the SYSLMOD DD statement, which points to that library) is to be combined with D13 as input to the third load module.

15. This ENTRY statement assigns an entry point named START to the load module.

16. This NAME statement assigns the module name CD13 to the load module containing C13 and D13 on TEST.LOAD.

17. /* denotes the end of the Linkage Editor input.

Figure 16 (continued).

Execution of Load Modules

Because all of the load modules now ready for execution are in the library named TEST.LOAD, the JOBLIB DD statement required for execution of each of the load modules to be tested is the same (see "Definition of a Library"). Therefore, the job stream required to execute any load module will contain a JOB card, a JOBLIB DD statement pointing to TEST.LOAD, an EXEC statement where PGM='member name to be tested', followed by the appropriate DD statements for that particular program. (See Figure 17 for an illustration of a job stream to test programs in TEST. LOAD). While it is possible to test multiple load modules in one JOB, each as a separate job step, all using only one JOBLIB statement, it should be noted that once an abnormal end of one execution is encountered, the ensuing job steps will be bypassed.

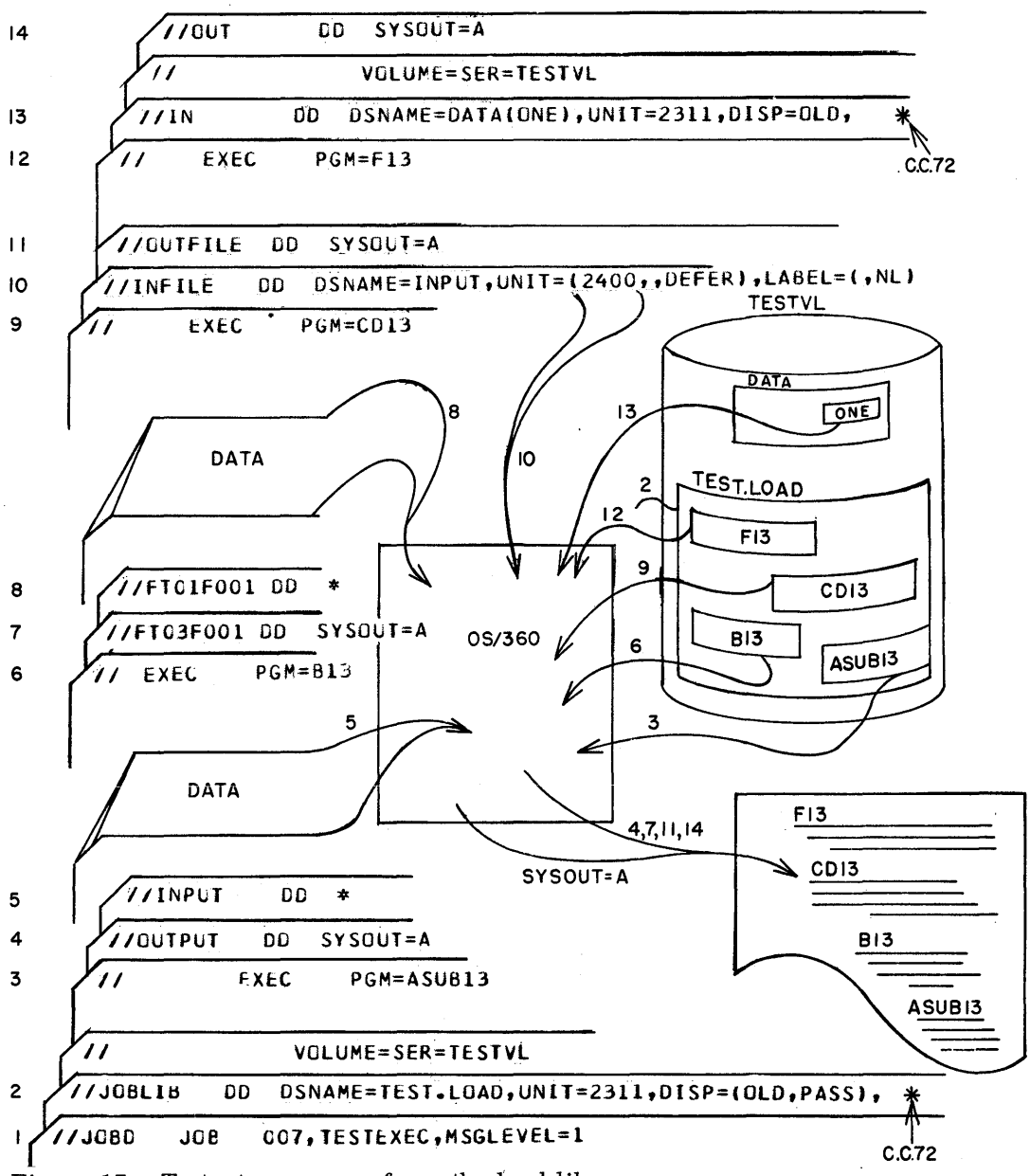


Figure 17. To test programs from the load library

1. The JOB statement indicates that a new job, JOBD, follows.

2. JOBLIB indicates that before searching the Link library for the programs to be executed, the library TEST.LOAD on volume TESTVL should be searched.

3. The EXEC statement causes the program ASUB13 to be executed.

4. This DD statement indicates that the output (assigned the ddname OUTPUT by the programmer) of ASUB13 is to go on the printer.

5. This DD statement specifies that the input data (INPUT) for ASUB13 follows in the job stream.

6. The second job step indicated by this EXEC statement causes program B13 to be read into core from TEST.LOAD and executed.

7. FT03F001 is the ddname assigned by FORTRAN to the output data set for B13, the printer.

8. FT01F001 is the ddname assigned by FORTRAN to the DD statement that specifies that B13's input data follows in the job stream.

9. The third job step causes program CD13 to be executed.

10. INFILE specifies that the input data set for CD13, INPUT, is on tape.

11. OUTFILE specifies that the results of CD13 are to be printed.

12. The fourth job step causes program F13 to be executed from TEST.LOAD.

13. IN, the ddname defining the input data set for F13, indicates that it is a member named ONE in the PDS named DATA.

14. OUT specifies that the results of F13 are to be printed.

Figure 17 (continued).

Job Stream

```
                //TESTPROC JOB  007,INSTLTEST,MSGLEVEL=1
COMPILE         //          EXEC  PROC = (A)
                //    (A)   .SYSIN DD    DSNAME=TEST.SOURCE ((B))
                //    (A)   .(C)       DD DSNAME=TEST.OBJECT ((B))

LINKEDIT        //          EXEC  PROC=TESTLINK,PARM.TESTLINK='(D)'
                //TESTLINK.SYSLIN DD*
                  INCLUDE OBJPDS ((B))
                  NAME      (E) (R)
                /*

EXECUTE         //TEST B    JOB   007,INSTLEXEC,MSGLEVEL=1
                //JOBLIB    DD    DSNAME=TEST.LOAD,DISP=(OLD,PASS).
                                  VOLUME=SER=TESTVL
                //          EXEC  PGM =(E)
                //    (F)   DD
                //    (F)   DD
                /*
```

(A) = Language Procedure Name

FORTRAN	PROC=TESTFORT
COBOL	PROC=TESTCOBL
ASSEMBLER	PROC=TESTASSM

(B) = Member name, that is, name of program to be compiled

(C) = Compiler output ddname

FORTRAN	SYSLIN (See Figure 14)
COBOL	SYSPUNCH
ASSEMBLER	SYSPUNCH

(D) = Optional Linkage Editor parameters

(E) = Member name to be assigned to load module. E may equal B

(F) = DD statements required to specify input and output data sets for execution of the program.

Figure 18. Generalized compile-Linkage-Edit — Execute procedure

LIBRARY MAINTENANCE

Three types of maintenance are required to keep the three libraries (source, object, load) to a manageable size:

1. Reducing the PDS's extent requirements
2. Purging unused members
3. Punching, listing, and deleting completed programs

Reducing Extent Requirements

The frequency with which the installation would wish to reduce the extents of a PDS depends on the volume of testing being performed and the original size of the PDS. Additional extents may be required, as new members (programs) are added or updated in a library. As an example, if a source program named MATRIX13 were originally put into the source library and subsequently changed through the CHGSORCE procedure, the original space for the MATRIX13 module would be unavailable for use.

Probably on a shift basis or daily basis, the installation would want to obtain a picture of the situation. In order to do this, a procedure is included here called TESTPEEK. This procedure allows the printing of the TESTVL Volume Table of Contents, and the contents of each library. Figure 20 illustrates the TESTPEEK procedure. Note that only one card is required in the job stream to obtain the listings, because the control statements for the utility IEHLIST are located in SYS1.PROCLIB, cataloged under the name CNLPEEK2. The control statements are called by the SYSIN DD statement in TESTPEEK, which, of course, is also in SYS1.PROCLIB. An examination of the output, with particular attention to the number of extents in each library, may lead to the decision to reorganize the libraries if they contain much space that is unavailable for use.

To perform this function, we MOVE (see C28-6586) the TESTVL volume to itself. This particular utility program for each PDS specified, examines the directory and moves it to the new PDS. It also places the members in the top of the new PDS as illustrated in Figure 19.

To perform this for all the partitioned data sets on the entire TESTS volume, a procedure called CLEAN can be used, which is illustrated in Figure 21. Note that again only one card is required in the job stream to perform the CLEAN procedure, because both the CLEAN procedure and the IEHMOVE control statement required by it are on SYS1.PROCLIB. The SYSIN DD in CLEAN calls the control statement which is cataloged under the name CLEAN1.

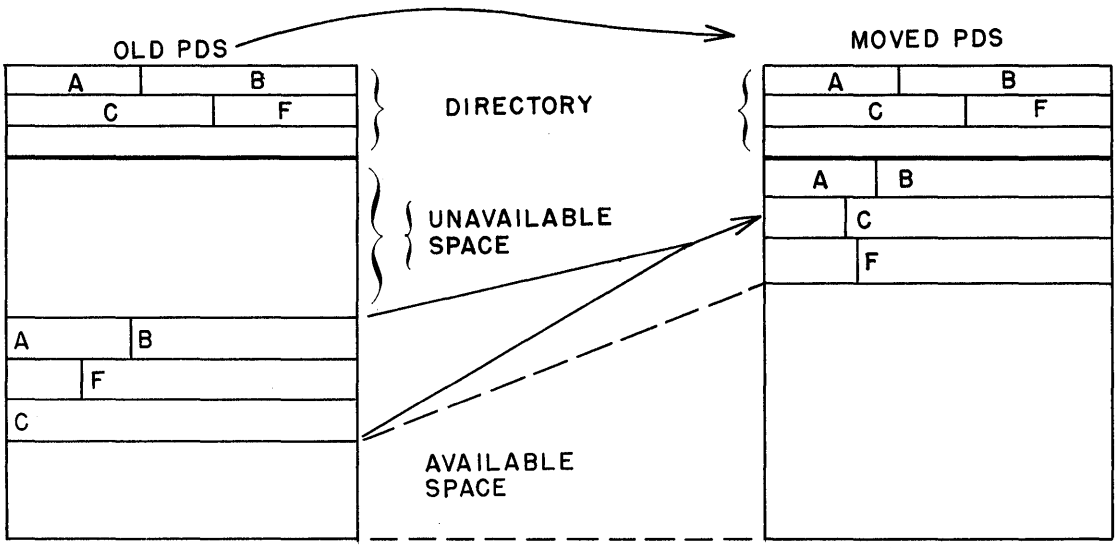


Figure 19. Increasing the available space in a PDS

Purging Unused Members

From the listing received from the TESTPEEK procedure, it will be desirable to audit the usefulness and timeliness of the various modules (programs).

If it is determined, for instance, that a particular module is no longer useful, the installation may run the utility IEHPROGM and scratch a particular member from all libraries. A procedure for this has not been included in this document, but it could be similar to the last three steps in the SORCEDOC procedure (see Figure 22).

Completed Programs

After a program has completed its required testing and is performing satisfactorily, it can be (1) moved to LINKLIB or a specific JOBLIB; (2) used to obtain a copy on tape or cards, or to obtain a listing; etc.

One of the most common joint functions performed on a completed program would be to (1) list the source code, (2) punch a source deck, and (3) delete the program member from the source, object, and load libraries. The SORCEDOC procedure, (see Figure 22) together with the cataloged control statements that are also in SYS1.PROCLIB (see

Figure 23) will perform all these functions. To reduce the number of cards required in the job stream (only four are required to print and punch a program while deleting it from three libraries), the SORCEDOC procedure uses several features of OS/360:

1. The ability to execute a multiple-step procedure. (SORCEDOC contains a number of EXEC statements.)

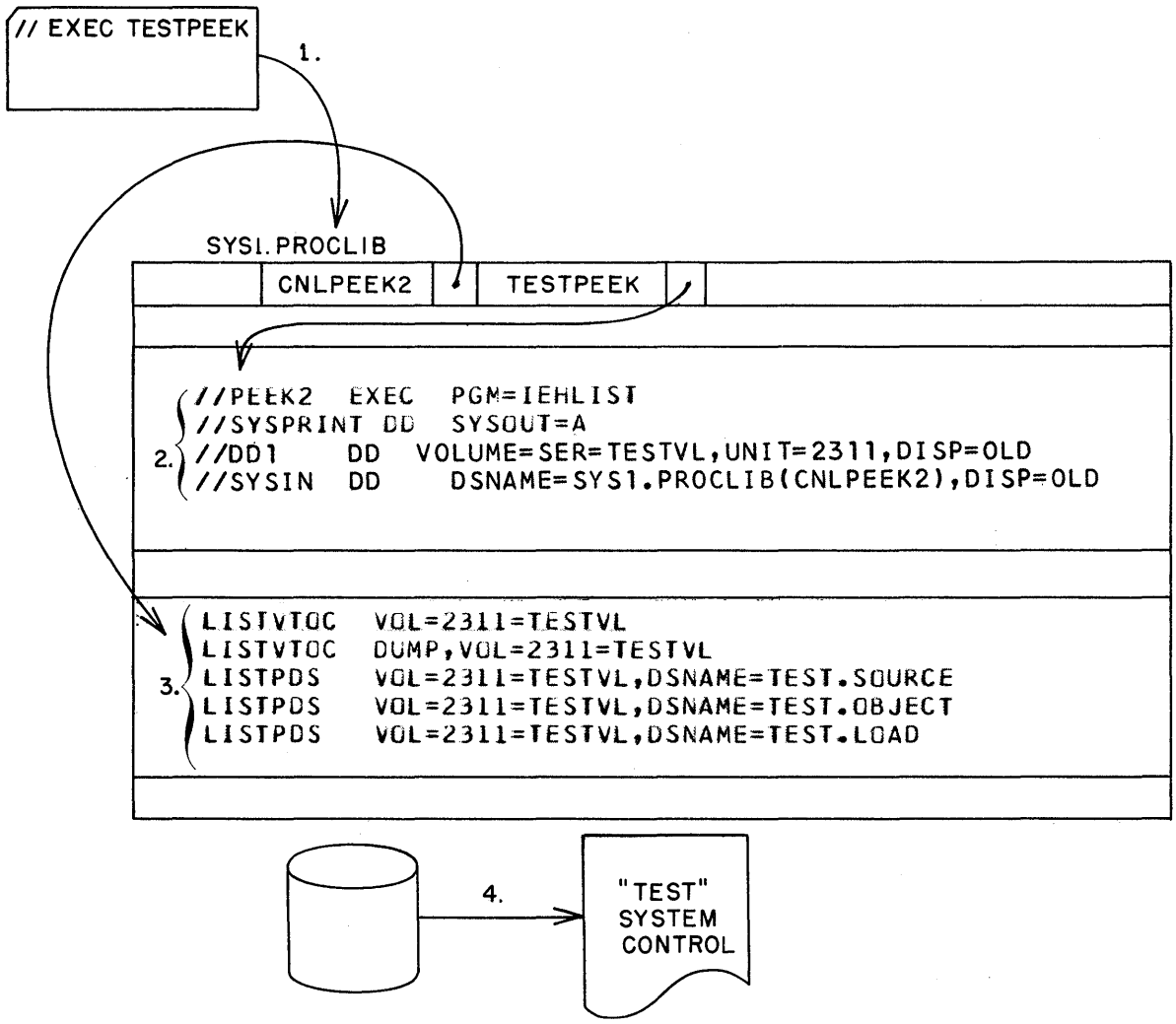
2. The ability to call utility control statements from a library rather than placing them in the job stream.

3. The ability to specify that a sequential data set is to be concatenated with a PDS member, and the ability to continue a utility control statement. (See "Data Set Utilities" in C28-6586 for concatenation restrictions.)

In this case (Figure 22) the name of the program to be printed, punched, and scratched (entered in the job stream) is recorded in a newly created temporary data set called TEMP (see point 1 in Figure 22) by the utility IEBGENER. This temporary data set then supplies the name of the program to the other utility programs in SORCEDOC. Note that the SYSIN DD statements in steps 2-6 of SORCEDOC call a utility control statement (a member of a PDS) from SYSIN.PROCLIB. The following DD statement, since it has no ddname, concatenated this temporary data set with the control statement. Also note that each utility control statement is prepared with an = sign in cc 71 (following MEMBER or MEMBER NAME) and a continuation indicator in cc 72. Therefore, each utility in steps 2-6 of SORCEDOC looks for the member name in TEMP after reading the = sign of the control statement. (See Figure 24 for a detailed illustration of the concatenation of the data sets and continuation of the utility control statement.)

Further, it is important to observe that to print/punch a member using the utility Print/Punch program, the detailed statement must be written MEMBER NAME = XXXX. In the IEHPROGM utility, to scratch a member, it must be specified as..., MEMBER = XXXX.

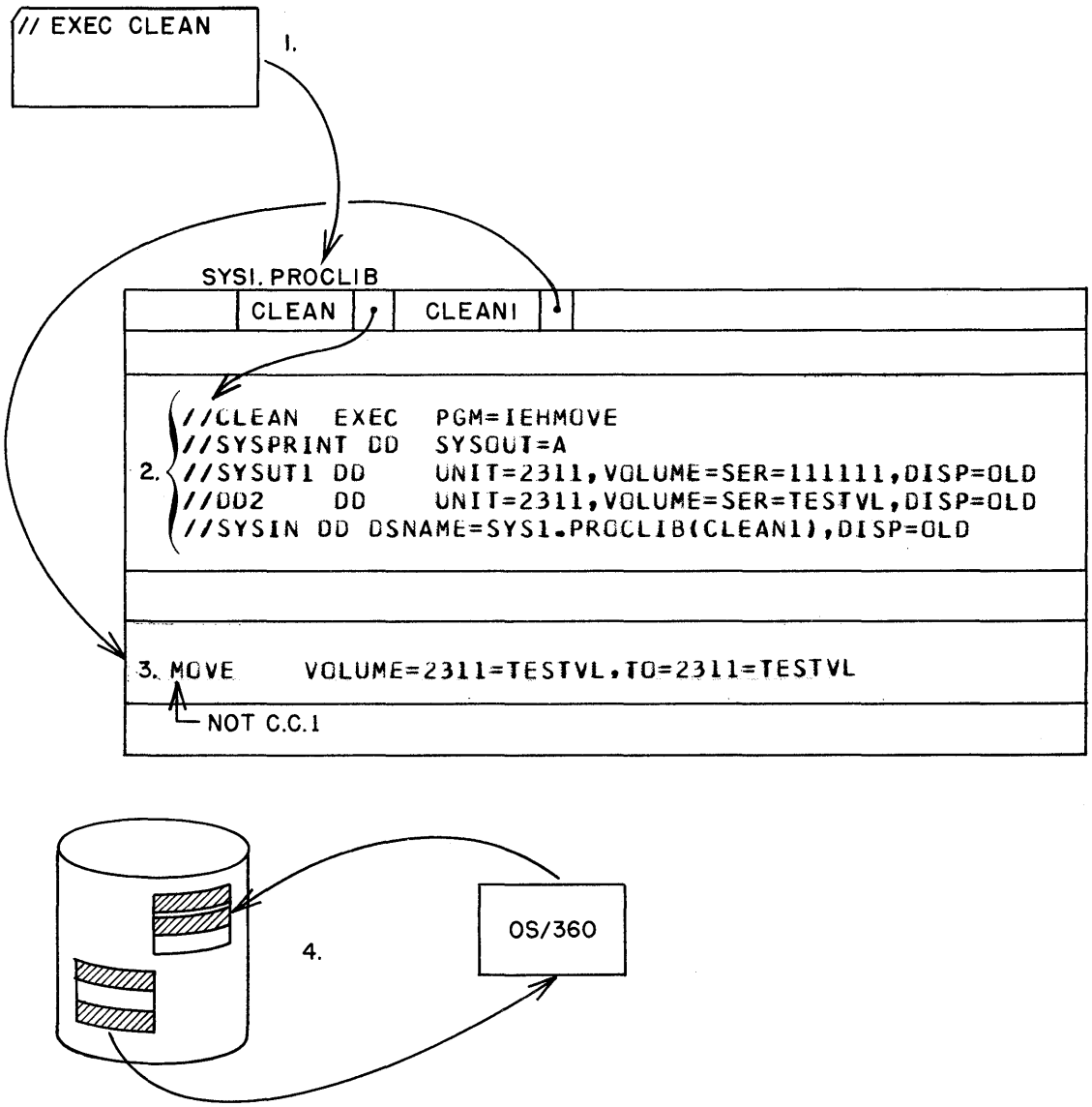
One of the most important features of the SORCEDOC approach is that it protects the user from inadvertently scratching a library. If, for example, a nonexistent member is specified or a member specification was omitted from the job stream, the utility will not scratch the library, since no member name was specified for the MEMBER or MEMBER NAME parameter.



1. Only one card required to obtain listings.
2. The procedure TESTPEEK located in SYS1.PROCLIB.
3. The control statements for the utility IEHLIST to list the desired data. Note that these five statements are located in SYS1.PROCLIB under the name CNLPEEK2. These statements are called by the DD Statement in TESTPEEK. A SYSIN DD * cannot reside in a cataloged procedure.
4. Output results.

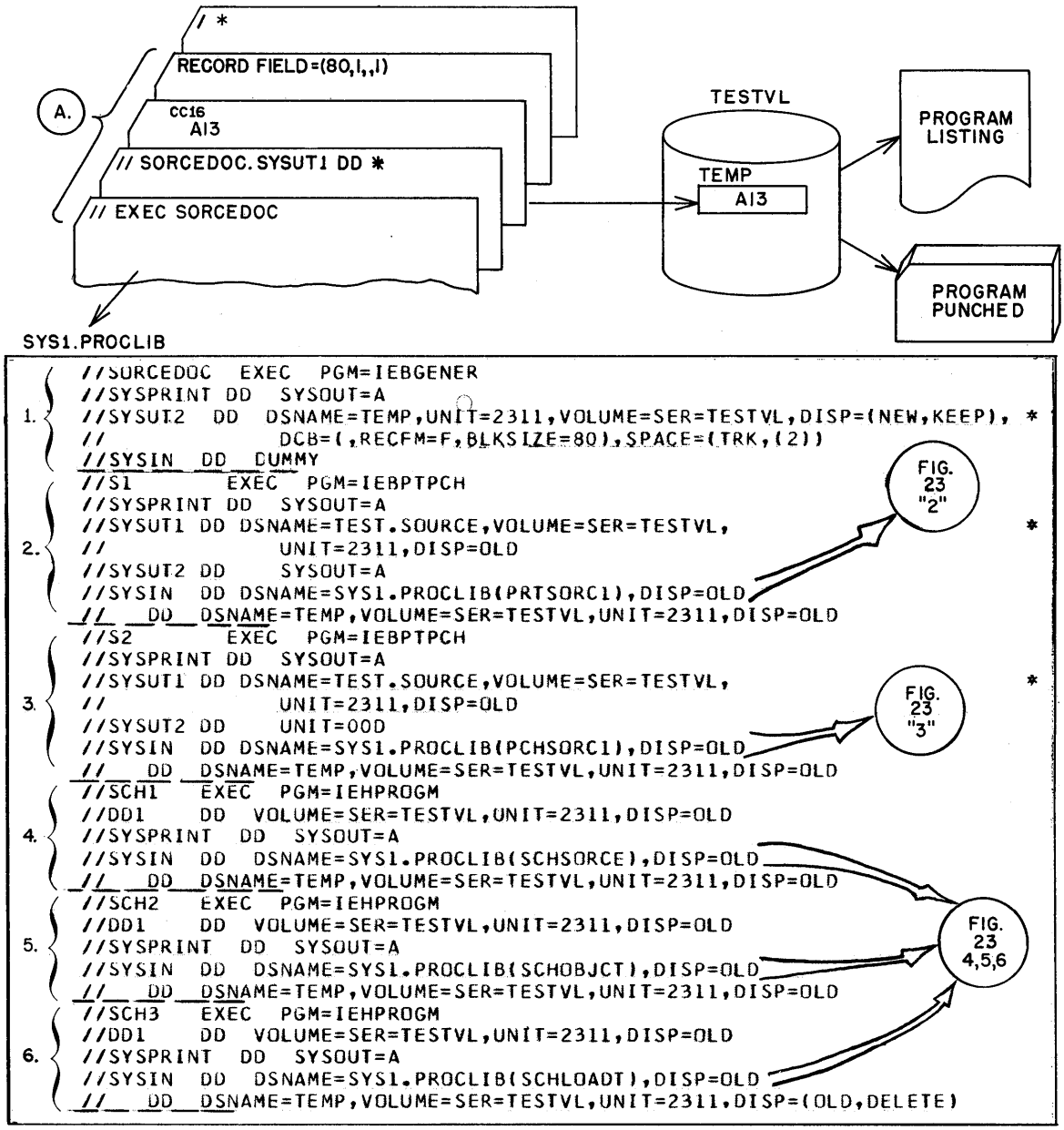
Note: In the TESTs environment it would be desirable to have the SYS1.PROCLIB on the TESTVL volume. This would allow the procedures for TESTs to be mounted only when the testing is in process and would leave system residence SYS1.PROCLIB space open for more universal procedures. It should be noted, however, that if the SYS1.PROCLIB were on TESTVL and pointed to at IPL time, DD cards in this writeup that reference SYS1.PROCLIB would require ~~Two~~ additional parameters that is, VOLUME=SER=TESTVL. UNIT=2311 This would eliminate a catalog search.

Figure 20. To list TESTS system control data



1. One card required to invoke the CLEAN procedure.
2. The CLEAN procedure is located in SYS1.PROCLIB.
3. Control statement for the utility IEHMOVE located in SYS1.PROCLIB under the name CLEAN1. This single statement moves TESTS volume to itself.
4. Conceptually, the action that takes place. The running time depends on the number of data sets and members within the data sets. The old data sets are deleted.

Figure 21. To reduce extent requirements on a volume



A. JCL required to execute procedure SORCEDOC.

1. Brings program name (in this example, A13) from card reader and stores it in a newly created sequential data set called TEMP.

2. Prints program — A13 (source).

3. Punches program — A13 (source).

4,5,6. Scratches member A13 from TEST.SOURCE, TEST.OBJECT, TEST.LOAD, respectively.

Figure 22. To print and punch a source program and delete it from the source, object and load libraries

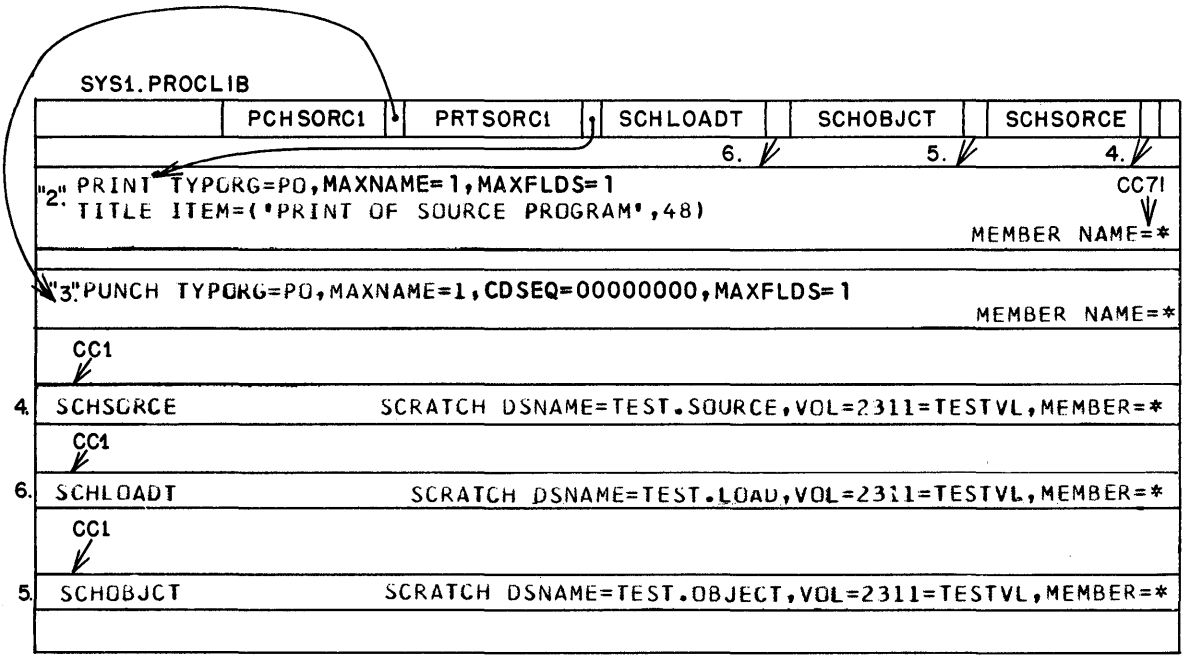
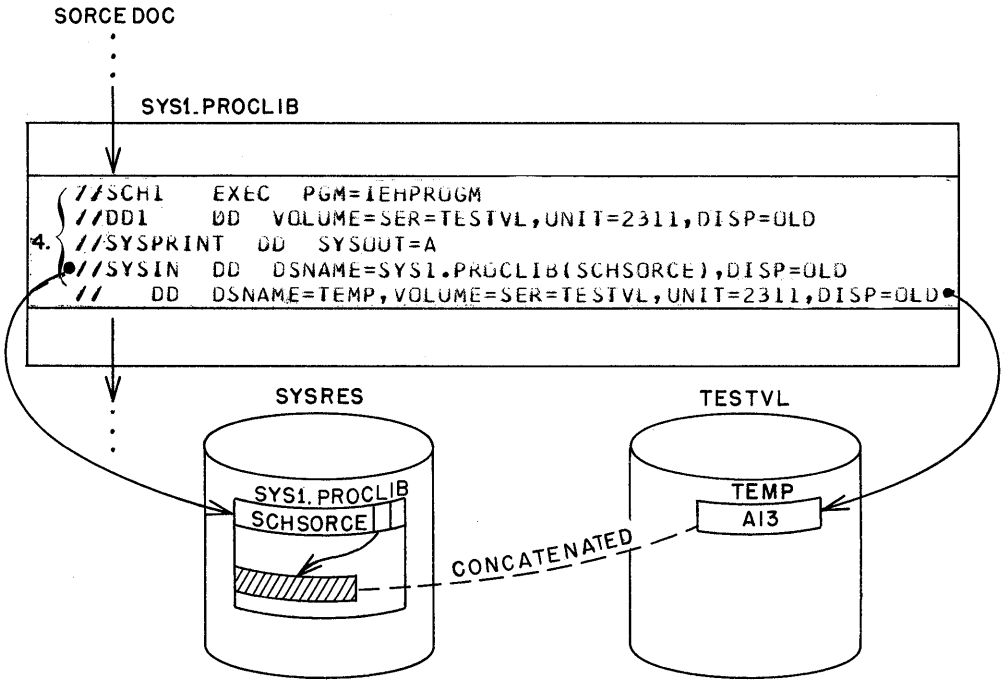
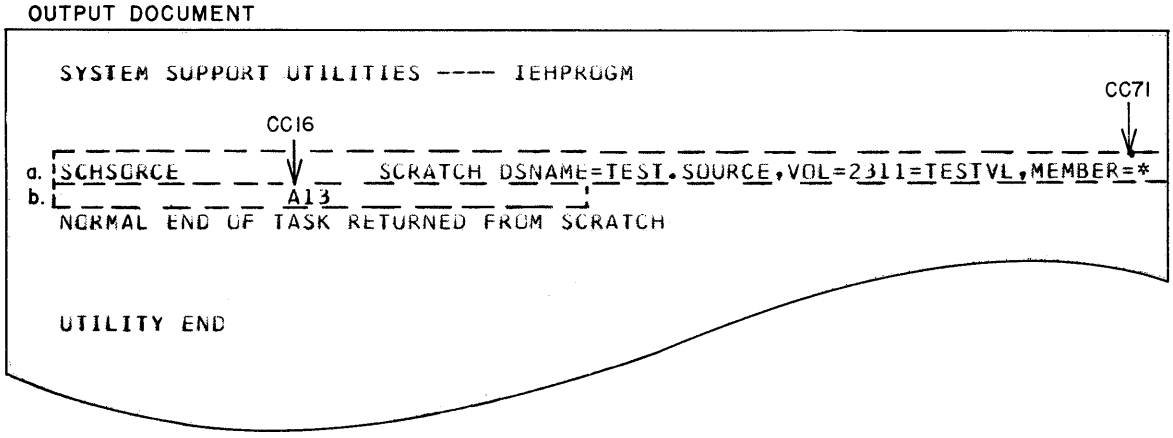


Figure 23. PRINT, PUNCH, and DELETE





4. See Figures 22 and 23, the fourth step.
   a. Utility control statement to scratch an unnamed member. The named member is found on TEMP, which is concatenated with the control statement.
   b. The concatenated sequential data set containing the member name.

Figure 24. To concatenate utility control statements

LIBRARY BACKUP – AUDIT TRAIL

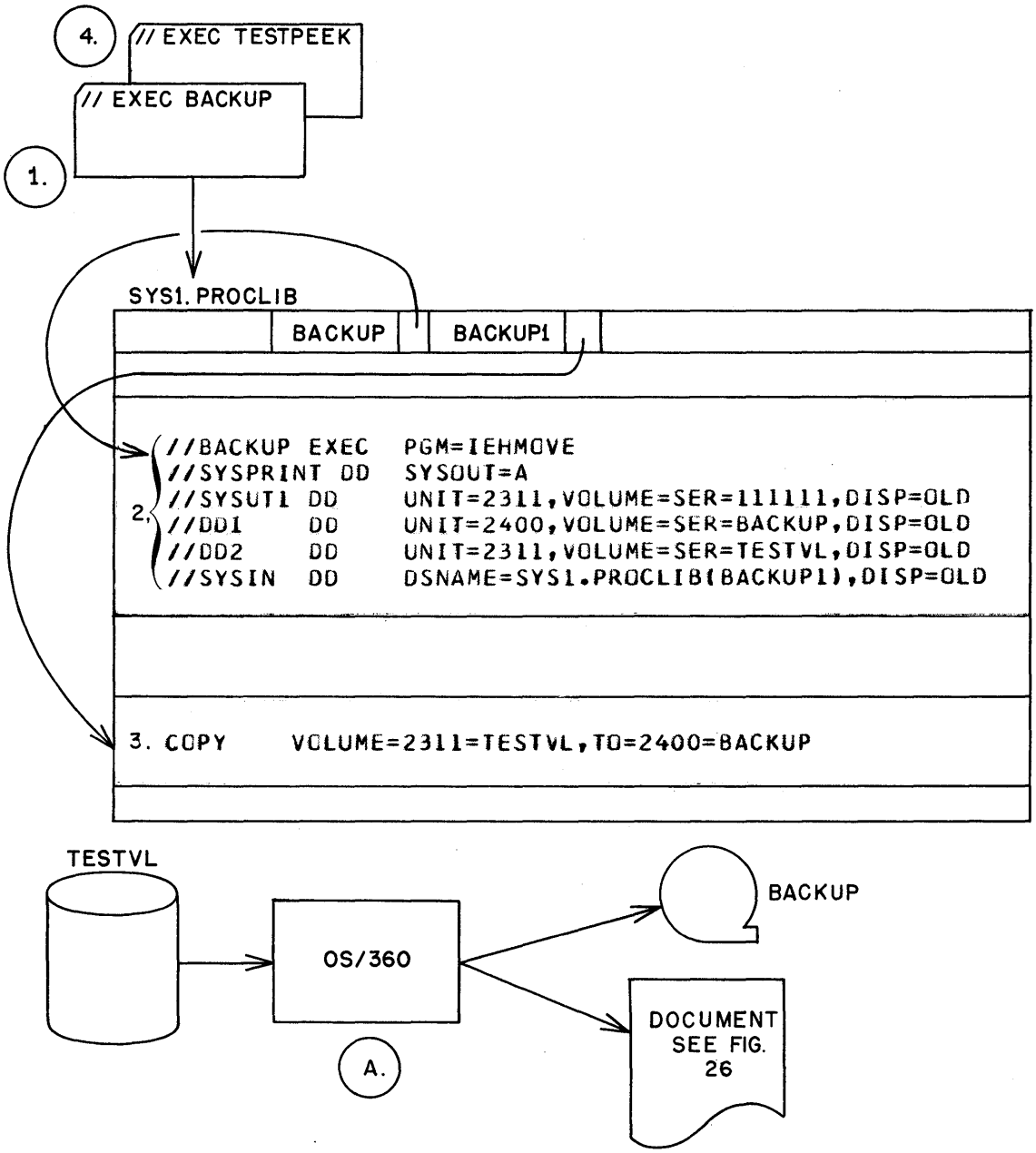
It is apparent that with such a system as described in this example, a means of protection against unforeseen circumstances is mandatory.

Backup Copy

It will be desirable to obtain a "backup" of the TESTS volume as well as a listing of its condition at the time a copy is made. The frequency with which a backup copy should be made will depend upon the volume of testing, but presumably a copy would be made at least once per shift or at the end of a large test run where multiple tests were performed.

The overall procedure for obtaining a backup copy of the TESTS volume is shown in Figure 25. Before executing the BACKUP procedure, a standard volume label of "BACKUP" must be written on a tape reel, with the eleventh byte an EBCDIC zero. (see c28 6626)

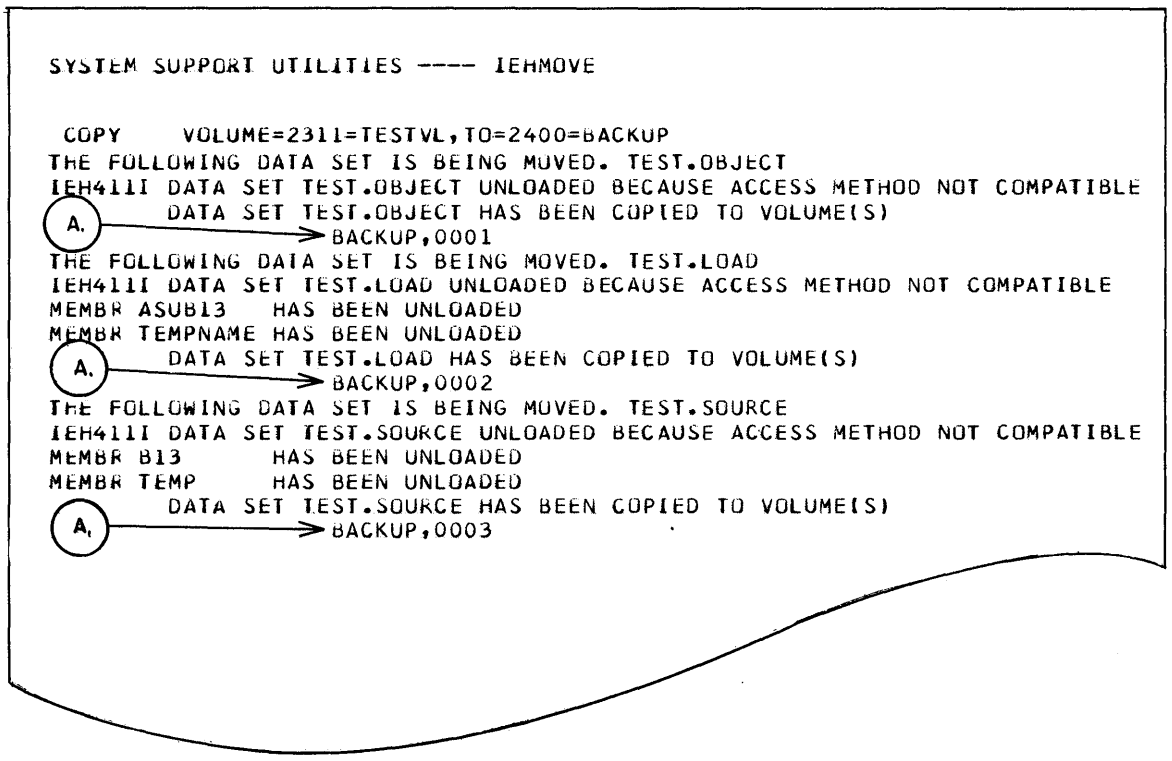
Excluding the listing received from TESTPEEK, Figure 26 illustrates the document received when the BACKUP procedure is executed.



1. Invokes the BACKUP procedure.
2. JCL for the BACKUP procedure.
3. Utility control statement for IEHMOVE.
4. Invokes the TESTPEEK procedure (see Figure 20).

A. Overall flow.

Figure 25. To obtain a backup copy of the TESTS volume



A. Note: These are sequence numbers assigned to the data sets on tape. These numbers will be used to retrieve the libraries (see Figure 28).

Note also that all data sets have been put on tape in an "unloaded" version (C28-6586). This is perfectly all right, because, when they are returned to disk, they are returned as they were originally.

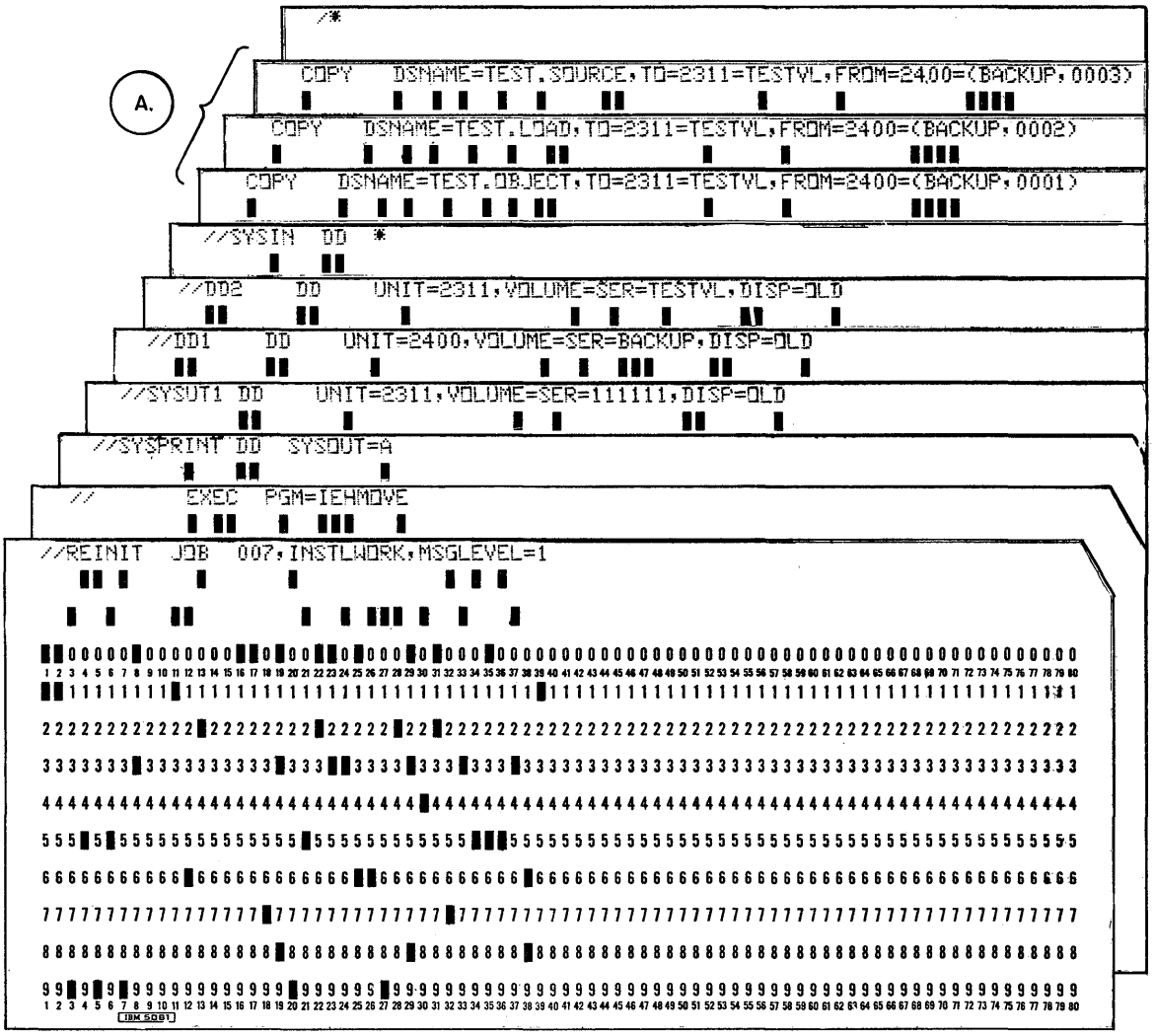
Figure 26. Document received from BACKUP procedure

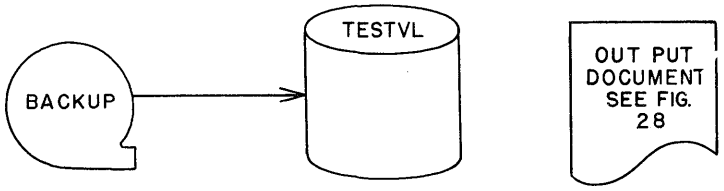
Reinitialize TESTS

If the TESTS volume should be damaged, it must be reinstated to its condition at the time the last BACKUP procedure was executed. Since this reinitializing will be performed infrequently, the job control statements to accomplish this are maintained in a card deck rather than in SYS1.PROCLIB.

After a volume has been initialized using DASDI (see Figure 6), the MOVE/COPY utility for data sets will copy the three data sets (source, object, load) onto the disk volume (see Figure 27). If additional data sets were on the original volume, these could be retrieved at this time by reviewing the listing from the BACKUP procedure and observing the sequence number of the data set (Figure 26). Figure 28 shows the results of this copy of data sets from tape to TESTVL.

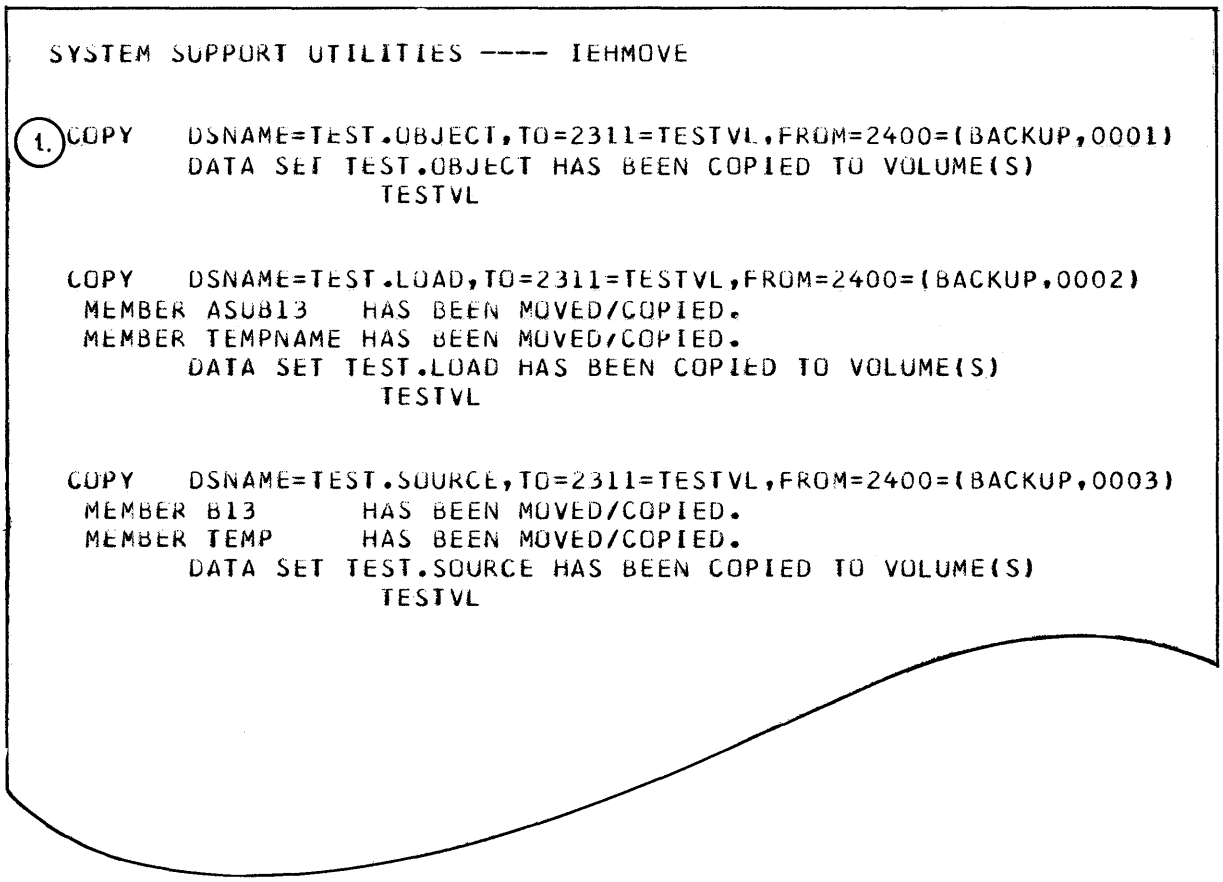
Note: The MOVE/COPY volume utility will MOVE/COPY with direct access as the FROM device only. Since FROM (in the MOVE/COPY volume utility) may not refer to a non-direct access device such as tape, we must use the MOVE/COPY for data sets rather than volume in order to retrieve the data sets from tape.





A. Three COPY utility statements to retrieve the three TESTS libraries. Notice that the sequence number (BACKUP, 0001) specification corresponds to the sequence number assigned when BACKUP was executed (see Figure 26).

Figure 27. To reinitialize the TESTS volume



1. Note: Although there were no members in this library, the data set still exists, and the original space that had been allocated is still in effect.

Figure 28. Document received from reinitializing the TESTS volume (see Figure 27 for execution)

MULTIPLE JOB FLOW IN "TESTS"

Figure 29 illustrates the processing of multiple programs through the TESTS environment. The purpose of this figure is to illustrate the logical flow, rather than the actual format of the statements. The operations involve programs A13, SUBA13, B13, C13, and D13.

A13. To be modified and reassembled.

SUBA13. To be entered as a new source module and link edited with A13 to form the new executable module ASUB13.

B13. Entered as a new source module to be compiled by COBOL and executed.

C13 and D13. Both have individually completed the test cycle and are to be combined into a new load module (CD13) for execution. Both are members of the object and load module libraries and could be combined in either format. This example combines the object modules.

The processing illustrated in Figure 29 has been separated into four jobs:

JOBA enters the source decks into the source library or modifies a module already there.

JOBB assembles or compiles the source modules onto the object library.

JOBC link-edits the object modules into the load library.

JOBD executes the programs from the JOBLIB (TEST. LOAD) in successive job steps.

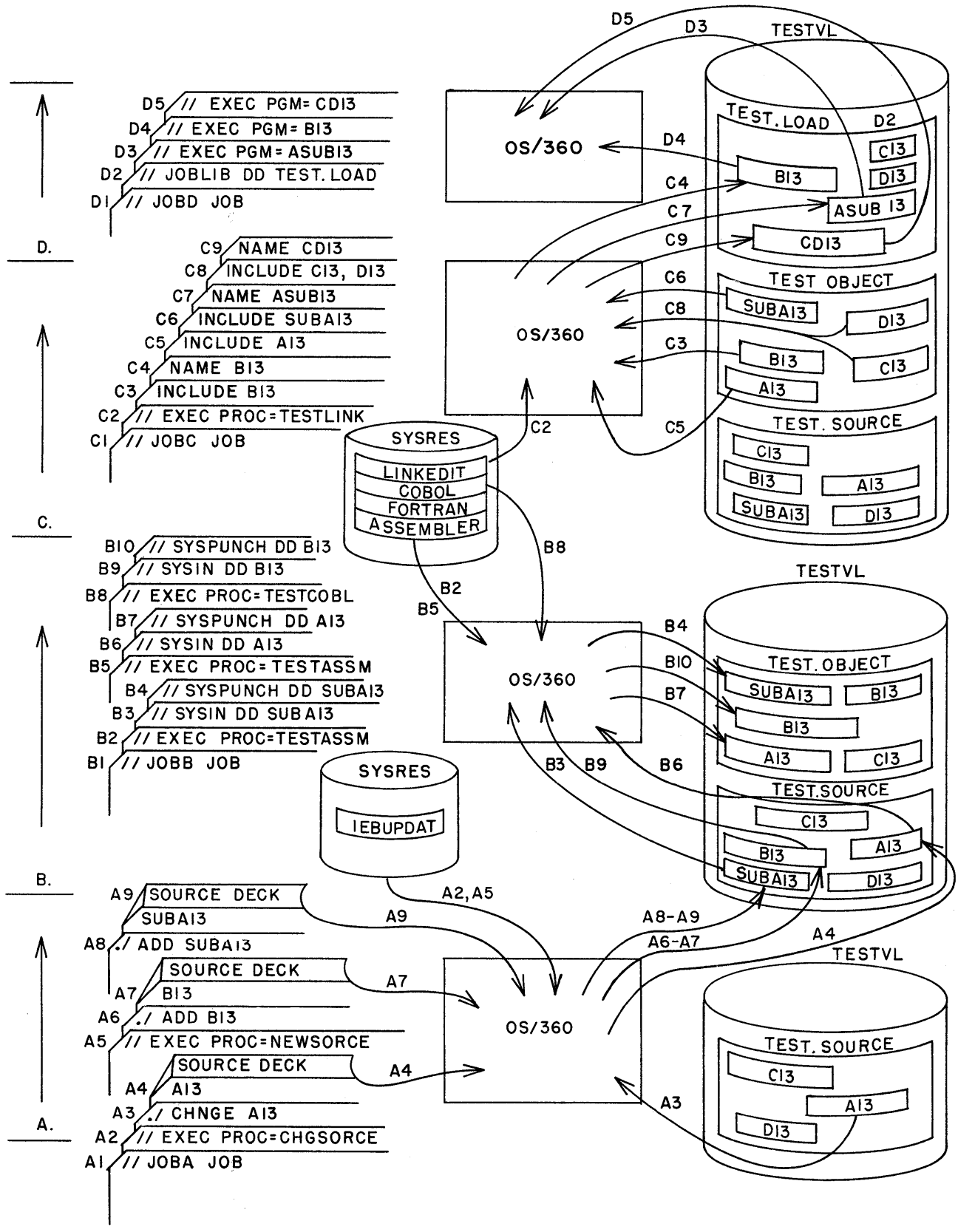


Figure 29. Job flow of multiple programs from source to execution

Job Stream A	Processing Incurred
A1. //JOBA JOB	Job statement indicating start of JOBA.
A2. //EXEC PROC=CHGSORCE	Invokes the cataloged procedure to update programs already on the source library.
A3. ./ CHNGE A13	Indicates the name of the program (A13) to be updated in the source library.
A4. SOURCE for A13	The source statements which will update A13.
A5. //EXEC PROC=NEWSORCE	Invokes the cataloged procedure to enter new programs in the source library.
A6. ./ ADD B13	Indicates the name of the program (B13) to be added to TEST.SOURCE.
A7. B13	The deck of source statements comprising B13.
A8. ./ ADD SUBA13	(See A6.) In this case SUBA13 is the new subroutine to be added to TEST.SOURCE.
A9. SUBA13	Source statements for SUBA13.
Job Stream B	
B1. //JOBB JOB	Specifies JOBB, in this application the language translation.
B2. //EXEC PROC=TEST ASSM	Invokes the cataloged procedure for Assembler.
B3. //SYSIN DD SUBA13	Represents the DD statement which specifies the name of the module (SUBA13) to be assembled.
B4. //SYSPUNCH DD SUBA13	Specifies to the assembler the name (SUBA13) to be given the output module on the object library.
B5. //EXEC PROC=TESTASSM	(See line B2.) Second job step.
B6. //SYSIN DD A13	(See B3.) In this case, program A13 which has just been modified is to be recompiled.
B7. //SYSPUNCH DD A13	(See B4.) The new object module will replace the one named A13 previously placed in the object library.
B8. //EXEC PROC=TESTCOBL	Invokes the cataloged procedure for COBOL.
B9. //SYSIN DD B13	Specifies that the name of the COBOL source module to be compiled is B13.
B10. //SYSPUNCH DD B13	Specifies that B13 is the name to be assigned to the compiled object module.
Job Stream C	Processing Incurred
C1. //JOBC JOB	Job statement-start of JOBC. In this case, only one job step occurs.
C2. //EXEC PROC=TESTLINK	Invokes the cataloged procedure to link-edit object modules into load modules.
C3. INCLUDE B13	Linkage Editor control statement indicating that the name of the first object module to be link edited is B13.
C4. NAME B13	Specifies that the resultant new load module which becomes a member of TEST.LOAD is to be named B13.
C5. INCLUDE A13	Linkage Editor control statement indicating that the second object module to be link-edited is called A13.
C6. INCLUDE SUBA13	(See C5.) Indicates that SUBA13 is to be link-edited with A13.
C7. NAME ASUB13 (R)	Specifies that the resultant load module consisting of A13 and SUBA13 is to be entered into TEST.LOAD and ASUB13.
C8. INCLUDE (C13,D13) (Performs same function as C5 and C6)	Linkage Editor control statement indicating that the input for this load module will be object modules C13 and D13.
C9. NAME CD13 (R)	Linkage Editor control statement specifying the resultant load module consisting of C13 and D13 should be entered on TEST.LOAD as CD13.
Job Stream D	
D1. //JOBD JOB	Job statement indicating start of JOBD.
D2. //JOBLIB DD TEST.LOAD	JOBLIB points to the library, TEST.LOAD, containing the programs to be executed in this job step.
D3. //EXEC PGM=ASUB13 (DD statements for ASUB13 not shown)	Specifies that the first job step will execute program ASUB13, followed by the appropriate DD statements defining the input and output data sets for this program.
D4. //EXEC PGM=B13 (DD statements for B13 not shown)	The second job step will execute program B13. (See line 3.)
D5. //EXEC PGM=CD13 (DD statements for CD13 not shown)	The third job step will execute program CD13. (See line 3.)

Figure 29 (continued).

Printed in U.S.A. C20-1663-0

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
(USA Only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

READER'S COMMENTS

IBM System/360 Operating System
User Libraries (C20-1663-0)

Your comments regarding this publication will help us improve future editions. Please comment on the usefulness and readability of the publication, suggest additions and deletions, and list specific errors and omissions.

USEFULNESS AND READABILITY

SUGGESTED ADDITIONS AND DELETIONS

ERRORS AND OMISSIONS (give page numbers)

Name_____

Title or Position_____

Address_____

FOLD ON TWO LINES, STAPLE AND MAIL
No Postage Necessary if Mailed in U.S.A.

fold fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N.Y.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
112 East Post Road
White Plains, N. Y. 10601

Attention: Technical Publications

fold fold

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
(USA Only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

Printed in U.S.A. C20-1663-0