Honeywell

LEVEL 6

SOFTWARE

GCOS 6 MOD 400 PROGRAMMER'S GUIDE

SERIES 60 (LEVEL 6)

GCOS 6 MOD 400 PROGRAMMER'S GUIDE

SUBJECT

Descriptions of Mod 400 Operating Environments, User Access to the System, and Selected Examples of the Use of System Software Components

SOFTWARE SUPPORTED

This publication supports Release 0100 of the Series 60 (Level 6) GCOS 6 MOD 400 Operating System; see the Manual Directory of the latest *GCOS 6 MOD 400 System Concepts* manual (Order No. CB20) for information as to later releases supported by this manual.

ORDER NUMBER

CB22, Rev. 0

January 1978

Preface

The purpose of this manual is to provide the user with programmer-oriented information regarding the various operating environments available under the GCOS 6 Mod 400 Operating System and programmer procedures for terminal startup and access to the system. Also contained in this manual are examples of the use of various system software components: the editor, macro preprocessor, assembler, COBOL and FORTRAN compilers, and the sort program.

This material is presented in 9 sections, as outlined in the Introduction. The Introduction also presents suggested usages of the Mod 400 manual set for application programmers, system programmers, and operators.

File No.: 1S23

MANUAL DIRECTORY

The following publications constitute the GCOS 6 manual set. The Manual Directory in the latest GCOS 6 MOD 400 System Concepts manual lists the current revision number and addenda (if any) for each manual in the set.

Order No.	*Manual Title*
CB01	*GCOS 6 Program Preparation*
CB02	*GCOS 6 Commands*
CB03	*GCOS 6 Communications Processing*
CB04	*GCOS 6 Sort/Merge*
CB05	*GCOS 6 Data File Organizations and Formats*
CB06	*GCOS 6 System Messages*
CB07	*GCOS 6 Assembly Language Reference*
CB08	*GCOS 6 System Service Macro Calls*
CB09	*GCOS 6 RPG Reference*
CB10	*GCOS 6 Intermediate COBOL Reference*
CB20	*GCOS 6 MOD 400 System Concepts*
CB21	*GCOS 6 MOD 400 Program Execution and Checkout*
CB22	*GCOS 6 MOD 400 Programmer's Guide*
CB23	*GCOS 6 MOD 400 System Building*
CB24	*GCOS 6 MOD 400 Operator's Guide*
CB25	*GCOS 6 MOD 400 FORTRAN Reference*
CB26	*GCOS 6 MOD 400 Entry-Level COBOL Reference*
CB30	*Remote Batch Facility User's Guide*
CB31	*Data Entry Facility User's Guide*
CB33	*Level 6/Level 6 File Transmission*
CB34	*Level 6/Level 62 File Transmission*
CB35	*Level 6/Level 64 (Release 0300) File Transmission*
CB36	*Level 6/Level 66 File Transmission*
CB37	*Level 6/Series 200/2000 File Transmission*
CB38	*Level 6/BSC 2780 File Transmission*
CB39	*Level 6/Level 64 (Release 0220) File Transmission*

In addition, the following documents provide general hardware information:

Order No.	*Manual Title*
AS22	*Honeywell Level 6 Minicomputer Handbook*
AT04	*Level 6 System and Peripherals Operation Manual*

Contents

Section 1. Introduction

Section 2. Operating Environments

Section 3. User Terminal Startup

Section 4. User Access to the System

Section 5. Using the Editor

Section 6. Using the Assembler and Macro Preprocessor

Section 7. Using the COBOL Compiler

Section 8. Using the FORTRAN Compiler

Section 9. Using the Sort

Figures

Section 1
Introduction

The GCOS 6 Mod 400 operating system for the Models 6/30 and 6/40 minicomputers provides a comprehensive set of system services which form a base for executing user-written applications, Honeywell-supplied applications, and program development tools. It provides an online, interrupt-driven operation for multiple users and a single, low-priority batch operation typically used for program development and associated activities.

A number of different operating environments are possible, controlled in part by options exercised at system configuration, and in part by options chosen by the system operator at startup or at various times during the operating day. These environments are more fully described in Section 2, "Operating Environments."

Access to the system by users can be achieved in a variety of ways, again depending in part on system configuration options selected. These options are concerned mainly with the definition of local and/or remote terminal devices and how they are connected to the system. These are described in Section 3, "User Terminal Startup." Other access options, normally under the control of the system operator, are concerned with the procedures by which a user identifies himself (logs in) to the system through a connected terminal. This subject is treated in Section 4, "User Access to the System."

The remaining sections comprise descriptions and examples of the use of various system components: the Editor (Section 5), the Assembler and Macro Preprocessor (Section 6), the COBOL Compiler (Section 7), the FORTRAN Compiler (Section 8), and the Sort component (Section 9). Each of these sections presents terminal and/or line printer listings representing the actions performed. In these listings, heading lines may vary in detail depending on the component that initiated the listing or, in some cases, may be omitted. However, in actual use, the user will see heading lines consisting of three major fields of information, as shown below.

1. System Identification: GCOS6 MOD400- $\genfrac{}{}{0pt}{}{S}{L}$ rrr-mm/dd/hhmm

   S — SAF
   L — LAF
   rrr — Release number of the operating system
   mm/dd/hhmm — Date/time when operating system was created (month, day, hour, and minute)

2. Component Identification: xxxxx-rrrr-mm/dd/hhmm

   xxxxx — Component name
   rrrr — Revision number of component
   mm/dd/hhmm — Date/time that specified revision of component was created (month, day, hour, and minute)

3. Time of program execution: yyyy/mm/dd hhmm:ss.t

   Date/time of program execution (year, month, day, hour, minute, second, and tenth of second)

GUIDE TO USING THE MANUAL SET

This guide to the manuals is arranged according to functions that might be performed by an applications programmer, a systems programmer, or an operator. As used in this guide, the applications programmer writes applications programs; the system programmer configures the system and defines the environment for each application; and the operator operates the system from the operator terminal. These functions could be performed by three different persons or by the same person serving in the different capacities.

APPLICATIONS PROGRAMMER'S MANUAL GUIDE

Figure 1-1 illustrates the suggested sequence in using the manuals. If you wish to start using the system by writing an application program, begin by using the *Programmer's Guide* manual. It illustrates: (1) various ways to gain access to the system, (2) a sample Editor session, and (3) for application languages, the procedure for performing program preparation and execution. Working with the small subset of commands used in the examples is a good approach to learning the system command set. This approach for getting started assumes that a system programmer has already configured and started up a suitable application environment. While using the system, you may wish to familiarize yourself with the system facilities described in the *System Concepts* manual.

Through examples, the *Programmer's Guide* illustrates how to use the system facilities. Other manuals provide reference material. The *Program Preparation* manual contains Editor directives (statements) to create and update an application language source unit. For each of the languages the appropriate language reference manuals contain the description of the language statements. Operating system dependencies, if any, that affect how you write the application are described in the *Programmer's Guide*. If the application uses communications, refer to the *Communications Processing* manual. Read the *Data File Organizations and Formats* manual if you require a better understanding of a language-supported file organization that is to be used in an application, or if you must calculate the size of a data file. You can use Monitor macro calls, as described in the *System Service Macro Calls* manual, in assembly language programs. Before your program can be entered for execution, it must be linked as described in the *Program Execution and Checkout* manual.

For program compilation or assembly and execution, the procedures described in the *Programmer's Guide* might be sufficient. To obtain more control over the execution of your program or utilize the system facilities more completely or efficiently, use the commands described in the *Commands* manual. If you wish to use the operator terminal, read the *Operator's Guide*. In many cases, the description of commands must be supplemented by system concepts described in the *System Concepts* manual. Rather than read all the conceptual material at one time, you may find it more meaningful to refer to it in conjunction with the appropriate reference material. The *Commands* manual also describes the utilities. An assembly language program, the Patch, Debug, and Dump utilities are described in the *Program Execution and Checkout* manual; file transmission from Level 6 to a host system is described in the *File Transmission* manual appropriate to the host system. Error messages and return status codes are listed in the *System Messages* manual.

SYSTEM PROGRAMMER'S MANUAL GUIDE

Figure 1-2 illustrates the suggested sequence for using the manuals. The *System Building* manual provides you with the configuration directives (statements) and startup procedures to configure and start up a MOD 400, a Remote Batch Facility (RBF), or a Data Entry Facility (DEF) system. You must know the conceptual material in the *System Concepts* manual in order to successfully use the configuration directives. To tailor an applications environment suitable for the intended application, use the operator commands described in the *Operator's Guide* manual. Error messages are listed in the *System Messages* manual. If you are working with an application that runs under the BES operating system, the *System Concepts* manual contains MOD 400 and BES compatibility considerations.

OPERATOR'S MANUAL GUIDE

Figure 1-3 illustrates the suggested sequence for using the manuals. Specific operator job functions must be determined by each installation; a large system might have a person assigned as an operator; a small system might have each programmer also act as an operator. The *Operator's Guide* indicates the system procedures performed through the operator terminal and describes operator commands used in system operation.

The *Programmer's Guide* contains examples using commands (described in the *Commands* manual) that are similar to operator commands. The *System Concepts* manual provides an understanding of the operating system. Note that the *Operator's Guide* describes using the

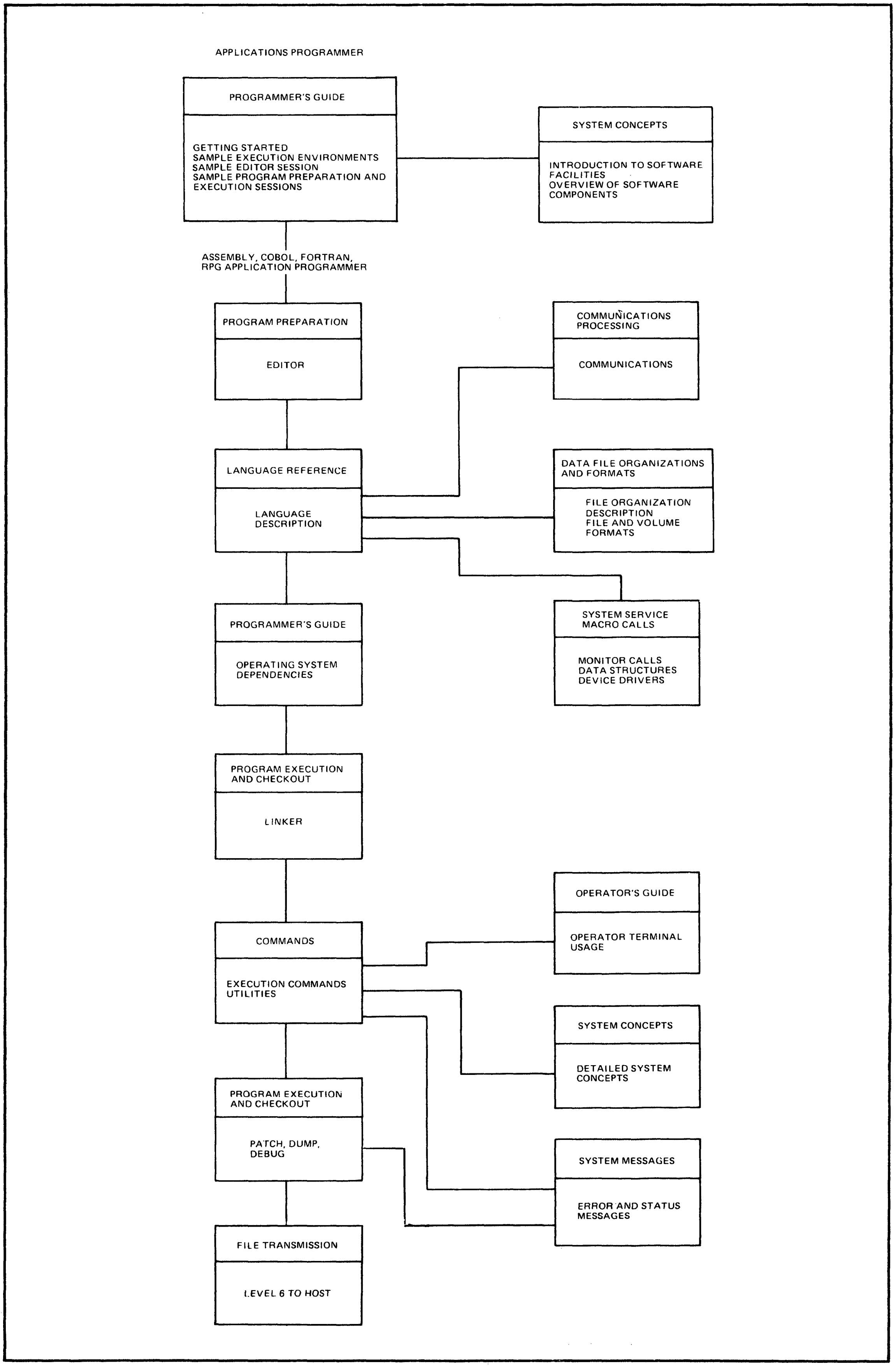


Figure 1-1. Applications Programmer Guide to Manuals

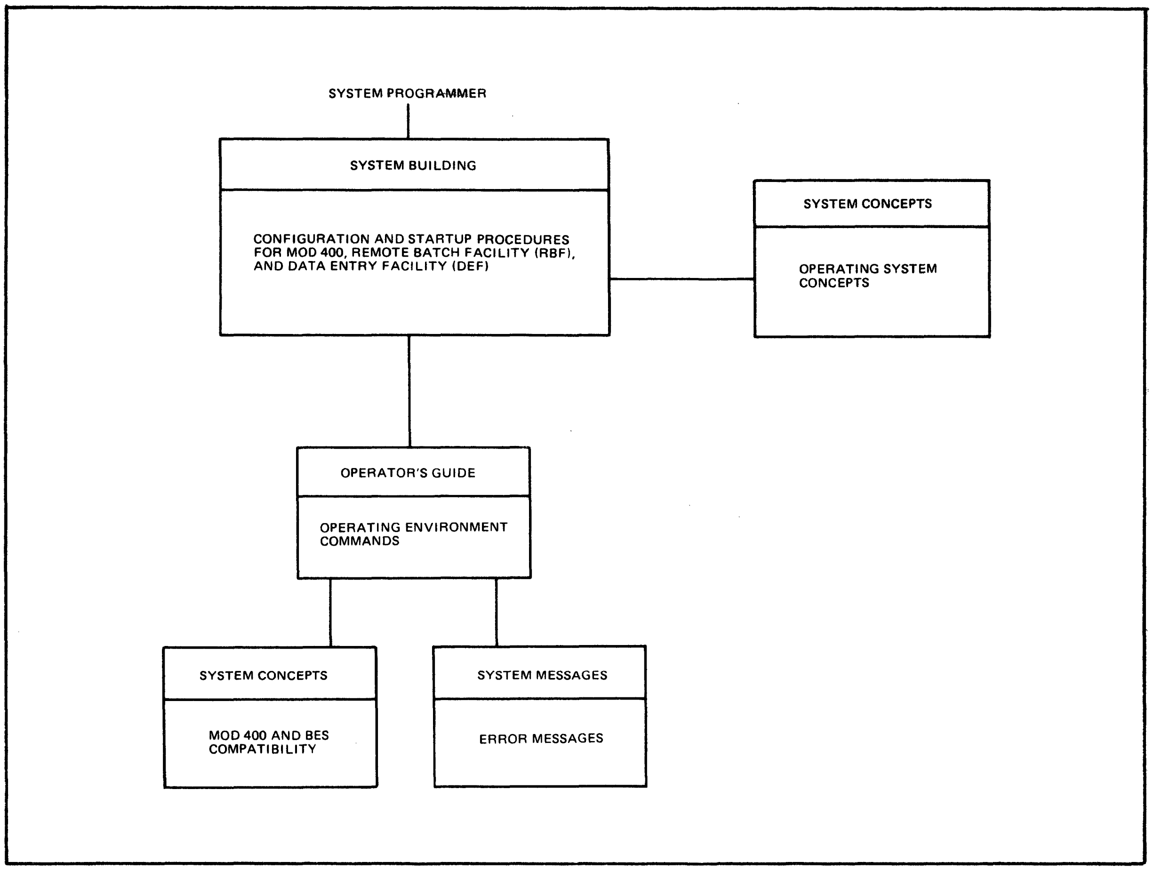

SYSTEM PROGRAMMER
SYSTEM BUILDING
CONFIGURATION AND STARTUP PROCEDURES FOR MOD 400, REMOTE BATCH FACILITY (RBF), AND DATA ENTRY FACILITY (DEF)
SYSTEM CONCEPTS
OPERATING SYSTEM CONCEPTS
OPERATOR'S GUIDE
OPERATING ENVIRONMENT COMMANDS
SYSTEM CONCEPTS
MOD 400 AND BES COMPATIBILITY
SYSTEM MESSAGES
ERROR MESSAGES

Figure 1-2. System Programmer Guide to Manuals

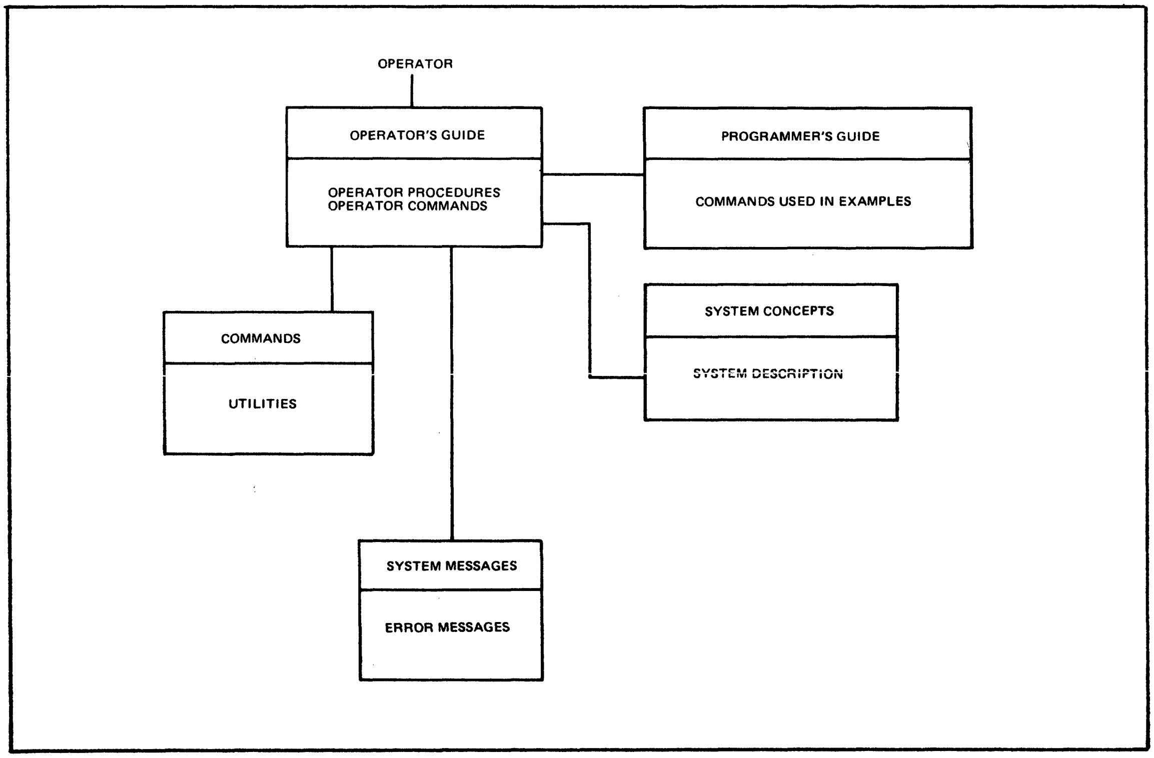


Figure 1-3. Operator Guide to Manuals

operator terminal for operator functions to enter operator commands to the system task group, or for user functions to enter commands to a user task group. To run the utilities, use the commands (described in the *Commands* manual) entered through the operator terminal functioning as a user terminal. Error messages are listed in the *System Messages* manual.

RBF AND DEF USER MANUAL GUIDE

Figure 1-4 illustrates the suggested sequence for using the manuals. The system programmer configuration functions have been done and the system is ready to be used for Remote Batch Facility (RBF) functions or Data Entry Facility (DEF) functions. The *Programmer's Guide* manual provides sample login execution environments typical of ones that might be at your facility. The *Remote Batch Facility User's Guide* is used for RBF operations and the *Data Entry Facility User's Guide* is used for DEF operations.

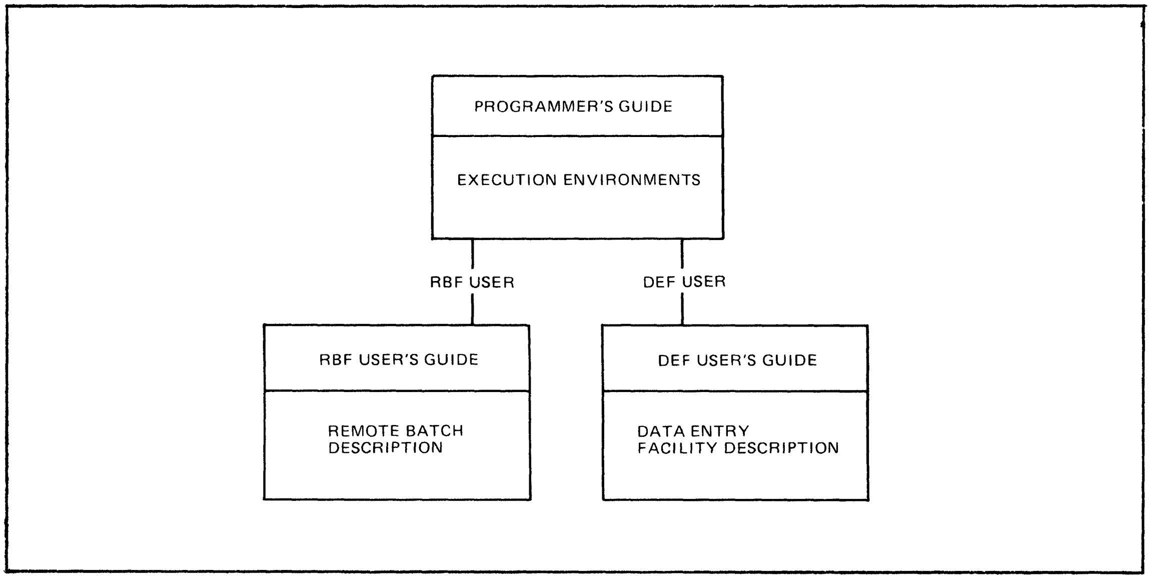


Figure 1-4. RBF and DEF User Guide to Manuals

Section 2
Operating Environments

The Mod 400 operating system allows a wide variety of operating environments, ranging from a single operator-controlled configuration to one in which the operator, other users, or a combination if both can control the configuration at any time during the operating day. This range of operating environments is described in this section.

OPERATOR-ONLY ENVIRONMENT

This environment is one in which a designated operator and a limited number of users (typically programmers developing application programs) use the system on a first-come first-served basis for developing and testing programs. All work is done through the operator terminal, through either the system task group or a single online task group created by the system startup procedure. Certain functions can be performed through either of the two task groups; others can be done only through the system task group or the online task group — refer to the *Operator's Guide* and the *Commands* manuals for details on which functions can be performed from each task group.

ALL-ONLINE ENVIRONMENT

An all-online environment is one in which one or more users can concurrently use the facilities of the operating system to perform interactive tasks of any kind permitted by the command language described in the *Commands* manual, plus any user applications that can be invoked through the command processor. This latter category consists of user programs in the form of bound units that are called from a task group in which the command processor is declared as the lead task when the task group is created. A task group can also be created by the operator or another online user, declaring the application bound unit as the lead task; in this case the creation of the task group and its activation results directly in the execution of the declared bound unit, without the need to enter its name as a command.

An example of this kind of environment is one in which several task groups have the command processor as lead task and one or more other task groups have specific application programs such as the Data Entry Facility and user-created programs as lead tasks. The former task groups can be used for editing source program files, entering requests for jobs to be run in the batch task group (see below), requesting printouts of files, etc. Concurrent with these activities can be the execution of the user application programs constituting the latter set of task groups. From the user's point of view, each task group has the appearance of having control of the system.

ONLINE/BATCH ENVIRONMENT

This environment differs from the all-online environment only in that, in addition to the creation of the online task groups, a batch task group has also been created by the designated operator from the operator terminal. Once this task group has been created, any online task group having the command processor as its lead task can enter requests for jobs to be run through the use of the EBR (ENTER BATCH REQUEST) command. Typical of such batch jobs would be requests for compilations, links, application program checkout runs, and the like.

Creation and utilization of the batch task group requires the existence of at least the designated operator terminal, through which the batch task group is created and through which requests to it can be entered. Jobs run in the batch task group are normally controlled by a previously created file containing commands directing the execution of the jobs, and not by interactive dialog from a terminal. Section 4 contains additional information on the use of the batch task group.

DEDICATED APPLICATION ENVIRONMENT

This is an environment in which system startup or operator action subsequent to startup results in the creation of one or more task groups in which a user application, and not the command processor, is the lead task. In such an environment no interactive processing takes place; rather, whatever processing occurs is dependent on the nature of the application — e.g., data entry, an inventory application, etc.

MIXED ENVIRONMENT

The Mod 400 system does not restrict the user to any one of the foregoing environments at any given time. Given a large enough system, any of these can be combined with any others to provide concurrent interactive, batch, and dedicated operations on a selected terminal basis. That is, a selected set of terminals can be associated with interactive tasks, while others can be related to the dedicated application tasks.

Section 3
User Terminal Startup

Terminal startup procedures vary according to the type of terminal and whether the terminal is a noncommunications or a communications terminal. A noncommunications terminal is one that is connected to the system through the multiple device controller (MDC), while a communications terminal is connected through the multiline communications processor (MLCP). An MLCP-connected terminal can be connected either through a modem or through a dial-up telephone line. In the former case, when the modem is made ready, the terminal is connected and ready for operation. With a dial-up connection, the user must dial the number which connects the telephone to the system and wait for the signal that indicates the connection is made. For an MDC-connected terminal, simply turning on the power to the terminal suffices to connect it to the system. Subsequent actions depend on whether or not the listener/login processor is activated, and whether the terminal is declared by the operator to be associated with a specific task group.

STARTUP WITH THE LOGIN FACILITY

If the operator has activated the listener/login processor, and the terminal being started up is one which is monitored by the login processor, then the user must log in using the procedures described in the *Operator's Guide* manual after performing the actions required for physically connecting the terminal to the system. When the connection is made to an MLCP-connected terminal, the system will display a system identification message, a message of the day if one is defined, and indicate that it is ready to accept a login request. For an MDC-connected terminal, displays occur only if the terminal is active when the login processor is activated.

TASK GROUP-SPECIFIC TERMINAL STARTUP

If a terminal is declared by the operator to be associated with a specific task group, and is not monitored by the login processor, then, when the terminal is connected, it is ready to accept whatever input or output is dictated by the logic of the task's execution. If the command processor is the lead task, a ready message will be issued, indicating that the terminal is ready to accept commands. If a user application is the lead task, it should issue a message to the terminal indicating that it has recognized the availability of the terminal and is ready for execution.

Section 4
User Access To The System

Once a terminal has been connected to the system as described in Section 3, a user can gain access to the system in any of several ways. Which of these ways is used at any given time depends upon operator actions taken during and after system startup. Examples of various access procedures are given in this section. Each example states any prerequisite operator actions which would have been performed.

ACCESS BY LOGGING IN

Configuration and system startup have been done. The operator, through the system task group, has activated the login function as described in the *Operator's Guide* manual. The terminal is one which is connected through the MDC, and was active when the login function was activated. The system message of the day has been displayed and the login prompter message has been printed. The user's login procedure at this point depends on the terminal login characteristics for this terminal. Procedures are described below.

DIRECT LOGIN TERMINAL

In addition to the message of the day, if the command processor is the lead task, the ready message will have been displayed, and no further action is required. The user can begin to enter commands.

If an application is the lead task, further action, if any, depends on the characteristics and logic of the application.

ABBREVIATED LOGIN TERMINAL

After the login prompter message has been issued, the user enters a one-character abbreviation such as

```
  A
```

If the login line corresponding to the abbreviation "A" indicates that the command processor is the lead task, the system responds with the ready message, and the user can then begin to enter commands.

If an application is the lead task, further action, if any, depends on the characteristics and logic of the application. A terminal that accepts an abbreviated login also accepts a full login command line.

FULL LOGIN TERMINAL

After the login prompter message has been issued, the user must enter a full login line as described in the *Commands* manual. If the login line specifies or implies that the command processor is the lead task, the system responds with the ready message, and the user can begin to enter commands.

If the lead task is an application, further action, if any, depends on the characteristics and logic of the application.

COMMAND PROCESSOR AS LEAD TASK

For a user named W. Smith to log in to the system, specifying the command processor as the lead task, a login line such as

```
    L   SMITHW   -HD     ^ VOL22>SMITHW
```

could be used, where, ^ VOL22>SMITHW is the working directory pathname. As soon as the ready message is displayed, the user can begin to enter commands for either serial or concurrent execution. In particular, if the -HD argument was not used, the working directory can be specified with a CWD (CHANGE WORKING DIRECTORY) command

```
    CWD     ^ VOL22>SMITHW
```

APPLICATION AS LEAD TASK

For the same user to log in specifying a task other than the command processor, his login line could be

```
    L   SMITHW   -PO   MS_UPDATE   -HD     ^ VOL22> SMITHW
```

where MS_UPDATE is the name of the bound unit which is to be the lead task. The bound unit is located in the directory ^ VOL22>SMITHW. If it is located in some other directory, a full pathname must be used as the argument, such as

```
    L   SMITHW   -PO     ^ VOL23>MS_UPDATE   -HD     ^ VOL22>SMITHW
```

After the login line is processed, control rests with the application. It is strongly advised that the application issue some kind of message indicating that it has been successfully loaded and is ready to begin, or has begun, execution. It may be simply an informative message or a message requesting some action on the part of the terminal user.

ACCESS THROUGH THE OPERATOR OR ANOTHER USER

The system operator or an online user can create and activate an online task group through the use of the CG (CREATE GROUP) and EGR (ENTER GROUP REQUEST) commands, or through the SG (SPAWN GROUP) command. The application being run in this task group can be in the form of a series of commands implying either serial or concurrent execution, as shown below.

SERIAL EXECUTION OF APPLICATION TASKS

The operator or another user has created and activated a new online task group whose lead task is the command processor, and whose command-in file is the MDC- or MLCP-connected terminal being used by the new user.

As soon as the ready message is displayed, the new user can begin to enter commands. After each command request is terminated, indicated by the display of the ready message, control returns to command input level and another command can be entered. The following example shows the entry of commands to initiate a COBOL compilation, the assignment of the user-out file to a line printer, and the printing of the COBOL compilation listings.

```
    COBOL   PROGA   [ctl_arg]     Invoke the COBOL compiler
          .
          .
          .
    compiler responses
          .
          .
          .
```

```
RDY:                       Indicates end of compilation
FO   >SPD>LPT01            Assign user-out to line printer
RDY:                       Indicates assignment complete
PR   PROGA.L               Invoke PRINT command to print compilation output
RDY:                       Indicates printout complete
```

CONCURRENT EXECUTION OF APPLICATION TASKS

The operator or another user has created and activated a new online task group whose lead task is the command processor, and whose command-in file is the MDC- or MLCP-connected terminal being used by the new user.

As soon as the ready message is displayed, the new user can begin to enter commands. This example shows the entry of commands to initiate a COBOL compilation, the assignment of the user-out file to a line printer, and the printing of a file which is unrelated to the compilation, and thus has no time dependency upon completion of the compilation.

```
ST  1  -EFN  COBOL  [ctl_arg] Invoke COBOL compiler task at relative level 1
RDY:                          Indicates completion of the ST command;
                              compilation is in progress
FO  >SPD>LPT01                Assign user-out file to line printer
RDY:                          Indicates assignment complete
ST  3  -EFN  PR  -ARG   FILE1 Invoke PRINT task to print FILE1, unrelated to
                              compilation
RDY:                          Indicates completion of the ST command; printing
                              is in progress concurrent with compilation
```

This is an example of multitasking. The responses from the COBOL compiler, indicated in the previous example, will be interspersed with other input and output lines, depending on when they occur in relation to these lines. The user should always ensure that a ready message has occurred in response to his last command entry before making another entry.

CONCURRENT EXECUTION FROM SEVERAL TASK GROUPS

Several task groups have been created and activated, each associated with a different command-in terminal, and each having the command processor as its lead task.

As soon as the ready message appears at each terminal, the user at that terminal can begin to enter commands to do serial or concurrent application execution. The task groups are concurrently active for execution and contend with each other for system resources. Each user appears to have control of the system.

EXECUTION OF AN APPLICATION FROM THE BATCH TASK GROUP

An application environment has been specified consisting of several online task groups and the batch task group (whose lead task is the command processor).

A user can enter one or more EBR (ENTER BATCH REQUEST) commands from each of the online task groups to obtain processing in the batch task group. These requests are queued and will be satisfied on a first-in first-out basis. The EBR requires a command-in file containing commands to be executed in the batch task group. The file is normally disk-resident in the user's working directory having previously been created. If a terminal were specified as the command-in device, the user at the terminal must wait to enter a command, until the command processor processes this EBR command. Otherwise, batch processing will stall waiting for this batch request to complete.

To request is to execute the command file, PAYR_IN, on directory ^ZSYSO1>IW. The application is to compile, link, and execute an application program PAYROLL.

```
EBR   PAYR_IN   -WD   ^ZSYSO1>IW
```

The file PAYR_IN contains the following commands:

```
COBOL   PAYROLL   -LO   -COUT   >SPD>LPT01
LINKER   PAYROLL   -COUT   >SPD>LPT01
LIB   ^ZSYSO2>ZCRT
LINK   PAYROLL
MAP;QT
GET   DEPT4   2
GET   >SPD>LPT02   3
PAYROLL
BYE
```

Any time after the file PAYR_IN has been created, it can be invoked through an EBR to control batch execution. The command file can contain any combination of legitimate commands, such as compile/link/execute sequences, including any necessary file control commands (GET, REMOVE); or file print/dump commands. The main constraint is that the commands be entered into the file in the same manner as if they were being executed from the online terminal, keeping in mind any time dependencies that might exist among various tasks. Responses from the invoked commands that would normally be written to the user terminal in an online environment are written to a file PAYR_IN.AO in the working directory of the user who issued the EBR command.

EXECUTION FROM THE DATA ENTRY FACILITY (DEF)

One or more task groups whose lead task is the Data Entry Facility (DEF) have been spawned.

When DEF has indicated at the terminal that it is ready to accept data entry actions, the user can begin to enter directives. No other preliminary actions are required.

Refer to the *Data Entry Facility User's Guide* manual for details on the operation of the Data Entry Facility.

It should be noted that the presence of the Data Entry Facility in no way restricts the presence of other online task groups or the batch task group. These functions can be carried on concurrently as described in the preceding paragraphs.

ACCESS THROUGH THE OPERATOR TERMINAL

A special case of system access is that in which all interactive and/or batch executions are initiated through the operator terminal. The major difference between this execution mode and those described previously is that the interface to the system is through the Operator Interface Manager (OIM), described in detail in the *Operator's Guide* manual. The most user-visible aspect of this mode is the issuance by the OIM of task group id designations and message numbers, which require, in many cases, task group id and message number entries from the operator terminal in response.

The operator terminal is the only way in which the system as initially delivered to the user can be accessed. Initial startup results in the creation of the system task group ($S) and one online task group ($H). For small system environments in which the operator and one or more users (e.g., programmers writing and debugging their own programs through the operator terminal on a first-come, first-served basis) share the operator terminal, this type of startup, appropriately modified for the physical system configuration, may be sufficient.

Typically, the operator in this kind of configuration could initiate other task groups in any of the combinations described in Section 2 through the system task group, and also use the $H task group for any function which is not normally done in the system task group (e.g., editing files, assembling or compiling, linking, debugging, and the like). In particular, in the originally-released system, the $H task group is used to construct new CLM_USER files for system startup, and STARTUP.EC files for use during the startup process. There is no requirement that the existence of the $H task group be maintained permanently — the originally-released STARTUP.EC file which results in the creation of the $H task group can be modified at any time to delete the function of creating this task group.

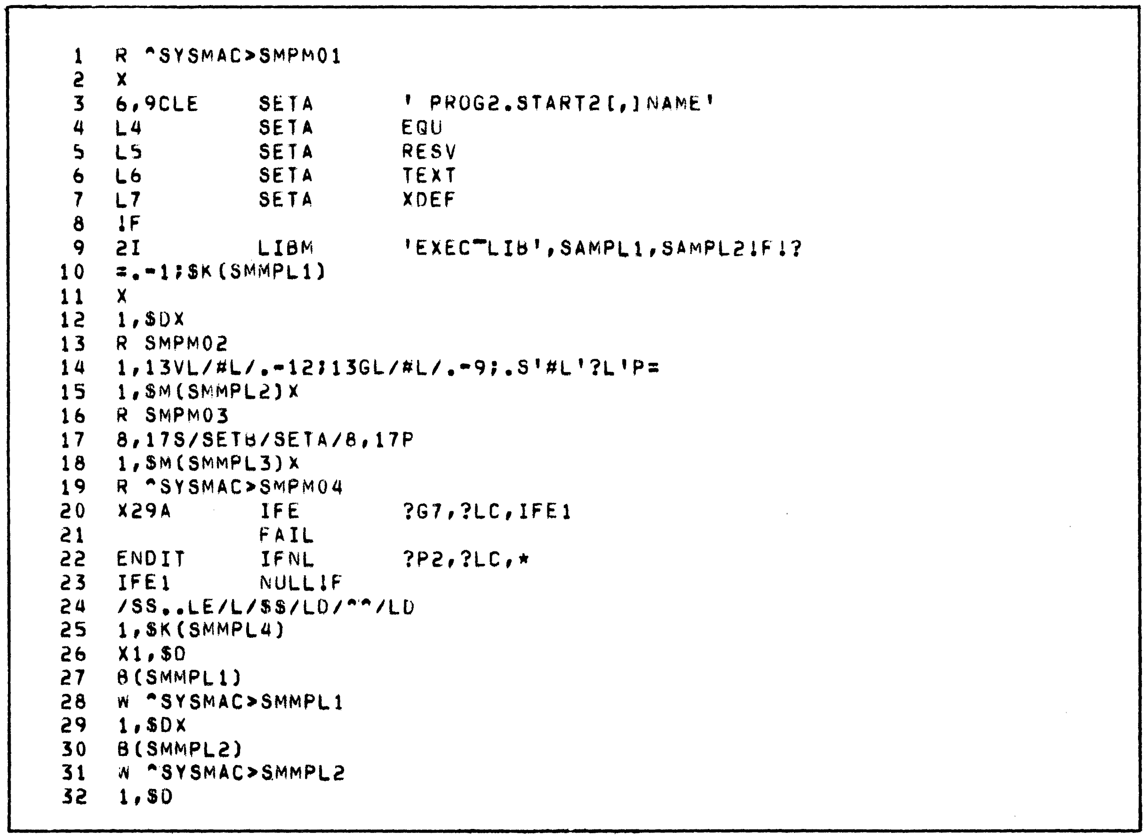
Any of the operations described above can be done through the operator terminal from an online task group such as $H, or any other online task group created as a function of system startup or at some later time, and specifying the operator terminal as its command-in file. Most of the examples in Sections 5 through 9 show operations using the online task group $H. They illustrate the issuance of the task group identification prefix by the OIM in operator terminal typeouts, and in some cases the changing of the OIM default task group identification to $H, eliminating the need to enter the prefix explicitly when issuing commands to this task group. If these same operations were done in a task group associated with a terminal other than the operator terminal, the prefixes would not be issued nor need to be specified at that terminal.

Section 5
Using the Editor

This section illustrates how Editor directives are used to modify the contents of four files, merge files into one file, and place macro routines in the macro library directory. The Editor directives are in file SMPCMDFL; the four files to be altered are SMPM01 (example 1), SMPM02 (example 2), SMPM03 (example 3), and SMPM04 (example 4). The examples are shown below. SMPM01 and SMPM02 are altered and written to files SMMPL1 and SMMPL2, respectively, and then combined to form file SMPMAC.P containing macro statements and calls to be processed by the macro preprocessor. SMPM03 and SMPM04 are altered and written as files SMMPL3 and SMMPL4, respectively; they are again altered and written as macro library routine files SAMPL1 and SAMPL2, respectively, into the MACRO<EXEC_LIB directory to be used during macro preprocessing. Editor output directed to the operator's terminal is shown in an operator's terminal typeout.

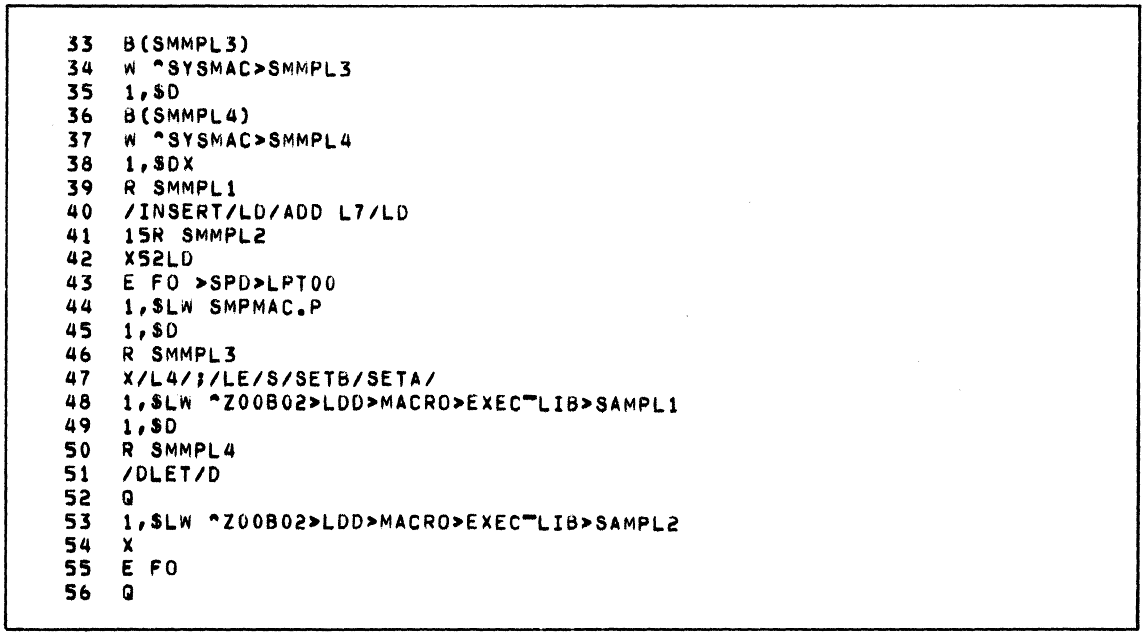
EDITOR DIRECTIVE DESCRIPTION

The following is a line-by-line explanation of the action taken by the Editor when it processes the directive file, ^SYSMAC>SMPCMDFL, of Figure 5-1, and an explanation of the operator terminal typeouts displayed in Figure 5-2. Editor directive lines are identified by line number. In the typeout, the response to these directives begins after the line ($H) EDIT-0100-11/21/0827. The default working directory is ^SYSMAC so that either a full pathname of the form ^SYSMAC>SMPM04 or simple pathname SMPM01 can be used.



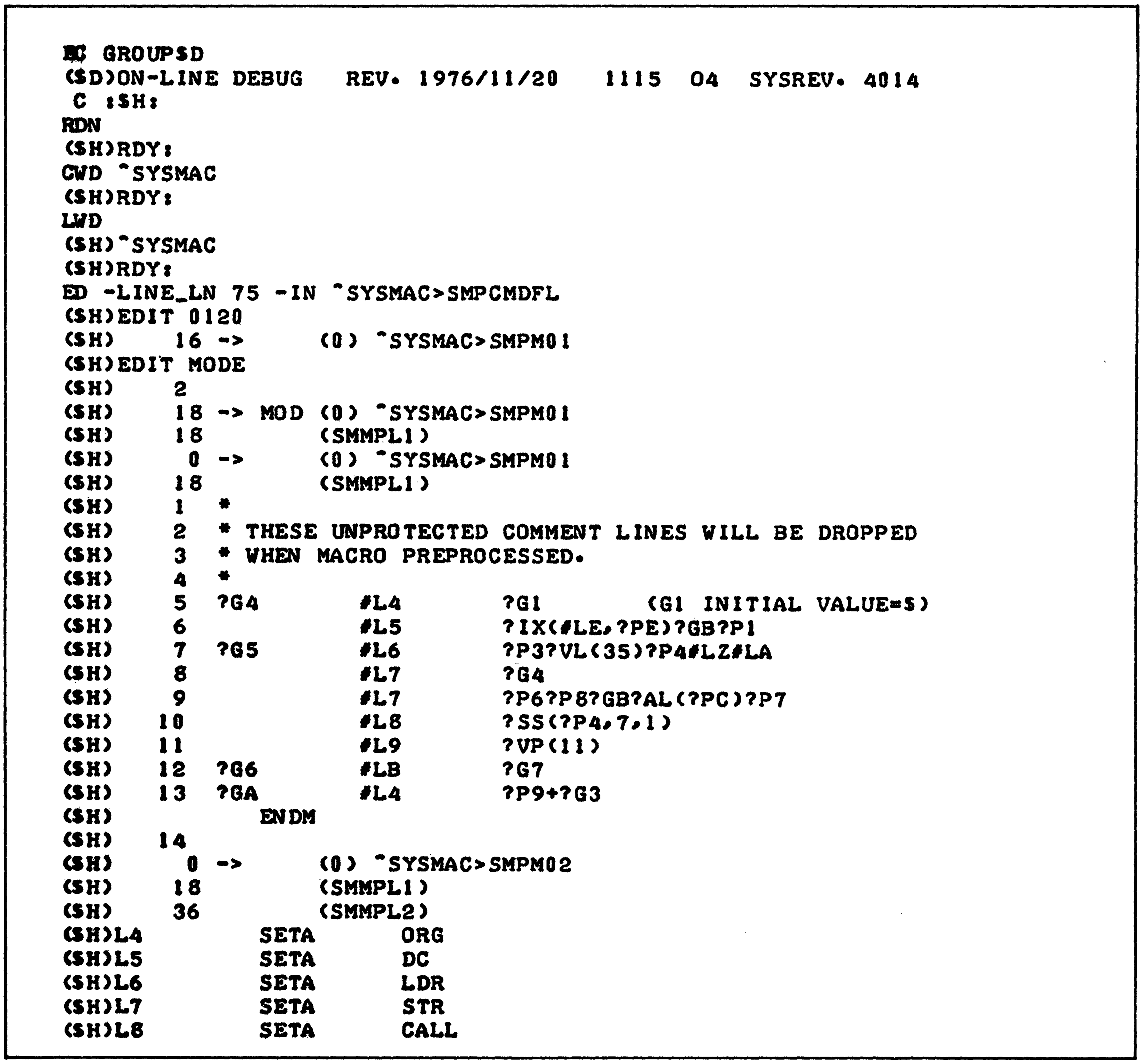
```
 1   R ^SYSMAC>SMPM01
 2   X
 3   6,9CLE       SETA        ' PROG2.START2[,]NAME'
 4   L4           SETA        EQU
 5   L5           SETA        RESV
 6   L6           SETA        TEXT
 7   L7           SETA        XDEF
 8   !F
 9   2I           LIBM        'EXEC¯LIB',SAMPL1,SAMPL2!F!?
10   =.-1;$K(SMMPL1)
11   X
12   1,$DX
13   R SMPM02
14   1,13VL/#L/.-12;13GL/#L/.-9;.S'#L'?L'P=
15   1,$M(SMMPL2)X
16   R SMPM03
17   8,17S/SETB/SETA/8,17P
18   1,$M(SMMPL3)X
19   R ^SYSMAC>SMPM04
20   X29A         IFE         ?G7,?LC,IFE1
21                FAIL
22   ENDIT        IFNL        ?P2,?LC,*
23   IFE1         NULL!F
24   /SS..LE/L/$$/LD/^^/LD
25   1,$K(SMMPL4)
26   X1,$D
27   B(SMMPL1)
28   W ^SYSMAC>SMMPL1
29   1,$DX
30   B(SMMPL2)
31   W ^SYSMAC>SMMPL2
32   1,$D
```

Figure 5-1. Sample Editor Directives in File SMPCMDFL



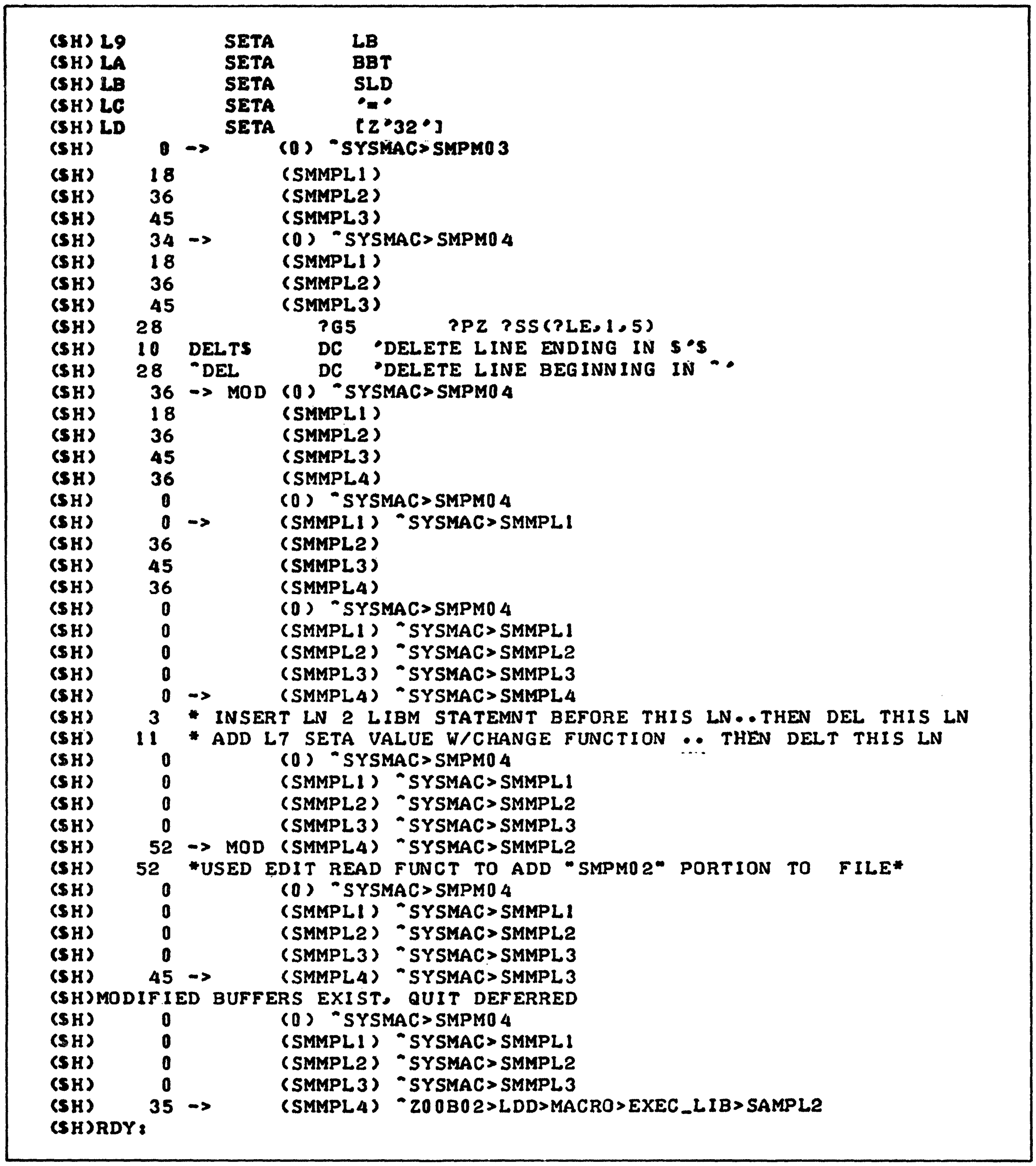
```
B(SMMPL3)
W ^SYSMAC>SMMPL3
1,$D
B(SMMPL4)
W ^SYSMAC>SMMPL4
1,$DX
R SMMPL1
/INSERT/LD/ADD L7/LD
15R SMMPL2
X52LD
E FO >SPD>LPT00
1,$LW SMPMAC.P
1,$D
R SMMPL3
X/L4/;/LE/S/SETB/SETA/
1,$LW ^Z00B02>LDD>MACRO>EXEC⁻LIB>SAMPL1
1,$D
R SMMPL4
/DLET/D
Q
1,$LW ^Z00B02>LDD>MACRO>EXEC⁻LIB>SAMPL2
X
E FO
Q
```

Figure 5-1 (cont). Sample Editor Directives in File SMPCMDFL



```
EC GROUPSD
($D)ON-LINE DEBUG   REV. 1976/11/20   1115  04  SYSREV. 4014
 C :$H:
RDN
($H)RDY:
CWD ^SYSMAC
($H)RDY:
LWD
($H)^SYSMAC
($H)RDY:
ED -LINE_LN 75 -IN ^SYSMAC>SMPCMDFL
($H)EDIT 0120
($H)     16 ->      (0) ^SYSMAC>SMPM01
($H)EDIT MODE
($H)     2
($H)     18 -> MOD (0) ^SYSMAC>SMPM01
($H)     18         (SMMPL1)
($H)      0 ->      (0) ^SYSMAC>SMPM01
($H)     18         (SMMPL1)
($H)     1  *
($H)     2  * THESE UNPROTECTED COMMENT LINES WILL BE DROPPED
($H)     3  * WHEN MACRO PREPROCESSED.
($H)     4  *
($H)     5  ?G4        #L4       ?G1       (G1 INITIAL VALUE=$)
($H)     6             #L5       ?IX(#LE,?PE)?GB?P1
($H)     7  ?G5        #L6       ?P3?VL(35)?P4#LZ#LA
($H)     8             #L7       ?G4
($H)     9             #L7       ?P6?P8?GB?AL(?PC)?P7
($H)    10             #L8       ?SS(?P4,7,1)
($H)    11             #L9       ?VP(11)
($H)    12  ?G6        #LB       ?G7
($H)    13  ?GA        #L4       ?P9+?G3
($H)          ENDM
($H)    14
($H)      0 ->      (0) ^SYSMAC>SMPM02
($H)     18         (SMMPL1)
($H)     36         (SMMPL2)
($H)L4        SETA      ORG
($H)L5        SETA      DC
($H)L6        SETA      LDR
($H)L7        SETA      STR
($H)L8        SETA      CALL
```

Figure 5-2. Terminal Responses from Sample Editor Directives of Figure 5-1



```
($H) L9          SETA       LB
($H) LA          SETA       BBT
($H) LB          SETA       SLD
($H) LC          SETA       '='
($H) LD          SETA       [Z'32']
($H)      0 ->       (0) ^SYSMAC>SMPM03
($H)     18          (SMMPL1)
($H)     36          (SMMPL2)
($H)     45          (SMMPL3)
($H)     34 ->       (0) ^SYSMAC>SMPM04
($H)     18          (SMMPL1)
($H)     36          (SMMPL2)
($H)     45          (SMMPL3)
($H)    28              ?G5         ?PZ ?SS(?LE,1,5)
($H)    10  DELT$       DC   'DELETE LINE ENDING IN $'$
($H)    28  ^DEL        DC   'DELETE LINE BEGINNING IN ^'
($H)     36 -> MOD (0) ^SYSMAC>SMPM04
($H)     18          (SMMPL1)
($H)     36          (SMMPL2)
($H)     45          (SMMPL3)
($H)     36          (SMMPL4)
($H)      0          (0) ^SYSMAC>SMPM04
($H)      0 ->       (SMMPL1) ^SYSMAC>SMMPL1
($H)     36          (SMMPL2)
($H)     45          (SMMPL3)
($H)     36          (SMMPL4)
($H)      0          (0) ^SYSMAC>SMPM04
($H)      0          (SMMPL1) ^SYSMAC>SMMPL1
($H)      0          (SMMPL2) ^SYSMAC>SMMPL2
($H)      0          (SMMPL3) ^SYSMAC>SMMPL3
($H)      0 ->       (SMMPL4) ^SYSMAC>SMMPL4
($H)      3  * INSERT LN 2 LIBM STATEMNT BEFORE THIS LN..THEN DEL THIS LN
($H)    11   * ADD L7 SETA VALUE W/CHANGE FUNCTION .. THEN DELT THIS LN
($H)      0          (0) ^SYSMAC>SMPM04
($H)      0          (SMMPL1) ^SYSMAC>SMMPL1
($H)      0          (SMMPL2) ^SYSMAC>SMMPL2
($H)      0          (SMMPL3) ^SYSMAC>SMMPL3
($H)     52 -> MOD (SMMPL4) ^SYSMAC>SMMPL2
($H)    52   *USED EDIT READ FUNCT TO ADD "SMPM02" PORTION TO  FILE*
($H)      0          (0) ^SYSMAC>SMPM04
($H)      0          (SMMPL1) ^SYSMAC>SMMPL1
($H)      0          (SMMPL2) ^SYSMAC>SMMPL2
($H)      0          (SMMPL3) ^SYSMAC>SMMPL3
($H)     45 ->       (SMMPL4) ^SYSMAC>SMMPL3
($H)MODIFIED BUFFERS EXIST, QUIT DEFERRED
($H)      0          (0) ^SYSMAC>SMPM04
($H)      0          (SMMPL1) ^SYSMAC>SMMPL1
($H)      0          (SMMPL2) ^SYSMAC>SMMPL2
($H)      0          (SMMPL3) ^SYSMAC>SMMPL3
($H)     35 ->       (SMMPL4) ^Z00B02>LDD>MACRO>EXEC_LIB>SAMPL2
($H)RDY:
```

Figure 5-2 (cont). Terminal Responses from Sample Editor Directives of Figure 5-1

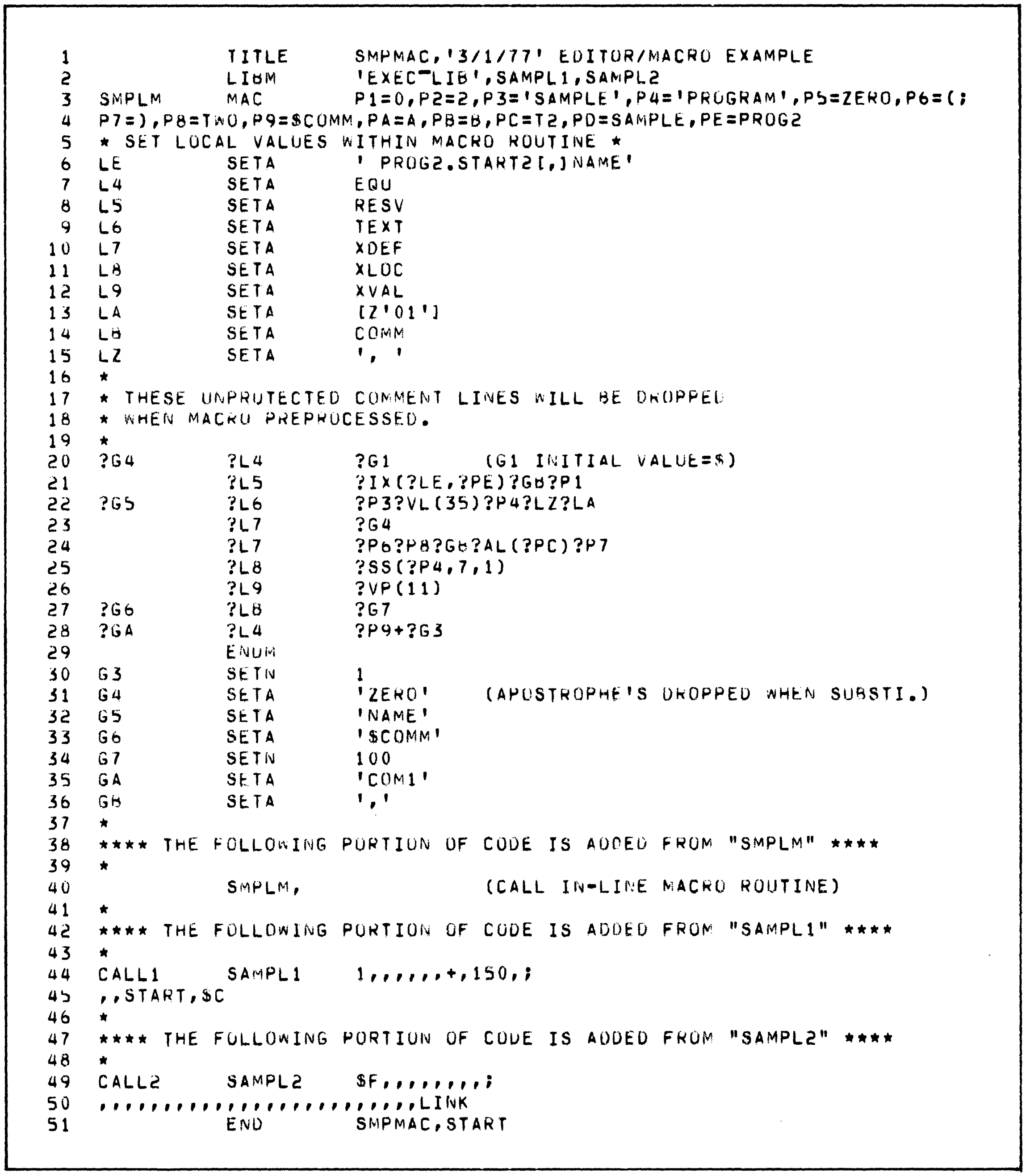
Line No.	Editor Directive Description	Terminal Typeout
1	R ^SYSMAC>SMPM01 Read the 16-line file (example 1) into the current buffer (0).	
2	X Display the status of the current buffer (denoted by →). Sixteen lines were read into current buffer (0).	16->(0) ...
3	6,9Ctext Change lines 6 through 9 of the buffer with this text for line 6.	
4	Text for line 7.	
5	Text for line 8.	
6	Text for line 9.	
7	Text, this additional line is inserted after the previous four lines were changed. Note that the Editor recognizes tab characters.	
8	!F Terminate input mode and enter edit mode.	
9	2Itext!F!? Insert text before line 2. Terminate input mode. Display current mode.	EDIT MODE
10	=.-1;$K(SMMPL1) Display current line pointer.	2
	Move the current line pointer back one line to a new current line pointer position. Copy the lines from that position to the last line in the current buffer into auxiliary buffer, SMMPL1.	
11	X Display status of current and auxiliary buffers. There are 18 lines in current buffer (0), which has been modified (MOD) since it was read in, and 18 lines in auxiliary buffer SMMPL1.	18-> MOD (0) ... 18 (SMMPL1)
12	1,$DSX Delete the first through last line of the current buffer (0). Display status of buffers.	0->(0) ... 18 (SMMPL1)
13	R SMPM02 Read 36-line file (example 2) into the current buffer (0).	
14	1,13VL/#L/.-12;13GL/#L/.-9;S'#L'?L'P= For lines 1 through 13, display line numbers and all lines that do not contain the expression #L.	1* 2*THESE 4*
	Move the current line pointer back 12 lines from line 14 to line 2. For lines 2 through 13, display all lines and their line numbers containing the expression #L.	5?G4#L4 ... . . . 13?GA #L4 ...

Line No.	Editor Directive Description	Terminal Typeout
	Move the current line pointer back nine lines to line 5 and substitute ?L for #L.	
	Print current line.	ENDM
	Print current line pointer value.	14
15	1,$M(SMMPL2)X	
	Move line 1 through the last line of the current buffer to auxiliary buffer SMMPL2. The contents of the current buffer (0) are erased. Display the status of the buffers.	0->(0) ... 18 (SMMPL1) 36(SMMPL2)
16	R SMPM03	
	Read 45-line file (example 3) into current buffer (0).	
17	8,17S/SETB/SETA/8,17P	
	For lines 8 through 17, substitute SETA for SETB.	
	Print lines 8 through 17 without line numbers.	L4 SETA ... . . . LD SETA ...
18	1,$M(SMMPL3)X	
	Move line 1 through last line of the current buffer into auxiliary buffer SMMPL3 and erase buffer (0).	
	Display buffer status.	0->(0) ... 18 (SMMPL1) 36 (SMMPL2) 45 (SMMPL3)
19	R ^SYSMAC>SMP04	
	Read the 34-line file (example 4) into the current buffer (0).	
20	X29A	
	Display buffer status, 34 lines are currently in buffer (0).	34->(0) ... 18 (SMMPL1) . . . 45 (SMMPL3)
	Append, after line 29, four lines of text. Text for line 30.	
21	Text for line 31.	
22	Text for line 32.	
23	text!F	
	Last line of text (line 33).	
	Terminate input mode and enter edit mode.	
24	/SS..LE/L/$$/LD/^ ^/LD	
	Search the current buffer for the first occurrence of the expression SS..LE, where .. are any two characters.	
	List the line and its line number.	28?G5 ?PZ ?SS(?LE...

Line No.	Editor Directive Description	Terminal Typeout
	Locate a line that ends with $ as the last character. The first dollar sign is escaped using a nonprinting C,i.e.,!C$$. The second dollar sign retains its special meaning, and indicates: locate the last character in a line ending in dollar sign.	
	List the line and its number; then delete the line.	10DELT$ DC...D'$'
	Locate a line beginning with ^. The second circumflex is escaped by using the nonprinting !C,i.e., ^ !C ^ .	
	List, then delete the line and its line number.	28 ^ DEL DC 'DELETE ...
25	1,$K(SMMPL4)	
	Copy the current buffer contents from first through last line into auxiliary buffer SMMPL4.	
26	X1,$D	
	Display the status of the buffers.	36->MOD(0) ... 18 ... . . . 36 (SMMPL4)
	Delete first through last line of current buffer.	
27	B(SMMPL1)	
	The auxiliary buffer, SMMPL1, is made the current buffer prior to writing.	
28	W ^ SYSMAC>SMMPL1	
	Write the current buffer contents as a file whose pathname is ^ SYSMAC>SMMPL1.	
29	1,$DX	
	Delete the first through last line of the current buffer.	
	Display the buffer status. The pointer points to current buffer, SMMPL1.	0(0) ... 0->(SMMPL1) ... . . . 36(SMMPL4)
30	B(SMMPL2)	
	The auxiliary buffer, SMMPL2, is made the current buffer prior to writing.	
31	W ^ SYSMAC>SMMPL2	
	Write the current buffer contents as a file whose pathname is ^ SYSMAC>SMMPL2.	
32	1,$D	
	Delete the first through last line of the current buffer.	
33	B(SMMPL3)	
	The auxiliary buffer, SMMPL3, is made the current buffer prior to writing.	

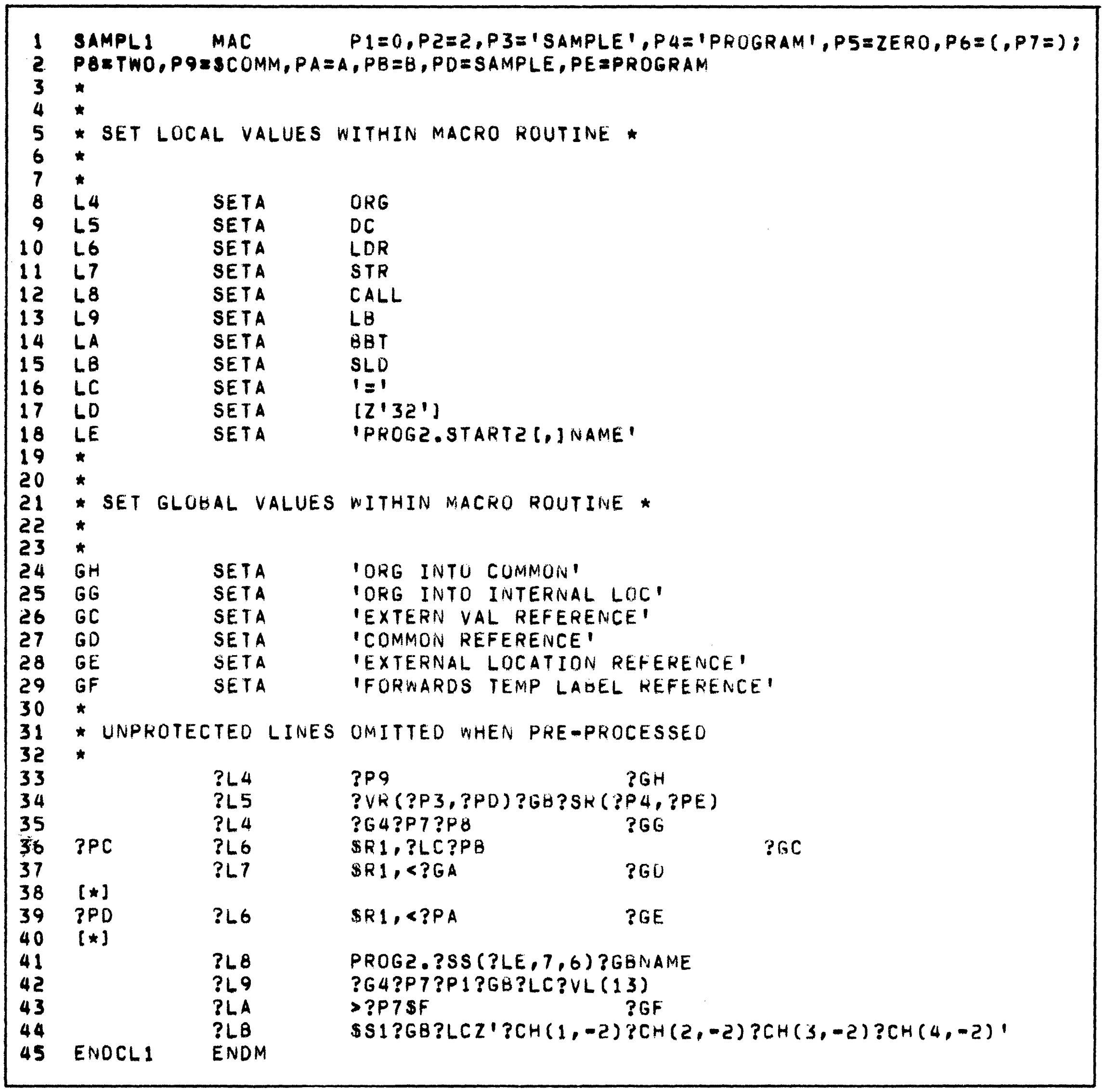
Line No.	Editor Directive Description	Terminal Typeout
34	W ^ SYSMAC>SMMPL3 Write the current buffer contents as a file whose pathname is ^ SYSMAC>SMMPL3.	
35	1,$D Delete the first through last line of the current buffer.	
36	B(SMMPL4) The auxiliary buffer, SMMPL4, is made the current buffer prior to writing.	
37	W ^ SYSMAC>SMMPL4. Write the current buffer contents as a file whose pathname is ^ SYSMAC>SMMPL4.	
38	1,$DX Delete the first through last line of the current buffer.	
	Display the status of the buffers. SMMPL4 is the current buffer. All the buffers have been cleared.	0 (0) ... 0 (SMMPL1) .. . . . 0->(SMMPL4) ...
39	R SMMPL1 Read the file SMMPL1 into the current buffer, SMMPL4.	
40	/INSERT/LD//ADD L7/LD Locate the first line containing the expression, INSERT, list it and its line number, and then delete it.	3* INSERT LN ...
	Starting at the current line, locate the first line containing the expression, ADD L7, list it and its line number, and then delete it.	11* ADD L7 SETA ...
41	15R SMMPL2 Read the file, SMMPL2, into the current buffer after line 15 of the buffer. Two files are being merged.	
42	X52LD Display the status of the buffers. Current buffer, SMMPL4, now has 52 lines.	0 (0) ... . . . 52->MOD(SMMPL4)...
	List line 52 then delete it.	52* USED EDIT ...
43	E FO >SPD>LPT00 The Execute directive allows you to execute the ECL command FO to change the output file from the operator's terminal to the line printer.	
44	1,$LW SMPMAC.P List the first through last line of current buffer on the line printer (Figure 5-3). Write the current buffer as a file whose pathname is SMPMAC.P.	

Line No.	Editor Directive Description	Terminal Typeout
45	1,$D Delete the first through last line of the current buffer.	
46	R SMMPL3 Read the file, SMMPL3, into the current buffer, SMMPL4.	
47	X/L4/;/LE/S/SETB/SETA/ Display the status of the buffers. Locate the first line containing the expression, L4. Starting with the line containing L4 through the line containing the expression LE, substitute SETA for all occurrences of SETB.	0 (0) ... . . . 45->(SMMPL4) ...
48	1,$LW ^Z00B02>LDD>MACRO>EXEC_LIB>SAMPL1 List the first through last line of the current buffer on the line printer (Figure 5-4). Write the current buffer as a library routine file whose pathname is ^Z00B02>LDD>MACRO>EXEC_◇%÷>SAMPL1.	
49	1,$D Delete the first through last line of the current buffer.	
50	R SMMPL4 Read the file, SMMPL4, into the current buffer.	
51	/DLET/D Locate and delete the line containing the expression DLET.	
52	Q Quit. The quit is deferred since a buffer has been modified and has not been written to a file. You have *one* more chance to write the contents of the current buffer as a file.	MODIFIED BUFFERS EXIST ...
53	1,$LW ^Z00B02>LDD>MACRO>EXEC_LIB>SAMPL2 List the first through last line of the current buffer on the line printer (Figure 5-5). Write the current buffer contents as a library routine file whose pathname is ^Z00B02>LDD>MACRO>EXEC_LIB>SAMPL2.	
54	X Display buffer status. Status is always displayed on the operator's terminal even though the output file is the printer.	0 (0) ... . . . 35->(SMMPL4)
55	E FO The Execute directive allows you to execute the ECL comand FO to change the output file from the line printer back to the operator's terminal.	
56	Q Quit. Exit from the Editor.	



```
          TITLE     SMPMAC,'3/1/77' EDITOR/MACRO EXAMPLE
          LIBM      'EXEC‾LIB',SAMPL1,SAMPL2
SMPLM     MAC       P1=0,P2=2,P3='SAMPLE',P4='PROGRAM',P5=ZERO,P6=(;
P7=),P8=TWO,P9=$COMM,PA=A,PB=B,PC=T2,PD=SAMPLE,PE=PROG2
* SET LOCAL VALUES WITHIN MACRO ROUTINE *
LE        SETA      ' PROG2.START2[,]NAME'
L4        SETA      EQU
L5        SETA      RESV
L6        SETA      TEXT
L7        SETA      XDEF
L8        SETA      XLOC
L9        SETA      XVAL
LA        SETA      [Z'01']
LB        SETA      COMM
LZ        SETA      ', '
*
* THESE UNPROTECTED COMMENT LINES WILL BE DROPPED
* WHEN MACRO PREPROCESSED.
*
?G4       ?L4       ?G1       (G1 INITIAL VALUE=$)
          ?L5       ?IX(?LE,?PE)?GB?P1
?G5       ?L6       ?P3?VL(35)?P4?LZ?LA
          ?L7       ?G4
          ?L7       ?P6?P8?GB?AL(?PC)?P7
          ?L8       ?SS(?P4,7,1)
          ?L9       ?VP(11)
?G6       ?LB       ?G7
?GA       ?L4       ?P9+?G3
          ENDM
G3        SETN      1
G4        SETA      'ZERO'    (APOSTROPHE'S DROPPED WHEN SUBSTI.)
G5        SETA      'NAME'
G6        SETA      '$COMM'
G7        SETN      100
GA        SETA      'COM1'
GB        SETA      ','
*
**** THE FOLLOWING PORTION OF CODE IS ADDED FROM "SMPLM" ****
*
          SMPLM,              (CALL IN-LINE MACRO ROUTINE)
*
**** THE FOLLOWING PORTION OF CODE IS ADDED FROM "SAMPL1" ****
*
CALL1     SAMPL1    1,,,,,,+,150,;
,,START,$C
*
**** THE FOLLOWING PORTION OF CODE IS ADDED FROM "SAMPL2" ****
*
CALL2     SAMPL2    $F,,,,,,,,;
,,,,,,,,,,,,,,,,,,,,,,,,,LINK
          END       SMPMAC,START
```

Figure 5-3. Sample of Unexpanded Assembly Language Program with Macro Calls and Statements (SMPMAC.P)



```
SAMPL1    MAC       P1=0,P2=2,P3='SAMPLE',P4='PROGRAM',P5=ZERO,P6=(,P7=);
P8=TWO,P9=$COMM,PA=A,PB=B,PD=SAMPLE,PE=PROGRAM
*
*
* SET LOCAL VALUES WITHIN MACRO ROUTINE *
*
*
L4        SETA      ORG
L5        SETA      DC
L6        SETA      LDR
L7        SETA      STR
L8        SETA      CALL
L9        SETA      LB
LA        SETA      BBT
LB        SETA      SLD
LC        SETA      '='
LD        SETA      [Z'32']
LE        SETA      'PROG2.START2[,]NAME'
*
*
* SET GLOBAL VALUES WITHIN MACRO ROUTINE *
*
*
GH        SETA      'ORG INTO COMMON'
GG        SETA      'ORG INTO INTERNAL LOC'
GC        SETA      'EXTERN VAL REFERENCE'
GD        SETA      'COMMON REFERENCE'
GE        SETA      'EXTERNAL LOCATION REFERENCE'
GF        SETA      'FORWARDS TEMP LABEL REFERENCE'
*
* UNPROTECTED LINES OMITTED WHEN PRE-PROCESSED
*
          ?L4       ?P9                 ?GH
          ?L5       ?VR(?P3,?PD)?GB?SR(?P4,?PE)
          ?L4       ?G4?P7?P8           ?GG
?PC       ?L6       $R1,?LC?PB                    ?GC
          ?L7       $R1,<?GA            ?GD
[*]
?PD       ?L6       $R1,<?PA            ?GE
[*]
          ?L8       PROG2.?SS(?LE,7,6)?GBNAME
          ?L9       ?G4?P7?P1?GB?LC?VL(13)
          ?LA       >?P7$F              ?GF
          ?LB       $S1?GB?LCZ'?CH(1,-2)?CH(2,-2)?CH(3,-2)?CH(4,-2)'
ENDCL1    ENDM
```

Figure 5-4. Sample of Unexpanded Macro Routine (SAMPL1) Contained in EXEC_LIB Directory

```
 1  SAMPL2     MAC        P1=0,P2=2,P3='SAMPLE',P4='PROGRAM',P5=ZERO,P6=(,P7=);
 2  P8=TWO,P9=$COMM,PA=A,PB=B
 3  * SET LOCAL VALUES WITHIN MACRO ROUTINE *
 4  L4         SETA       >=[Z'1300']
 5  LA         SETA       IOLD
 6  LD         SETA       $R1
 7  LE         SETA       'PROG2.START[,]NAME'
 8  LG         SETA       SOR
 9  LC         SETN       -32768
10  LP         SETN       32767
11  LQ         SETN       0
12  LI         SETA       BEZ
13  LY         SETA       HLT
14  LZ         SETA       ','
15  * SET GLOBAL VALUES WITHIN MACRO ROUTINE *
16  G7         SETN       -32768
17  G2         SETA       'BACKWARDS TEMP LABEL REFERENCE'
18  G5         SETA       CTRL
19  *
20  * UNPROTECTED LINES OMITTED WHEN PRE-PROCESSED
21  *
22  ?P1        ?LA        ?P5?LZ?L4,=?LD
23  ?P1        ?LA        ?P5?LZ?L4,=?LD
24             ?LG        ?LD,?VG(3)
25             ?LI        ?LD,-$C   ?G2
26             ?LY
27             ?G5        ?PZ ?SS(?LE,1,5)
28             IFE        ?G7,?LC,IFE1
29             FAIL
30  ENDIT      IFNL       ?P2,?LC,*
31  IFE1       NULL
32             IFNE       ?G7,?LP,GTEND
33             FAIL
34  GTEND      GOTO       ENDIT
35  ENDCL2     ENDM
```

Figure 5-5. Sample of Unexpanded Macro Routine (SAMPL2) Contained in EXEC_LIB Directory

Example 1:

File SMPM01 before Editing

```
 1             TITLE      SMPMAC,'3/1/77' EDITOR/MACRO EXAMPLE
 2  * INSERT LN 2 LIBM STATEMNT BEFORE THIS LN..THEN DEL THIS LN
 3  SMPLM      MAC        P1=0,P2=2,P3='SAMPLE',P4='PROGRAM',P5=ZERO,P6=(;
 4  P7=),P8=TWO,P9=$COMM,PA=A,PB=B,PC=T2,PD=SAMPLE,PE=PROG2
 5  * SET LOCAL VALUES WITHIN MACRO ROUTINE *
 6  LE         USE CHANGE FUNCTION TO ADD SETA VALUE FOR THIS LN
 7  L4         USE CHANGE FUNCTION TO ADD SETA VALUE FOR THIS LN
 8  L5         USE CHANGE FUNCTION TO ADD SETA VALUE FOR THIS LN
 9  L6         USE CHANGE FUNCTION TO ADD SETA VALUE FOR THIS LN
10  * ADD L7 SETA VALUE W/CHANGE FUNCTION .. THEN DELT THIS LN
11  L8         SETA       XLOC
12  L9         SETA       XVAL
13  LA         SETA       [Z'01']
14  LB         SETA       COMM
15  LZ         SETA       ', '
16  *USED EDIT READ FUNCT TO ADD "SMPM02" PORTION TO  FILE*
```

Example 2:

File SMPM02 before Editing

```
 1   *
 2   * THESE UNPROTECTED COMMENT LINES WILL BE DROPPED
 3   * WHEN MACRO PREPROCESSED.
 4   *
 5   ?G4        #L4        ?G1        (G1 INITIAL VALUE=$)
 6              #L5        ?IX(#LE,?PE)?GB?P1
 7   ?G5        #L6        ?P3?VL(35)?P4#LZ#LA
 8              #L7        ?G4
 9              #L7        ?P6?P8?GB?AL(?PC)?P7
10              #L8        ?SS(?P4,7,1)
11              #L9        ?VP(11)
12   ?G6        #L8        ?G7
13   ?GA        #L4        ?P9+?G3
14              ENDM
15   G3         SETN       1
16   G4         SETA       'ZERO'     (APOSTROPHE'S DROPPED WHEN SUBSTI.)
17   G5         SETA       'NAME'
18   G6         SETA       '$COMM'
19   G7         SETN       100
20   GA         SETA       'COM1'
21   GB         SETA       ','
22   *
23   **** THE FOLLOWING PORTION OF CODE IS ADDED FROM "SMPLM" ****
24   *
25              SMPLM,                (CALL IN-LINE MACRO ROUTINE)
26   *
27   **** THE FOLLOWING PORTION OF CODE IS ADDED FROM "SAMPL1" ****
28   *
29   CALL1      SAMPL1     1,,,,,,,+,150,;
30   ,,START,$C
31   *
32   **** THE FOLLOWING PORTION OF CODE IS ADDED FROM "SAMPL2" ****
33   *
34   CALL2      SAMPL2     $F,,,,,,,,;
35   ,,,,,,,,,,,,,,,,,,,,,,,,,,LINK
36              END        SMPMAC,START
```

Example 3:

File SMPM03 before Editing

```
 1  SAMPL1       MAC          P1=0,P2=2,P3='SAMPLE',P4='PROGRAM',P5=ZERO,P6=(,P7=);
 2  P8=TWO,P9=$COMM,PA=A,PB=B,PD=SAMPLE,PE=PROGRAM
 3  *
 4  *
 5  * SET LOCAL VALUES WITHIN MACRO ROUTINE *
 6  *
 7  *
 8  L4           SETB         ORG
 9  L5           SETB         DC
10  L6           SETB         LDR
11  L7           SETB         STR
12  L8           SETB         CALL
13  L9           SETB         LB
14  LA           SETB         BBT
15  LB           SETB         SLD
16  LC           SETB         '='
17  LD           SETB         [Z'32']
18  LE           SETB         'PROG2.START2[,]NAME'
19  *
20  *
21  * SET GLOBAL VALUES WITHIN MACRO ROUTINE *
22  *
23  *
24  GH           SETA         'ORG INTO COMMON'
25  GG           SETA         'ORG INTO INTERNAL LOC'
26  GC           SETA         'EXTERN VAL REFERENCE'
27  GD           SETA         'COMMON REFERENCE'
28  GE           SETA         'EXTERNAL LOCATION REFERENCE'
29  GF           SETA         'FORWARDS TEMP LABEL REFERENCF'
30  *
31  * UNPROTECTED LINES OMITTED WHEN PRE-PROCESSED
32  *
33               ?L4          ?P9                         ?GH
34               ?L5          ?VR(?P3,?PD)?GB?SR(?P4,?PE)
35               ?L4          ?G4?P7?P8                   ?GG
36  ?PC          ?L6          $R1,?LC?P8                                    ?GC
37               ?L7          $R1,<?GA                    ?GD
38  [*]
39  ?PD          ?L6          $R1,<?PA                    ?GE
40  [*]
41               ?L8          PROG2.?SS(?LE,7,6)?GHNAME
42               ?L9          ?G4?P7?P1?GB?LC?VL(13)
43               ?LA          >?P7$F                      ?GF
44               ?LB          $S1?GB?LCZ'?CH(1,-2)?CH(2,-2)?CH(3,-2)?CH(4,-2)'
45  ENDCL1       ENDM
```

Example 4:

File SMPM04 before Editing

```
 1   SAMPL2      MAC        P1=0,P2=2,P3='SAMPLE',P4='PROGRAM',P5=ZERO,P6=(,P7=);
 2   P8=TWO,P9=$COMM,PA=A,PB=B
 3   * SET LOCAL VALUES WITHIN MACRO ROUTINE *
 4   L4          SETA       >=[Z'1300']
 5   LA          SETA       IOLD
 6   LD          SETA       $R1
 7   LE          SETA       'PROG2.START[,]NAME'
 8   LG          SETA       SOH
 9   LC          SETN       -32768
10   DELT$       DC   'DELETE LINE ENDING IN $'$
11   LP          SETN       32767
12   LQ          SETN       0
13   LI          SETA       BEZ
14   LY          SETA       HLT
15   LZ          SETA       ','
16   * SET GLOBAL VALUES WITHIN MACRO ROUTINE *
17   G7          SETN       -32768
18   G2          SETA       'BACKWARDS TEMP LABEL REFERENCE'
19   G5          SETA       CTRL
20   *
21   * UNPROTECTED LINES OMITTED WHEN PRE-PROCESSED
22   *
23   ?P1         ?LA        ?P5?LZ?L4,=?LD
24   ?P1         ?LA        ?P5?LZ?L4,=?LD
25               ?LG        ?LD,?VG(3)
26               ?LI        ?LD,-$C    ?G2
27               ?LY
28               ?G5        ?PZ ?SS(?LE,1,5)
29   ^DEL        DC   'DELETE LINE BEGINNING IN ^'
30               IFNE       ?G7,?LP,GTEND
31               FAIL
32   GTEND       GOTO       ENDIT
33   DLET        DC   'DELETE LINE BEFORE QUIT'
34   ENDCL2      ENDM
```

Section 6
Using the Assembler and Macro Preprocessor

This section illustrates the use of the assembler to construct programs containing macro calls. Two assembly language samples are presented. One illustrates the output of different assembly language program processors. The other illustrates an assembly language program that contains multiple tasks. Both samples provide the operator terminal session listings that contain the commands to invoke the system software.

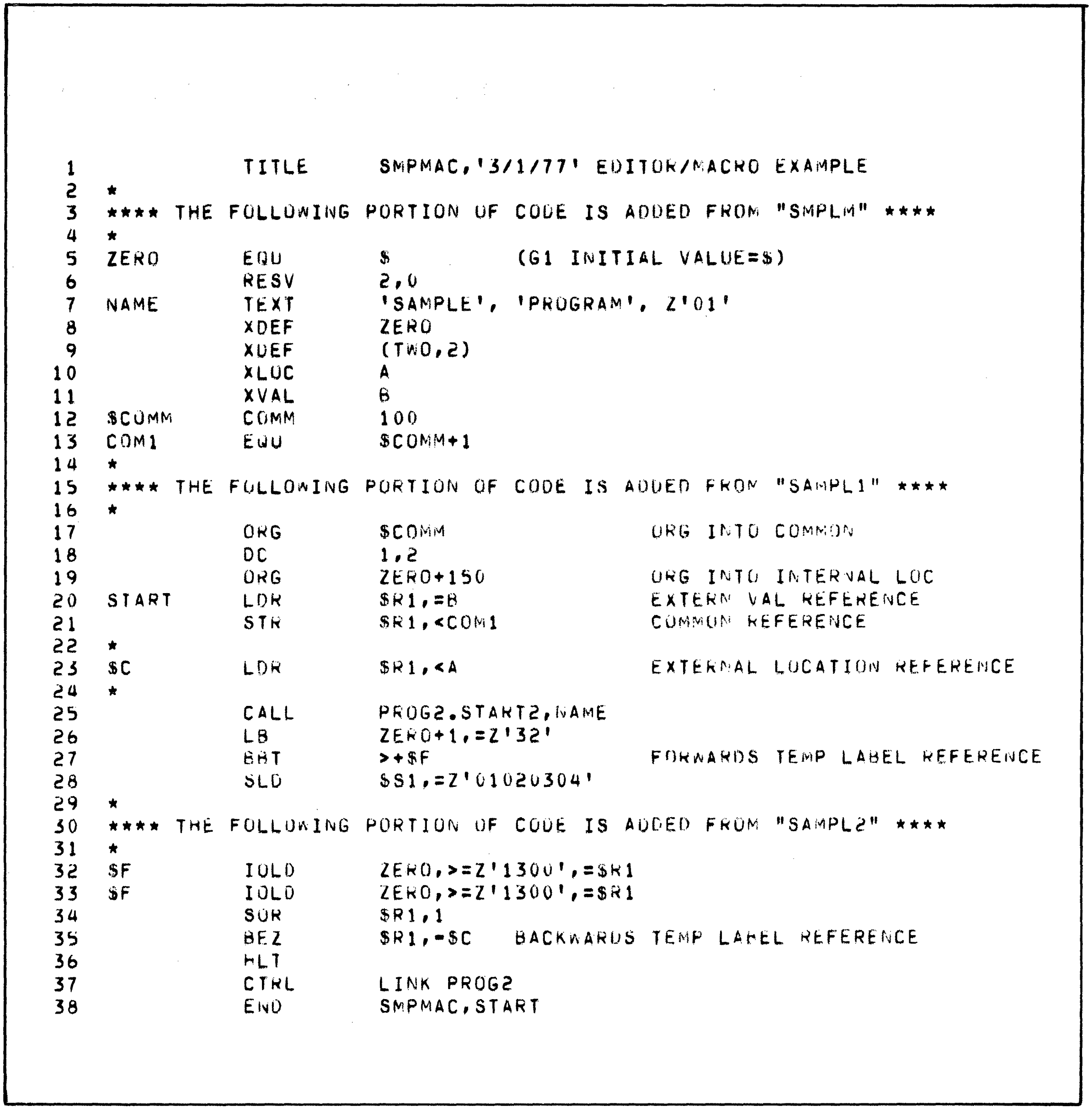
SAMPLE ASSEMBLY LANGUAGE SESSION (SMPMAC)

Figure 6-1 contains a sample operator terminal session to preprocess, assemble with cross-reference, and link the assembly language program SMPMAC. It is assumed that the Honeywell-supplied startup has been done prior to this session.

The typeout illustrates the following points. The working directory is changed to SYSMAC where the program's files are located. The file, SMPMAC, that is processed by the Macro Preprocessor is in Figure 5-3. The macro routines, SAMPL1 and SAMPL2, called by the program are listed in Figures 5-4 and 5-5. A listing of the output file SMPMAC.A from the macro preprocessor is shown in Figure 6-2.

```
CWD ↑SYSMAC
($H)RDY:
↑ZSYS51>SYSLIB2>MACROP  ↑SYSMAC>SMPMAC  -SZ  10
($H)MACROP-0100-11/17/1404
($H)0000 ERR COUNT
($H)RDY:
ASSEM  ↑SYSMAC>SMPMAC  -SZ  10  -SAF  -LE  -XREF  -COUT  >SPD>LPT00
($H)ASSEM-0100-11/17/1346
($H)0000 ERR COUNT
($H)ASSEM: (020105) 1D 1380 0000 0000
($H)>SPD>LPT00
($H)RDY:
FO  >SPD>LPT00
($H)RDY:
LINKER  ↑SYSMAC>SMPMAC  -COUT  >SPD>LPT00  -SZ  10
($H)LINKER-0100-11/23/1258
LDEF  A,X'100'
VDEF  B,X'2'
LINK  SMPMAC
MAP
QT
($H)SAF OR SLIC PROG2.0      NT FND
($H)SAF OR SLIC PROG2.0      NT FND
($H)ROOT  SMPMAC
($H)LINK DONE
($H)RDY:
```

Figure 6-1. Sample Terminal Session (SMPMAC)

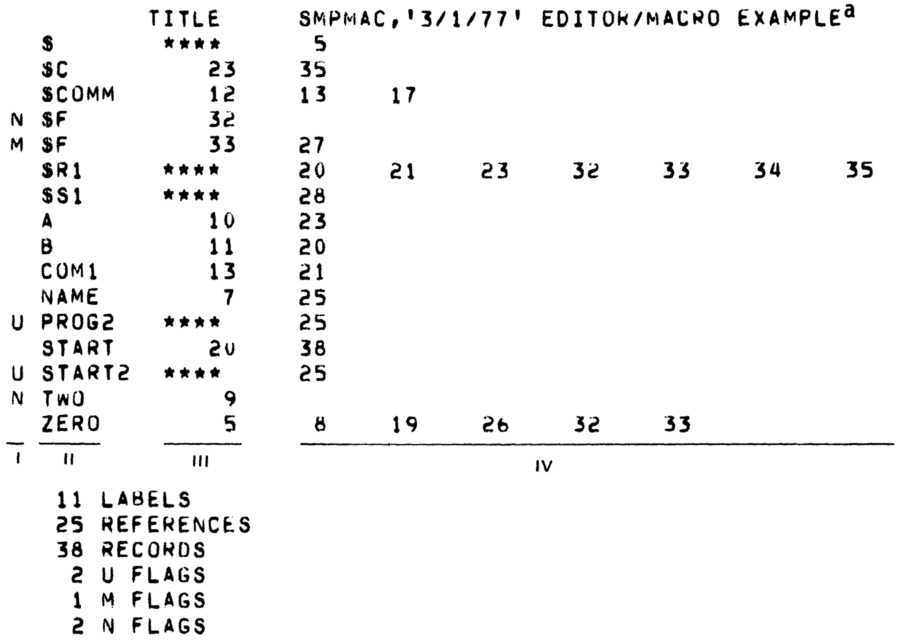


```
               TITLE     SMPMAC,'3/1/77' EDITOR/MACRO EXAMPLE
*
**** THE FOLLOWING PORTION OF CODE IS ADDED FROM "SMPLM" ****
*
ZERO           EQU       $              (G1 INITIAL VALUE=$)
               RESV      2,0
NAME           TEXT      'SAMPLE', 'PROGRAM', Z'01'
               XDEF      ZERO
               XDEF      (TWO,2)
               XLOC      A
               XVAL      B
$COMM          COMM      100
COM1           EQU       $COMM+1
*
**** THE FOLLOWING PORTION OF CODE IS ADDED FROM "SAMPL1" ****
*
               ORG       $COMM                    ORG INTO COMMON
               DC        1,2
               ORG       ZERO+150                 ORG INTO INTERNAL LOC
START          LDR       $R1,=B                   EXTERN VAL REFERENCE
               STR       $R1,<COM1                COMMON REFERENCE
*
$C             LDR       $R1,<A                   EXTERNAL LOCATION REFERENCE
*
               CALL      PROG2.START2,NAME
               LB        ZERO+1,=Z'32'
               BBT       >+$F                     FORWARDS TEMP LABEL REFERENCE
               SLD       $S1,=Z'01020304'
*
**** THE FOLLOWING PORTION OF CODE IS ADDED FROM "SAMPL2" ****
*
$F             IOLD      ZERO,>=Z'1300',=$R1
$F             IOLD      ZERO,>=Z'1300',=$R1
               SUR       $R1,1
               BEZ       $R1,-$C   BACKWARDS TEMP LABEL REFERENCE
               HLT
               CTRL      LINK PROG2
               END       SMPMAC,START
```

Figure 6-2. Macro Preprocessor Output (SMPMAC)

Figure 6-3 illustrates a cross-reference listing produced by the Assembler. See the *Program Preparation and Checkout* manual for an explanation of cross-reference symbols.

Figure 6-4 illustrates the Assembler listing of the assembled Macro Preprocessor output. Figure 6-5 illustrates the link map of the previously assembled program SMPMAC.



```
          TITLE      SMPMAC,'3/1/77' EDITOR/MACRO EXAMPLE[a]
  $        ****       5
  $C          23     35
  $COMM       12     13     17
N $F          32
M $F          33     27
  $R1      ****      20     21     23     32     33     34     35
  $S1      ****      28
  A           10     23
  B           11     20
  COM1        13     21
  NAME         7     25
U PROG2    ****      25
  START       20     38
U START2   ****      25
N TWO          9
  ZERO         5      8     19     26     32     33
_ _____    ______    ______________________________________________
I  II        III                        IV

   11 LABELS
   25 REFERENCES
   38 RECORDS
    2 U FLAGS
    1 M FLAGS
    2 N FLAGS
```

Legend:

I — Optional error flag:

- M — Designated label occurs more than once in the label field in the module; i.e., the label is multiply defined.
- U — Designated label is not defined;**** is also included in the definition field.
- N — Designated label is not referenced in the module.

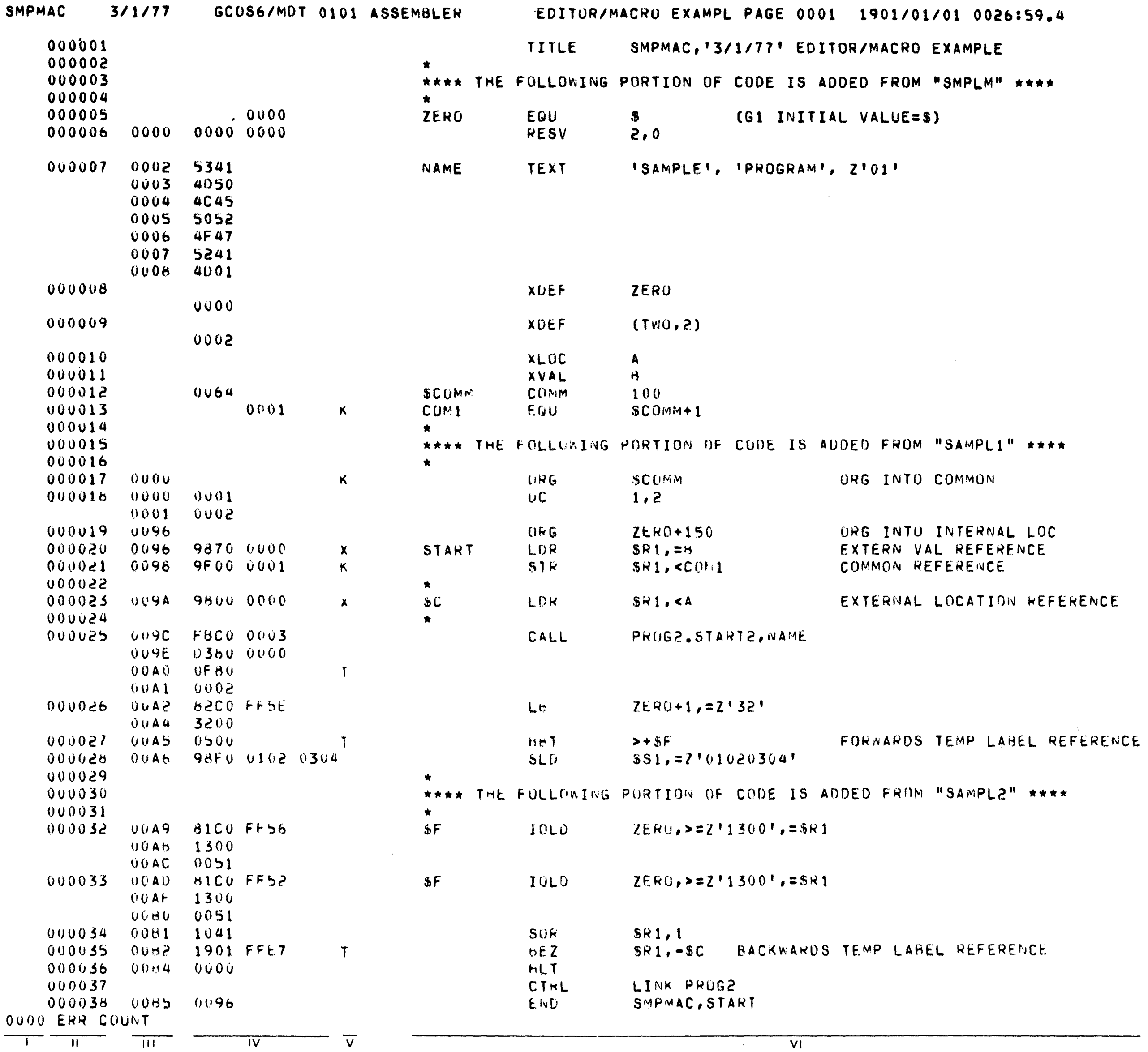
II — Identifiers (e.g., registers) and an alphabetical list of all labels in the assembly language source module. Identifiers do not have to be defined and are never flagged.

III — Number of the line in which the symbolic name is defined in the module. Asterisks (****) indicate that the symbolic name was not defined in this module.

IV — Number of each line that contains a reference to the symbolic name.

[a] The contents of the assembly program TITLE statement become the heading for the cross-reference listing.

Figure 6-3. Cross-Reference Listing (SMPMAC)



```
SMPMAC     3/1/77     GCOS6/MDT 0101 ASSEMBLER       EDITOR/MACRO EXAMPL PAGE 0001  1901/01/01 0026:59.4

    000001                                      TITLE     SMPMAC,'3/1/77' EDITOR/MACRO EXAMPLE
    000002                            *
    000003                            **** THE FOLLOWING PORTION OF CODE IS ADDED FROM "SMPLM" ****
    000004                            *
    000005            . 0000          ZERO      EQU       $         (G1 INITIAL VALUE=$)
    000006  0000  0000 0000                     RESV      2,0

    000007  0002  5341                NAME      TEXT      'SAMPLE', 'PROGRAM', Z'01'
            0003  4050
            0004  4C45
            0005  5052
            0006  4F47
            0007  5241
            0008  4001
    000008                                      XDEF      ZERO
                  0000
    000009                                      XDEF      (TWO,2)
                  0002
    000010                                      XLOC      A
    000011                                      XVAL      B
    000012        0064                $COMM     COMM      100
    000013             0001     K     COM1      EQU       $COMM+1
    000014                            *
    000015                            **** THE FOLLOWING PORTION OF CODE IS ADDED FROM "SAMPL1" ****
    000016                            *
    000017  0000                K               ORG       $COMM               ORG INTO COMMON
    000018  0000  0001                          DC        1,2
            0001  0002
    000019  0096                                ORG       ZERO+150            ORG INTO INTERNAL LOC
    000020  0096  9870 0000     X     START     LDR       $R1,=B              EXTERN VAL REFERENCE
    000021  0098  9F00 0001     K               STR       $R1,<COM1           COMMON REFERENCE
    000022                            *
    000023  009A  9800 0000     X     $C        LDR       $R1,<A              EXTERNAL LOCATION REFERENCE
    000024                            *
    000025  009C  FBC0 0003                     CALL      PROG2.START2,NAME
            009E  D360 0000
            00A0  0F80          T
            00A1  0002
    000026  00A2  82C0 FF5E                     LB        ZERO+1,=Z'32'
            00A4  3200
    000027  00A5  0500          T               BBT       >+$F                FORWARDS TEMP LABEL REFERENCE
    000028  00A6  98F0 0102 0304                SLD       $S1,=Z'01020304'
    000029                            *
    000030                            **** THE FOLLOWING PORTION OF CODE IS ADDED FROM "SAMPL2" ****
    000031                            *
    000032  00A9  81C0 FF56           $F        IOLD      ZERO,>=Z'1300',=$R1
            00AB  1300
            00AC  0051
    000033  00AD  81C0 FF52           $F        IOLD      ZERO,>=Z'1300',=$R1
            00AF  1300
            00B0  0051
    000034  00B1  1041                          SOR       $R1,1
    000035  00B2  1901 FFE7     T               BEZ       $R1,-$C   BACKWARDS TEMP LABEL REFERENCE
    000036  00B4  0000                          HLT
    000037                                      CTRL      LINK PROG2
    000038  00B5  0096                          END       SMPMAC,START
0000 ERR COUNT
 I     II     III      IV       V                                  VI
```

Legend:

I — Optional error flag(s):

A — Operand field format error
C — Numeric conversion error
D — Short displacement out of range
E — Illegal address expression
F — Illegal forward reference
H — Improper header
L — Label field format error
M — Multiply-defined symbol
N — No matching left parenthesis
O — Illegal operation code
P — Assembler control statement error
Q — Address<0 or ⩾32K
R — Illegal register reference
S — Improper statement format
T — Truncation warning for string constant
U — Undefined symbol
X — Expression too complex
Z — Conditional assembly error

There can be up to four error flags per line. If there are more than four errors, only the first four are listed and included in the error count; subsequent errors are ignored.

II — Record number; a 6-digit decimal representation corresponding to the sequential count of the number of logical records read.

III — Program counter; 4-digit hexadecimal representation of the relative address of the corresponding source statement on the right-hand side of the listing.

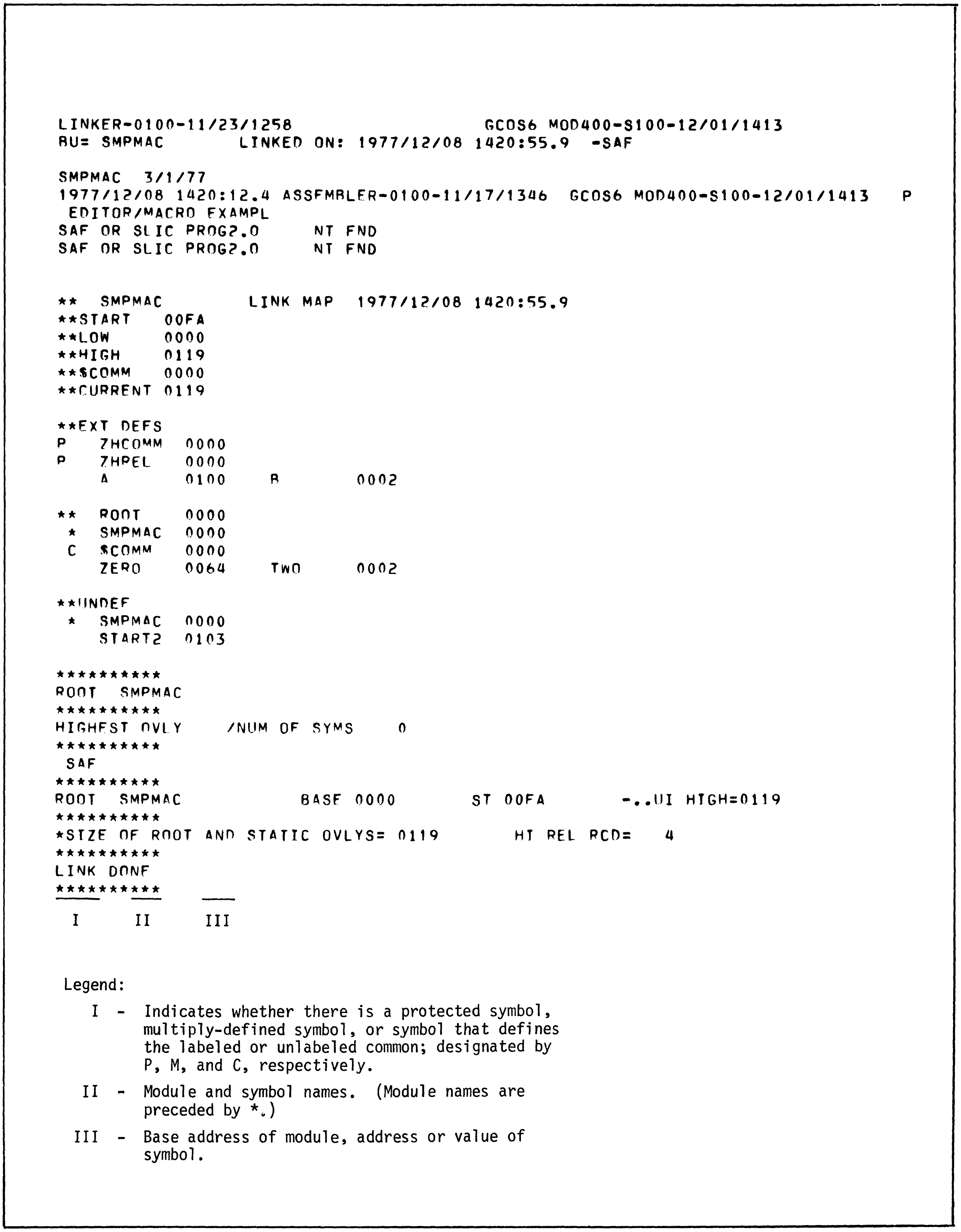
IV — Machine code; 4-, 8-, or 12-digit hexadecimal representation of the corresponding assembly language instruction or Assembler control statement shown on the right-hand side of the listing.

V — Type flag; 1-character flag (preferably a nonhexadecimal digit) that specifies the label type of the referenced symbol:

K — Common
T — Temporary
X — External
P — P-relative reference to external or common symbol

VI — Verbatim representation, including comments, of the source statement, as defined in the Assembly Language manual.

Figure 6-4. Assembler Output Listing (SMPMAC)



```
LINKER-0100-11/23/1258                    GCOS6 MOD400-S100-12/01/1413
BU= SMPMAC        LINKED ON: 1977/12/08 1420:55.9  -SAF

SMPMAC  3/1/77
1977/12/08 1420:12.4 ASSEMBLER-0100-11/17/1346  GCOS6 MOD400-S100-12/01/1413   P
 EDITOR/MACRO EXAMPL
SAF OR SLIC PROG2.0      NT FND
SAF OR SLIC PROG2.0      NT FND


**  SMPMAC         LINK MAP  1977/12/08 1420:55.9
**START    00FA
**LOW      0000
**HIGH     0119
**$COMM    0000
**CURRENT  0119

**EXT DEFS
P   ZHCOMM   0000
P   ZHPEL    0000
    A        0100     B        0002

**  ROOT     0000
 *  SMPMAC   0000
 C  $COMM    0000
    ZERO     0064     TWO      0002

**UNDEF
 *  SMPMAC   0000
    START2   0103

**********
ROOT  SMPMAC
**********
HIGHEST OVLY    /NUM OF SYMS    0
**********
 SAF
**********
ROOT  SMPMAC            BASE 0000       ST 00FA       -..UI HIGH=0119
**********
*SIZE OF ROOT AND STATIC OVLYS= 0119        HI REL RCD=   4
**********
LINK DONE
**********
____   ___     ____
  I     II      III
```

Legend:

I - Indicates whether there is a protected symbol, multiply-defined symbol, or symbol that defines the labeled or unlabeled common; designated by P, M, and C, respectively.

II - Module and symbol names. (Module names are preceded by *.)

III - Base address of module, address or value of symbol.

Figure 6-5. Linker Output Listing (SMPMAC)

SAMPLE ASSEMBLY LANGUAGE MULTITASK PROGRAM (BRDCST)

Figure 6-6 contains a sample terminal session to compile, link, execute, and start debugging the assembly language, program BRDCST, on that system. A specialized system is configured with CLM_USER containing the following configuration directives:

```
DEVICE KSR00,5,0,X'0500',CONSOLE
MEMPOOL S,,5000
DEVICE CDR00,26,26,X'1300'
MEMPOOL E,AA,14336,,BB,11264
MEMPOOL B,,8850


SYS 60,16,SSIP,3
DEVICE DSK01,17,17,X'480'
DEVICE DSK02,18,18,X'1200'
DEVICE DSK03,19,19,X'1280'
DEVICE LPT00,20,20,X'1380',LPT00
QUIT
```

The typeout illustrates the following points. A task group, $H, is spawned. Editor is used to print the file containing BRDCST source text, a portion of which is shown in Example 1. The Macro Preprocessor, required for processing of $IORB, $CRTSK, and $RQTSK macro calls, is not on the directory search path and a full pathname must be used. A task group, BC, in which to execute BRDCST is created and BRDCST is loaded for execution.

Example 1 is a listing of BRDCST. It is presented to illustrate how tasks are created and invoked in an assembly language multitask program.

```
SG $H H.L.A 38 >SPD>CONSOLE -OUT >SPD>CONSOLE -POOL AA -WD ^ZOOBOO
($S)RDY:
 C :$H:
RDN
($H)GROUP READY
($S)GROUP $H DID NOT ACCEPT INPUT
RDN
($H)RDY:
CWD ^ETSCOM>MAN_EX
($H)RDY:
FO >SPD>LPTOO
ED
($H)EDIT rrrr-mm/dd/hhmm
R BRDCST.P
1,$P
Q
FO >SPD>CONSOLE
($H)RDY:
^Zrrr02>SYSLIB2>MACROP BRDCST -SZ 20 -IC
($H)MACROP rrrr-mm/dd/hhmm
($H)0000 ERR COUNT
($H)RDY:
ASSEM BRDCST -SIZE 1 -COUT >SPD>LPTOO
($H)ASSEM rrrr-mm/dd/hhmm
($H)0000 ERR COUNT
($H)RDY:
LINKER BRDCST -C >SPD>LPTOO -S 2
($H)LINKER rrrr BU=BRDCST LINKED ON: yyyy/mm/dd nnmm:ss.t
IN ^ETSCOM>MAN_EX
LN BRDCST
START BRDCST
MP
QT
($H)ROOT >BRDCST
($H)LINK DONE
($H)RDY:
 C :$S:
CG BC 40 -LRN 30 -POOL BB -EFN ^ETSCOM>MAN_EX>BRDCST
($S)RDY:
EGR BC B.E.N >SPD>CONSOLE -WD ^ETSCOM -OUT >SPD>CONSOLE
($S)RDY:
```

Figure 6-6. Sample Terminal Session (BRDCST)

Example 1:

Partial Program Listing of BRDCST

```
          TITLE BRDCST
          LIBM         >LDD>MACRO>EXEC_LIB'
*
*         THIS TEST PROGRAM IS A
*         MEDIA TRANSCRIPTION TEST.
*         IT CAN EXECUTE AS AN
*         ON-LINE OR BATCH
*         DRIVER TEST......
            :                 :
            :                 :
DEVTBL    RESV  0
          DC    <CRDBLK          LRN 26 )
          DC    <TTYBLK          LRN 13 }  POINTERS TO DEVICE I/O
          DC    <DSKBLI          LRN 17 }  REQUEST BLOCKS
          DC    <PRTBLK          LRN 20 )
          DC     <DSKBLO          LRN 18
          DC     <TTYOUT          LRN 12          POINTERS TO DEVICE I/O
          DC     <ASRINP          LRN 10          REQUEST BLOCKS
          DC     <ASROUT          LRN 11
*
CRDBLK    $IORB 26,WAIT,,BUFFER,,80
*
TTYBLK    $IORB 13,WAIT,,BUFFER,,80
*                                                 I/O REQUEST BLOCK DEFINITIONS
DSKBLI    $IORB 17,WAIT,,BUFFER,,80               USING $IORB MACRO CALLS
*
DSKBLO    $IORB 18,WAIT,,BUFFER,,80
*
            :              :

BUFFER    RESV  80
          RESV  80
          RESV  80
TILRN     DC    26                                BUFFER, LRN AND LEVEL
TO1LRN    DC    27                                DEFINITIONS
TILVL     DC    1
TO1LVL    DC    2
            :              :

*
*               TASK 1 (INPUT) REQUEST BLOCK LRN 26
*
TASK01    $TRB  26,WAIT,,INSTRT
*                                                 TASK REQUEST BLOCK
*               TASK 2 (OUTPUT) REQUEST BLOCK LRN 27   DEFINITIONS USING
*                                                 $TRB MACRO CALLS
TASK02    $TRB  27,WAIT,,OUTSTR
*
            :              :

*
*         SET USER BIT TO INDICATE DISK
BRDCST    LDR   $R2,=Z'0021'
          LDR   $R1,<DSKBLI+$AF                   PROGRAM ENTRY POINT AND
          STH   $R2,=$R1                          INITIALIZATION CODE
          STR   $R1,<DSKBLI+$AF
            :              :

*
*         CREATE INPUT, OUTPUT TASKS
          $CRTSK TILRN,TILVL,INSTRT               TASK CREATION USING
          $CRTSK TO1LRN,TO1LVL,OUTSTR             $CRTSK MACRO CALLS
            :              :

*         INPUT TASK REQUEST
*
*
INPUTR    $RQTSK TASK01                           ENTRY OF TASK REQUESTS
*                                                 USING $RQTSK MACRO CALLS
*
*         OUTPUT TASK REQUEST
*
          $RQTSK TASK02
*
            :              :

             TASK EXECUTION CODE

            :              :
*
*
*         XDEFS AND XLOCS                         XDEFS AND XLOCS FOR
*                                                 LINKER PROCESSING
*         XDEF BRDCST
          END  BRDCST
```

Section 7
Using the COBOL Compiler

This section illustrates the use of the COBOL compiler to construct programs written in the COBOL language. It shows how to load a source program from a card deck into a mass storage COBOL source file and how to subsequently invoke the compiler to process the source program from the mass storage file. Two samples are presented; one shows the procedure for running an application, and the other is the ouput listing from an actual compilation and link.

SAMPLE CARD-TO-DISK-FILE PROGRAM (CARDIN)

Example 1 is a sample COBOL source program that places data read from cards onto a disk file. The following paragraphs illustrate a procedure for creating application files, loading source, compiling, linking, and executing. System startup has created the application task group $H. After startup, the current working directory is ^Zrrr01>SYSLIB1. The following summarizes the contents of volumes used in commands:

Volume	**Device Unit**	**Contents**
Zrrr00	DSK00	Bootstrap, Monitor, Linker
Zrrr01	DSK01	SYSLIB1, SYSLIB2
Zrrr04	DSK02	COBOL Compiler
VOL03	DSK03	Application Files

Example 1:

Program Listing of CARDIN

```
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. CARDIN.
000030 ENVIRONMENT DIVISION.
000040 CONFIGURATION SECTION.
000050 INPUT-OUTPUT SECTION.
000060 FILE-CONTROL.
000070     SELECT CARD ASSIGN TO OA-CARD-READER.
000080     SELECT MASTER ASSIGN TO OC-MSD.
000090 DATA DIVISION.
000100 FILE SECTION.
000110 FD  CARD LABEL RECORDS OMITTED.
000113 01  CARD-REC PIC X(80).
000116 FD  MASTER LABEL RECORDS OMITTED.
000120 01  MASTER-REC PIC X(80).
000130 PROCEDURE DIVISION.
000140 CARDIN.
000150     OPEN INPUT CARD.
000160     OPEN OUTPUT MASTER.
000170 LOOP.
000180     READ CARD RECORD AT END GO TO EOF.
000190     MOVE CARD-REC TO MASTER-REC.
000200     WRITE MASTER-REC.
000210     GO TO LOOP.
000220 EOF.
000230     CLOSE CARD.
000240     CLOSE MASTER.
000250     STOP RUN.
000260 END COBOL
```

VOLUME AND FILE CREATION

```
ΔCΔ  :$H:
RDN
CV   >SPD>DSK03 -FT VOL03                              Format volume VOL03
CD    ^VOL03>SOURCE                                    Create directories for
CD    ^VOL03>OBJECT                                    source, object and user files
CD    ^VOL03>FILES
CF    ^VOL03>FILES>OLD_MASTER  -N_REL  -RSZ  128       Create user file
FO   >SPD>LPT00
LS   -PN   ^VOL03                                      List contents of created
LS   -PN   ^VOL03>OBJECT                               directories
LS   -PN   ^VOL03>FILES
FO
```

SOURCE LOADING

The following illustrates loading a source deck using the CP command. Place the source decks for CARDIN in the card reader in the following sequence:

CARDIN source deck
EOF (11-5-8-9) card

Enter the following command:

```
CP   >SPD>CDR00 ^VOL03>SOURCE>CARDIN.C
```

COMPILING WITH COBOL

In the following ECL commands, the working directory is ^VOL03>OBJECT. The search path for bound units (executable programs) is current working directory, LIB1, then LIB2, where their pathname is initially ^Zrrr01>SYSLIB1. The COBOL Compiler is not in any directories in the search path; its full pathname must be given. However, the command

```
COBOL   <SOURCE>CARDIN...
```

can be used if you change the pathname for the directory LIB2 by issuing an operator command to the system task group using:

```
Δ$SΔCSD  -LIB2    ^Zrrr04
```

(where Δ is exactly one space)

^VOL03>OBJECT will contain temporary work files required for the compiler and the created object files used by the Linker. The compiler argument LD will list source, data map, and errors; LO will, in addition, list the object text.

To compile CARDIN enter the following:

```
CWD     ^VOL03>OBJECT
COBOL   <SOURCE>CARDIN   -LO   -COUT   >SPD>LPT00
```

LINKING

The working directory is still ^VOL03>OBJECT. The Linker LIB directive directs the Linker to search the secondary directory for COBOL run-time routines (ZCRT) required for linking. To link, enter the following commands:

```
LINKER CARDIN -COUT >SPD>LPT00 -SIZE 4
  LIB   ^Zrrr04>ZCRT
  LINK  CARDIN
  MAP;QT
```

EXECUTING

The internal file names 0A and 0C translate to logical file numbers 01 and 03, respectively, and must be associated with the pathnames or the physical devices through a GET or ASSOC command. To execute the program, enter CARDIN.

Enter the following commands:

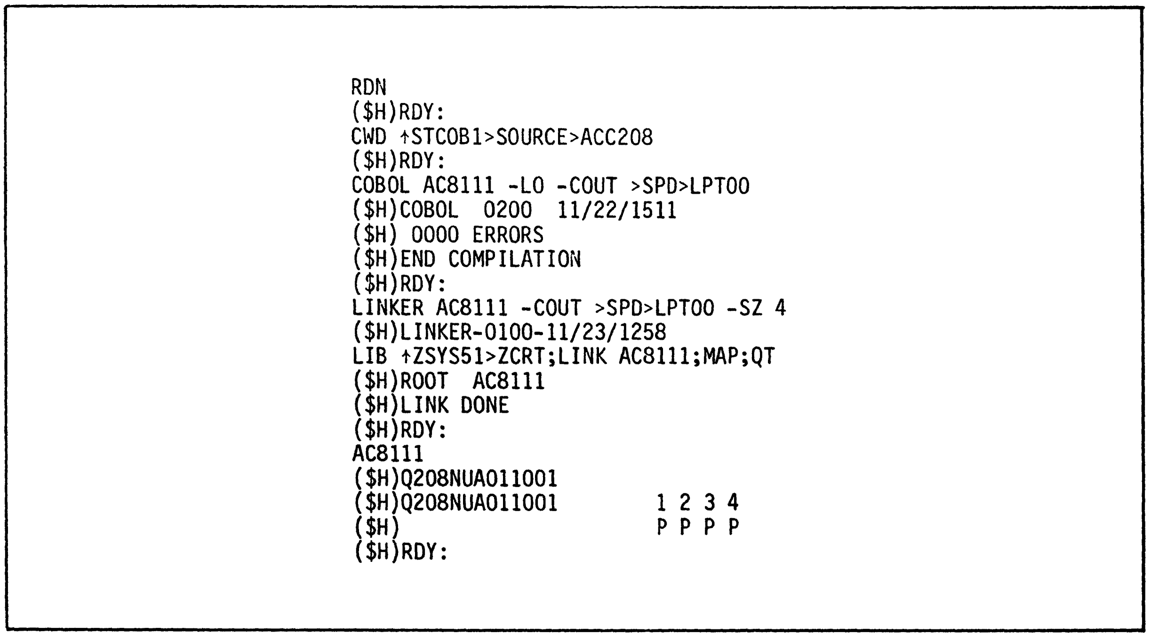
```
GET 01 >SPD>CDR00
GET 03 ^VOL03>FILES>OLD_MASTER
CARDIN
```

SAMPLE COBOL TERMINAL SESSION (AC8111)

Figure 7-1 illustrates an operator terminal session in which a system is configured, and the COBOL program AC8111 is compiled, linked, and executed on that system. The Entry Level COBOL compiler is specified in this session. To secify the Intermediate COBOL compiler, change the COBOL command and the LINKER LIB directive as follows.

```
COBOLI AC8111 -LO -COUT >SPD>LPT00
LIB ^ZSYS51>ZCIRT;LINK AC8111;MAP;QT
```

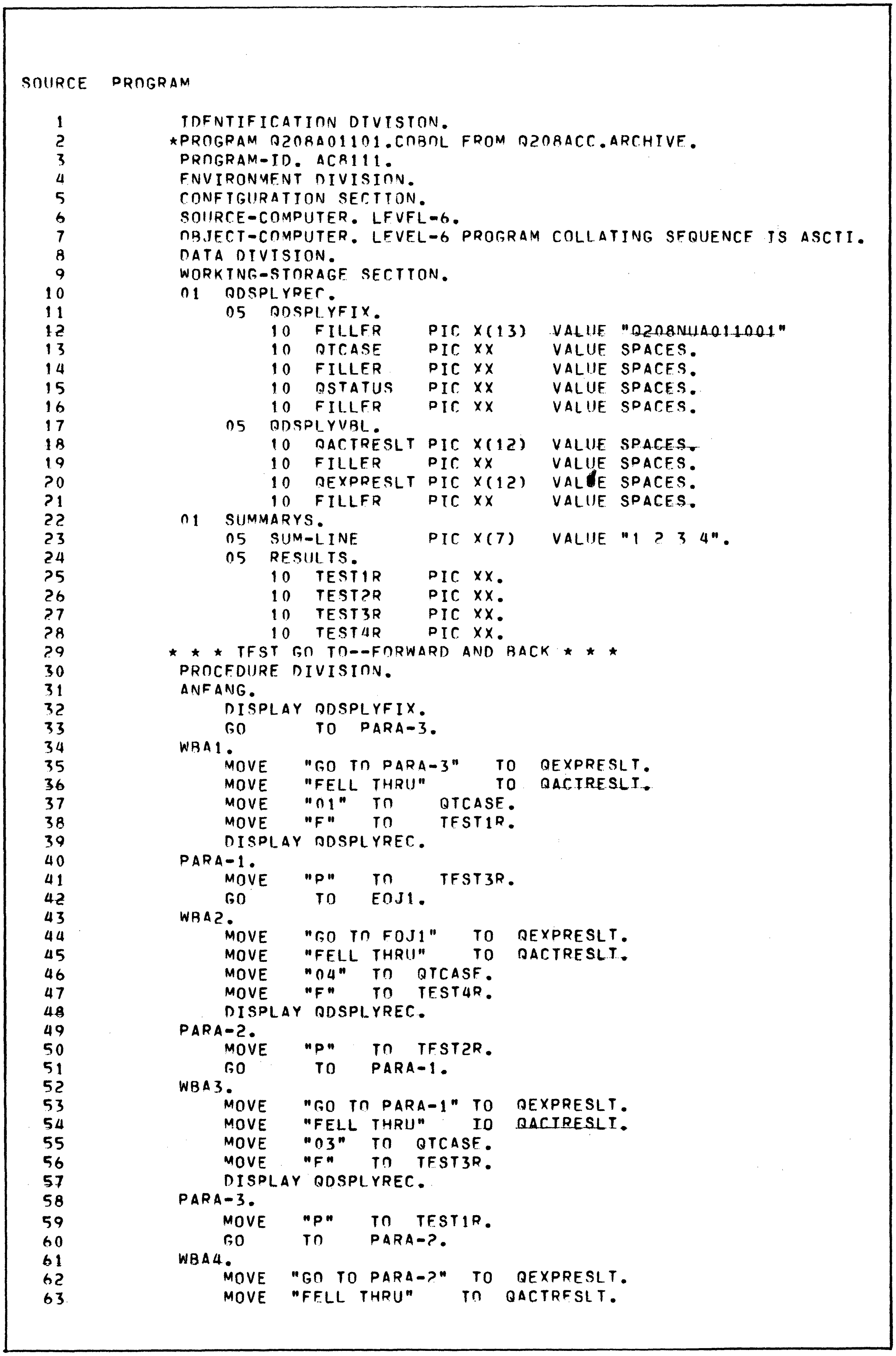
The LINKER LIB directive directs the Linker to search the secondary directory for COBOL run-time routines required for linking. To execute the program, enter AC8111.



```
RDN
($H)RDY:
CWD ↑STCOB1>SOURCE>ACC208
($H)RDY:
COBOL AC8111 -LO -COUT >SPD>LPT00
($H)COBOL  0200  11/22/1511
($H) 0000 ERRORS
($H)END COMPILATION
($H)RDY:
LINKER AC8111 -COUT >SPD>LPT00 -SZ 4
($H)LINKER-0100-11/23/1258
LIB ↑ZSYS51>ZCRT;LINK AC8111;MAP;QT
($H)ROOT  AC8111
($H)LINK DONE
($H)RDY:
AC8111
($H)Q208NUA011001
($H)Q208NUA011001        1 2 3 4
($H)                     P P P P
($H)RDY:
```

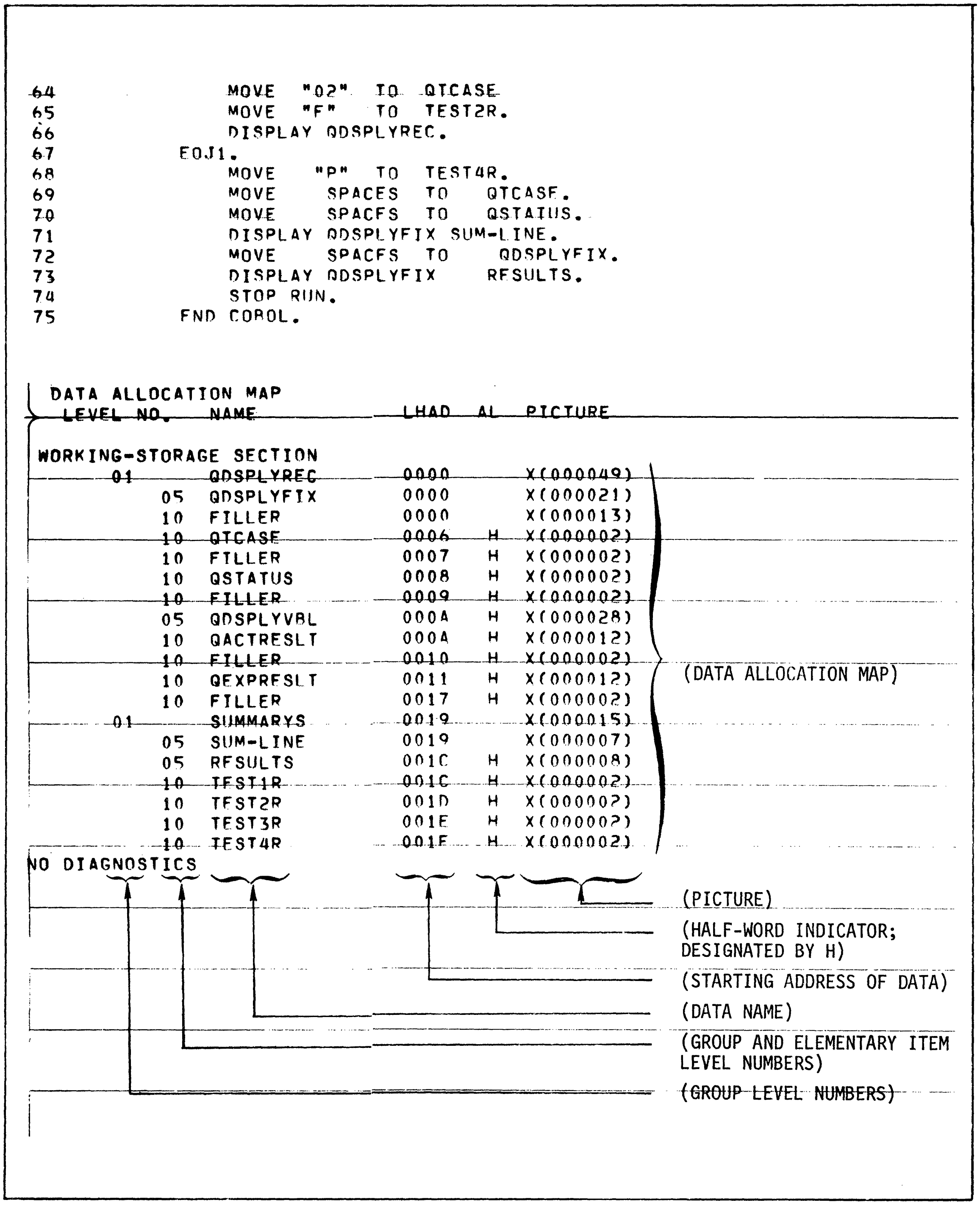
Figure 7-1. Sample Terminal Session (AC8111)

Figure 7-2 is a listing of the program AC8111, its compiled object text, and the output from the Linker. The program was compiled using the entry-level compiler.



```
SOURCE  PROGRAM

          IDENTIFICATION DIVISION.
         *PROGRAM Q208A01101.COBOL FROM Q208ACC.ARCHIVE.
          PROGRAM-ID. AC8111.
          ENVIRONMENT DIVISION.
          CONFIGURATION SECTION.
          SOURCE-COMPUTER. LEVEL-6.
          OBJECT-COMPUTER. LEVEL-6 PROGRAM COLLATING SEQUENCE IS ASCII.
          DATA DIVISION.
          WORKING-STORAGE SECTION.
          01  QDSPLYREC.
              05  QDSPLYFIX.
                  10  FILLER     PIC X(13)  VALUE "Q208NUA011001"
                  10  QTCASE     PIC XX     VALUE SPACES.
                  10  FILLER     PIC XX     VALUE SPACES.
                  10  QSTATUS    PIC XX     VALUE SPACES.
                  10  FILLER     PIC XX     VALUE SPACES.
              05  QDSPLYVBL.
                  10  QACTRESLT PIC X(12)  VALUE SPACES.
                  10  FILLER     PIC XX     VALUE SPACES.
                  10  QEXPRESLT PIC X(12)  VALUE SPACES.
                  10  FILLER     PIC XX     VALUE SPACES.
          01  SUMMARYS.
              05  SUM-LINE       PIC X(7)   VALUE "1 2 3 4".
              05  RESULTS.
                  10  TEST1R     PIC XX.
                  10  TEST2R     PIC XX.
                  10  TEST3R     PIC XX.
                  10  TEST4R     PIC XX.
         * * * TEST GO TO--FORWARD AND BACK * * *
          PROCEDURE DIVISION.
          ANFANG.
              DISPLAY QDSPLYFIX.
              GO      TO  PARA-3.
          WBA1.
              MOVE   "GO TO PARA-3"   TO  QEXPRESLT.
              MOVE   "FELL THRU"      TO  QACTRESLT.
              MOVE   "01"  TO     QTCASE.
              MOVE   "F"   TO     TEST1R.
              DISPLAY QDSPLYREC.
          PARA-1.
              MOVE   "P"   TO     TEST3R.
              GO      TO   EOJ1.
          WBA2.
              MOVE   "GO TO EOJ1"   TO  QEXPRESLT.
              MOVE   "FELL THRU"    TO  QACTRESLT.
              MOVE   "04"  TO  QTCASE.
              MOVE   "F"   TO  TEST4R.
              DISPLAY QDSPLYREC.
          PARA-2.
              MOVE   "P"   TO  TEST2R.
              GO      TO   PARA-1.
          WBA3.
              MOVE   "GO TO PARA-1" TO  QEXPRESLT.
              MOVE   "FELL THRU"    TO  QACTRESLT.
              MOVE   "03"  TO  QTCASE.
              MOVE   "F"   TO  TEST3R.
              DISPLAY QDSPLYREC.
          PARA-3.
              MOVE   "P"   TO  TEST1R.
              GO      TO      PARA-2.
          WBA4.
              MOVE  "GO TO PARA-2"  TO  QEXPRESLT.
              MOVE  "FELL THRU"    TO  QACTRESLT.
```

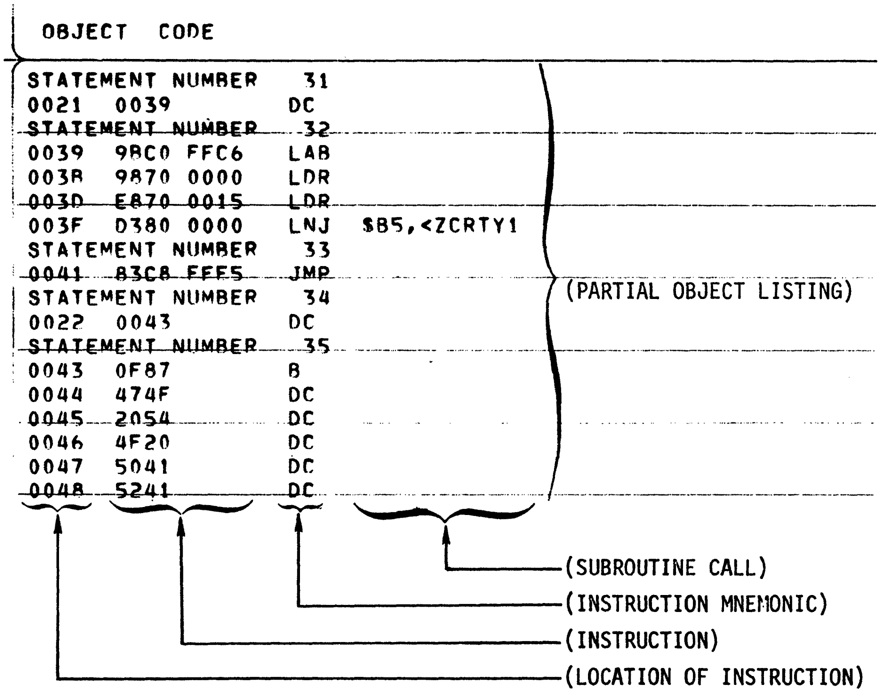
Figure 7-2. Sample Listings for AC8111



```
64              MOVE  "02"  TO  QTCASE
65              MOVE  "F"   TO  TEST2R.
66              DISPLAY QDSPLYREC.
67          EOJ1.
68              MOVE   "P"  TO  TEST4R.
69              MOVE    SPACES  TO   QTCASE.
70              MOVE    SPACES  TO   QSTATUS.
71              DISPLAY QDSPLYFIX SUM-LINE.
72              MOVE    SPACES  TO    QDSPLYFIX.
73              DISPLAY QDSPLYFIX    RESULTS.
74              STOP RUN.
75          END COBOL.


 DATA ALLOCATION MAP
  LEVEL NO.    NAME                LHAD  AL  PICTURE

WORKING-STORAGE SECTION
      01       QDSPLYREC           0000      X(000049)
          05   QDSPLYFIX           0000      X(000021)
          10   FILLER              0000      X(000013)
          10   QTCASE              0006  H   X(000002)
          10   FILLER              0007  H   X(000002)
          10   QSTATUS             0008  H   X(000002)
          10   FILLER              0009  H   X(000002)
          05   QDSPLYVBL           000A  H   X(000028)
          10   QACTRESLT           000A  H   X(000012)
          10   FILLER              0010  H   X(000002)
          10   QEXPRESLT           0011  H   X(000012)
          10   FILLER              0017  H   X(000002)
      01       SUMMARYS            0019      X(000015)
          05   SUM-LINE            0019      X(000007)
          05   RESULTS             001C  H   X(000008)
          10   TEST1R              001C  H   X(000002)
          10   TEST2R              001D  H   X(000002)
          10   TEST3R              001E  H   X(000002)
          10   TEST4R              001F  H   X(000002)
NO DIAGNOSTICS
```

(DATA ALLOCATION MAP)
(PICTURE)
(HALF-WORD INDICATOR; DESIGNATED BY H)
(STARTING ADDRESS OF DATA)
(DATA NAME)
(GROUP AND ELEMENTARY ITEM LEVEL NUMBERS)
(GROUP LEVEL NUMBERS)

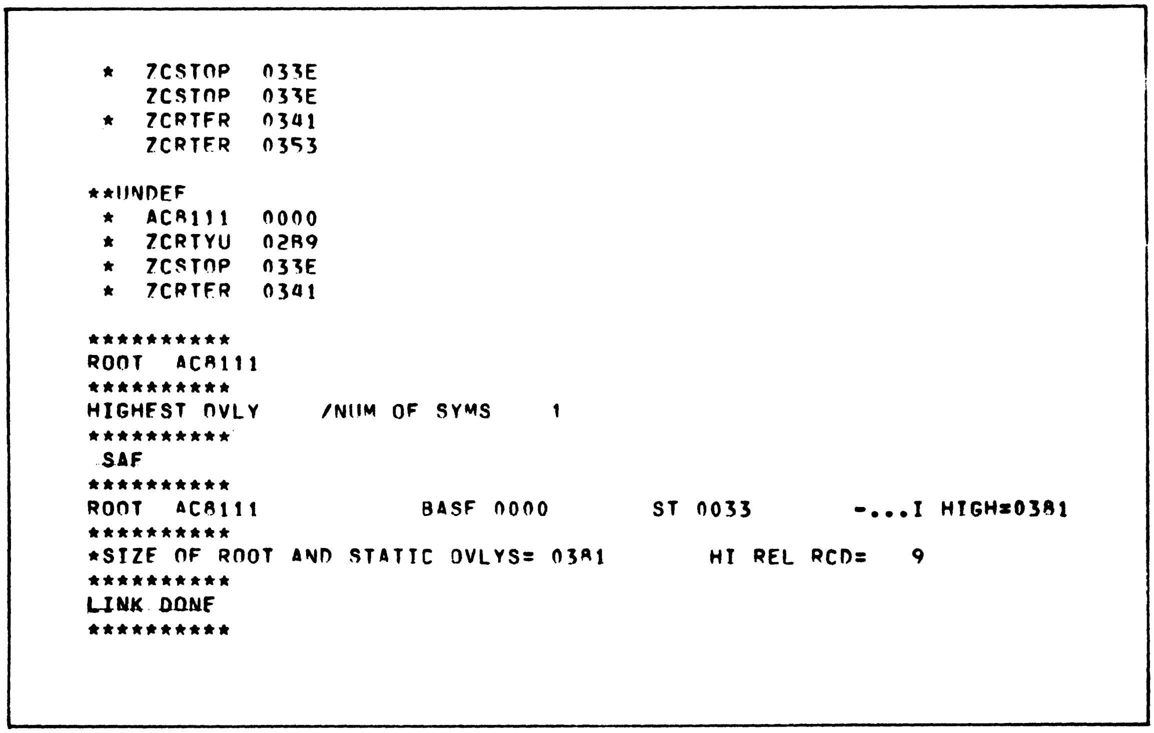
Figure 7-2 (cont). Sample Listings for AC8111



```
 OBJECT  CODE

STATEMENT NUMBER   31
0021  0039          DC
STATEMENT NUMBER   32
0039  9BC0 FFC6     LAB
003B  9870 0000     LDR
003D  E870 0015     LDR
003F  D380 0000     LNJ  $B5,<ZCRTY1
STATEMENT NUMBER   33
0041  83C8 FFF5     JMP
STATEMENT NUMBER   34
0022  0043          DC
STATEMENT NUMBER   35
0043  0F87          B
0044  474F          DC
0045  2054          DC
0046  4F20          DC
0047  5041          DC
0048  5241          DC
```

```
LINKER-0100-11/23/1258                      GCOS6 MOD400-S100-11/29/0620
BU= AC8111         LINKED ON: 1901/01/01 0002:44.1  -SAF

AC8111  01/01/01
 COBOL REV. 0200 DATE 01/01/01 TIME 0000 .

ZCRTYU  770208
HRS ASSEMBLER 2.49  06/02/77  1340.3 EDT THU
(C)  COPYRIGHT 1976 BY HONEYWELL INFORMATION SYSTEMS INC

ZCSTOP  770208
HRS ASSEMBLER 2.49  06/02/77  1336.9 EDT THU
(C)  COPYRIGHT 1976 BY HONEYWELL INFORMATION SYSTEMS INC

ZCRTER  770208
HRS ASSEMBLER 2.49  06/02/77  1934.4 EDT THU
(C)  COPYRIGHT 1976 BY HONEYWELL INFORMATION SYSTEMS INC


**  AC8111          LINK MAP  1901/01/01 0002:44.1
**START    0033
**LOW      0000
**HIGH     0381
**CURRENT  0381

**EXT DEFS
P    ZHCOMM  0000
P    ZHREL   0000

**   ROOT    0000
 *   AC8111  0000
     AC8111  0033     ZCMAIN  0031
 *   ZCRTYU  02B9
     ZCRTY1  02F1     ZCRTY2  0312    ZCRTY3  0334
```

Figure 7-2 (cont). Sample Listings for AC8111



```
     *   ZCSTOP   033E
         ZCSTOP   033E
     *   ZCRTER   0341
         ZCRTER   0353

    **UNDEF
     *   AC8111   0000
     *   ZCRTYU   02B9
     *   ZCSTOP   033E
     *   ZCRTER   0341

    **********
    ROOT  AC8111
    **********
    HIGHEST OVLY     /NUM OF SYMS     1
    **********
     SAF
    **********
    ROOT  AC8111              BASE 0000       ST 0033        -...I HIGH=0381
    **********
    *SIZE OF ROOT AND STATIC OVLYS= 0381       HI REL RCD=   9
    **********
    LINK DONE
    **********
```

Figure 7-2 (cont). Sample Listings for AC8111

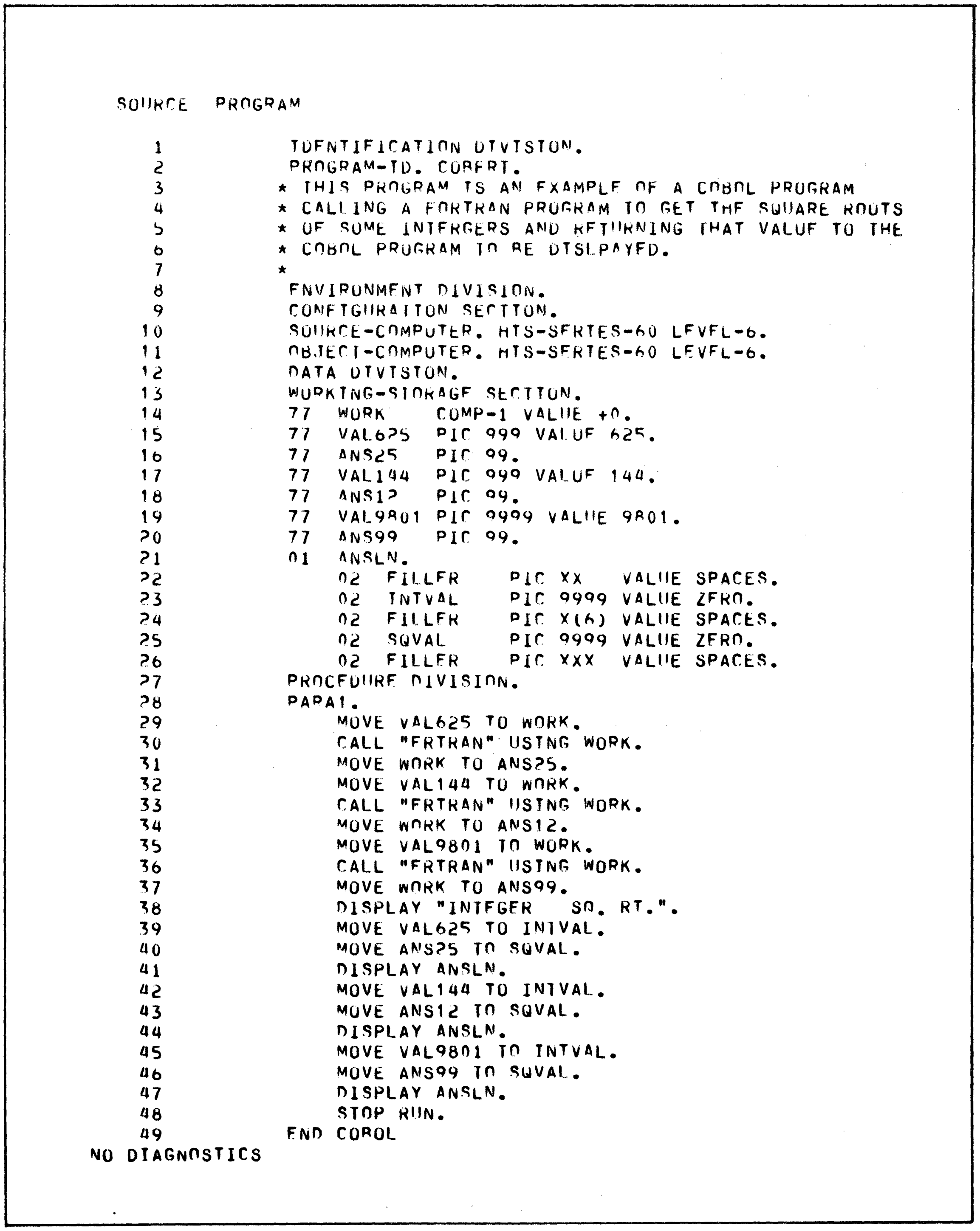
CALLING FORTRAN ROUTINES FROM AN ENTRY-LEVEL COBOL MAIN PROGRAM

Entry-Level COBOL programs can call FORTRAN subroutines and conversely. This enables a COBOL application to utilize the features of the FORTRAN language, such as the intrinsic routines, and FORTRAN run-time libraries.

The COBOL main program must be linked with all the called FORTRAN routines to form one bound unit. The FORTRAN routines and libraries must either be in the working directory or one of the libraries searched by the Linker, as specified by the Linker LIB and LIBn directives.

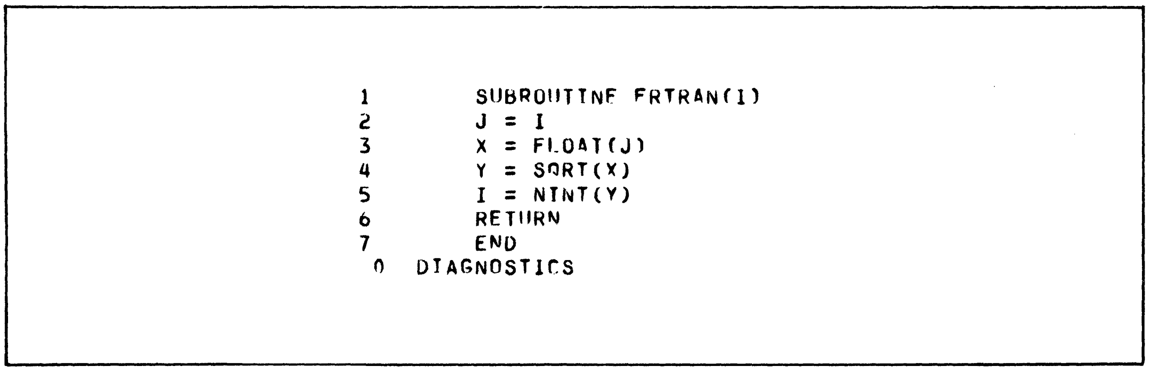
Figure 7-3 is a sample Entry-Level COBOL source program, COBFRT, whose function is to calculate and print the square roots of three integers. Since the COBOL library does not have a square root routine, a FORTAN subroutine, FRTRAN in Figure 7-4, is used to convert the passed COBOL integer argument values to read values and call the FORTRAN square root routine.

The commands entered from the operator terminal are listed in Figure 7-5. COBFRT.O and FRTRAN.O are both in the working directory FRTCOB, the COBOL run-time library, ZCRT, is in the directory specified by the Linker directive LIB, and the FORTRAN run-time library, ZFRT, is in the directory specified by LIB2. The system volume, ZSYS51, contains the FORTRAN and COBOL compilers, ZFRT, ZCRT and the operating system software. Volume FRTCOB contains the source modules (COBFRT.C and FRTRAN.F), the object modules (COBRFT.O and FRTRAN.O) and the linked bound unit COBFRT.



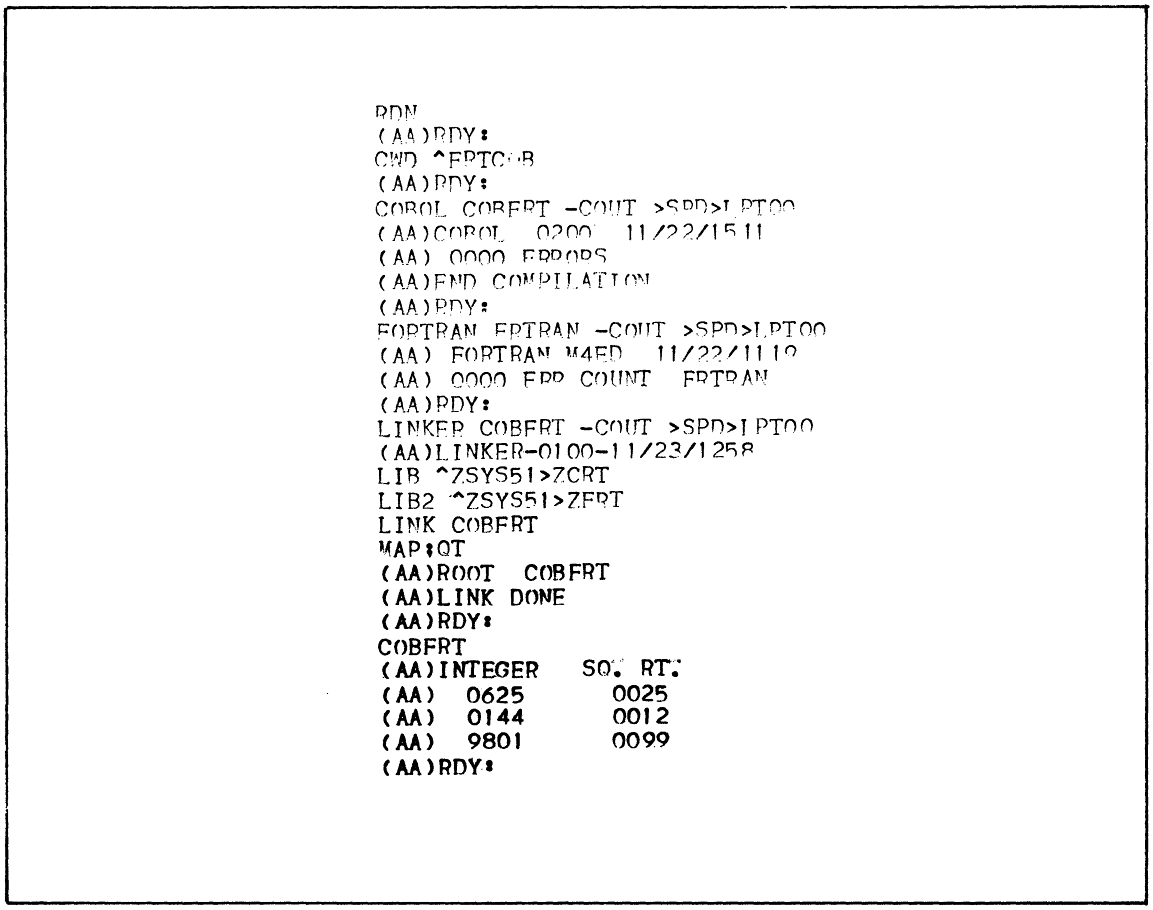
```
 SOURCE  PROGRAM

     1          IDENTIFICATION DIVISION.
     2          PROGRAM-ID. COBFRT.
     3         * THIS PROGRAM IS AN EXAMPLE OF A COBOL PROGRAM
     4         * CALLING A FORTRAN PROGRAM TO GET THE SQUARE ROUTS
     5         * OF SOME INTERGERS AND RETURNING THAT VALUE TO THE
     6         * COBOL PROGRAM TO BE DISLPAYED.
     7         *
     8          ENVIRONMENT DIVISION.
     9          CONFIGURAITON SECTION.
    10          SOURCE-COMPUTER. HIS-SERIES-60 LEVEL-6.
    11          OBJECT-COMPUTER. HIS-SERIES-60 LEVEL-6.
    12          DATA DIVISION.
    13          WORKING-STORAGE SECTION.
    14          77  WORK     COMP-1 VALUE +0.
    15          77  VAL625   PIC 999 VALUE 625.
    16          77  ANS25    PIC 99.
    17          77  VAL144   PIC 999 VALUE 144.
    18          77  ANS12    PIC 99.
    19          77  VAL9801 PIC 9999 VALUE 9801.
    20          77  ANS99    PIC 99.
    21          01  ANSLN.
    22              02  FILLER     PIC XX    VALUE SPACES.
    23              02  INTVAL     PIC 9999 VALUE ZERO.
    24              02  FILLER     PIC X(6) VALUE SPACES.
    25              02  SQVAL      PIC 9999 VALUE ZERO.
    26              02  FILLER     PIC XXX  VALUE SPACES.
    27          PROCEDURE DIVISION.
    28          PARA1.
    29              MOVE VAL625 TO WORK.
    30              CALL "FRTRAN" USING WORK.
    31              MOVE WORK TO ANS25.
    32              MOVE VAL144 TO WORK.
    33              CALL "FRTRAN" USING WORK.
    34              MOVE WORK TO ANS12.
    35              MOVE VAL9801 TO WORK.
    36              CALL "FRTRAN" USING WORK.
    37              MOVE WORK TO ANS99.
    38              DISPLAY "INTEGER   SQ. RT.".
    39              MOVE VAL625 TO INTVAL.
    40              MOVE ANS25 TO SQVAL.
    41              DISPLAY ANSLN.
    42              MOVE VAL144 TO INTVAL.
    43              MOVE ANS12 TO SQVAL.
    44              DISPLAY ANSLN.
    45              MOVE VAL9801 TO INTVAL.
    46              MOVE ANS99 TO SQVAL.
    47              DISPLAY ANSLN.
    48              STOP RUN.
    49          END COBOL
NO DIAGNOSTICS
```

Figure 7-3. COBOL Listing of COBFRT



```
      SUBROUTINE FRTRAN(I)
      J = I
      X = FLOAT(J)
      Y = SQRT(X)
      I = NINT(Y)
      RETURN
      END
 0  DIAGNOSTICS
```

Figure 7-4. FORTRAN Listing of FRTRAN



```
RDN
(AA)RDY:
CWD ^FRTCOB
(AA)RDY:
COBOL COBFRT -COUT >SPD>LPT00
(AA)COBOL  0200  11/22/1511
(AA) 0000 ERRORS
(AA)END COMPILATION
(AA)RDY:
FORTRAN FRTRAN -COUT >SPD>LPT00
(AA) FORTRAN M4ED  11/22/1119
(AA) 0000 ERR COUNT  FRTRAN
(AA)RDY:
LINKER COBFRT -COUT >SPD>LPT00
(AA)LINKER-0100-11/23/1258
LIB ^ZSYS51>ZCRT
LIB2 ^ZSYS51>ZFRT
LINK COBFRT
MAP:QT
(AA)ROOT  COBFRT
(AA)LINK DONE
(AA)RDY:
COBFRT
(AA)INTEGER   SQ. RT.
(AA)  0625      0025
(AA)  0144      0012
(AA)  9801      0099
(AA)RDY:
```

Figure 7-5. Operator Terminal Session for COBFRT

Section 8
Using the FORTRAN Compiler

This section illustrates the use of the FORTRAN compiler to construct programs written in the FORTRAN language, and to perform FORTRAN chaining.

SAMPLE FORTRAN TERMINAL SESSION (MATINV)

Figure 8-1 illustrates an operator terminal session in which the FORTRAN program MATINV is compiled, linked, and executed. The FORTRAN compiler is on the search path, and, therefore, a full pathname is not needed to locate the compiler. The Linker LIB directive directs the linker to search the secondary directory for FORTRAN runtime routines (ZFRT) required for linking. Two files, unit number 2 and 3, are associated with device pathnames. To execute the program, enter MATINV. Figure 8-2 shows the MATINV source listing and the linker output when the program was linked.

```
FORTRAN  VL782 SUB1 MATINV -COUT >SPD>LPT00
($H) FORTRAN
($H) 0000 ERR COUNT  MATINV
($H)BDY:
LINKER  VOL2 TEST MATINV -COUT >SPD>LPT00
($H)
LINKER   ...
LIB  ZF0400 ZFRT
LINK MATINV;MP;QT
($H)ROOT  MATINV
($H)LINK DONE
($H)RDY:
ASSOC 2 >SPD>CDR00
($H)RDY:
ASSOC 3 >SPD>LPT00
($H)RDY:
MATINV
($H) STOP
($H)RDY:
```

Figure 8-1. Sample Terminal Session (MATINV)

FORTRAN CHAINING

A method of creating and controlling execution of overlays within FORTRAN programs to conserve memory space, commonly known as CHAINing, can be used wherein an executable bound unit (overlay) is executed as a chain prior to invoking the next chain.

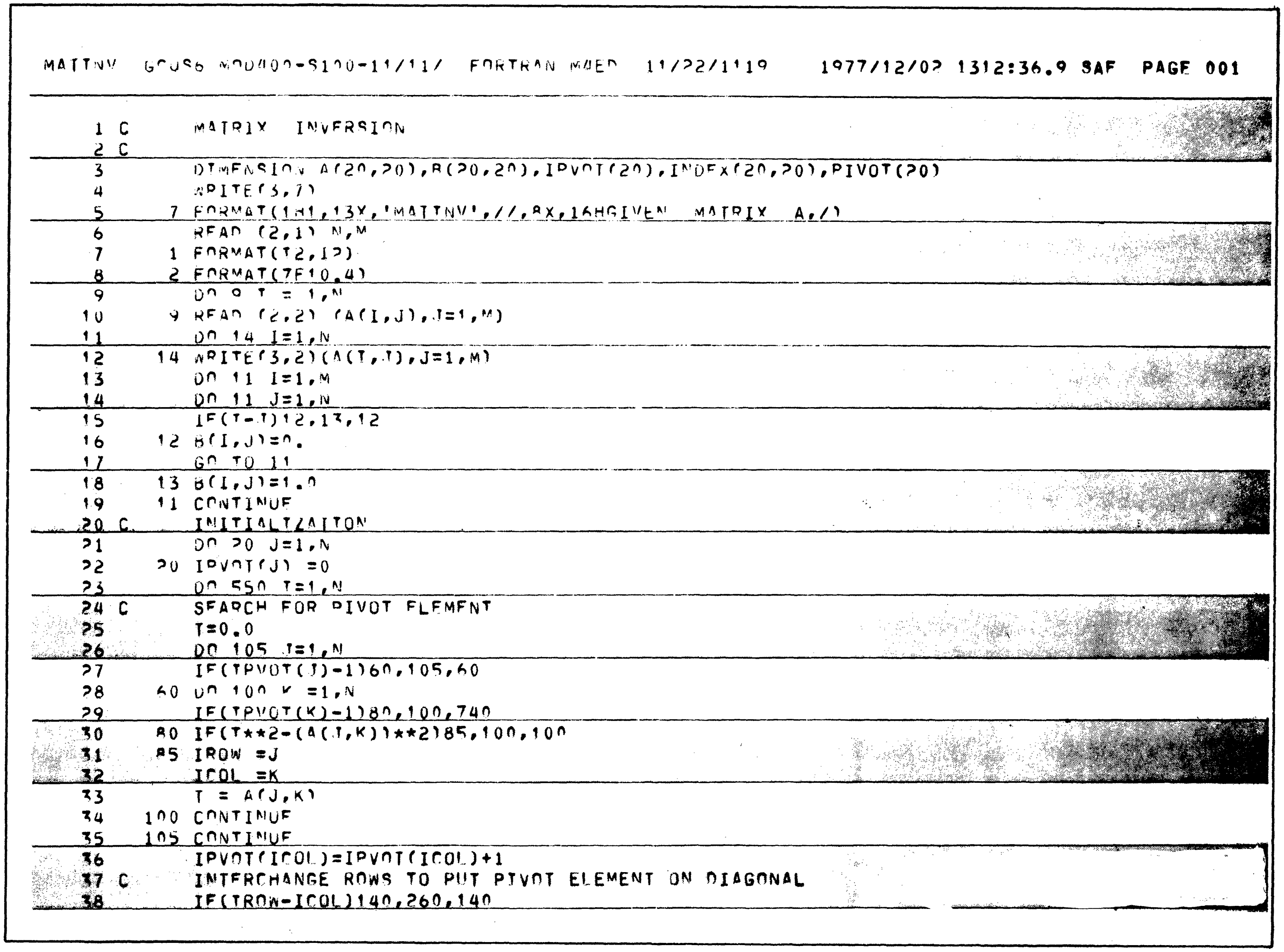
The source statement for referencing a chain is:

CALL   CHAIN(e)

where e is an integer expression resulting in a value greater than or equal to zero, identifying the chain to be loaded. Proper linking results in a bound unit with overlays that require minimum memory for execution.

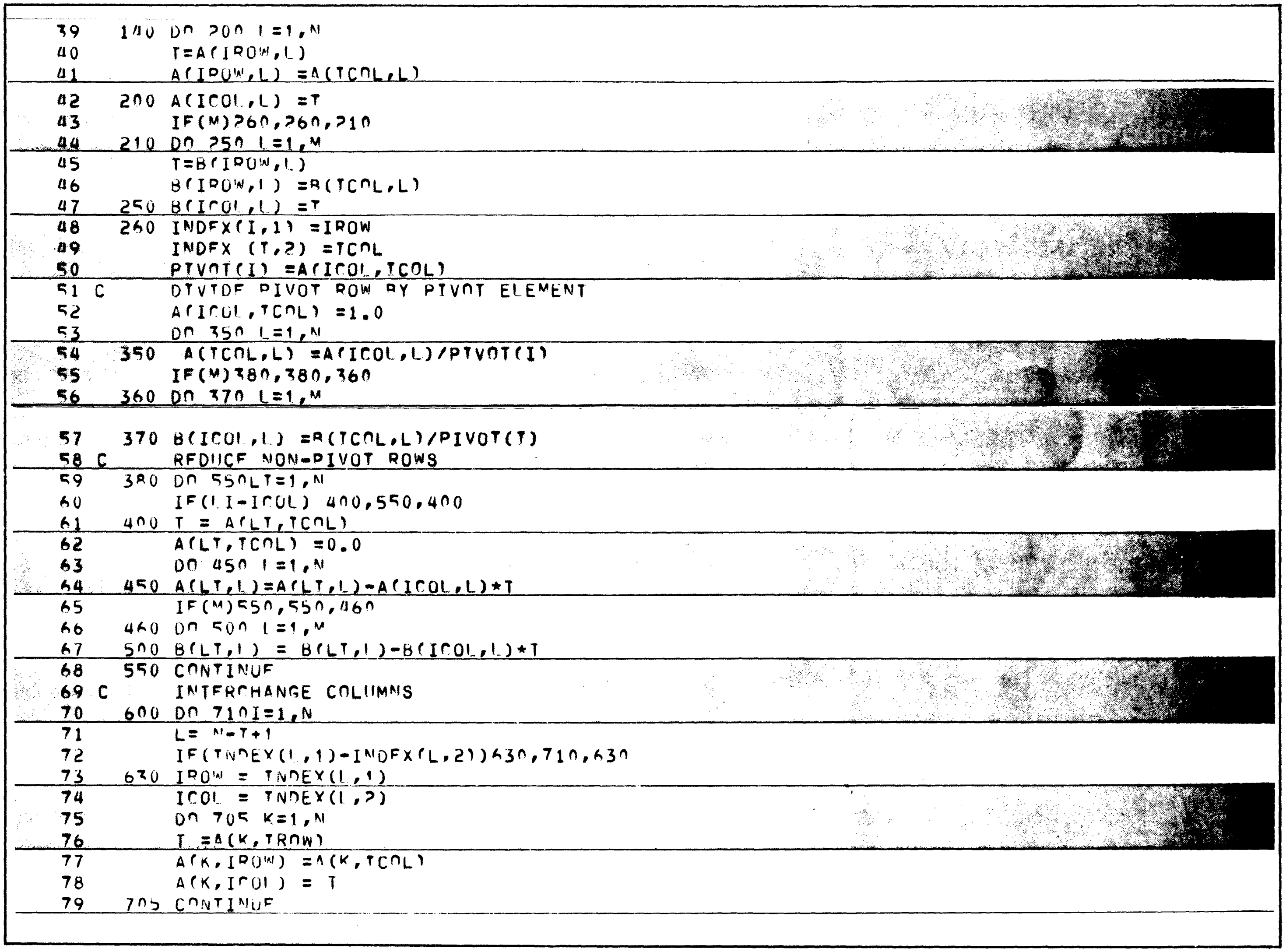
Although there are no rules for defining the best method of segmenting a FORTRAN application into chains, the following should be considered:

1. The largest chain determines the overall memory requirement.
2. Any chain may be called by any other chain as many times as required.



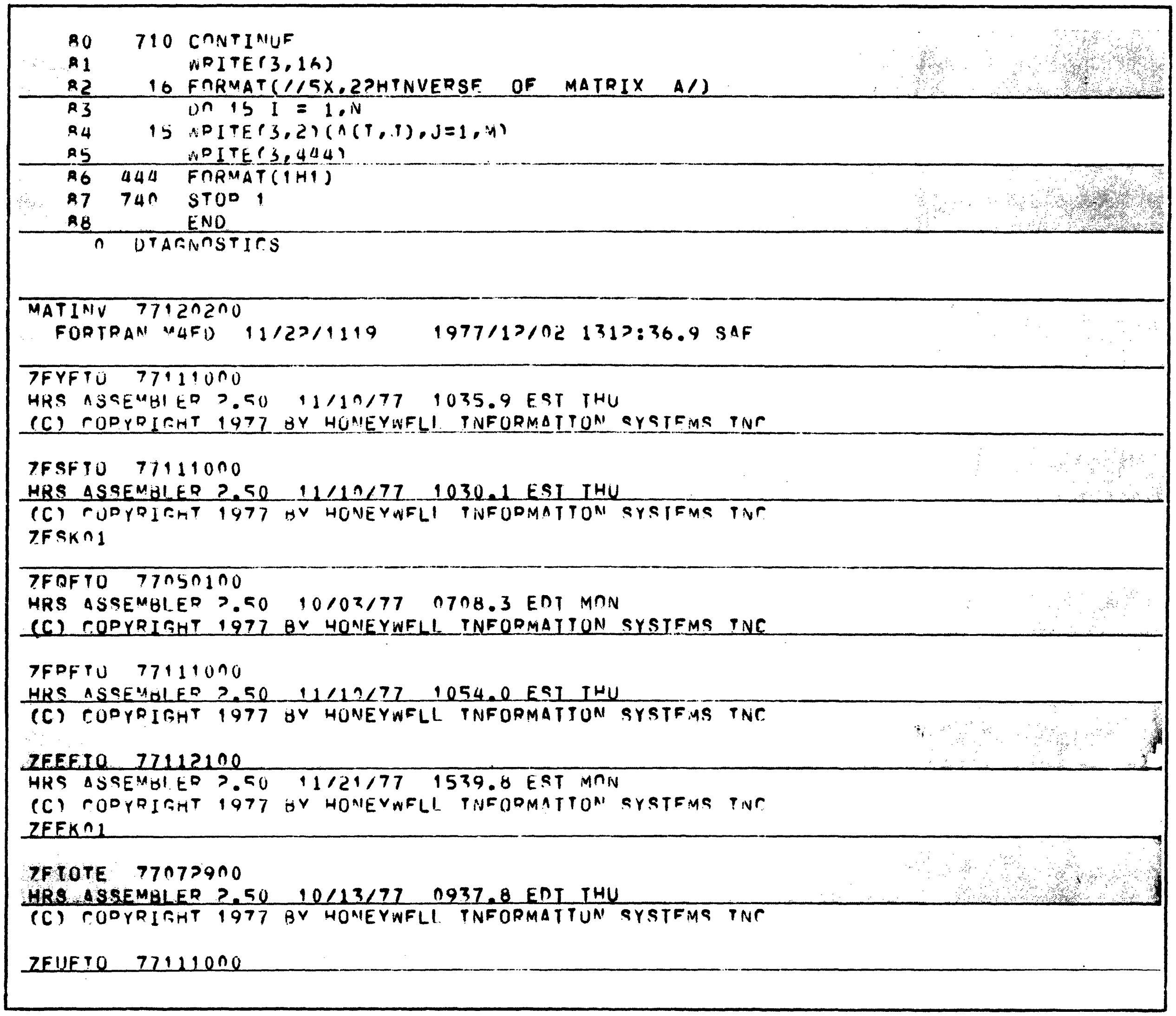
```
MATINV   GCOS6 MOD400-S100-11/11/  FORTRAN M4ER  11/22/1119     1977/12/02 1312:36.9 SAF  PAGE 001

C     MATRIX  INVERSION
C
      DIMENSION A(20,20),B(20,20),IPVOT(20),INDEX(20,20),PIVOT(20)
      WRITE(3,7)
    7 FORMAT(1H1,13X,'MATINV',//,8X,16HGIVEN  MATRIX  A,/)
      READ (2,1) N,M
    1 FORMAT(I2,I2)
    2 FORMAT(7F10.4)
      DO 9 I = 1,N
    9 READ (2,2) (A(I,J),J=1,M)
      DO 14 I=1,N
   14 WRITE(3,2)(A(I,J),J=1,M)
      DO 11 I=1,M
      DO 11 J=1,N
      IF(I-J)12,13,12
   12 B(I,J)=0.
      GO TO 11
   13 B(I,J)=1.0
   11 CONTINUE
C     INITIALIZAITON
      DO 20 J=1,N
   20 IPVOT(J) =0
      DO 550 I=1,N
C     SEARCH FOR PIVOT ELEMENT
      T=0.0
      DO 105 J=1,N
      IF(IPVOT(J)-1)60,105,60
   60 DO 100 K =1,N
      IF(IPVOT(K)-1)80,100,740
   80 IF(T**2-(A(J,K))**2)85,100,100
   85 IROW =J
      ICOL =K
      T = A(J,K)
  100 CONTINUE
  105 CONTINUE
      IPVOT(ICOL)=IPVOT(ICOL)+1
C     INTERCHANGE ROWS TO PUT PIVOT ELEMENT ON DIAGONAL
      IF(IROW-ICOL)140,260,140
```

Figure 8-2. Source and Linker Output Listing (MATINV)



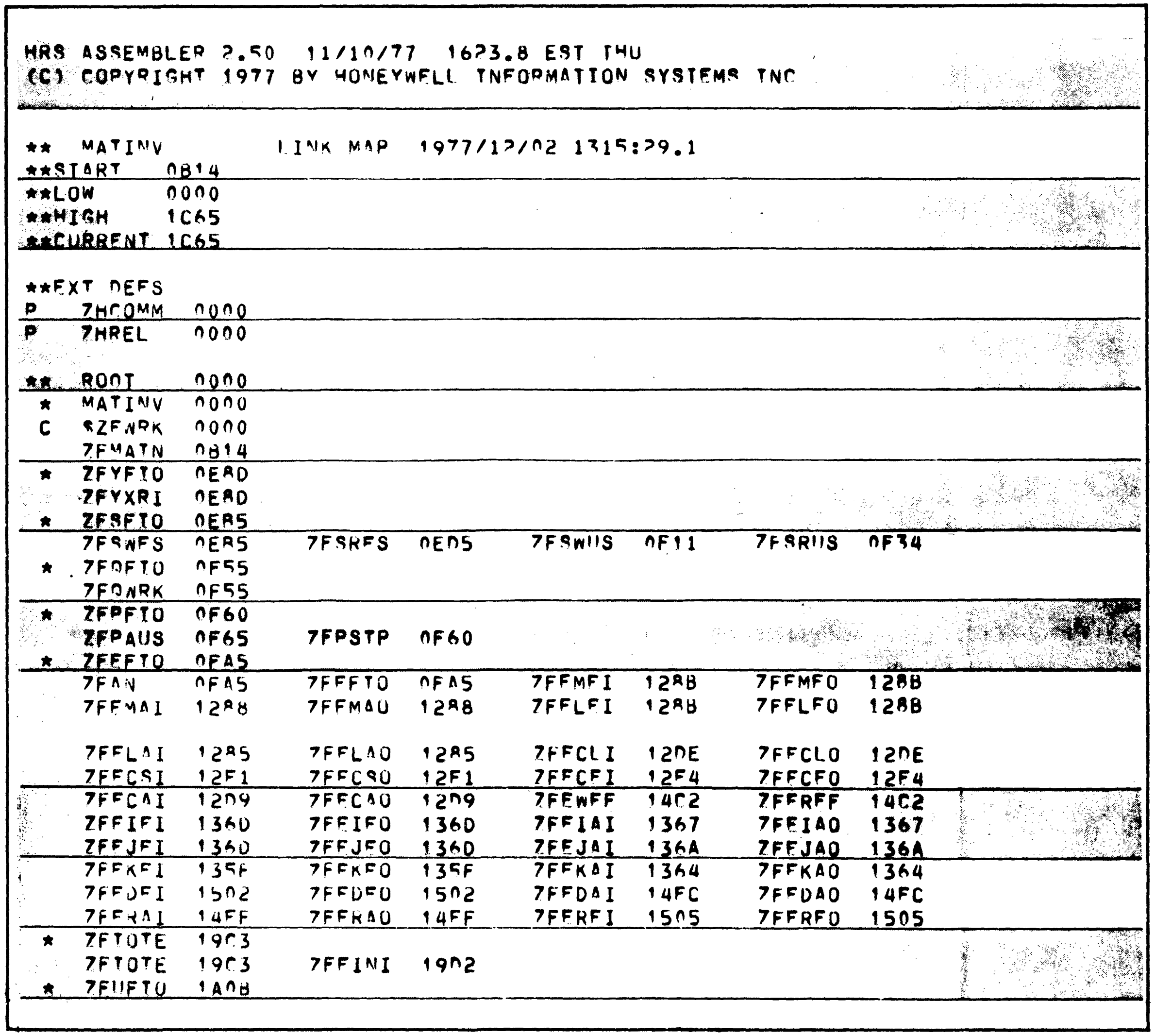
```
  140 DO 200 L=1,N
      T=A(IROW,L)
      A(IROW,L) =A(ICOL,L)
  200 A(ICOL,L) =T
      IF(M)260,260,210
  210 DO 250 L=1,M
      T=B(IROW,L)
      B(IROW,L) =B(ICOL,L)
  250 B(ICOL,L) =T
  260 INDEX(I,1) =IROW
      INDEX (I,2) =ICOL
      PIVOT(I) =A(ICOL,ICOL)
C     DIVIDE PIVOT ROW BY PIVOT ELEMENT
      A(ICOL,ICOL) =1.0
      DO 350 L=1,N
  350  A(ICOL,L) =A(ICOL,L)/PIVOT(I)
      IF(M)380,380,360
  360 DO 370 L=1,M
  370 B(ICOL,L) =B(ICOL,L)/PIVOT(I)
C     REDUCE NON-PIVOT ROWS
  380 DO 550LT=1,N
      IF(LT-ICOL) 400,550,400
  400 T = A(LT,ICOL)
      A(LT,ICOL) =0.0
      DO 450 L=1,N
  450 A(LT,L)=A(LT,L)-A(ICOL,L)*T
      IF(M)550,550,460
  460 DO 500 L=1,M
  500 B(LT,L) = B(LT,L)-B(ICOL,L)*T
  550 CONTINUE
C     INTERCHANGE COLUMNS
  600 DO 710I=1,N
      L= N-I+1
      IF(INDEX(L,1)-INDEX(L,2))630,710,630
  630 IROW = INDEX(L,1)
      ICOL = INDEX(L,2)
      DO 705 K=1,N
      T =A(K,IROW)
      A(K,IROW) =A(K,ICOL)
      A(K,ICOL) = T
  705 CONTINUE
```

Figure 8-2 (cont). Source and Linker Output Listing (MATINV)



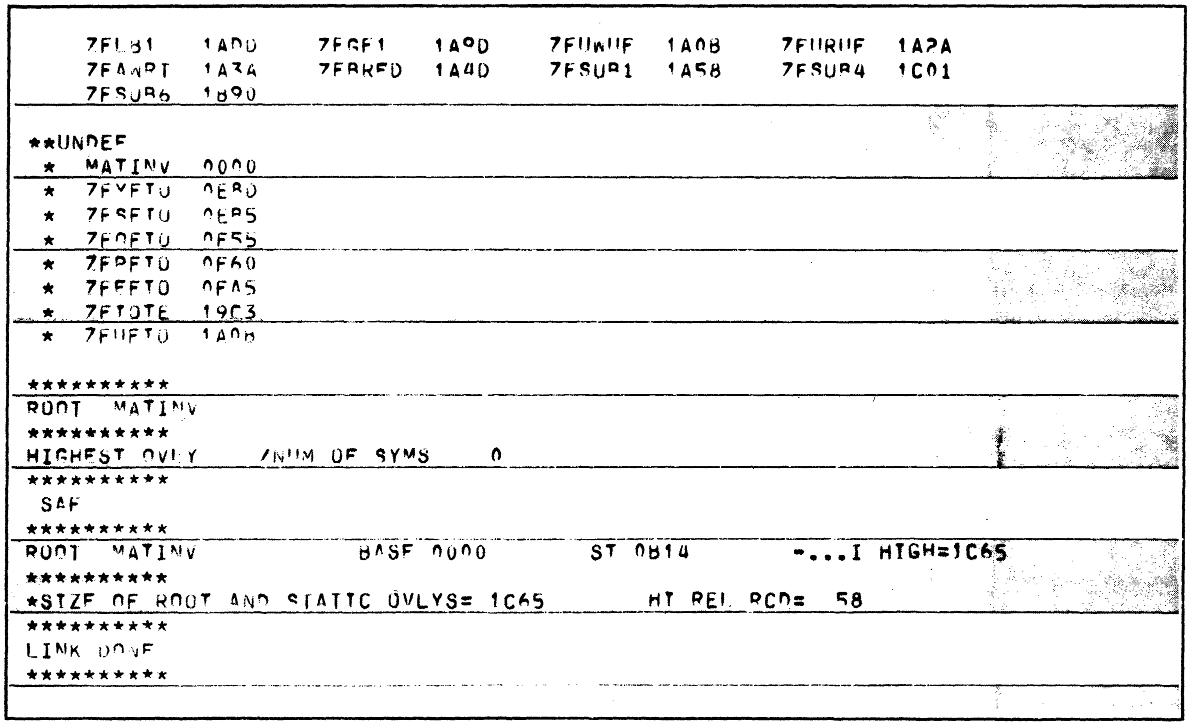
```
   80    710 CONTINUE
   81          WRITE(3,16)
   82       16 FORMAT(//5X,22HINVERSE  OF  MATRIX  A/)
   83          DO 15 I = 1,N
   84       15 WRITE(3,2)(A(I,J),J=1,M)
   85          WRITE(3,444)
   86   444    FORMAT(1H1)
   87   740    STOP 1
   88          END
     0  DIAGNOSTICS


MATINV  77120200
  FORTRAN M4FD  11/22/1119     1977/12/02 1312:36.9 SAF

ZFYFTO  77111000
HRS ASSEMBLER 2.50  11/10/77  1035.9 EST THU
(C) COPYRIGHT 1977 BY HONEYWELL INFORMATION SYSTEMS INC

ZFSFTO  77111000
HRS ASSEMBLER 2.50  11/10/77  1030.1 EST THU
(C) COPYRIGHT 1977 BY HONEYWELL INFORMATION SYSTEMS INC
ZFSK01

ZFQFTO  77050100
HRS ASSEMBLER 2.50  10/03/77  0708.3 EDT MON
(C) COPYRIGHT 1977 BY HONEYWELL INFORMATION SYSTEMS INC

ZFPFTO  77111000
HRS ASSEMBLER 2.50  11/10/77  1054.0 EST THU
(C) COPYRIGHT 1977 BY HONEYWELL INFORMATION SYSTEMS INC

ZFEFTO  77112100
HRS ASSEMBLER 2.50  11/21/77  1539.8 EST MON
(C) COPYRIGHT 1977 BY HONEYWELL INFORMATION SYSTEMS INC
ZFEK01

ZFIOTE  77072900
HRS ASSEMBLER 2.50  10/13/77  0937.8 EDT THU
(C) COPYRIGHT 1977 BY HONEYWELL INFORMATION SYSTEMS INC

ZFUFTO  77111000
```

Figure 8-2 (cont). Source and Linker Output Listing (MATINV)



```
HRS ASSEMBLER 2.50  11/10/77  1623.8 EST THU
(C) COPYRIGHT 1977 BY HONEYWELL INFORMATION SYSTEMS INC

**  MATINV          LINK MAP  1977/12/02 1315:29.1
**START   0814
**LOW     0000
**HIGH    1C65
**CURRENT 1C65

**EXT DEFS
P   ZHCOMM   0000
P   ZHREL    0000

**  ROOT     0000
 *  MATINV   0000
 C  $ZFWRK   0000
    ZFMAIN   0814
 *  ZFYFIO   0E8D
    ZFYXRI   0E8D
 *  ZFSFIO   0E85
    ZFSWFS   0E85    ZFSRFS   0ED5    ZFSWUS   0F11    ZFSRUS   0F34
 *  ZFQFIO   0F55
    ZFQWRK   0F55
 *  ZFPFIO   0F60
    ZFPAUS   0F65    ZFPSTP   0F60
 *  ZFEFIO   0FA5
    ZFAN     0FA5    ZFEFIO   0FA5    ZFFMFI   128B    ZFFMFO   128B
    ZFFMAI   1288    ZFFMAO   1288    ZFFLFI   128B    ZFFLFO   128B

    ZFFLAI   1285    ZFFLAO   1285    ZFFCLI   12DE    ZFFCLO   12DE
    ZFFCSI   12F1    ZFFCSO   12F1    ZFFCFI   12F4    ZFFCFO   12F4
    ZFFCAI   12D9    ZFFCAO   12D9    ZFEWFF   14C2    ZFFRFF   14C2
    ZFFIFI   136D    ZFFIFO   136D    ZFFIAI   1367    ZFFIAO   1367
    ZFFJFI   136D    ZFFJFO   136D    ZFEJAI   136A    ZFFJAO   136A
    ZFFKFI   135F    ZFFKFO   135F    ZFFKAI   1364    ZFFKAO   1364
    ZFFDFI   1502    ZFFDFO   1502    ZFFDAI   14FC    ZFFDAO   14FC
    ZFFRAI   14FF    ZFFRAO   14FF    ZFFRFI   1505    ZFFRFO   1505
 *  ZFTOTE   19C3
    ZFTOTE   19C3    ZFFINI   19D2
 *  ZFUFIO   1A0B
```

Figure 8-2 (cont). Source and Linker Output Listing (MATINV)

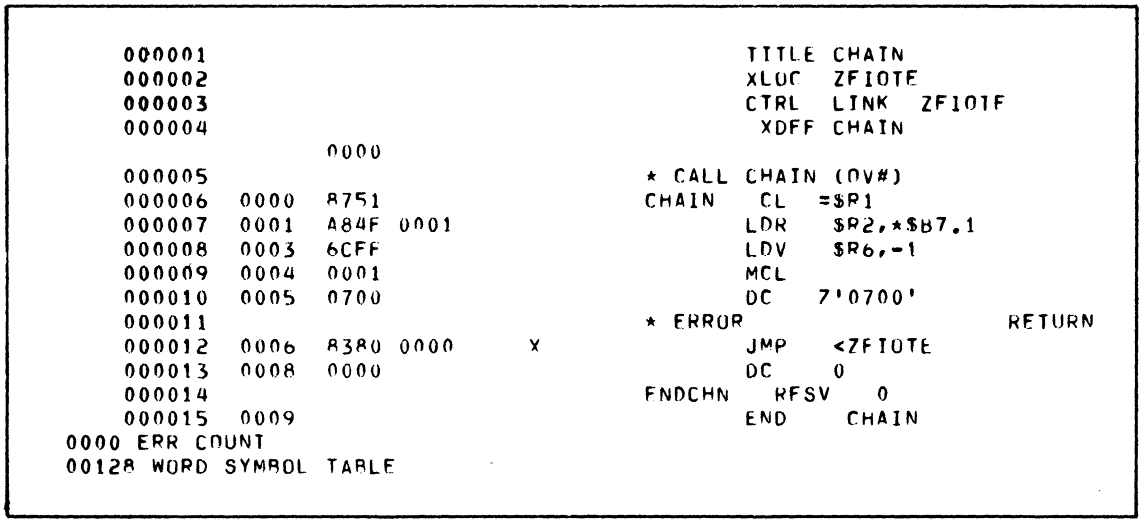


```
    ZFLB1     1ADD      ZFGF1     1A9D      ZFUWUF    1A0B      ZFURUF    1A2A
    ZFAWRT    1A3A      ZFBRED    1A4D      ZFSUB1    1A58      ZFSUB4    1C01
    ZFSUB6    1B90

**UNDEF
 *  MATINV    0000
 *  ZFYFTU    0E8D
 *  ZFSFTU    0EB5
 *  ZFOFTU    0F55
 *  ZFPFTO    0F60
 *  ZFFFTO    0FA5
 *  ZFIOTE    19C3
 *  ZFUFTO    1A0B

**********
ROOT   MATINV
**********
HIGHEST OVLY      /NUM OF SYMS      0
**********
 SAF
**********
ROOT   MATINV              BASE 0000         ST 0B14         -...I HIGH=1C65
**********
*SIZE OF ROOT AND STATIC OVLYS= 1C65         HT REL RCD=  58
**********
LINK DONE
**********
```

Figure 8-2 (cont.). Source and Linker Output Listing (MATINV)

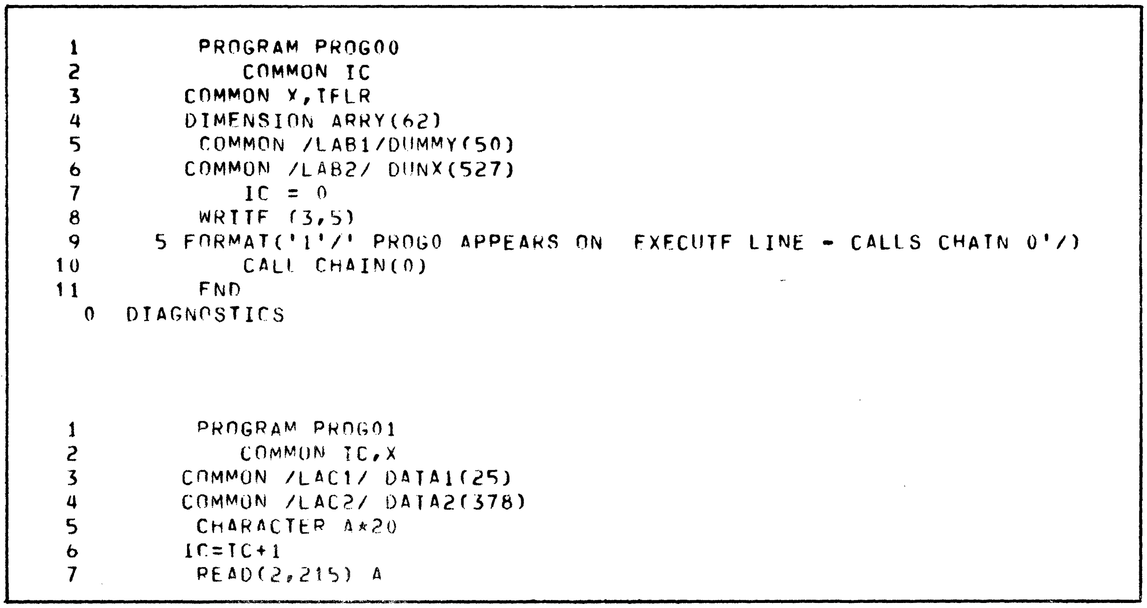
3. The first statement executed in a loaded chain is always the first executable statement of the first main program in the chain. (A chain cannot begin with a subroutine.)
4. All data passed between chains must be in unlabeled or labeled COMMON blocks that have been defined within the root. Because of Linker constraints, the first occurrence of a COMMON block defines its size, therefore care must be exercised when using COMMON blocks of different sizes.
5. Within programs in a chain, either labeled or unlabeled COMMON may be freely used as a means of data communication.
6. Data statements (for data not in COMMON) within a program of a chain cause the data to be initialized each time the chain is loaded.
7. Files are common to all chains since the run-time work area is defined within the root.

Figure 8-3 shows an assembly-language program, CHAIN, whose function is to load the chains specified in the CALL statements of the FORTRAN programs shown in Figure 8-4. These latter programs call each other at various times and print messages indicating their loading and execution.



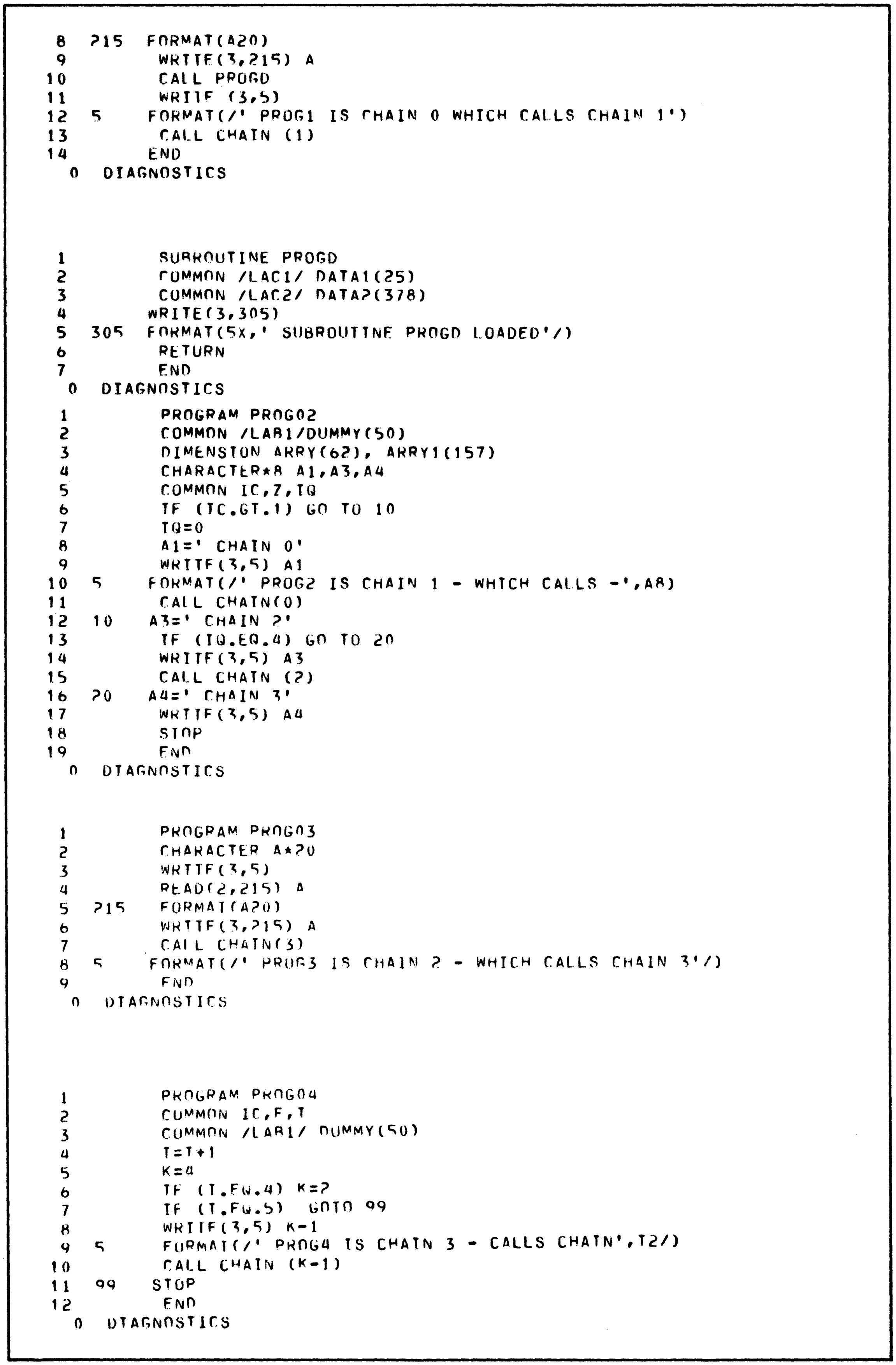
```
                                                       TITLE CHAIN
                                                       XLOC  ZFIOTE
                                                       CTRL  LINK  ZFIOTE
                                                        XDEF CHAIN
                 0000
                                                 * CALL CHAIN (OV#)
           0000  8751                            CHAIN   CL   =$R1
           0001  A84F 0001                             LDR   $R2,*$B7.1
           0003  6CFF                                  LDV   $R6,-1
           0004  0001                                  MCL
           0005  0700                                  DC    7'0700'
                                                 * ERROR                      RETURN
           0006  8380 0000      X                      JMP   <ZFIOTE
           0008  0000                                  DC    0
                                                 ENDCHN   RESV   0
           0009                                        END    CHAIN
0000 ERR COUNT
00128 WORD SYMBOL TABLE
```

Figure 8-3. Assembly Listing of Program CHAIN



```
       PROGRAM PROG00
          COMMON IC
      COMMON Y,IFLR
      DIMENSION ARRY(62)
       COMMON /LAB1/DUMMY(50)
      COMMON /LAB2/ DUNX(527)
          IC = 0
       WRITE (3,5)
    5 FORMAT('1'/' PROG0 APPEARS ON  EXECUTE LINE - CALLS CHAIN 0'/)
          CALL CHAIN(0)
       END
 0  DIAGNOSTICS



       PROGRAM PROG01
          COMMON IC,X
      COMMON /LAC1/ DATA1(25)
      COMMON /LAC2/ DATA2(378)
       CHARACTER A*20
      IC=IC+1
       READ(2,215) A
```

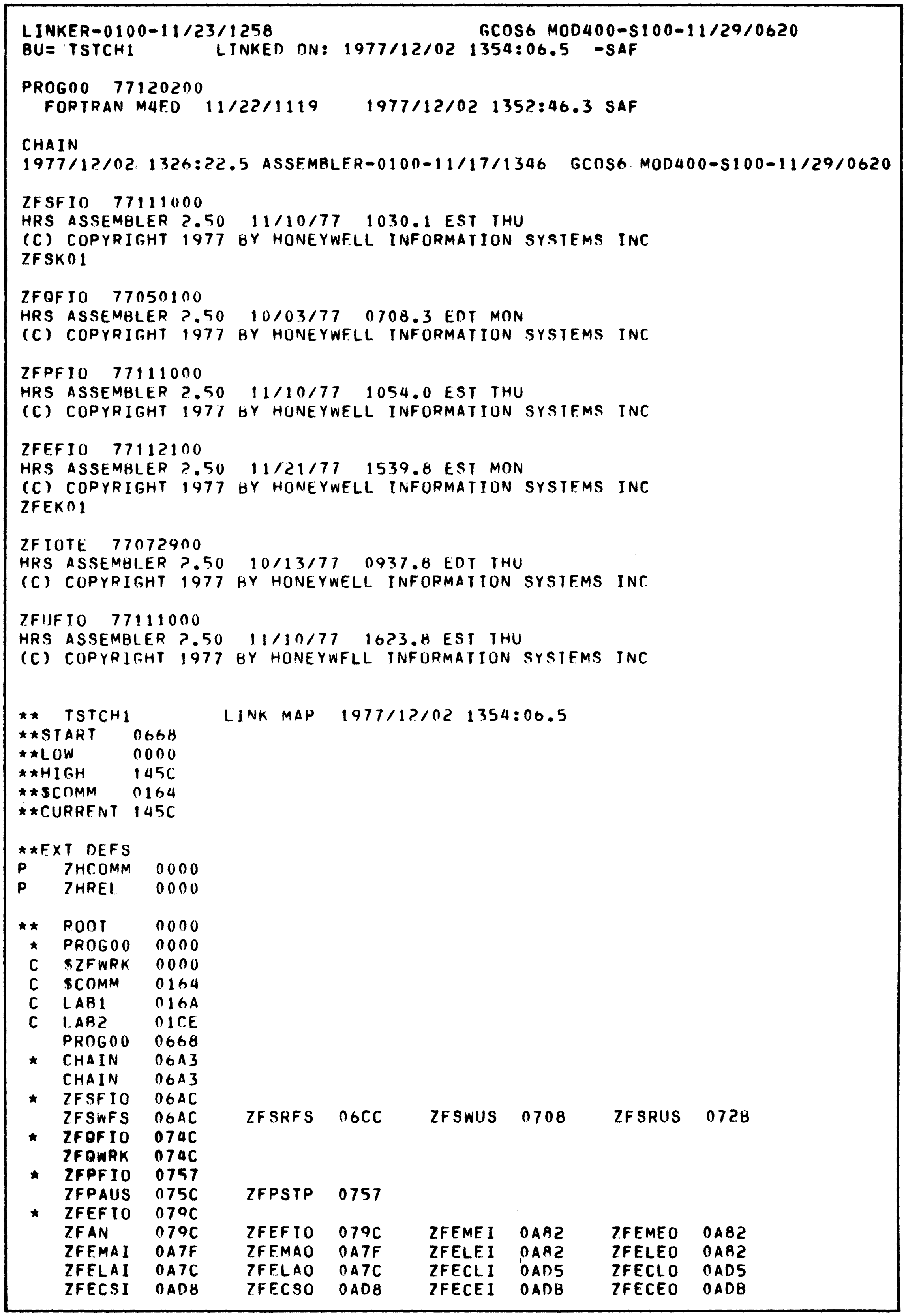
Figure 8-4. FORTRAN Programs Calling the CHAIN Function



```
  8   215   FORMAT(A20)
  9          WRITE(3,215) A
 10          CALL PROGD
 11          WRITE (3,5)
 12   5     FORMAT(/' PROG1 IS CHAIN 0 WHICH CALLS CHAIN 1')
 13          CALL CHAIN (1)
 14         END
   0   DIAGNOSTICS



  1          SUBROUTINE PROGD
  2          COMMON /LAC1/ DATA1(25)
  3          COMMON /LAC2/ DATA2(378)
  4         WRITE(3,305)
  5   305   FORMAT(5X,' SUBROUTINE PROGD LOADED'/)
  6          RETURN
  7          END
  0   DIAGNOSTICS
  1          PROGRAM PROG02
  2          COMMON /LAB1/DUMMY(50)
  3          DIMENSION ARRY(62), ARRY1(157)
  4          CHARACTER*8 A1,A3,A4
  5          COMMON IC,Z,IQ
  6          IF (IC.GT.1) GO TO 10
  7          IQ=0
  8          A1=' CHAIN 0'
  9          WRITE(3,5) A1
 10   5     FORMAT(/' PROG2 IS CHAIN 1 - WHICH CALLS -',A8)
 11          CALL CHAIN(0)
 12   10    A3=' CHAIN 2'
 13          IF (IQ.EQ.4) GO TO 20
 14          WRITE(3,5) A3
 15          CALL CHAIN (2)
 16   20    A4=' CHAIN 3'
 17          WRITE(3,5) A4
 18          STOP
 19          END
   0   DIAGNOSTICS


  1          PROGRAM PROG03
  2          CHARACTER A*20
  3          WRITE(3,5)
  4          READ(2,215) A
  5   215    FORMAT(A20)
  6          WRITE(3,215) A
  7          CALL CHAIN(3)
  8   5     FORMAT(/' PROG3 IS CHAIN 2 - WHICH CALLS CHAIN 3'/)
  9          END
   0   DIAGNOSTICS



  1          PROGRAM PROG04
  2          COMMON IC,F,I
  3          COMMON /LAB1/ DUMMY(50)
  4          I=I+1
  5          K=4
  6          IF (I.EQ.4) K=2
  7          IF (I.EQ.5)  GOTO 99
  8          WRITE(3,5) K-1
  9   5      FORMAT(/' PROG4 IS CHAIN 3 - CALLS CHAIN',I2/)
 10          CALL CHAIN (K-1)
 11   99    STOP
 12          END
   0   DIAGNOSTICS
```

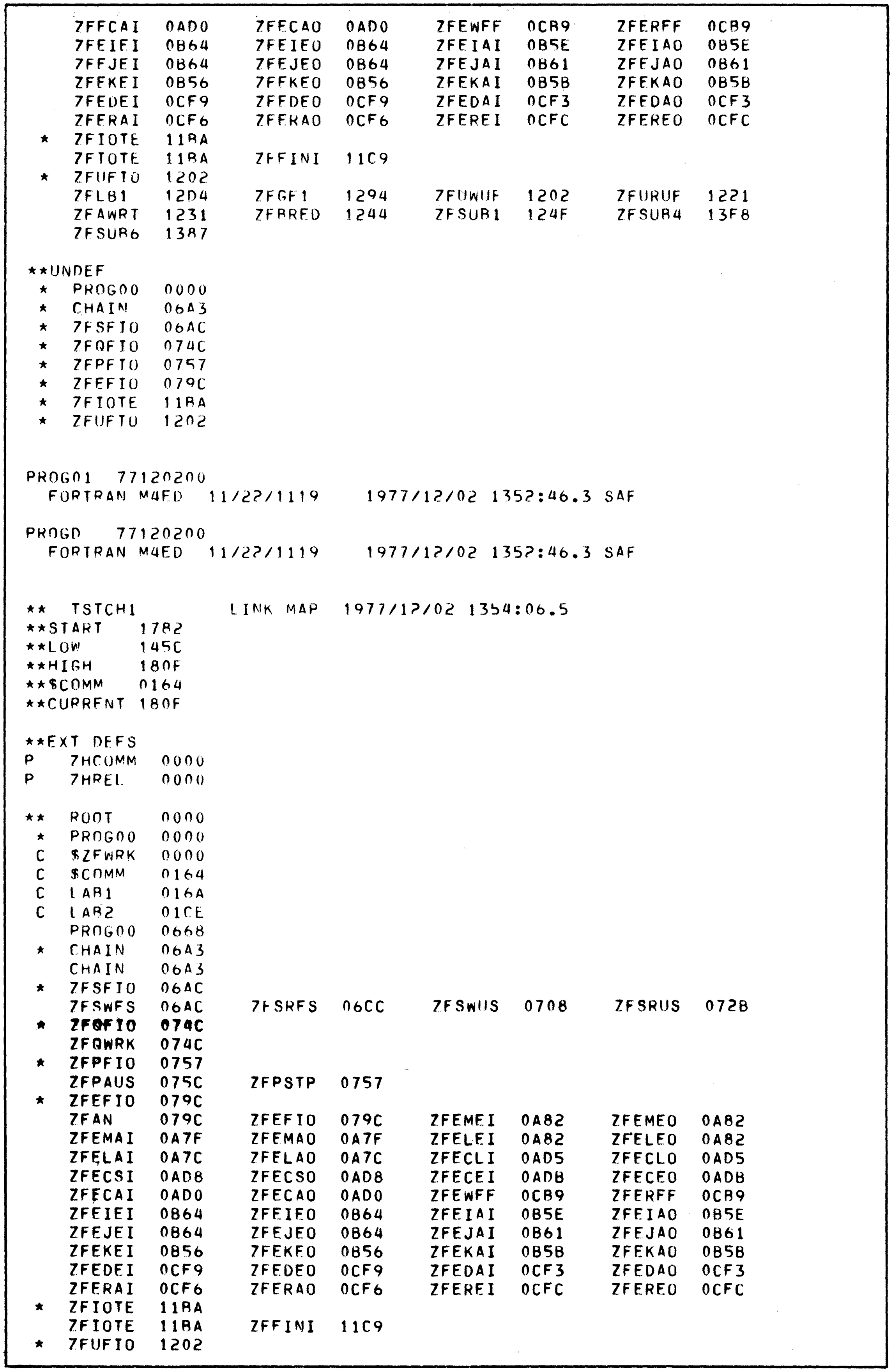
Figure 8-4 (cont). FORTRAN Programs Calling the CHAIN Function

Figure 8-5 is the output listing resulting from the linking of the programs constituting the chain.



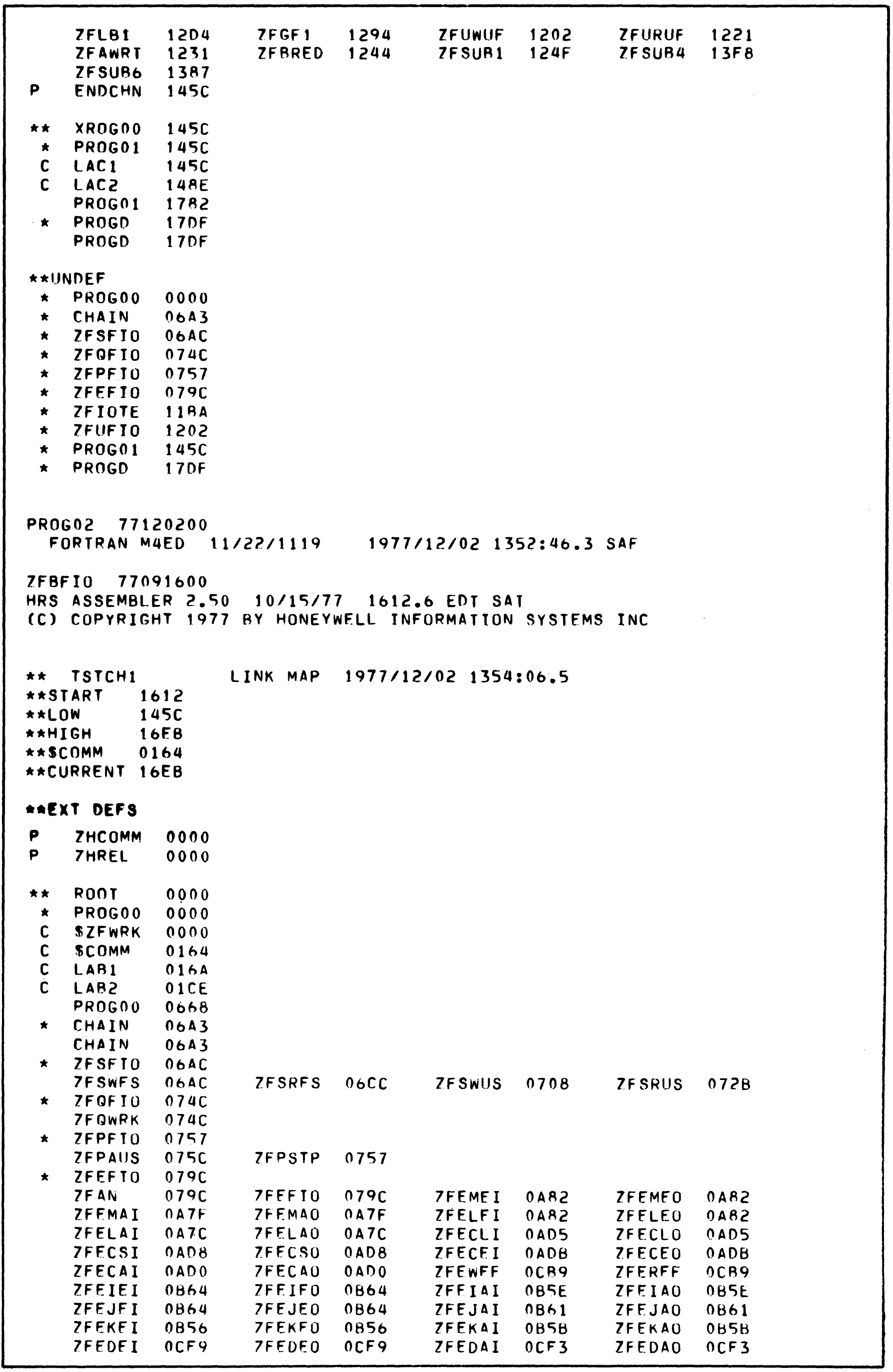
```
LINKER-0100-11/23/1258                     GCOS6 MOD400-S100-11/29/0620
BU= TSTCH1        LINKED ON: 1977/12/02 1354:06.5  -SAF

PROG00  77120200
  FORTRAN M4ED  11/22/1119    1977/12/02 1352:46.3 SAF

CHAIN
1977/12/02 1326:22.5 ASSEMBLER-0100-11/17/1346  GCOS6 MOD400-S100-11/29/0620

ZFSFIO  77111000
HRS ASSEMBLER 2.50  11/10/77  1030.1 EST THU
(C) COPYRIGHT 1977 BY HONEYWELL INFORMATION SYSTEMS INC
ZFSK01

ZFQFIO  77050100
HRS ASSEMBLER 2.50  10/03/77  0708.3 EDT MON
(C) COPYRIGHT 1977 BY HONEYWELL INFORMATION SYSTEMS INC

ZFPFIO  77111000
HRS ASSEMBLER 2.50  11/10/77  1054.0 EST THU
(C) COPYRIGHT 1977 BY HONEYWELL INFORMATION SYSTEMS INC

ZFEFIO  77112100
HRS ASSEMBLER 2.50  11/21/77  1539.8 EST MON
(C) COPYRIGHT 1977 BY HONEYWELL INFORMATION SYSTEMS INC
ZFEK01

ZFIOTE  77072900
HRS ASSEMBLER 2.50  10/13/77  0937.8 EDT THU
(C) COPYRIGHT 1977 BY HONEYWELL INFORMATION SYSTEMS INC

ZFUFIO  77111000
HRS ASSEMBLER 2.50  11/10/77  1623.8 EST THU
(C) COPYRIGHT 1977 BY HONEYWELL INFORMATION SYSTEMS INC


**  TSTCH1        LINK MAP  1977/12/02 1354:06.5
**START   0668
**LOW     0000
**HIGH    145C
**$COMM   0164
**CURRENT 145C

**EXT DEFS
P    ZHCOMM   0000
P    ZHREL    0000

**   ROOT     0000
 *   PROG00   0000
 C   $ZFWRK   0000
 C   $COMM    0164
 C   LAB1     016A
 C   LAB2     01CE
     PROG00   0668
 *   CHAIN    06A3
     CHAIN    06A3
 *   ZFSFIO   06AC
     ZFSWFS   06AC     ZFSRFS   06CC     ZFSWUS   0708     ZFSRUS   0728
 *   ZFQFIO   074C
     ZFQWRK   074C
 *   ZFPFIO   0757
     ZFPAUS   075C     ZFPSTP   0757
 *   ZFEFIO   079C
     ZFAN     079C     ZFEFIO   079C     ZFEMEI   0A82     ZFEMEO   0A82
     ZFEMAI   0A7F     ZFEMAO   0A7F     ZFELEI   0A82     ZFELEO   0A82
     ZFELAI   0A7C     ZFELAO   0A7C     ZFECLI   0AD5     ZFECLO   0AD5
     ZFECSI   0AD8     ZFECSO   0AD8     ZFECEI   0ADB     ZFECEO   0ADB
```

Figure 8-5. Linker Output for Chained Programs



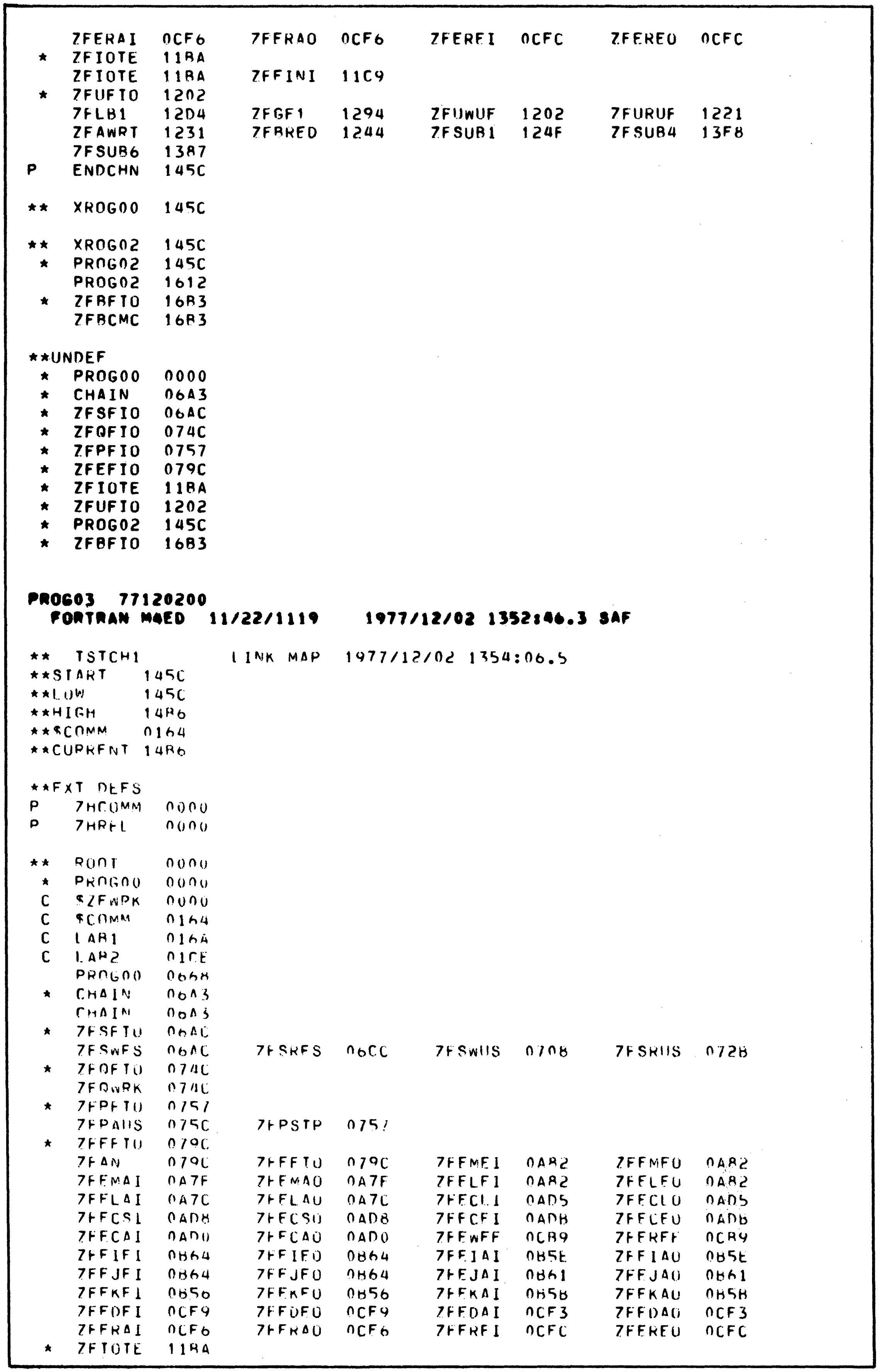
```
    ZFFCAI   0AD0      ZFECAO   0AD0      ZFEWFF   0CB9      ZFERFF   0CB9
    ZFEIEI   0B64      ZFEIEO   0B64      ZFEIAI   0B5E      ZFEIAO   0B5E
    ZFFJEI   0B64      ZFEJEO   0B64      ZFEJAI   0B61      ZFFJAO   0B61
    ZFEKEI   0B56      ZFFKEO   0B56      ZFEKAI   0B5B      ZFEKAO   0B5B
    ZFEDEI   0CF9      ZFFDEO   0CF9      ZFEDAI   0CF3      ZFEDAO   0CF3
    ZFERAI   0CF6      ZFERAO   0CF6      ZFEREI   0CFC      ZFEREO   0CFC
 *  ZFIOTE   11BA
    ZFIOTE   11BA      ZFFINI   11C9
 *  ZFUFIO   1202
    ZFLB1    12D4      ZFGF1    1294      ZFUWUF   1202      ZFURUF   1221
    ZFAWRT   1231      ZFBRED   1244      ZFSUB1   124F      ZFSUB4   13F8
    ZFSUB6   1387

**UNDEF
 *  PROG00   0000
 *  CHAIN    06A3
 *  ZFSFIO   06AC
 *  ZFQFIO   074C
 *  ZFPFIO   0757
 *  ZFEFIO   079C
 *  ZFIOTE   11BA
 *  ZFUFIO   1202


PROG01  77120200
  FORTRAN M4ED  11/22/1119     1977/12/02 1352:46.3 SAF

PROGD   77120200
  FORTRAN M4ED  11/22/1119     1977/12/02 1352:46.3 SAF


**  TSTCH1          LINK MAP  1977/12/02 1354:06.5
**START    1782
**LOW      145C
**HIGH     180F
**$COMM    0164
**CURRENT 180F

**EXT DEFS
P   ZHCOMM   0000
P   ZHREL    0000

**  ROOT     0000
 *  PROG00   0000
 C  $ZFWRK   0000
 C  $COMM    0164
 C  LAB1     016A
 C  LAB2     01CE
    PROG00   0668
 *  CHAIN    06A3
    CHAIN    06A3
 *  ZFSFIO   06AC
    ZFSWFS   06AC      ZFSRFS   06CC      ZFSWUS   0708      ZFSRUS   072B
 *  ZFQFIO   074C
    ZFQWRK   074C
 *  ZFPFIO   0757
    ZFPAUS   075C      ZFPSTP   0757
 *  ZFEFIO   079C
    ZFAN     079C      ZFEFIO   079C      ZFEMEI   0A82      ZFEMEO   0A82
    ZFEMAI   0A7F      ZFEMAO   0A7F      ZFELEI   0A82      ZFELEO   0A82
    ZFELAI   0A7C      ZFELAO   0A7C      ZFECLI   0AD5      ZFECLO   0AD5
    ZFECSI   0AD8      ZFECSO   0AD8      ZFECEI   0ADB      ZFECEO   0ADB
    ZFECAI   0AD0      ZFECAO   0AD0      ZFEWFF   0CB9      ZFERFF   0CB9
    ZFEIEI   0B64      ZFEIEO   0B64      ZFEIAI   0B5E      ZFEIAO   0B5E
    ZFEJEI   0B64      ZFEJEO   0B64      ZFEJAI   0B61      ZFEJAO   0B61
    ZFEKEI   0B56      ZFEKEO   0B56      ZFEKAI   0B5B      ZFEKAO   0B5B
    ZFEDEI   0CF9      ZFEDEO   0CF9      ZFEDAI   0CF3      ZFEDAO   0CF3
    ZFERAI   0CF6      ZFERAO   0CF6      ZFEREI   0CFC      ZFEREO   0CFC
 *  ZFIOTE   11BA
    ZFIOTE   11BA      ZFFINI   11C9
 *  ZFUFIO   1202
```

Figure 8-5 (cont). Linker Output for Chained Programs



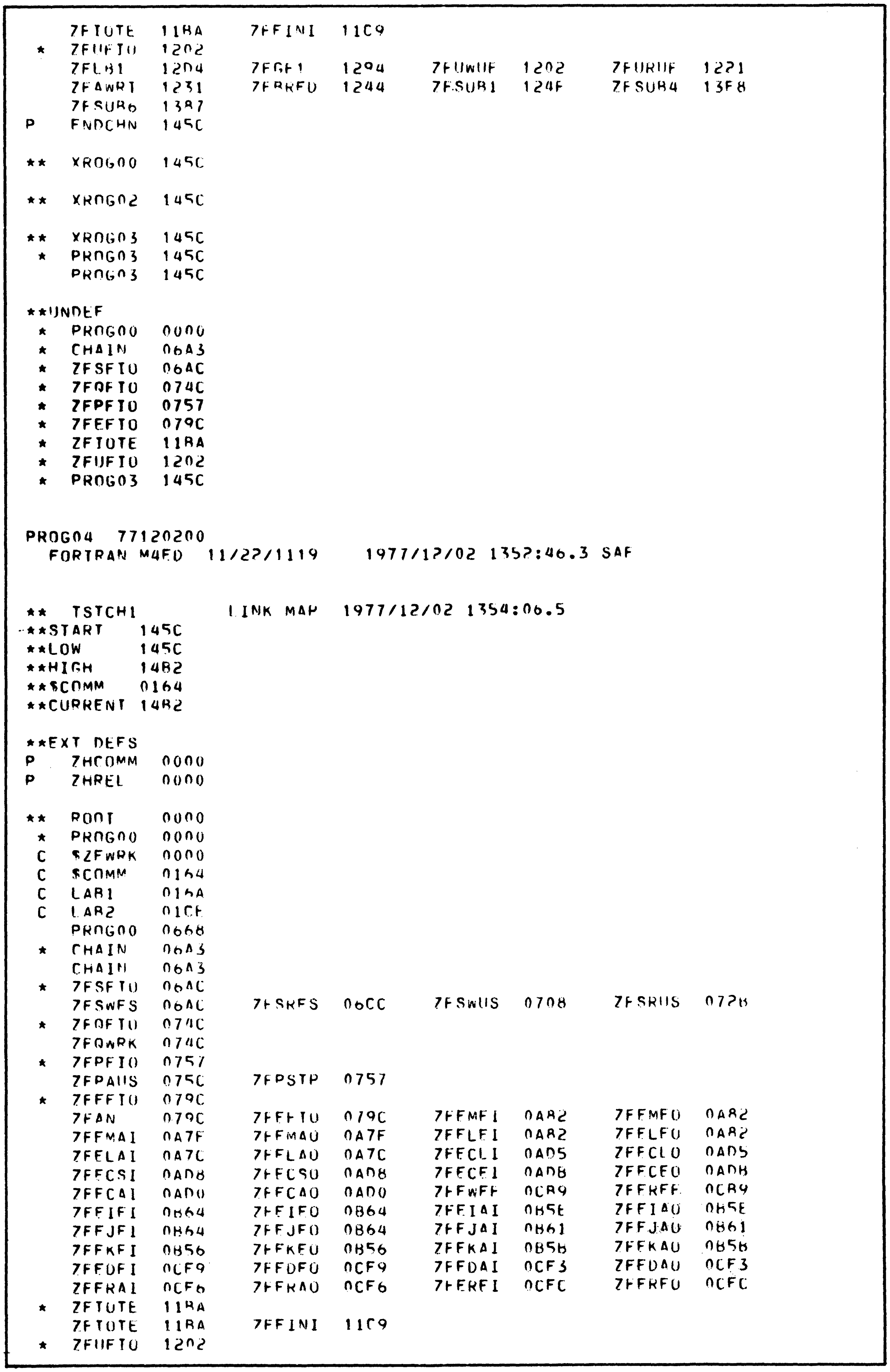
```
    ZFLB1    12D4      ZFGF1    1294      ZFUWUF   1202      ZFURUF   1221
    ZFAWRT   1231      ZFBRED   1244      ZFSUB1   124F      ZFSUB4   13F8
    ZFSUB6   1387
P   ENDCHN   145C

**  XROG00   145C
 *  PROG01   145C
 C  LAC1     145C
 C  LAC2     148E
    PROG01   1782
 *  PROGD    17DF
    PROGD    17DF

**UNDEF
 *  PROG00   0000
 *  CHAIN    06A3
 *  ZFSFIO   06AC
 *  ZFQFIO   074C
 *  ZFPFIO   0757
 *  ZFEFIO   079C
 *  ZFIOTE   118A
 *  ZFUFIO   1202
 *  PROG01   145C
 *  PROGD    17DF


PROG02  77120200
  FORTRAN M4ED  11/22/1119    1977/12/02 1352:46.3 SAF

ZFBFIO  77091600
HRS ASSEMBLER 2.50  10/15/77  1612.6 EDT SAT
(C) COPYRIGHT 1977 BY HONEYWELL INFORMATION SYSTEMS INC


**  TSTCH1         LINK MAP  1977/12/02 1354:06.5
**START   1612
**LOW     145C
**HIGH    16EB
**$COMM   0164
**CURRENT 16EB

**EXT DEFS
P   ZHCOMM   0000
P   ZHREL    0000

**  ROOT     0000
 *  PROG00   0000
 C  $ZFWRK   0000
 C  $COMM    0164
 C  LAB1     016A
 C  LAB2     01CE
    PROG00   0668
 *  CHAIN    06A3
    CHAIN    06A3
 *  ZFSFIO   06AC
    ZFSWFS   06AC      ZFSRFS   06CC      ZFSWUS   0708      ZFSRUS   072B
 *  ZFQFIO   074C
    ZFQWRK   074C
 *  ZFPFIO   0757
    ZFPAUS   075C      ZFPSTP   0757
 *  ZFEFIO   079C
    ZFAN     079C      ZFEFIO   079C      ZFEMEI   0A82      ZFEMEO   0A82
    ZFEMAI   0A7F      ZFEMAO   0A7F      ZFELFI   0A82      ZFELEO   0A82
    ZFELAI   0A7C      ZFELAO   0A7C      ZFECLI   0AD5      ZFECLO   0AD5
    ZFECSI   0AD8      ZFECSO   0AD8      ZFECEI   0ADB      ZFECEO   0ADB
    ZFECAI   0AD0      ZFECAO   0AD0      ZFEWFF   0CB9      ZFERFF   0CB9
    ZFEIEI   0B64      ZFEIFO   0B64      ZFEIAI   0B5E      ZFEIAO   0B5E
    ZFEJFI   0B64      ZFEJEO   0B64      ZFEJAI   0B61      ZFEJAO   0B61
    ZFEKFI   0B56      ZFEKFO   0B56      ZFEKAI   0B5B      ZFEKAO   0B5B
    ZFEDFI   0CF9      ZFEDEO   0CF9      ZFEDAI   0CF3      ZFEDAO   0CF3
```

Figure 8-5 (cont). Linker Output for Chained Programs



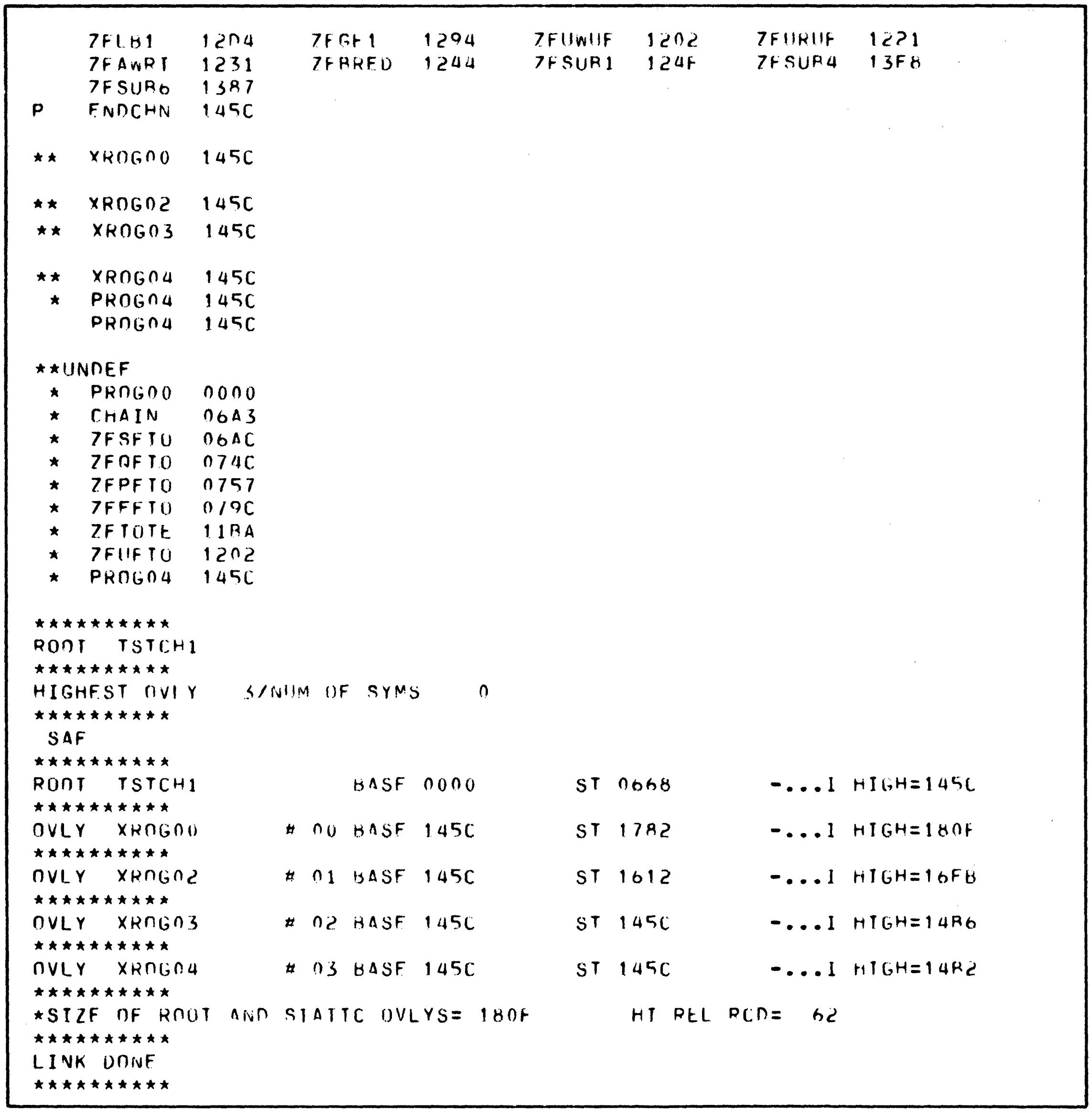
```
    ZFERAI   0CF6      ZFERAO   0CF6      ZFEREI   0CFC      ZFEREO   0CFC
 *  ZFIOTE   11BA
    ZFIOTE   11BA      ZFFINI   11C9
 *  ZFUFIO   1202
    ZFLB1    12D4      ZFGF1    1294      ZFUWUF   1202      ZFURUF   1221
    ZFAWRT   1231      ZFBRED   1244      ZFSUB1   124F      ZFSUB4   13F8
    ZFSUB6   1387
P   ENDCHN   145C

**  XROG00   145C

**  XROG02   145C
 *  PROG02   145C
    PROG02   1612
 *  ZFBFIO   16B3
    ZFBCMC   16B3

**UNDEF
 *  PROG00   0000
 *  CHAIN    06A3
 *  ZFSFIO   06AC
 *  ZFQFIO   074C
 *  ZFPFIO   0757
 *  ZFEFIO   079C
 *  ZFIOTE   11BA
 *  ZFUFIO   1202
 *  PROG02   145C
 *  ZFBFIO   16B3


PROG03   77120200
  FORTRAN M4ED  11/22/1119     1977/12/02 1352:46.3 SAF

**  TSTCH1          LINK MAP  1977/12/02 1354:06.5
**START   145C
**LOW     145C
**HIGH    14B6
**$COMM   0164
**CURRENT 14B6

**EXT DEFS
P   ZHCOMM   0000
P   ZHREL    0000

**  ROOT     0000
 *  PROG00   0000
 C  $ZFWPK   0000
 C  $COMM    0164
 C  LAB1     016A
 C  LAB2     01CE
    PROG00   0668
 *  CHAIN    06A3
    CHAIN    06A3
 *  ZFSFIO   06AC
    ZFSWFS   06AC      ZFSRFS   06CC      ZFSWUS   0708      ZFSRUS   072B
 *  ZFQFIO   074C
    ZFQWRK   074C
 *  ZFPFIO   0757
    ZFPAUS   075C      ZFPSTP   0757
 *  ZFEFIO   079C
    ZFAN     079C      ZFEFIO   079C      ZFEMEI   0A82      ZFEMEO   0A82
    ZFEMAI   0A7F      ZFEMAO   0A7F      ZFELFI   0A82      ZFELFO   0A82
    ZFELAI   0A7C      ZFELAO   0A7C      ZFECLI   0AD5      ZFECLO   0AD5
    ZFECSI   0AD8      ZFECSO   0AD8      ZFECFI   0ADB      ZFECFO   0ADB
    ZFECAI   0AD0      ZFECAO   0AD0      ZFEWFF   0CB9      ZFERFF   0CB9
    ZFEIFI   0B64      ZFEIFO   0B64      ZFEIAI   0B5E      ZFEIAO   0B5E
    ZFEJFI   0B64      ZFEJFO   0B64      ZFEJAI   0B61      ZFEJAO   0B61
    ZFEKFI   0B56      ZFEKFO   0B56      ZFEKAI   0B5B      ZFEKAO   0B5B
    ZFEDFI   0CF9      ZFEDFO   0CF9      ZFEDAI   0CF3      ZFEDAO   0CF3
    ZFERAI   0CF6      ZFERAO   0CF6      ZFEREI   0CFC      ZFEREO   0CFC
 *  ZFIOTE   11BA
```

Figure 8-5 (cont). Linker Output for Chained Programs



```
     ZFIOTE   11BA       ZFFINI   11C9
  *  ZFUFIO   1202
     ZFLB1    12D4       ZFGF1    1294       ZFUWUF   1202       ZFURUF   1221
     ZFAWRT   1231       ZFBRFU   1244       ZFSUB1   124F       ZFSUB4   13F8
     ZFSUB6   1387
P    ENDCHN   145C

**   XROG00   145C

**   XROG02   145C

**   XROG03   145C
  *  PROG03   145C
     PROG03   145C

**UNDEF
  *  PROG00   0000
  *  CHAIN    06A3
  *  ZFSFIO   06AC
  *  ZFQFIO   074C
  *  ZFPFIO   0757
  *  ZFEFIO   079C
  *  ZFIOTE   11BA
  *  ZFUFIO   1202
  *  PROG03   145C


PROG04  77120200
  FORTRAN M4FD  11/22/1119     1977/12/02 1352:46.3 SAF


**   TSTCH1           LINK MAP  1977/12/02 1354:06.5
**START    145C
**LOW      145C
**HIGH     14B2
**$COMM    0164
**CURRENT  14B2

**EXT DEFS
P    ZHCOMM   0000
P    ZHREL    0000

**   ROOT     0000
  *  PROG00   0000
 C   $ZFWRK   0000
 C   $COMM    0164
 C   LAB1     016A
 C   LAB2     01CE
     PROG00   0668
  *  CHAIN    06A3
     CHAIN    06A3
  *  ZFSFIO   06AC
     ZFSWFS   06AC       ZFSRFS   06CC       ZFSWUS   0708       ZFSRUS   0728
  *  ZFQFIO   074C
     ZFQWRK   074C
  *  ZFPFIO   0757
     ZFPAUS   075C       ZFPSTP   0757
  *  ZFFFIO   079C
     ZFAN     079C       ZFFFIO   079C       ZFFMFI   0A82       ZFFMFO   0A82
     ZFFMAI   0A7F       ZFFMAO   0A7F       ZFFLFI   0A82       ZFFLFO   0A82
     ZFELAI   0A7C       ZFFLAO   0A7C       ZFECLI   0AD5       ZFFCLO   0AD5
     ZFFCSI   0AD8       ZFFCSO   0AD8       ZFECFI   0ADB       ZFFCFO   0ADB
     ZFFCAI   0AD0       ZFFCAO   0AD0       ZFFWFF   0CB9       ZFFRFF   0CB9
     ZFFIFI   0B64       ZFFIFO   0B64       ZFFIAI   0B5E       ZFFIAO   0B5E
     ZFFJFI   0B64       ZFFJFO   0B64       ZFFJAI   0B61       ZFFJAO   0B61
     ZFFKFI   0B56       ZFFKFO   0B56       ZFFKAI   0B5B       ZFFKAO   0B5B
     ZFFDFI   0CF9       ZFFDFO   0CF9       ZFFDAI   0CF3       ZFFDAO   0CF3
     ZFFRAI   0CF6       ZFFRAO   0CF6       ZFERFI   0CFC       ZFFRFO   0CFC
  *  ZFIOTE   11BA
     ZFIOTE   11BA       ZFFINI   11C9
  *  ZFUFIO   1202
```

Figure 8-5 (cont). Linker Output for Chained Programs



```
     ZFLB1    12D4     ZFGF1    1294     ZFUWUF   1202     ZFURUF   1221
     ZFAWRT   1231     ZFBRED   1244     ZFSUB1   124F     ZFSUB4   13F8
     ZFSUB6   1387
P    ENDCHN   145C

**   XROG00   145C

**   XROG02   145C
**   XROG03   145C

**   XROG04   145C
 *   PROG04   145C
     PROG04   145C

**UNDEF
 *   PROG00   0000
 *   CHAIN    06A3
 *   ZFSFTO   06AC
 *   ZFQFTO   074C
 *   ZFPFTO   0757
 *   ZFFFTO   079C
 *   ZFTOTE   11BA
 *   ZFUFTO   1202
 *   PROG04   145C

**********
ROOT   TSTCH1
**********
HIGHEST OVLY    3/NUM OF SYMS     0
**********
 SAF
**********
ROOT   TSTCH1              BASE 0000        ST 0668        -...I HIGH=145C
**********
OVLY   XROG00        # 00 BASE 145C        ST 1782        -...I HIGH=180F
**********
OVLY   XROG02        # 01 BASE 145C        ST 1612        -...I HIGH=16FB
**********
OVLY   XROG03        # 02 BASE 145C        ST 145C        -...I HIGH=14B6
**********
OVLY   XROG04        # 03 BASE 145C        ST 145C        -...I HIGH=14B2
**********
*SIZE OF ROOT AND STATIC OVLYS= 180F        HI REL RCD=  62
**********
LINK DONE
**********
```

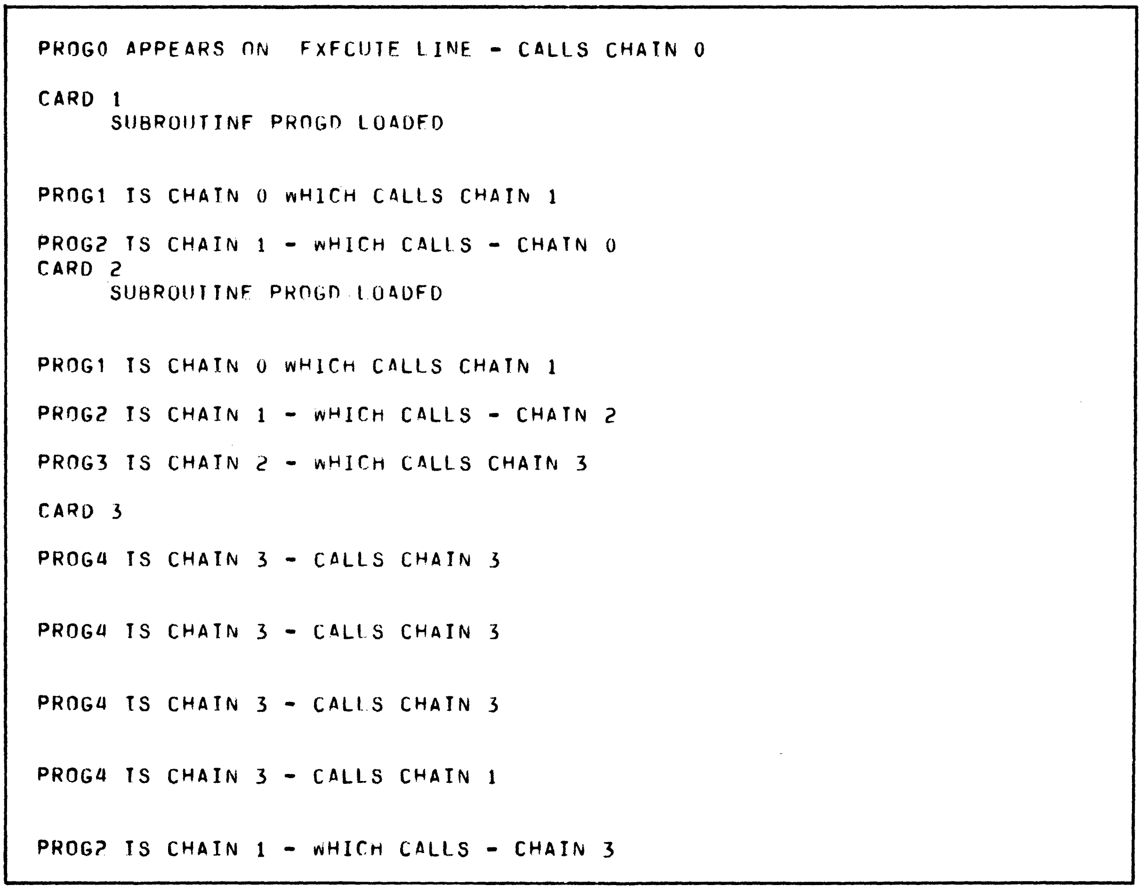
Figure 8-5 (cont). Linker Output for Chained Programs

Figure 8-6 illustrates the linker directives required to create the bound unit TSTCH1, comprising the programs listed in Figures 8-3 and 8-4.

Figure 8-7 shows the output resulting from the execution of the chained programs.

```
LIB ^VL5901>LDD>OBJECT>FRIOR
LINK  PROGOO
MAP
OVLY XROGOO          DEFINES OVERLAY 0 (CHAIN 0)
LDEF ENDCHN,$        DEFINES BASE FOR OVERLAYS
PROT ENDCHN
BASE ENDCHN
LINK PROG01
MAP
BASE ENDCHN
OVLY XROG01          DEFINES OVERLAY 1 (CHAIN 1)
LINK PROG02
MAP
OVLY XROG02          DEFINES OVERLAY 2 (CHAIN 2)
BASE ENDCHN
LINK PROG03
MAP
OVLY XROG03          DEFINES OVERLAY 3 (CHAIN 3)
BASE ENDCHN
LINK PROG04
MAP
QT
```

Figure 8-6. Linker Directives for Chained Programs

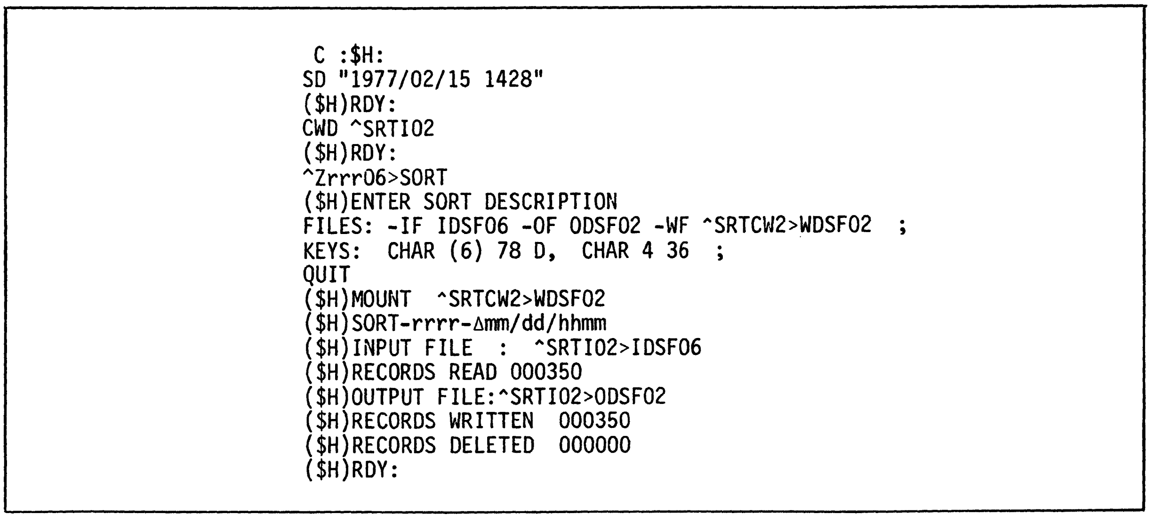


```
PROGO APPEARS ON  FXFCUTE LINE - CALLS CHAIN 0

CARD 1
     SUBROUTINF PROGD LOADFD


PROG1 IS CHAIN 0 WHICH CALLS CHAIN 1

PROG2 IS CHAIN 1 - WHICH CALLS - CHAIN 0
CARD 2
     SUBROUTINF PROGD LOADFD


PROG1 IS CHAIN 0 WHICH CALLS CHAIN 1

PROG2 IS CHAIN 1 - WHICH CALLS - CHAIN 2

PROG3 IS CHAIN 2 - WHICH CALLS CHAIN 3

CARD 3

PROG4 IS CHAIN 3 - CALLS CHAIN 3


PROG4 IS CHAIN 3 - CALLS CHAIN 3


PROG4 IS CHAIN 3 - CALLS CHAIN 3


PROG4 IS CHAIN 3 - CALLS CHAIN 1


PROG2 IS CHAIN 1 - WHICH CALLS - CHAIN 3
```

Figure 8-7. Execution Output from Chained Programs

Section 9
Using the Sort

Figure 9-1 contains a sample session at the operator terminal to sort a file using the Sort utility. Sort descriptors are entered through the operator terminal. Refer to the *Sort/Merge* manual for details on the use of the Sort component.



```
 C :$H:
SD "1977/02/15 1428"
($H)RDY:
CWD ^SRTI02
($H)RDY:
^Zrrr06>SORT
($H)ENTER SORT DESCRIPTION
FILES: -IF IDSF06 -OF ODSF02 -WF ^SRTCW2>WDSF02  ;
KEYS:  CHAR (6) 78 D,  CHAR 4 36  ;
QUIT
($H)MOUNT  ^SRTCW2>WDSF02
($H)SORT-rrrr-Δmm/dd/hhmm
($H)INPUT FILE  :  ^SRTI02>IDSF06
($H)RECORDS READ 000350
($H)OUTPUT FILE:^SRTI02>ODSF02
($H)RECORDS WRITTEN  000350
($H)RECORDS DELETED  000000
($H)RDY:
```

Figure 9-1. Sample Sort Terminal Session

The Sort utility is on volume Zrrr06 and the application files are on SRTI02. The Sort is invoked by entering the pathname, ^ Zrrr06>SORT. The Sort description statements are then entered. In this example, the work file is not mounted, and a message to mount ^ SRTCW2>WDSF02 is issued.

INDEX

INDEX

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE: SERIES 60 (LEVEL 6) GCOS 6 MOD 400 PROGRAMMER'S GUIDE

ORDER NO. CB22, REV. 0

DATED JANUARY 1978

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by appropriate technical personnel and action will be taken as required. If you require a written reply, check here and furnish complete mailing address below. □

FROM: NAME ______________________ DATE ____________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

____

CUT ALONG LINE

PLEASE FOLD AND TAPE –
NOTE: U. S. Postal Service will not deliver stapled forms

FOLD ALONG LINE

FIRST CLASS
PERMIT NO. 39531
WALTHAM, MA
02154

Business Reply Mail
Postage Stamp Not Necessary if Mailed in the United States

Postage Will Be Paid By:

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTENTION: PUBLICATIONS, MS 486

FOLD ALONG LINE

Honeywell

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

20682, 5578, Printed in U.S.A.