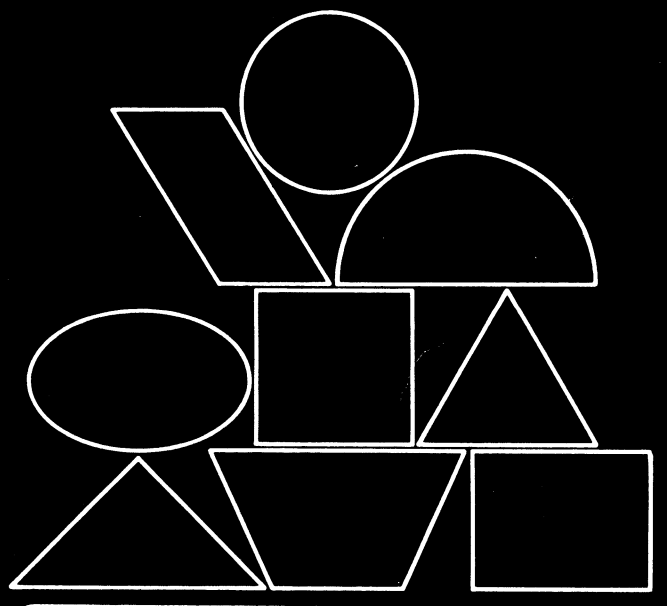
Data General

MP/OS OPERATING SYSTEM and UTILITIES

Self-Study Course



educational services

053-000005-00

NOTICE

Data General Corporation (DGC) has prepared this manual for use by DGC personnel and/or customers as a guide to the proper installation, operation, and maintenance of DGC equipment and software. The drawings and specifications contained herein are the property of DGC and shall neither be reproduced in whole or in part without DGC prior written approval nor be implied to grant any license to make, use, or sell equipment manufactured in accordance herewith.

DGC reserves the right to make changes without notice in the specifications and materials contained herein and shall not be responsible for any damages (including consequential) caused by reliance on the materials presented, including but not limited to typographical or arithmetic errors, company policy and pricing information. The information contained herein on DGC software is summary in nature. More detailed information on DGC software is available in current released publications.

MP/OS OPERATING SYSTEM
AND
UTILITIES SELF-STUDY
COURSE

TABLE OF CONTENTS

(Continued)

MODULE THREE QUIZ

MODULE FOUR . . . PROGRAM DEVELOPMENT

(Continued)

INTRODUCTION

PREREQUISITES

The student should have an understanding of basic Data Processing concepts and terminology equivalent to an introductory level course in programming or to Data General's S100.

ABSTRACT

The MP/OS Operating System and Utilities Self-Study Course is designed for the applications programmer, systems programmer, and system manager responsible for MP/OS program development and MP/OS system management. The Course instructs in the following major topics:

- MP/OS concepts and terminology
- CLI Command Line Interpreter for interacting with MP/OS at your console.
- SPEED Text Editor for entering and modifying source language files.
- Program Development in Assembly, MP/Fortran IV, and MP/Pascal
- Symbolic Debugger for on-line, executable program files.
- System Maintenance: Disk initialization, FIXUP, and the MOVE Utility.

COURSE FORMAT

This is a self-study, audio tape course. This allows you to:

1. learn what, when, and where you want to;
2. select lessons that fit your needs (you do not have to read it all);
3. learn the subject with or without a functioning Data General computer;
4. frequently determine your understanding of the subject matter.

OBJECTIVES

Upon completion of this course, the student will be able to:

1. Use the Command Line Interpreter (CLI) to:
   a) manage files
   b) control programs
   c) manage the user environment
   d) manage the system environment
   e) manage disk devices

2. Use the SPEED Text Editor to:
   a) enter source text
   b) modify source text
   c) store source text on disc devices.

3. Given a source program written in assembly, MP/Fortran IV, or MP/Pascal prepare the program for execution under MP/OS. Preparation includes editing, translation, and binding using the appropriate text editor, assembler, compiler, and binder command lines and options.

4. Use the Symbolic Debugger to display, modify, and test executable program files.

5. Use the Disk Initializer to software format disk media.

6. Use FIXUP to software repair disk media.

7. Use the MOVE utility to transfer files and back-up files.

8. Power-up, load, and on-line system devices and boot MP/OS into operation.

NOTE: Detailed objectives are provided with the introduction of each Module and Module segment.

DURATION

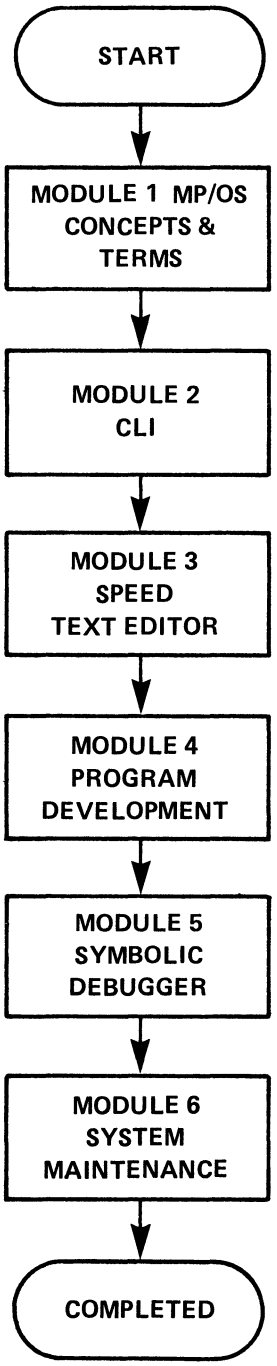
This is a self-paced course. Therefore, student completion times will vary. Diligent attention to the audio-tapes, quizzes, and lab exercises should require about four working days or thirty hours.

REQUIRED MATERIAL

This package is complete in itself. You will notice that it does *not* require access to a functioning MP/OS system. However, success is enhanced with unfettered access to a live system.

COURSE MAP

Below is a flowchart of the course modules. It is recommended that you do not alter the sequence of modules. *If you believe that you have mastery of the subject matter before completion of the module, then jump to the Module Quiz and test your abilities.*



NOTE: choose the segments appropriate for your language.

NOTE: Highly recommended for the assembly language programmer.

MODULE ONE
MP/OS CONCEPTS

MODULE ONE

Abstract

This module surveys the MP/OS operating system's concepts and facilities. Topics include how MP/OS manages main and peripheral storage, files, data, and I/O.

Objectives

Upon completion of this module, you will be able to:

1. Define, in your own words, the following MP/OS terms:
   - A) Program Stack
   - B) Levels
   - C) CLI
   - D) Pure Area
   - E) Impure Area
   - F) Swap
   - G) Chain
   - H) File
   - I) Directory
   - J) Pathname
   - K) Filename
   - L) Device Directory
   - M) Root Directory
   - N) Working Directory
   - O) I/O Channel
   - P) Multitasking
   - Q) Overlays
   - R) Searchlist
2. Given a list of filenames, identify the legal and illegal MP/OS filenames.
3. Given a filename with an MP/OS extension, state the file type.
4. Given a file's characteristics, state its appropriate attributes.
5. Given a sequence of I/O transfers, state which is fastest, slowest, most efficient, and least efficient.

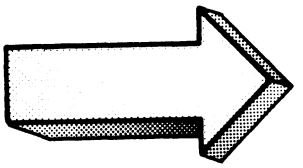
6. Describe the two main MP/OS memory areas. State the area of code that is common to all operating environments.

7. Differentiate between root directory, device directory, and working directory by stating a definition of each in your own words.

8. Given a directory structure, identify the valid and invalid pathnames.

Directions

1. Read the System Overview on the next page of the Student Guide.

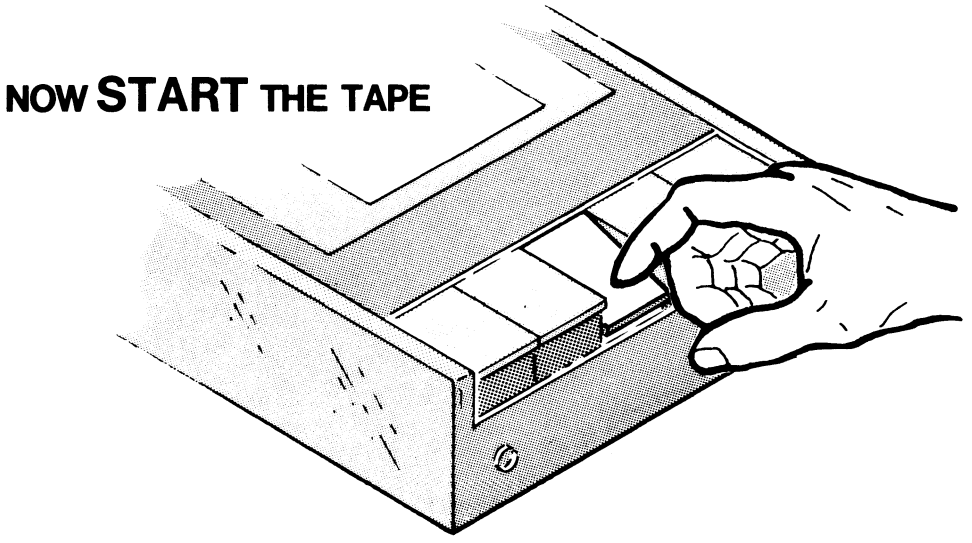


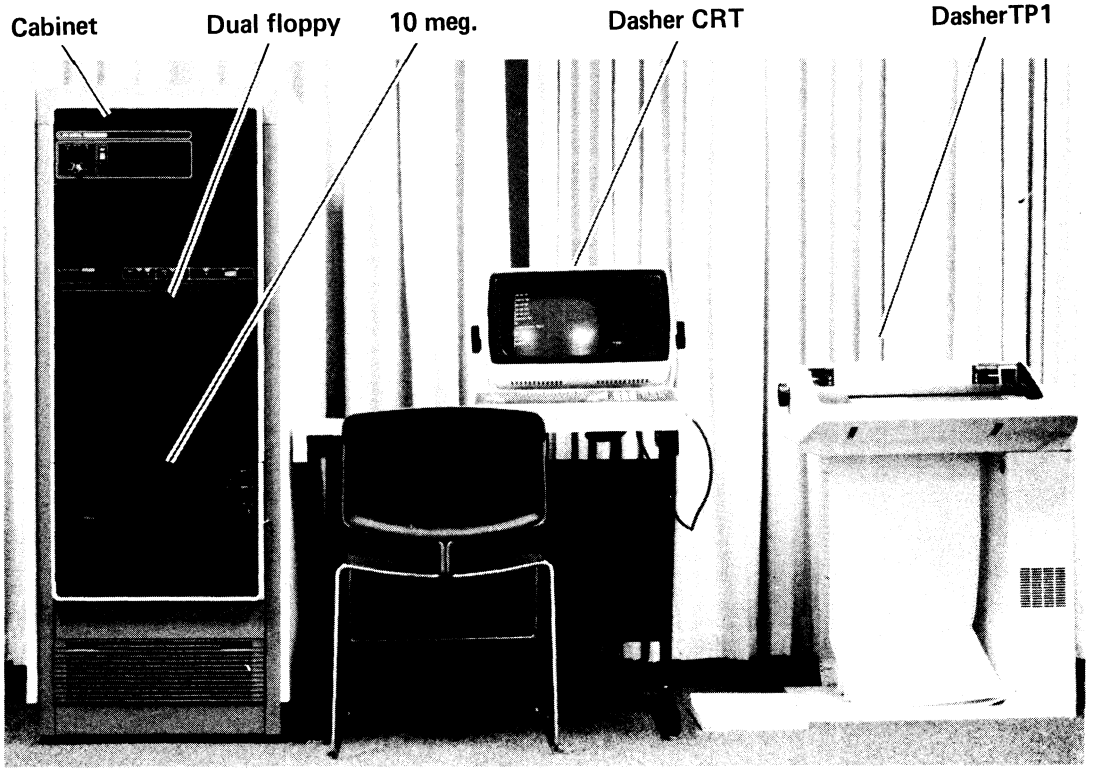
SYSTEM OVERVIEW

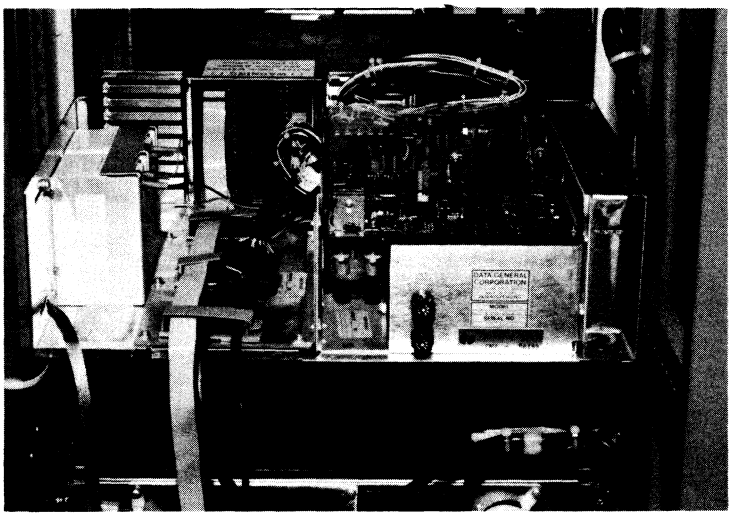
The MP/OS system is a general purpose operating system for the microNOVA line of computers. It provides features usually associated with larger computer systems, such as multitasking, memory management, and device independent I/O.

The MP/OS system can be used either for general purpose systems oriented toward program development, or for smaller stand-alone applications such as real-time process control. You can generate an MP/OS system containing a desired subset of the full system's power and tailor it to any configuration of memory boards and peripherals. You can put all software in read-only memory (ROM) to eliminate the need for mass storage or you can take advantage of the powerful MP/OS file management system for storing large amounts of data on disks.

Now turn to Figure 1-1 in the Student Guide and listen to the tape for Module One.



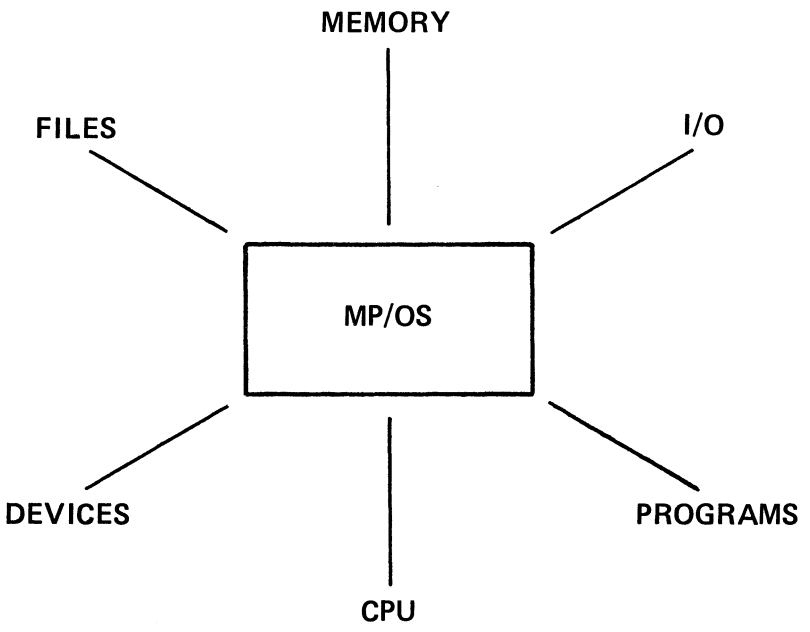




Board/Chassis
Process Control
Environment

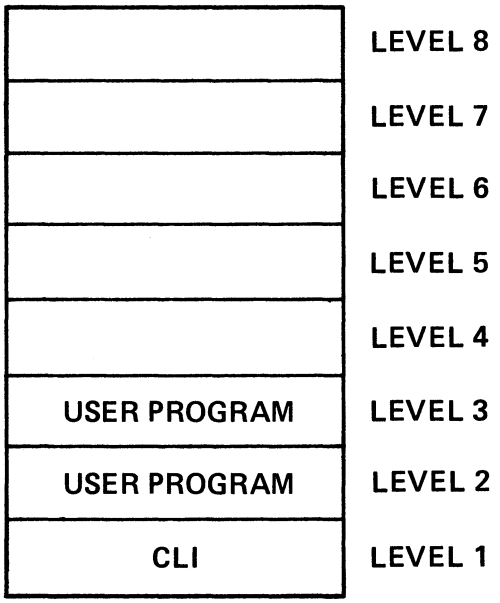
THE FLEXIBILITY OF MP-SYSTEMS

Figure 1-1



MP/OS AS MANAGER OF RESOURCES

Figure 1-2



MP/OS VIEW OF SYSTEM AND USER PROGRAMS

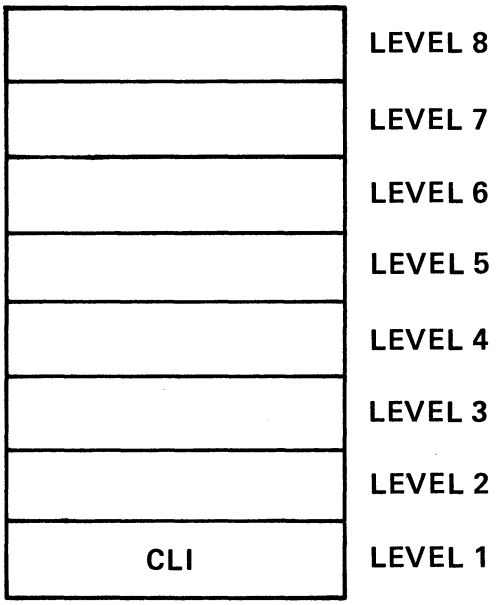
Figure 1-3

SYSTEM CALLS

1. CONTROL TRANSFER
2. INTERPROGRAM COMMUNICATION
3. BREAKFILE CREATION
4. MANIPULATE MEMORY ALLOCATIONS
5. OVERLAY MANAGEMENT
6. CONTROL PROGRAM TIMING
7. RESTART SYSTEM
8. INPUT/OUTPUT
9. CONTROL PERIPHERALS
10. MANAGE FILES

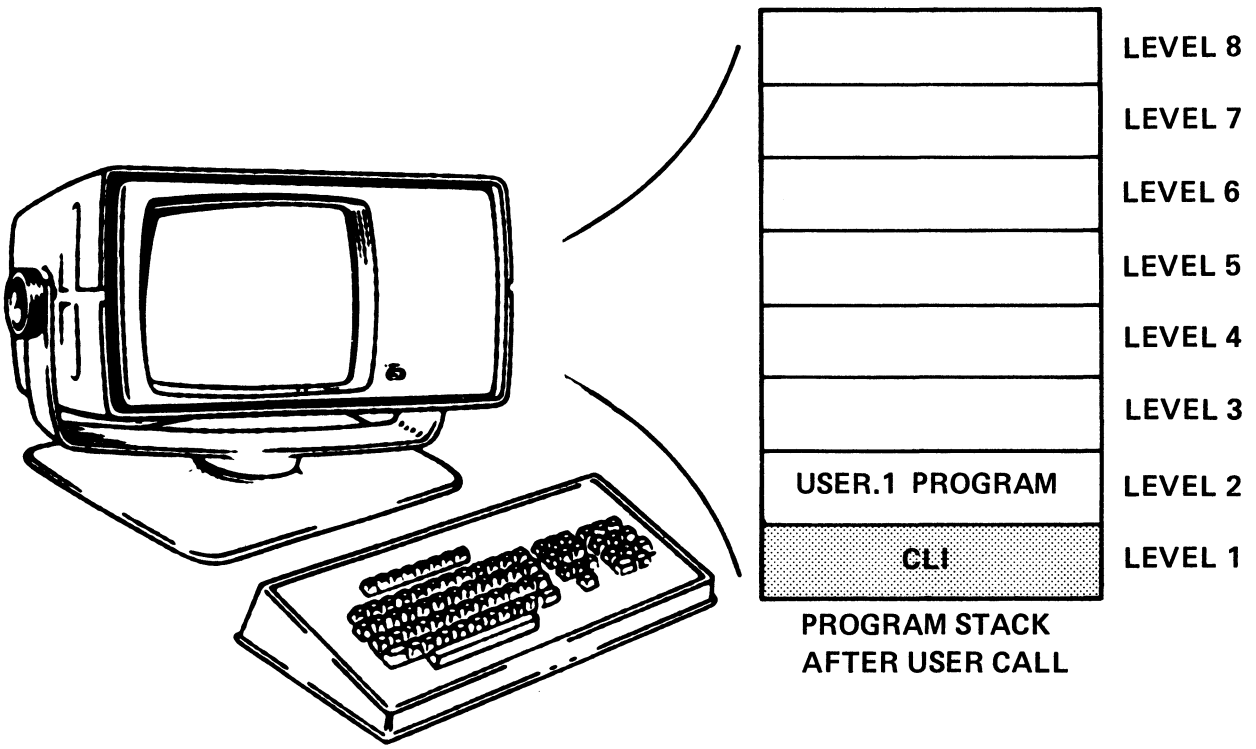
OUTLING OF SOME SYSTEM CALL FACILITIES

Figure 1-4



PROGRAM STACK AT INITIALIZATION

Figure 1-5

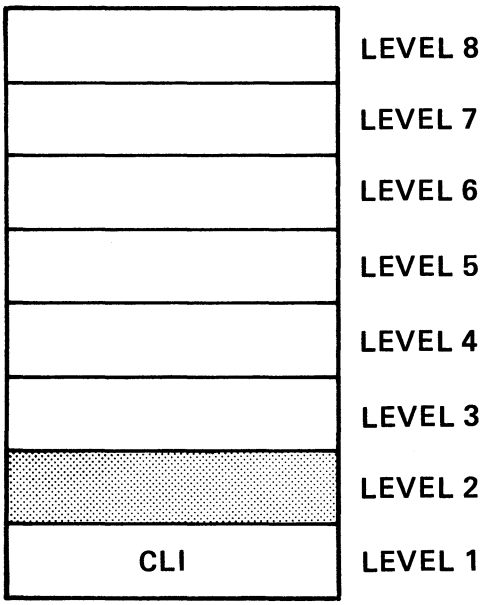


PARENT PROGRAM . . . CALLER

SON PROGRAM (OR DESCENDANT) . . . THE CALLED PROGRAM

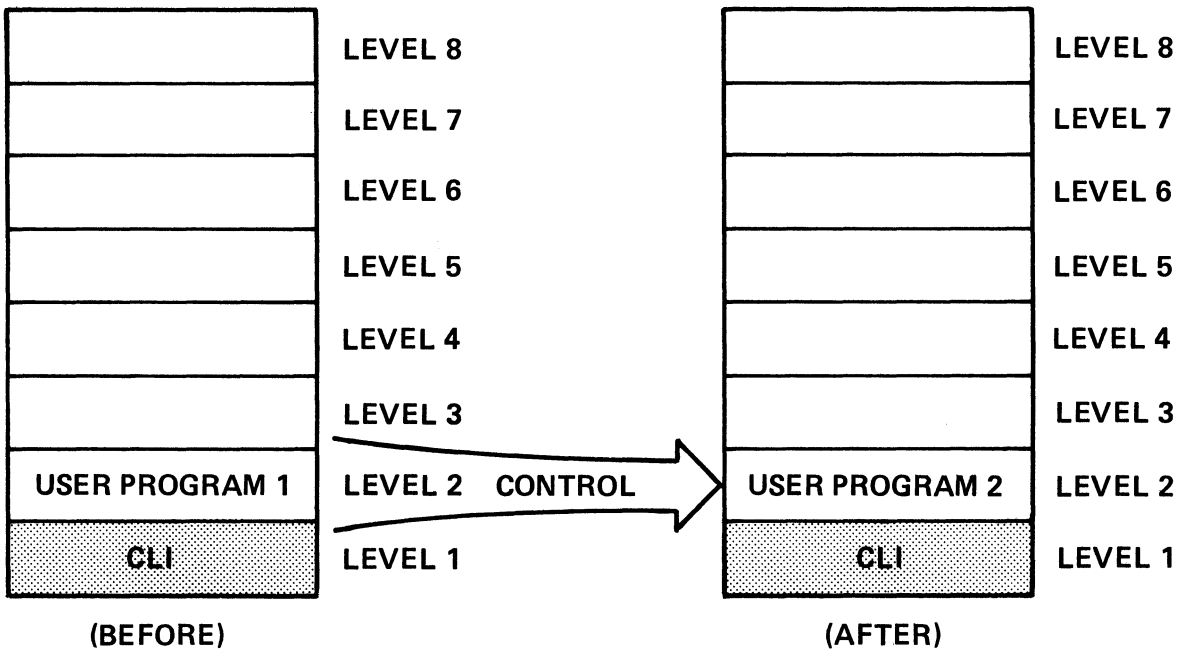
PROGRAM STACK DURING EXECUTION OF A SWAP

Figure 1-6



PROGRAM STACK AFTER
USER TERMINATION

Figure 1-7

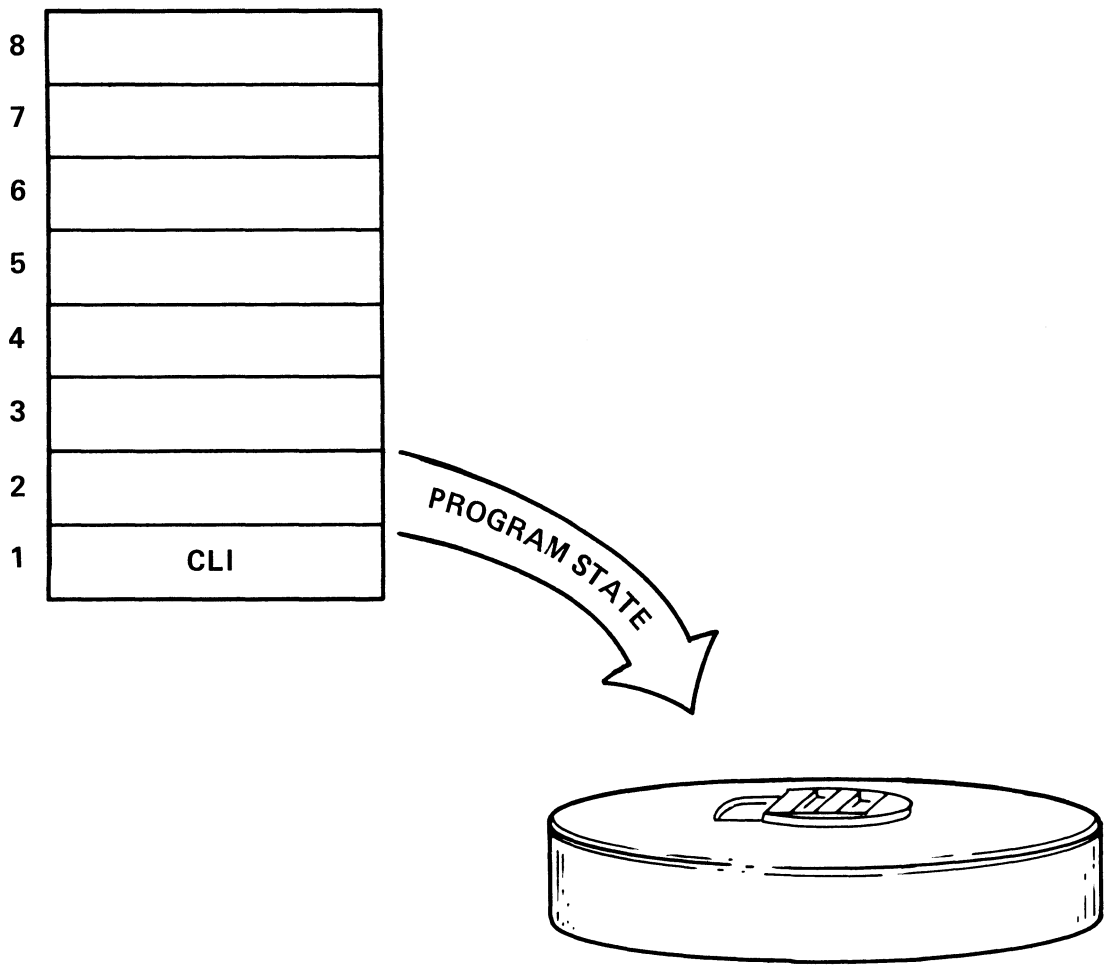


CHAIN – STATE OF CALLING PROGRAM IS LOST

SWAP – STATE OF CALLING PROGRAM IS SAVED AND RESTORED ON LEVEL TWO'S TERMINATION

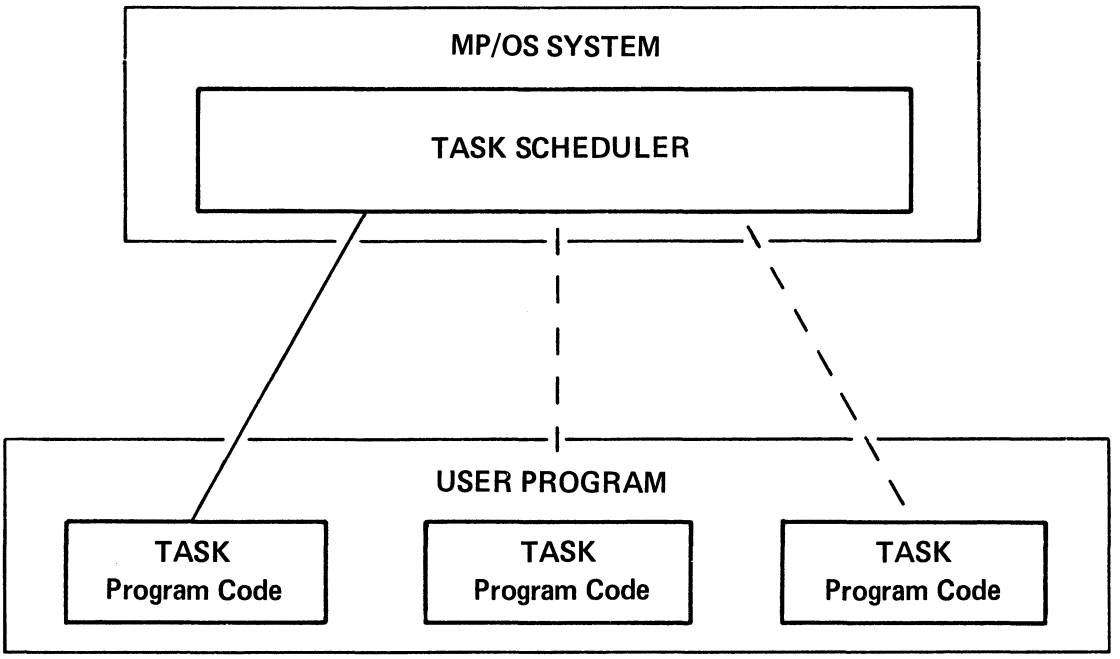
PROGRAM STACK DURING
A CHAIN OPERATION

Figure 1-8



USING A BREAK FILE TO STORE
A PROGRAM'S OPERATING STATE

Figure 1-9



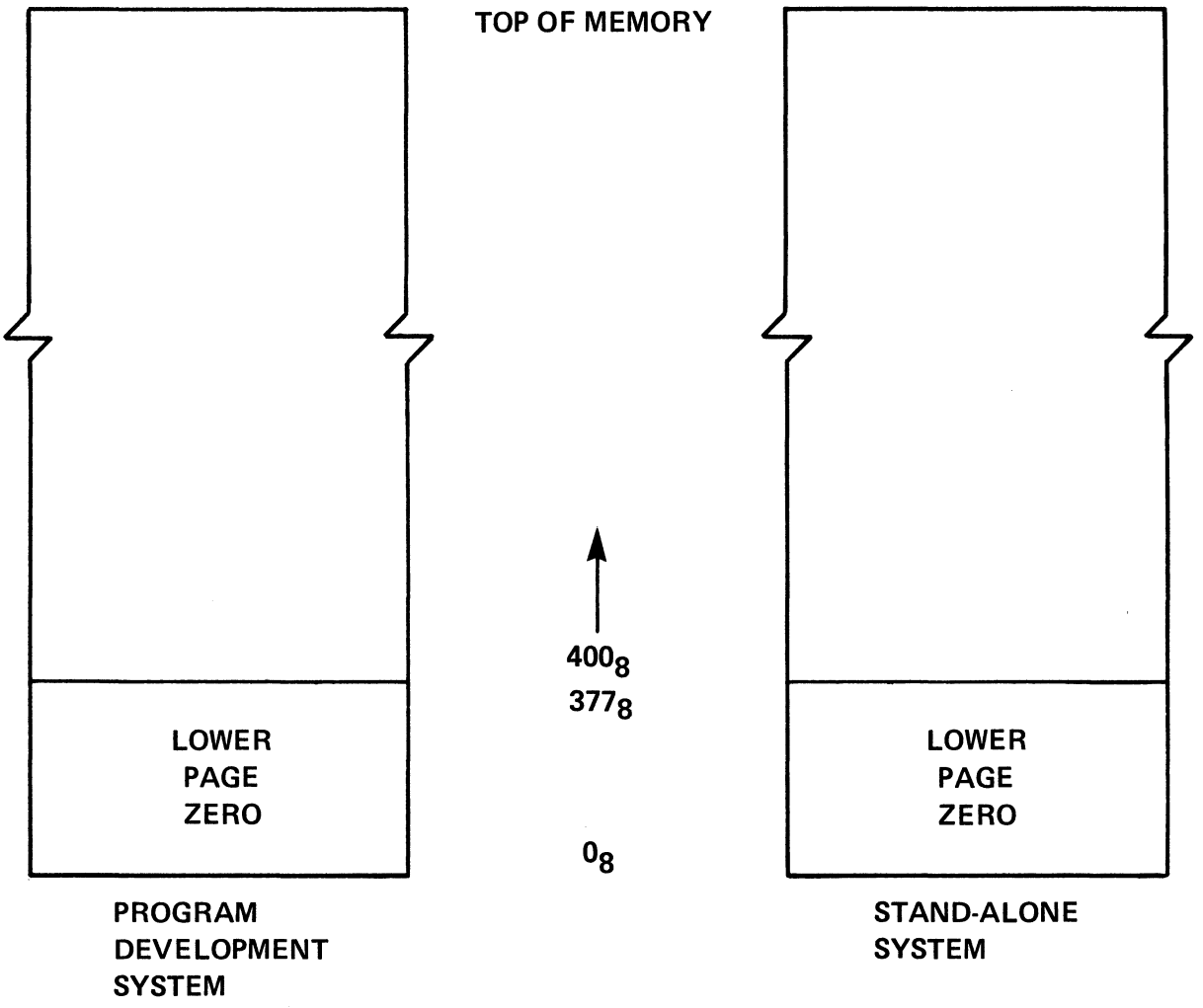
MULTITASKING – MULTIPLE TASKS (ASYNCHRONOUS PATHS OF EXECUTION) IN A SINGLE PROGRAM

MULTIPROGRAMMING – MULTIPLE PROGRAMS IN MEMORY FOR EXECUTION (NOT SUPPORTED BY MP/OS)

- 255 TASKS
- CREATE
- PRIORITIZE
- INTERCOMMUNICATION
- KILL

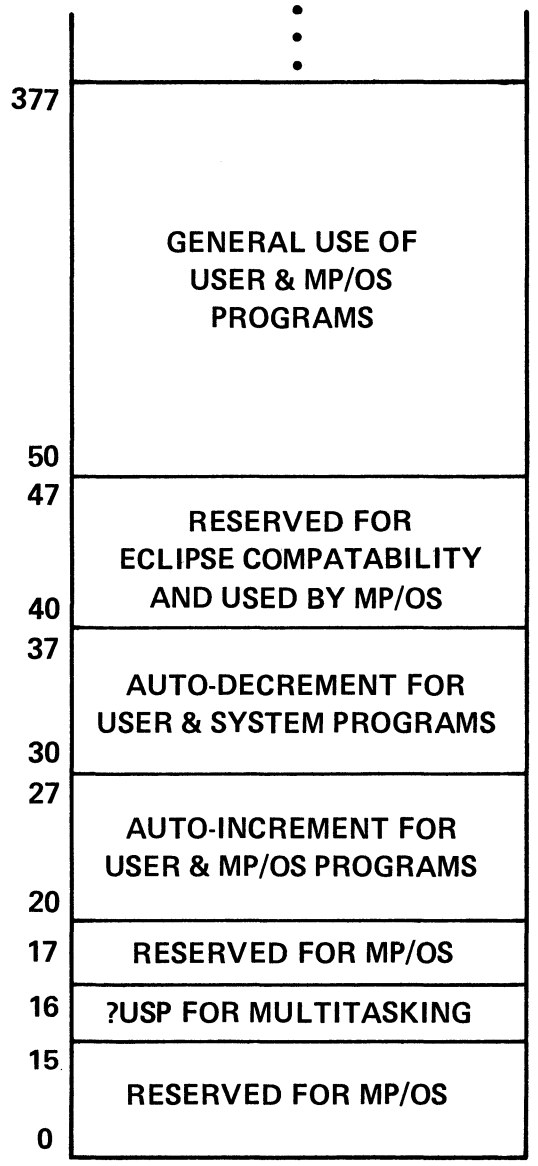
MULTITASKING

Figure 1-10



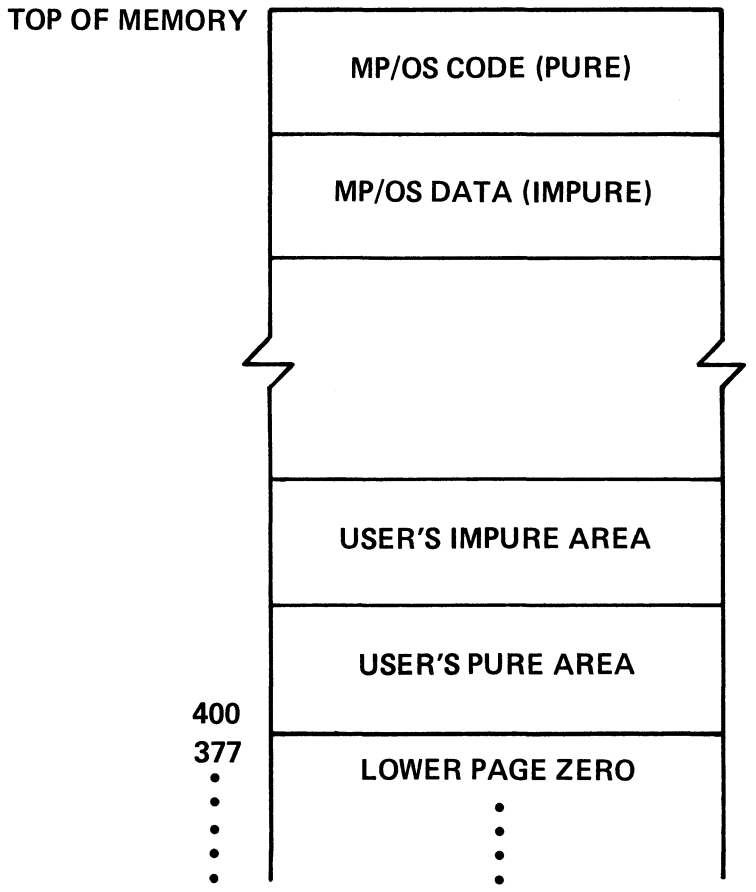
BASIC MEMORY ORGANIZATIONS

Figure 1-11



MEMORY CONFIGURATION FOR
DEVELOPMENT & STAND-ALONE SYSTEMS

Figure 1-12

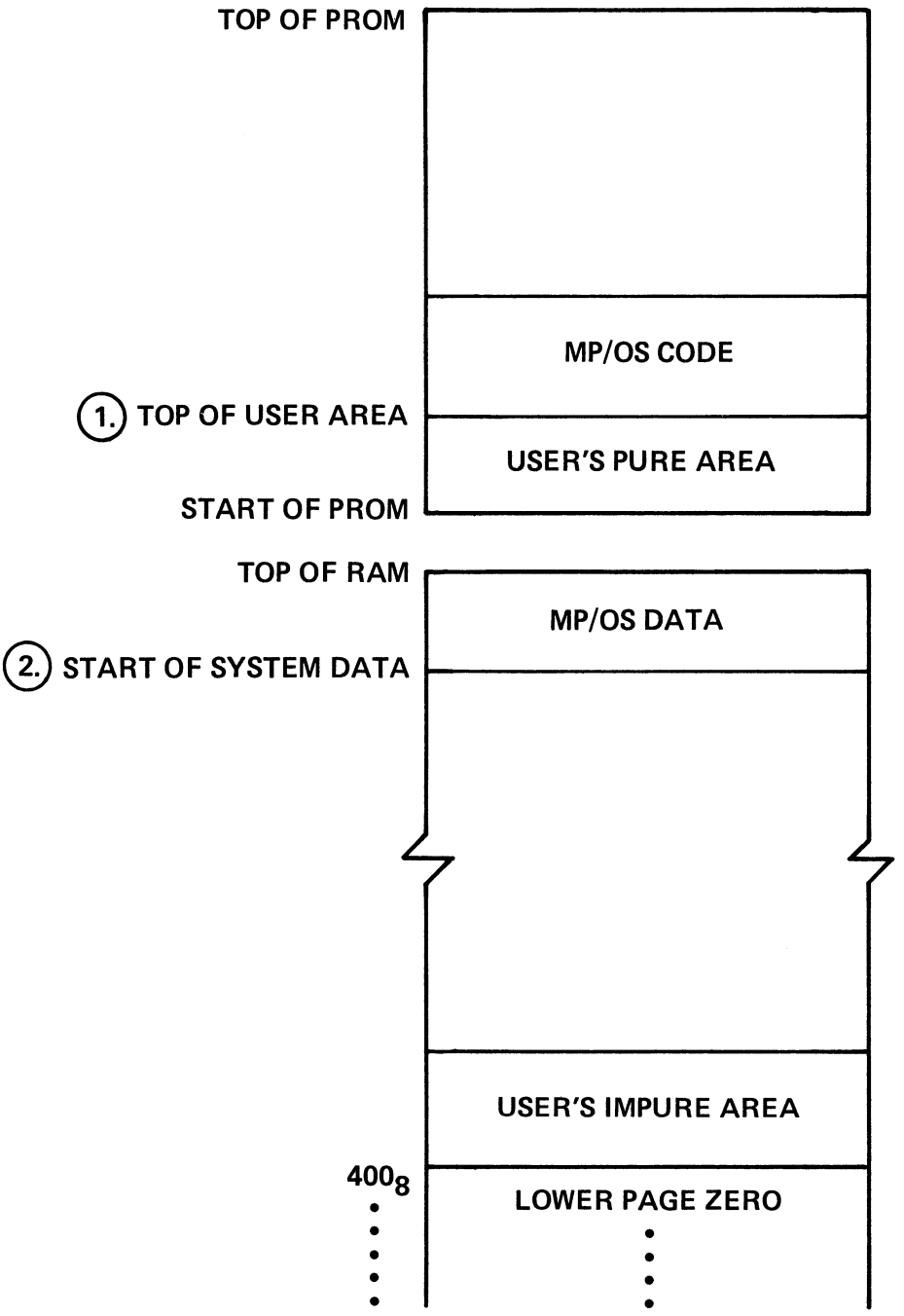


PURE AREA – PROGRAM AREA NEVER MODIFIED DURING EXECUTION AND THEREFORE, NOT SAVED

IMPURE AREA – PROGRAM AREA MODIFIED DURING EXECTION AND SAVED DURING SWAP

MEMORY CONFIGURATION
FOR A DEVELOPMENT SYSTEM

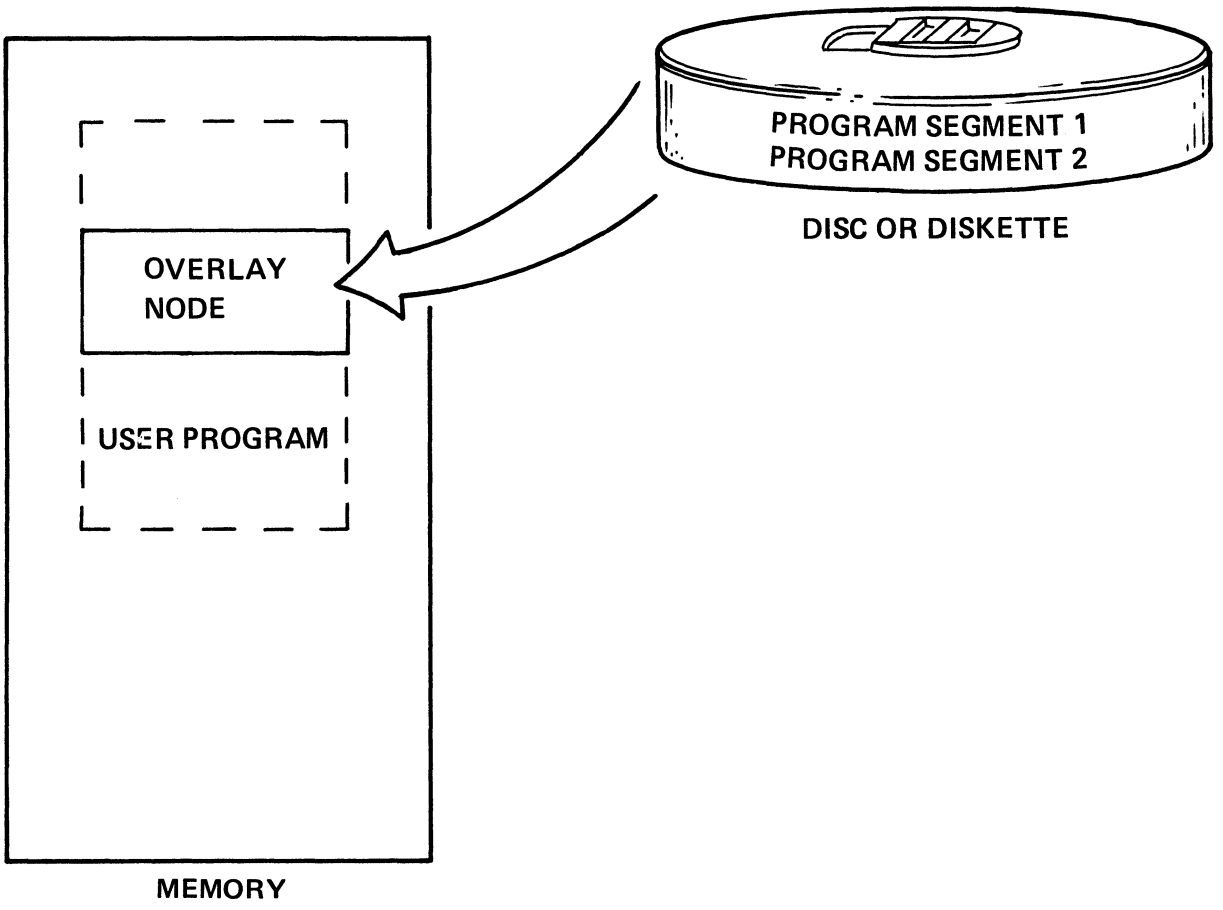
Figure 1-13



NOTE: Locations 375, 376, 377 are NOT available for access on some systems.

MEMORY CONFIGURATION FOR A PROM-BASED STAND-ALONE SYSTEM

Figure 1-14

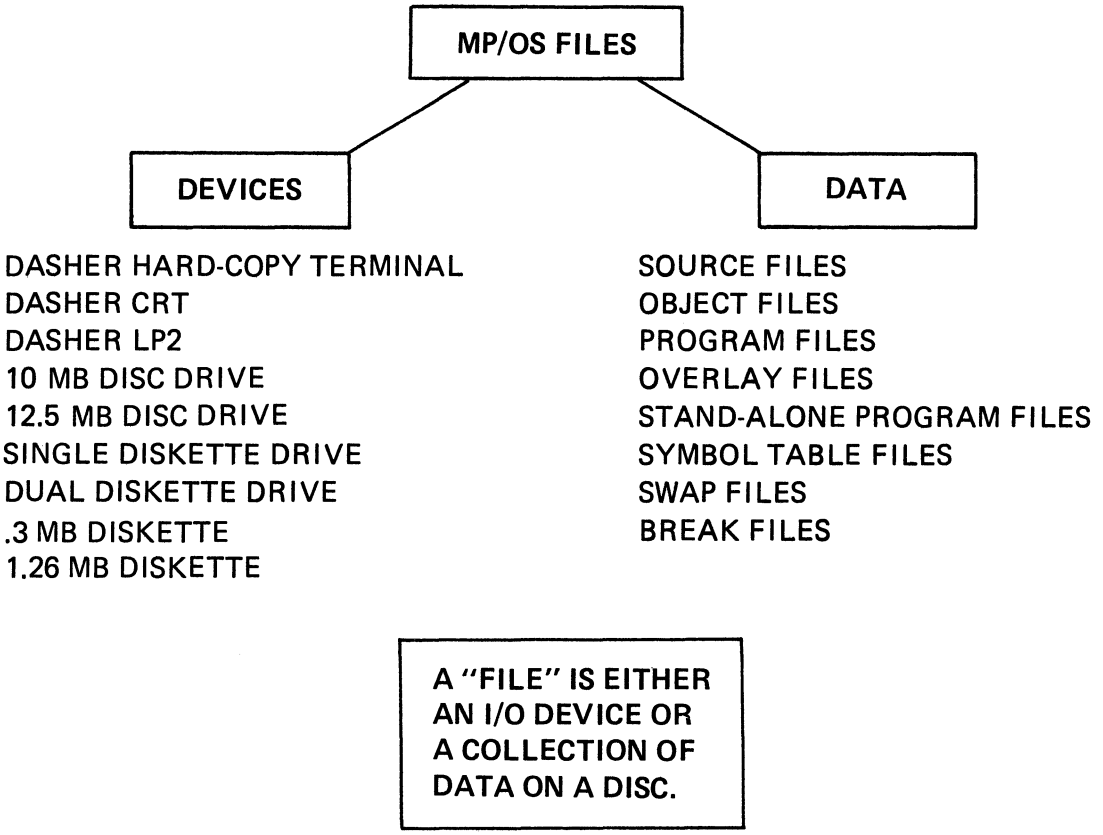


OVERLAYS – USED FOR INFREQUENTLY USED PROGRAM ROUTINES.
MORE THAN ONE .OL IS ALLOWED

NODE – MEMORY AREA RESERVED FOR OVERLAY ROUTINES.
MORE THAN ONE NODE IS ALLOWED

OVERLAYING

Figure 1-15



MP/OS DATA MANAGEMENT

Figure 1-16

FILENAMES

1. 1 to 15 Characters in Length
2. A thru Z Upper Case
3. a thru z Lower Case
4. 0 thru 9
5. ? (QUESTION MARK or "HOOK")
6. $
7. _ (UNDERSCORE)
8. . (PERIOD)

FILE NAMING RULES

Figure 1-17

LEGAL FILENAMES

1. NEWJERSEY
2. NEW_ JER$E.Y
3. NEWJERSEY.2
4. NewjeRsey
5. NEWJERSEY?

ILLEGAL FILENAMES

6. NEW JERSEY
7. NEWJE@SEY
8. NEW–JERSEY
9. NEWJERSEY#7
10. NE*JERSEY

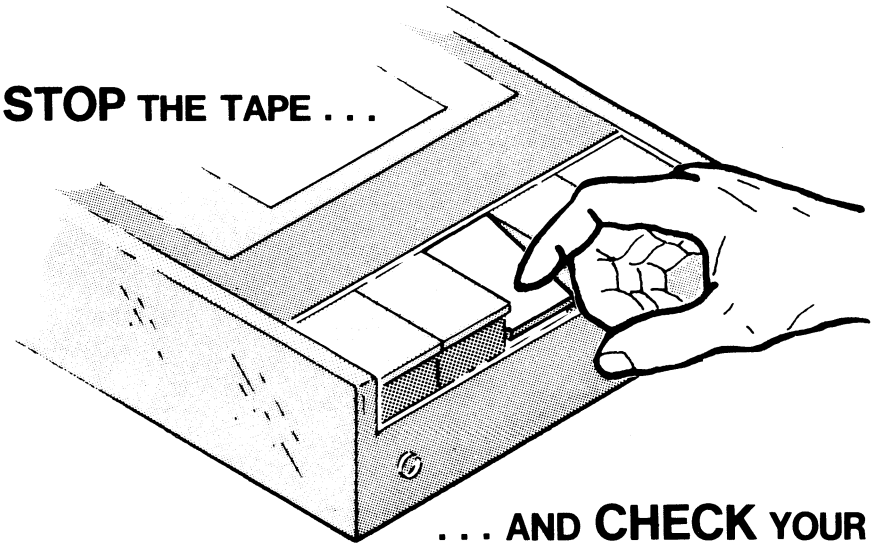
LEGAL AND ILLEGAL FILENAMES

Figure 1-18

TOPICS

- FILENAMES
- 1-15 CHARACTERS
- UPPER OR LOWER CASE
- VALID CHARACTERS (0 through 9, ?, –, .)

NOW **STOP** THE TAPE . . .



. . . AND **CHECK** YOUR PROGRESS

FILENAMES QUIZ

Identify the legal filenames by placing an "L" in the space provided and identify the illegal filenames by placing an "I" in the space provided.

L = LEGAL
I = ILLEGAL

1.____ PROGRAMONE

2.____ PROGRAM-ONE

3.____ PROGRAM__ONE

4.____ PROGRAM.1

5.____ PROGRAM ONE

6.____ PROGRAMONE?

7.____ PROGRAM*1

8.____ PROGRAM@1

9.____ program.one

10.____ PROGRAM:ONE

CHECK YOUR ANSWERS ON THE NEXT PAGE

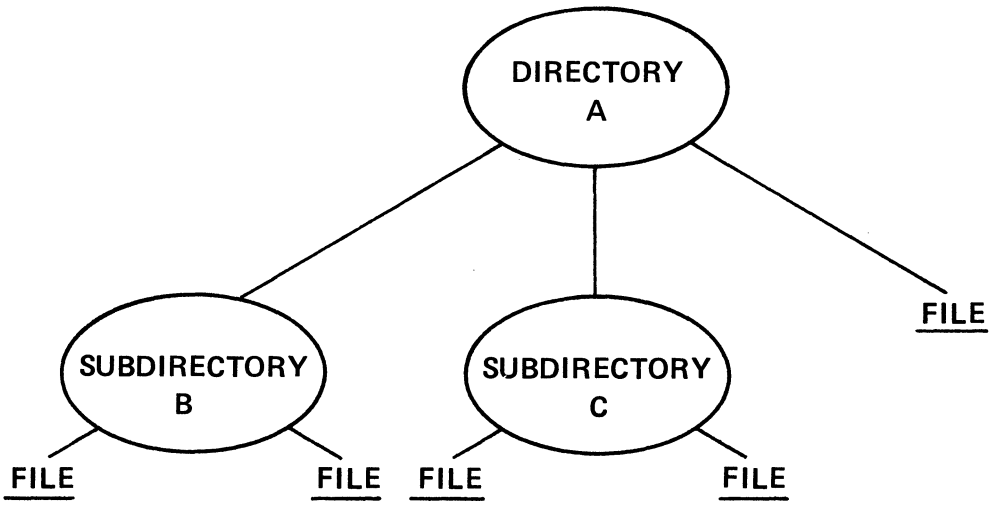
FILENAMES QUIZ

ANSWERS

L = LEGAL
I = ILLEGAL

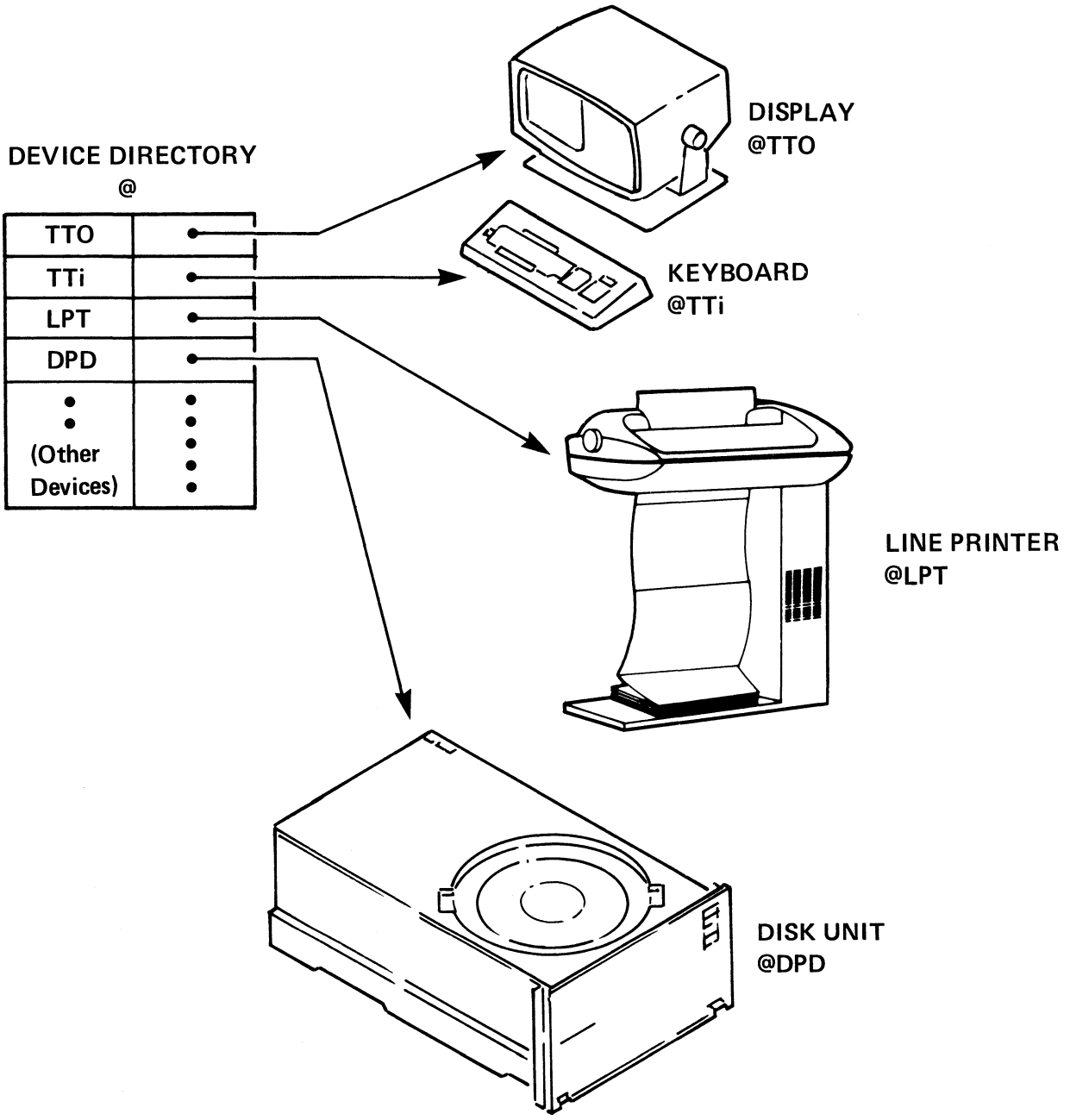
1 . . L . . . PROGRAMONE	All Upper-case, nothing fancy
2 . . I . . . PROGRAM-ONE	Hyphen is an illegal character
3 . . L . . PROGRAM_ONE	Under-score is legal.
4 . . L . . . PROGRAM.1	Numbers are legal characters
5 . . I . . . PROGRAM ONE	The space is an illegal character
6 . . L . . . PROGRAMONE?	The question mark is a legal character.
7 . . I . . . PROGRAM*1	The asterisk is an illegal character
8 . . I . . . PROGRAM@1	The "at" sign is an illegal character
9 . . L . . . program.one	
10 . I . . . PROGRAM:ONE	The colon is an illegal character.

YOU SHOULD GET 8 CORRECT OUT OF THE 10 QUESTIONS TO ACHIEVE MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN YOU UNDERSTAND THE CORRECT ANSWERS. THEN RETURN TO THE AUDIOTAPE.



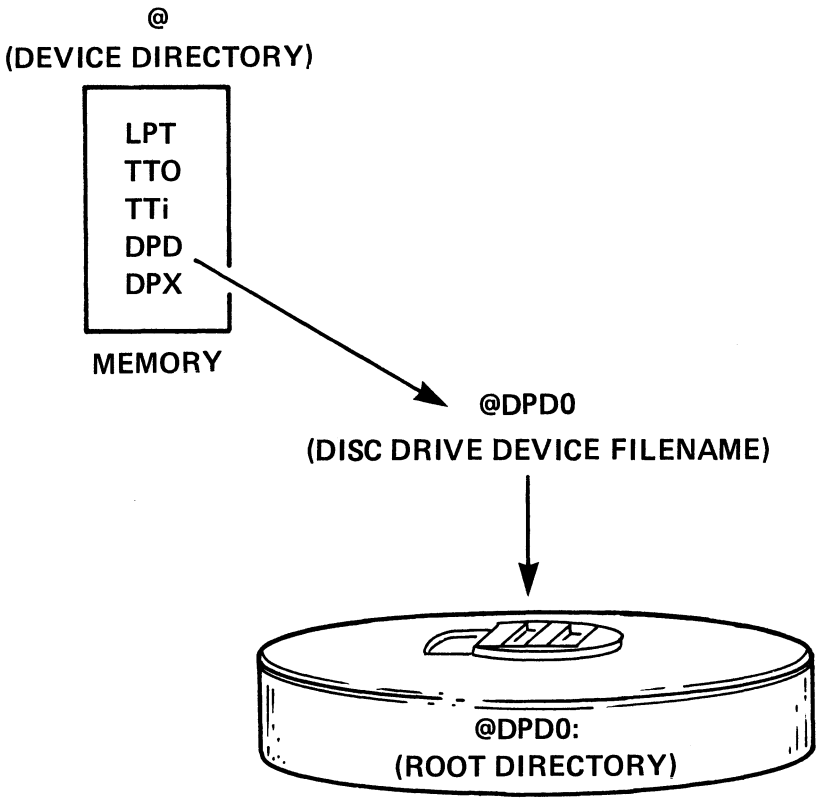
DIRECTORY FILES AND SUBDIRECTORY FILES

Figure 1-19



DEVICE DIRECTORY

Figure 1-20

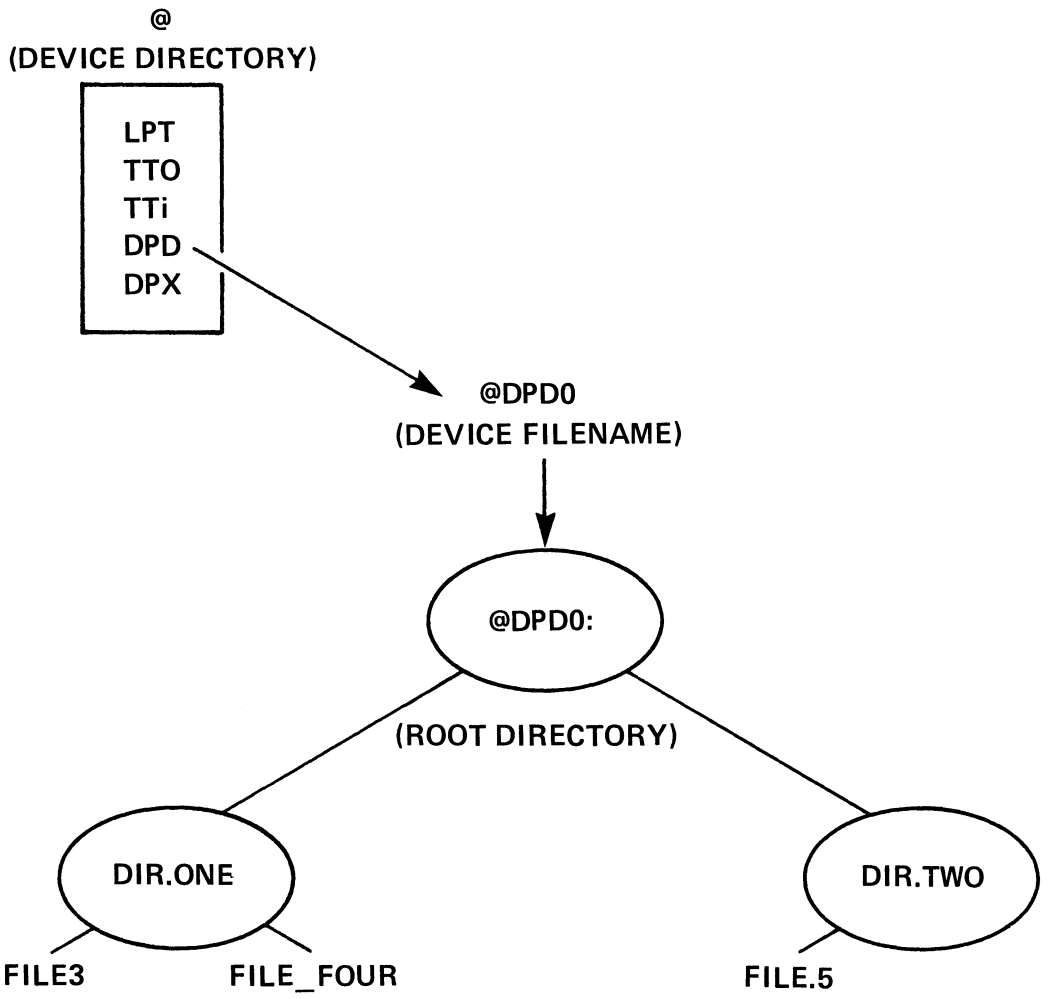


DEVICE DIRECTORY "@" LISTS ALL SYSTEM DEVICES BY FILENAME

ROOT DIRECTORY CONTAINS ALL FILES ON ONE DISC OR DISKETTE

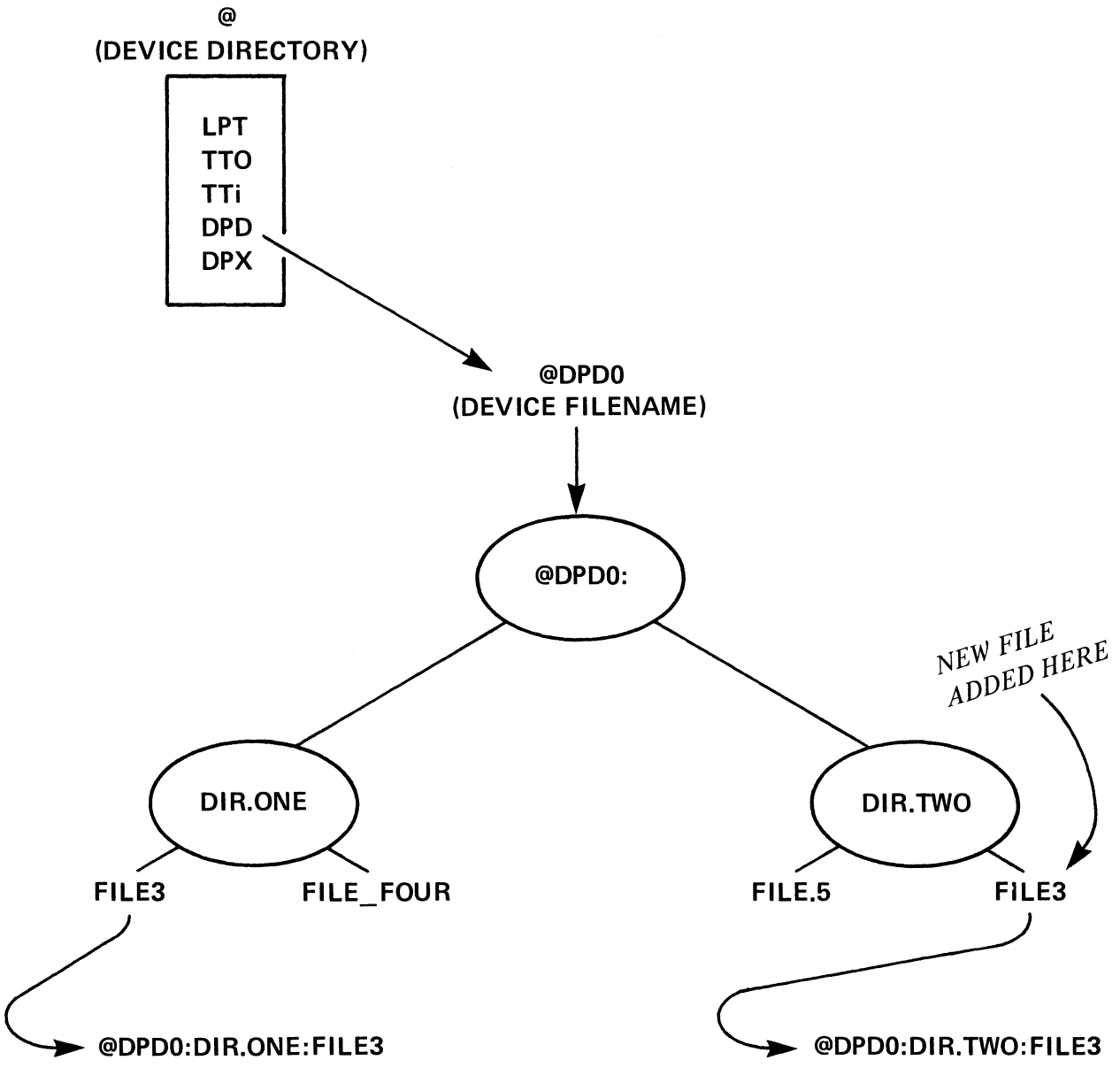
THE DEVICE DIRECTORY, DEVICE,
AND ROOT DIRECTORY

Figure 1-21



THE DEVICE DIRECTORY, DEVICE,
ROOT DIRECTORY, AND FILES.

Figure 1-22



PATHNAMES IDENTIFYING UNIQUE FILES

Figure 1-23

PATHNAME

1) A path through a Directory Structure to a particular file.

2) A series of filenames separated by colons.

3) Maximum length of 127 characters.

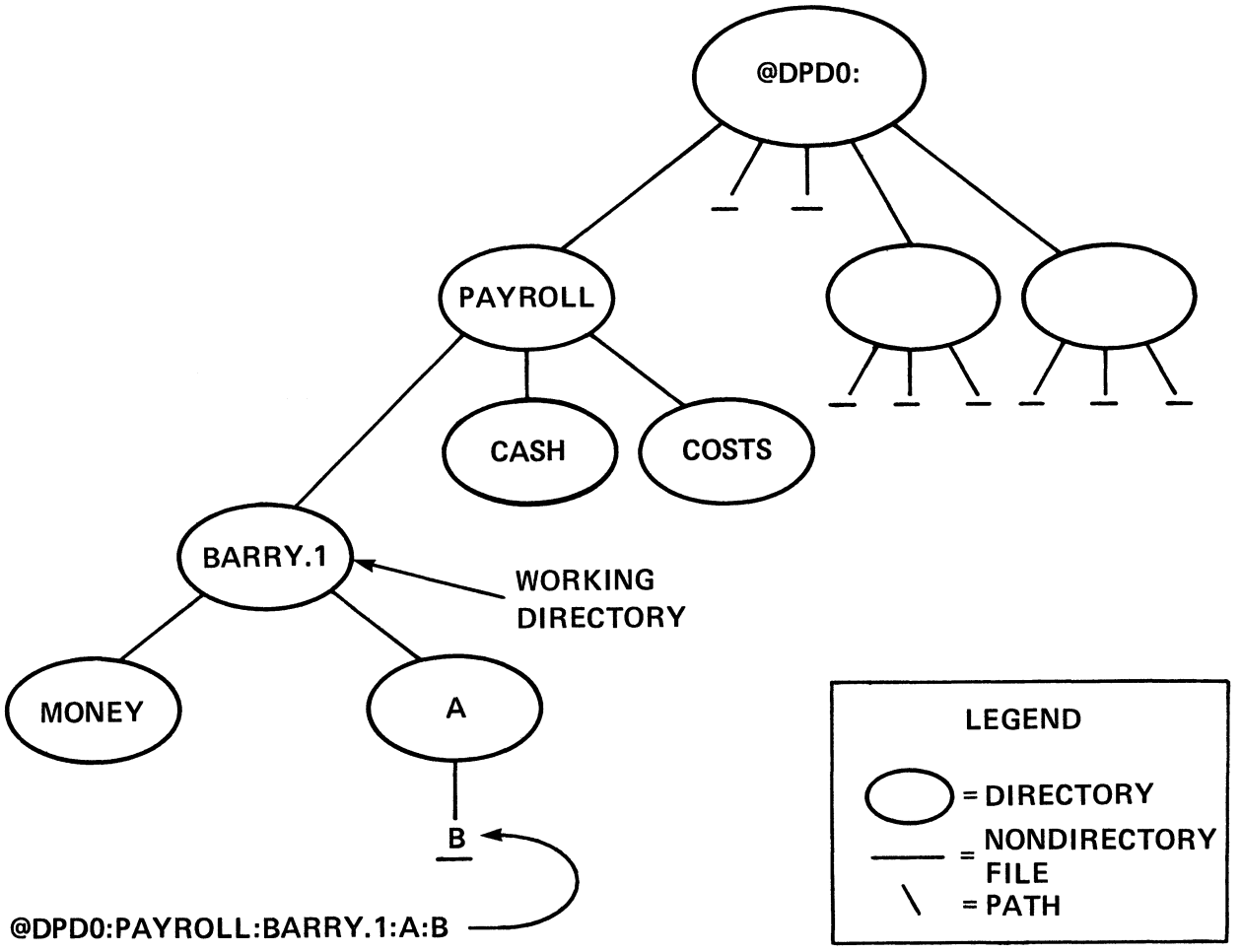
4) All filenames except last one must be directories.

5) Each directory in the pathname must be a subdirectory of the preceding directory.

6) Fully qualified pathname begins at root directory.

PATHNAME RULES

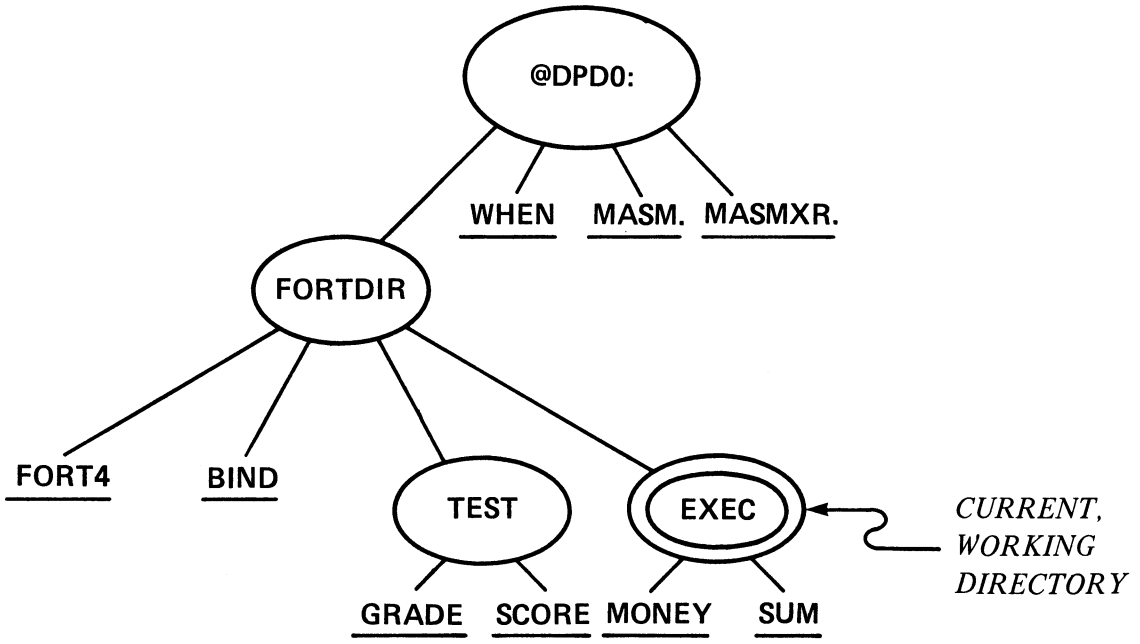
Figure 1-24



WORKING DIRECTORY – "CURRENT LOCATION" IN THE DIRECTORY STRUCTURE.
FILE SEARCHES BEGIN AT WORKING DIRECTORY.

DIAGRAM OF A COMPLEX FILE STRUCTURE

Figure 1-25



VALID PATHNAMES

1. @DPD0:FORTDIR
2. @DPD0:WHEN
3. MONEY
4. @DPD0:FORTDIR:FORT4
5. @DPD0:FORTDIR:TEST:SCORE

INVALID PATHNAMES

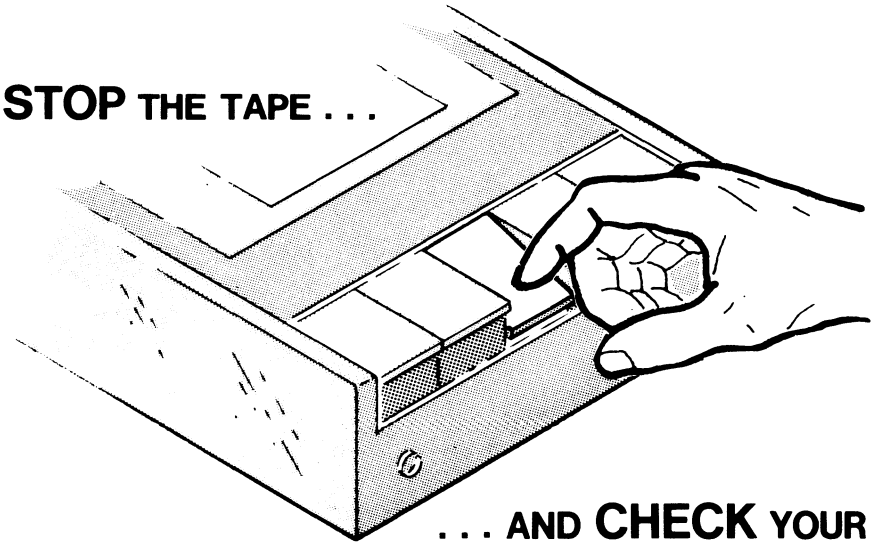
6. TEST:SCORE
7. MONEY:SUM
8. @DPD0:FORTDIR:MASM
9. @DPD0:FORTDIR:EXEC:MONEY:SUM
10. SCORE

Figure 1-26

TOPICS

- PATHNAMES
- FILENAMES SEPARATED BY :
- MAX OF 127 CHARACTERS.
- LAST FILENAME IS A NON-DIRECTORY FILE.

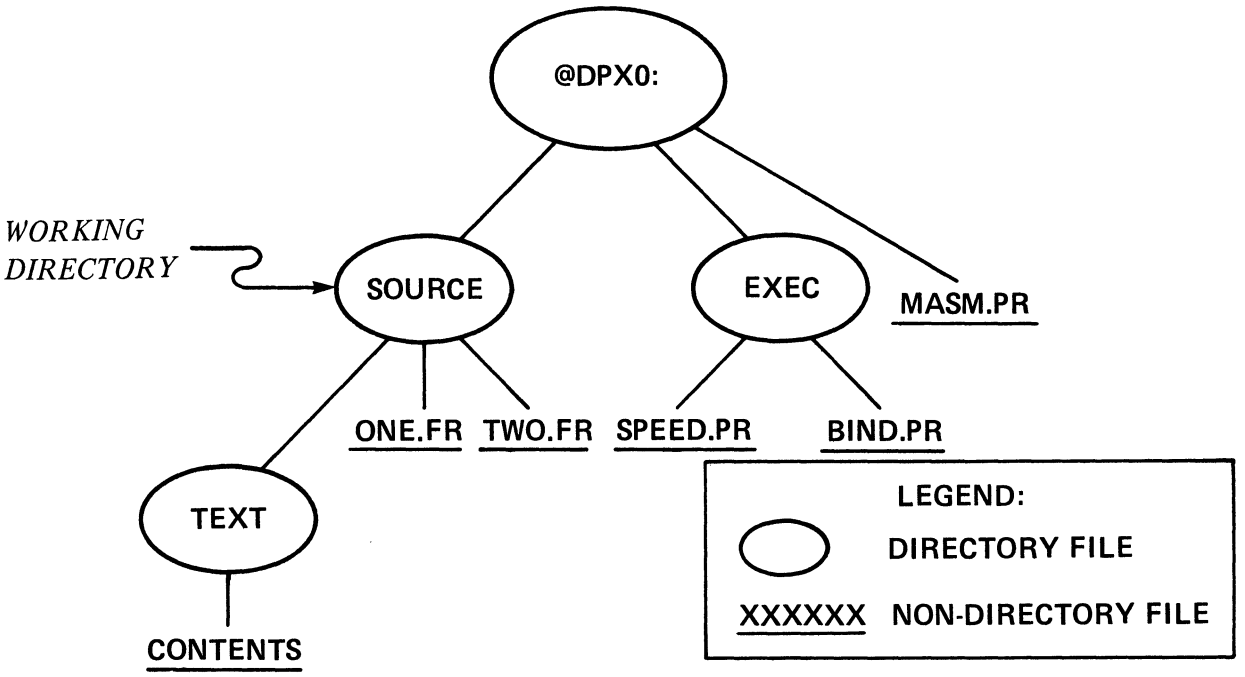
NOW **STOP** THE TAPE . . .



. . . AND **CHECK** YOUR PROGRESS

PATHNAMES QUIZ

Identify the valid and invalid pathnames for the following directory structure. Source is the working directory.



V = VALID
I = INVALID

1. ____ SOURCE:ONE.FR
2. ____ ONE.FR
3. ____ SOURCE:@DPX0:
4. ____ @DPX0:MASM.PR
5. ____ @DPX0:SOURCE:TEXT:CONTENTS
6. ____ SOURCE:EXEC:SPEED.PR
7. ____ SOURCE:EXEC:MASM.PR

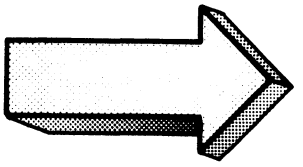
8. ____ TWO.FR

9. ____ SPEED.PR

10. ____ EXEC:SPEED.PR

CHECK YOUR ANSWERS

ON THE NEXT PAGE



PATHNAMES QUIZ

ANSWERS

V = VALID
I = INVALID

1. I . . . . SOURCE:ONE.FR SOURCE is not visible from SOURCE, just use ONE.FR.
2. V . . . . ONE.FR Since SOURCE is the working directory, this is a valid entry.
3. I . . . . SOURCE:@DPX0: This is reversed.
4. V . . . . @DPX0:MASM.PR Valid because it starts with the root directory.
5. V . . . . @DPX0:SOURCE:TEXT:CONTENTS A valid, fully qualified pathname
6. I . . . . SOURCE:EXEC:SPEED.PR Exec is not a subdirectory of source.
7. I . . . . SOURCE:EXEC:MASM.PR Exec is not a subdirectory of source.
8. V . . . . TWO.FR Two.FR is a file in the working directory.
9. I . . . . SPEED.PR SPEED.PR is not in the working directory.
10. I . . . . EXEC:SPEED.PR Add the @DPD0:.

YOU SHOULD GET 8 CORRECT OUT OF 10 QUESTIONS TO ACHIEVE MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN RETURN TO THE AUDIOTAPE.

SEARCHLIST

AN ORDERED LIST OF DIRECTORIES YOU WANT SEARCHED ANY TIME YOU REFERENCE A FILE NOT LISTED IN THE WORKING DIRECTORY.

SYSTEM MASTER DEVICE HOLDS OPERATING SYSTEM PROGRAMS. REFERENCED AS ":"

SYSTEM MASTER DEVICE

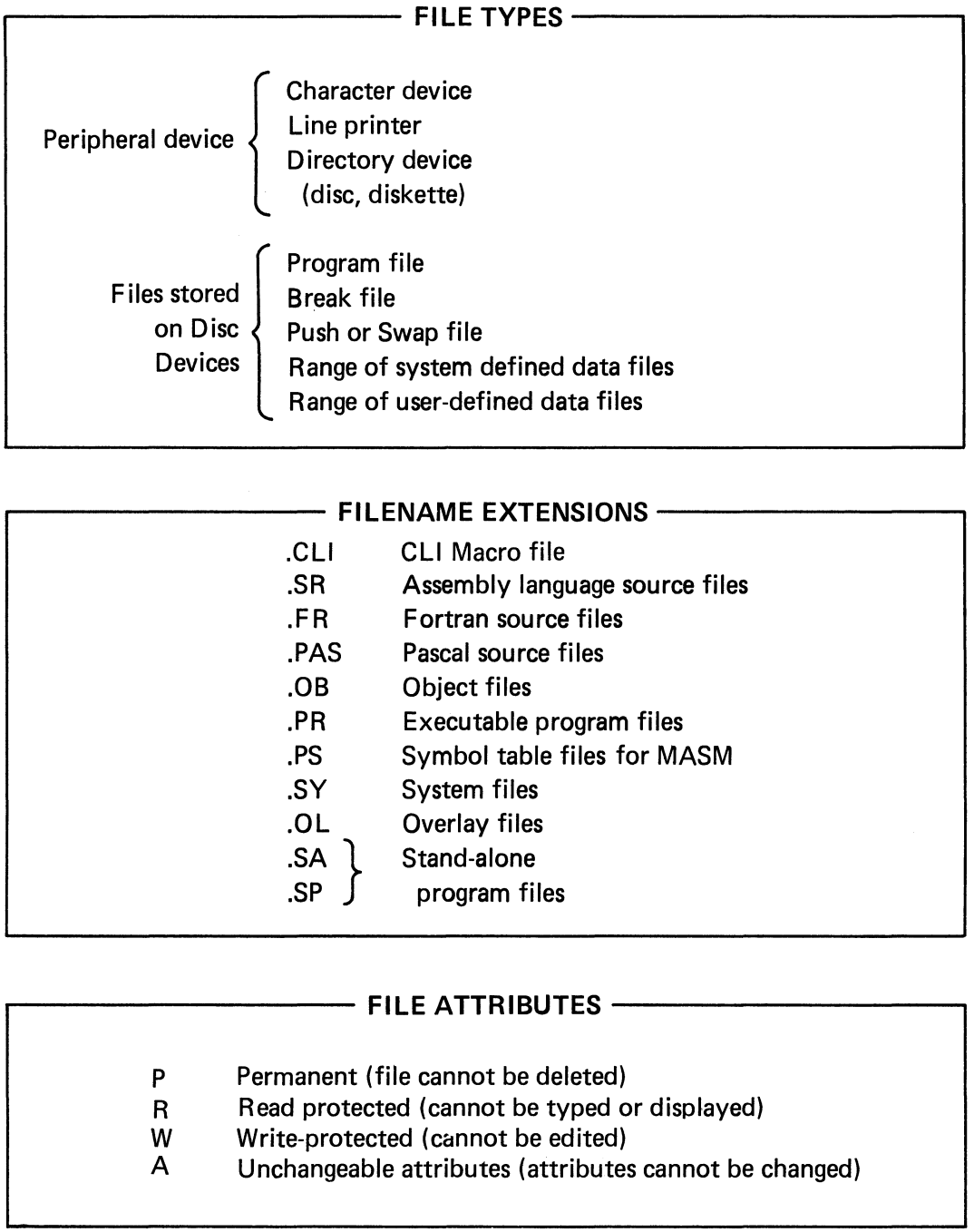
Figure 1-27

BLOCK = 512 BYTES

FILE ELEMENT = ONE OR MORE BLOCKS

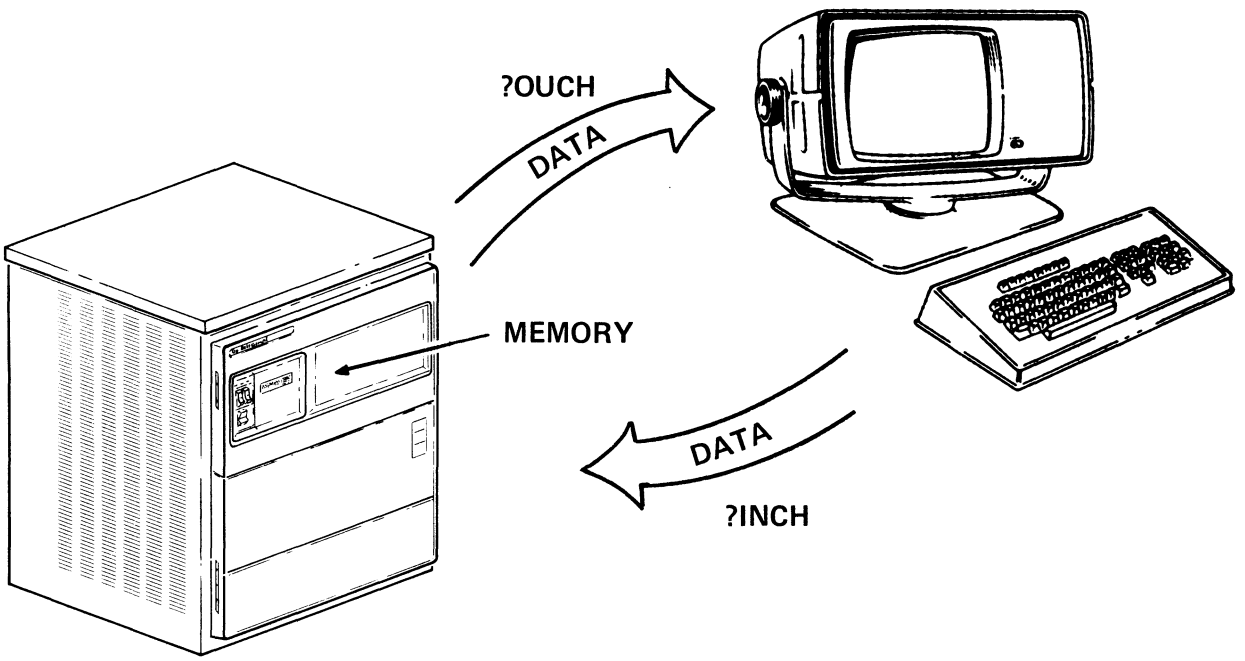
- LARGE ELEMENTS ALLOW FAST ACCESS
- SMALL ELEMENTS ALLOW EASY ALLOCATION
- MAXIMUM FILE SIZE DETERMINES IDEAL SIZE

Figure 1-28



FILE DESCRIPTORS

Figure 1-29



?INCH – INPUT DATA CHANNEL
?OUCH – OUTPUT DATA CHANNEL

I/O CHANNEL: SYSTEM-DEFINED DATA PATH

Figure 1-30

I/O TYPES

DYNAMIC I/O

- FAST TRANSFER
- FILE ⇌ MEMORY

BLOCK I/O (DYNAMIC)

- FASTEST TRANSFER
- DISC ⇌ MEMORY BUFFER

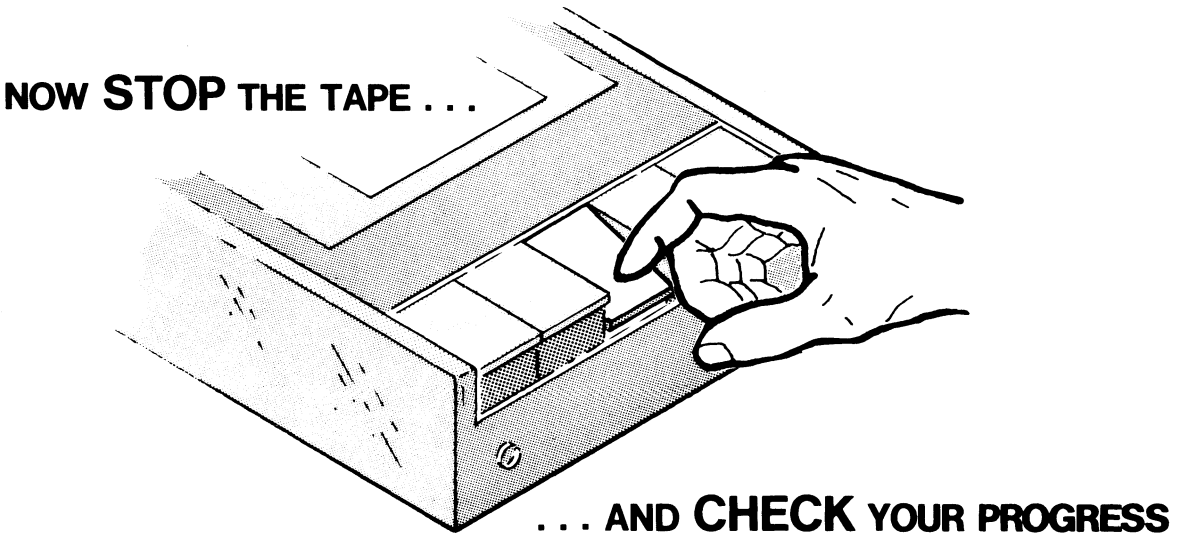
DATA SENSITIVE

- MEMORY ⇌ SYSTEM BUFFER

Figure 1-31

TOPICS

- SOFTWARE CONTROL
  - flexible resource manager
  - stack
  - system calls
  - swaps, chains, breakfiles
  - multitasking
- MEMORY CONFIGURATIONS
  - Lower Page Zero
  - Pure & impure program areas
  - Stand-alone vs. Program development configurations
  - Overlaying
- DATA MANAGEMENT
  - Directory and non-directory files.
  - Files as I/O device or disk data.
  - Device directory, root directory
  - System master device
  - Searchlist
  - File type, extension, attributes.



MODULE ONE QUIZ

Circle the best answer(s) to the following questions. Note that some questions require more than one answer.

1. MP/OS can be tailored to your application. Therefore, it can be used:
   A. in a stand-alone environment.
   B. in a program development environment.
   C. in a disk/diskette-based operation.
   D. as part of a distributed network.

2. MP/OS manages
   A. access to main memory.
   B. program execution.
   C. I/O.
   D. disc organization and access.

3. Information about each executing program is maintained in a:
   A. directory.
   B. program stack.
   C. level.
   D. channel command word.

4. The Command Line Interpreter (CLI) serves as:
   A. a job control language.
   B. directory maintenance utility.
   C. peripheral interface program.

5. When the system is initialized, a program named CLI.PR (CLI) is invoked and enters:
   A. at level one in the program stack
   B. a block
   C. a swap
   D. an element

6. When a descendant program is called in to execute and the parent program is saved:
    A. a chain occurs
    B. a terminate occurs.
    C. a swap occurs.
    D. a block occurs.

7. The memory locations containing code and data that are never modified during program execution is referred to as:
    A. lower page zero
    B. impure area
    C. pure area
    D. supervisor area

8. Locations 0 through 377 for any environment always consist of:
    A. pure area.
    B. impure area.
    C. lower page zero.
    D. supervisor area.

9. Which pair(s) can be in memory a the same time?
    A. Pure and impure areas.
    B. MP/OS code and data.
    C. System code and lower page zero.
    D. CLI and a compiler.

10. An MP/OS file may be a:
    A. collection of data.
    B. Dasher Line Printer.
    C. Program when resident on disc.
    D. non-directory file.

11. Which of the following is a legal MP/OS filename:
    A. BARRY_SMITH
    B. BARRY$.M_TH
    C. BARRY$._7?H
    D. barry.1

12. A file that may contain other files is called a:
    A. character-oriented device
    B. directory
    C. stack
    D. pathname

13. An expression used to reference a file within a directory is called a
    A. working directory
    B. sub-directory
    C. pathname
    D. device directory

14. The device directory:
    A. is referred to with "@".
    B. is the highest directory in the system.
    C. contains the filenames of all I/O devices.
    D. is a table in memory.

15. The highest directory on a disc device is the:
    A. device directory
    B. root directory
    C. parent directory
    D. working directory

16. Every program has a directory from which it executes. This directory is the:
    A. device directory
    B. root directory
    C. parent directory
    D. working directory

17. An easy way to access common files and programs is to use the:
    A. directory list
    B. device directory
    C. searchlist
    D. fully qualified pathname

18. A system-defined data path between a file and an application program is:
    A. a filename
    B. Block I/O
    C. an I/O channel
    D. Dynamic I/O

19. CLI is an executable program file. MP/OS would expect which extension to its filename:
    A. .OB
    B. .PR
    C. .PAS
    D. .SR

20. A permanent file that cannot be modified would have attributes:
    A. PA
    B. PR
    C. RW
    D. AW
    E. PW

21. The slowest I/O transfer is:
    A. Dynamic I/O
    B. Data-sensitive I/O
    C. Block I/O

22. MP/OS allows dividing your program into a number of subprograms called:
    A subroutines
    B. overlays
    C. tasks
    D. levels

CHECK YOUR ANSWERS
ON THE FOLLOWING
PAGES

MODULE ONE QUIZ

ANSWERS

1. A,B,C,D. MP/OS systems can be tailored to stand-alone, program development, disk/diskette-based operations, and part of a distributed network.

2. A,B,C,D. MP/OS provides efficient management of all CPU resources.

3. B. The program stack maintains information about the executing program. Internal tables (UST and TCB) also maintain information about the user's programs.

4. A,B,C. The CLI is your interactive interface to MP/OS requests and, as such, serves as JCL, directory maintenance, and peripheral interface.

5. A. CLI enters the program stack at level one at system initiation. If you create another file named CLI.PR and delete the original, the new file would be invoked automatically at initiation. This is not a recommended procedure.

6. C. A swap occurs when the parent program's state is saved. The parent's state is lost during a chain.

7. C. The pure area of a program consists of code and data that are never modified during program execution.

8. C. Lower page zero occupies addresses 0 through 377. The organization of the remainder of memory will vary with the environment and must also be impure.

9. A,B,C. Each of these occupy the memory at the same time, but they are prevented from overlapping by MP/OS. CLI can invoke the compiler, but won't be in memory at the same time.

10. A,B,C,D. A file refers to devices, data sets, and programs when resident on disc.

11. A,B,C,D. All are legal filenames.

12. B. A directory file may contain other files (and other directories).

13. C. A pathname references a file within a directory.

14. A,B,C,D The device directory, referred to as @, is a system table that lists all the devices in the system.

15. B. The Root Directory, referred to as ":", is the highest directory on each disc device.

16. D. Each program executes from a working directory. The working directory can easily be changed, it is not static.

17. C. The searchlist is an easy way to access common files and programs. It is a list of directories to be searched to reference a desired file.

18. C. An I/O channel is a system-defined data path between a file and the application program.

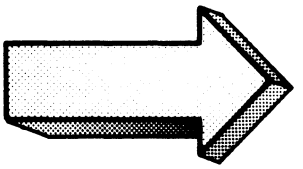
19. B. CLI.PR is the executable version of the Command Line Interpreter.

20. E. Attributes P (Permanent) and W (Write-protect) describe a file that cannot be deleted or modified. Note that the "A" attribute describes a file whose attributes cannot be changed, although an "A" file's contents may be modified.

21. B. Data-sensitive I/O is the slowest type of data transfer in an MP/OS system.

22. A,B,C. A program may be divided into subroutines, overlays, or tasks. The choice depends on the application.

A SCORE OF 19 CORRECT QUESTIONS OUT OF THE 22 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.



MODULE TWO
CLI

MODULE TWO

CLI

Abstract

This module is divided into the following sections:

- CLI concepts and functional capabilities
- How to use CLI
- Commands and options
- Command macros

Objectives

Upon completion of this module, you will be able to:

1. State the purpose of the Command Line Interpreter.

2. Given a list of console control character sequences:
   a) determine the function performed by either stating from memory or referencing the appropriate technical source;
   b) explain the results of each sequence

3. Given a list of command lines
   a) reference the function performed by each command;
   b) describe the options performed by appending switches to the command and/or arguments.
   c) describe the results of each command with and without options.

4. Given a processing problem, enter the appropriate CLI command to solve the situation.

5. Define, in your own words:
   a) working directory
   b) pathname
   c) file
   d) directory
   e) root
   f) parent directory
   g) searchlist
   h) system master device

6. Given a description of a system file structure:
   a) illustrate the file structure by sketching the relation between files;
   b) state the pathnames to each file.

7. Given a processing situation enter a Macro command to satisfy the situation.

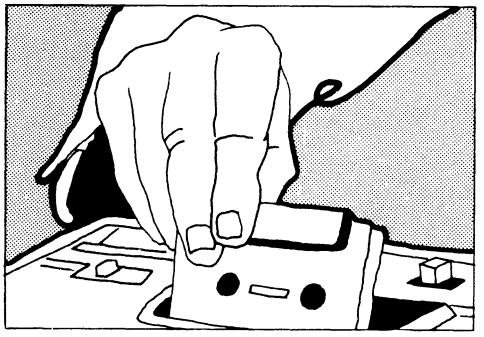
8. Given a list of CLI macro commands, describe the expected result of the operation.

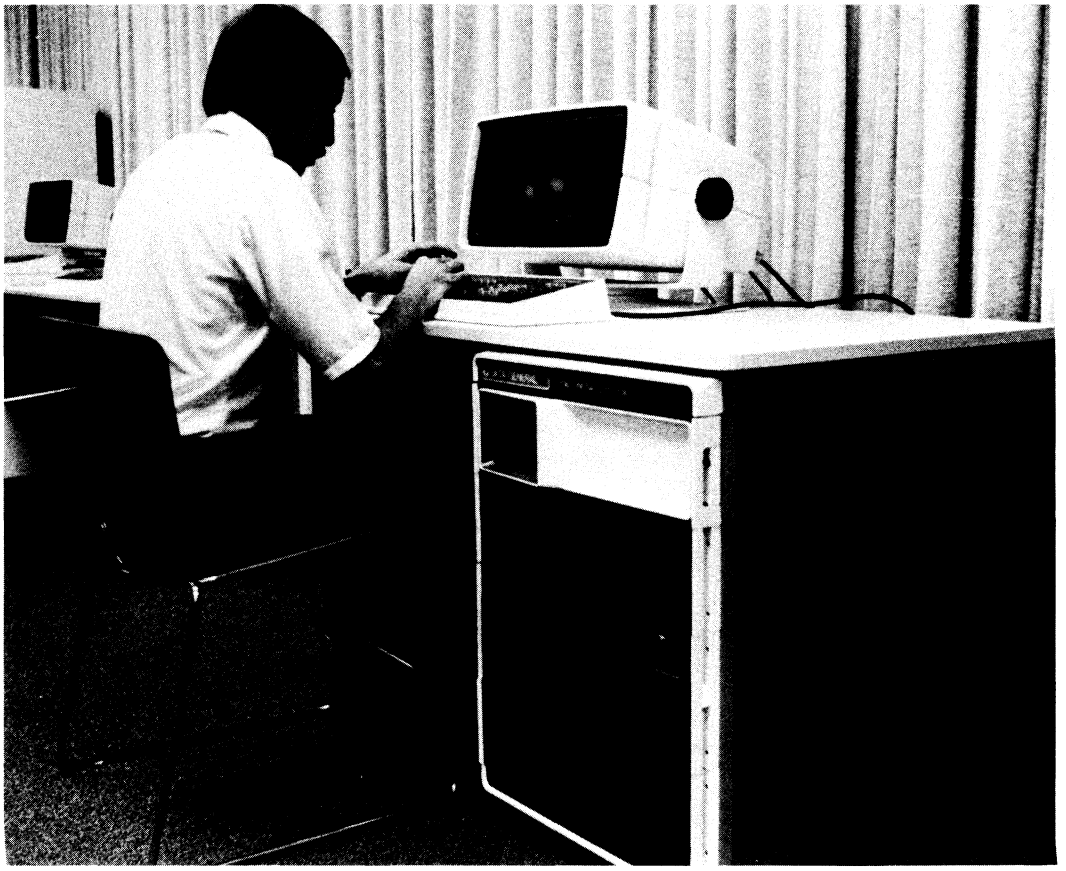
9. Given a CLI error situation:
   a) identify the possible cause of the error;
   b) reference the solution to the error;
   c) enter a CLI command or series of commands to correct the error.

Directions

1. Turn to figure 2-1 in your Student Guide.

2. Listen to the tape for Module Two.





FILE MANAGEMENT

CALL UTILITIES

SYSTEM ENVIRONMENT
MANAGEMENT

PROGRAM MANAGEMENT

CLI FUNCTIONS

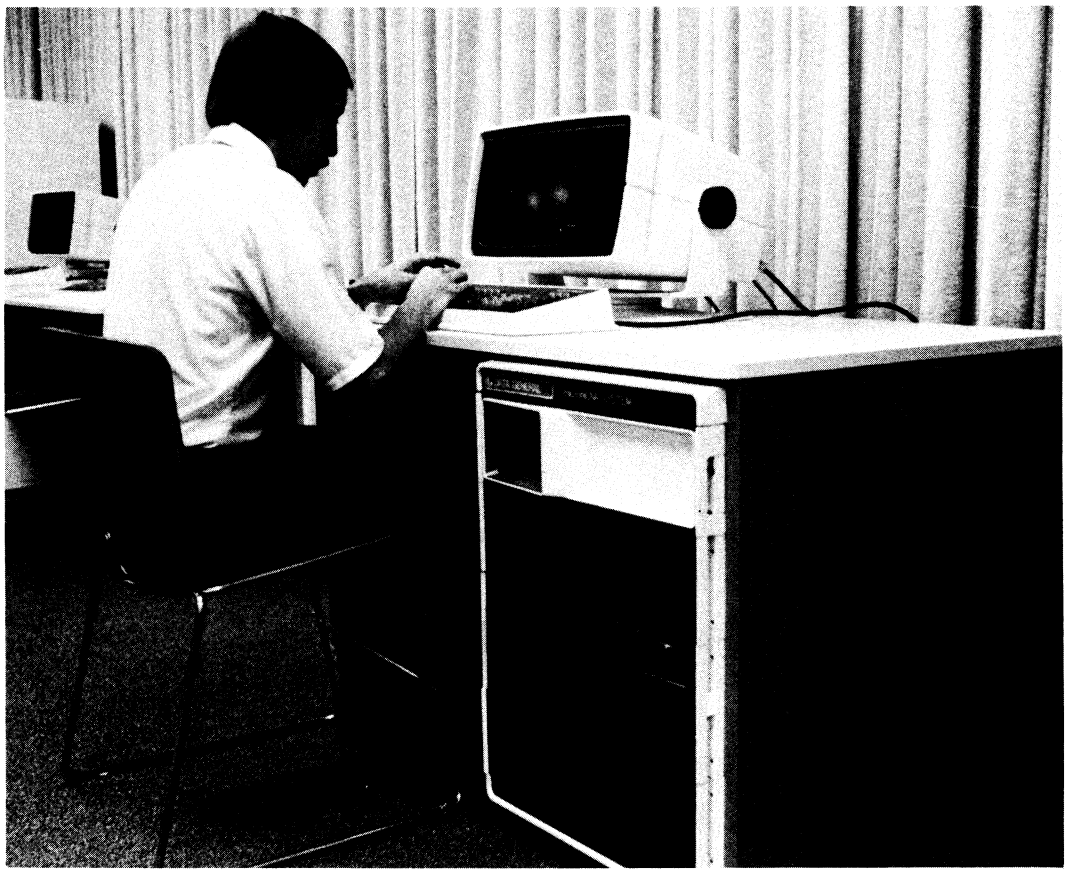
Figure 2-1

FILE MANAGEMENT

CREATE
DELETE
RENAME
COPY
PRINT
DISPLAY
ATTRIBUTES

CLI FILE MANAGEMENT FUNCTIONS

Figure 2-2



MP/OS UTILITIES

PROGRAM DEVELOPMENT

SPEED
MACROASSEMBLER
PASCAL COMPILER
FORTRAN COMPILER
BINDER
DEBUGGER

SYSTEM

DISK INITIALIZER
MOVE
LIBRARY FILE EDITOR

CLI PROGRAM CALLS

Figure 2-3

SYSTEM MANAGEMENT

DIRECTORY
SEARCHLIST
PATHNAME
DEVICE CHARACTERISTICS
SHUTDOWN

PROGRAM MANAGEMENT

CHAIN
SWAP
EXECUTE

CLI SYSTEM & PROGRAM MANAGEMENT

Figure 2-4

HOW TO ENTER COMMANDS

Directions

1. Read the following pages on using the console to communicate with CLI.
2. Try the CLI Command Entry Quiz.
3. Return to the tape for Module 2 after completing the reading and Quiz.

Using Control Characters

When you are working at the console, you may find you want to either change something you have typed or stop what is happening at the console. For example, you can use the DELETE or RUBOUT key (depending on your console) to correct typographical errors. DELETE/RUBOUT erases the character to the left of the cursor. In addition, you can use certain control characters to affect what is happening at the console. Control characters are transmitted when you press the CTRL key and some other key at the same time. We represent them as CTRL-x where x is the other key.

The following are control characters you might find useful:

- CTRL-U erases the current command line. This saves you typing a series of DELETEs if you change your mind about entering a command.

- CTRL-S suspends output to the console. On soft-copy devices it freezes the display of information on the screen; on hard-copy devices it stops the printing of information. You may want to use CTRL-S if the program you are running is generating a lot of output and you want to stop and examine it at some point.

- CTRL-Q cancels the effect of CTRL-S: on soft-copy devices it restarts the display of information on the screen; on hard-copy devices it restarts printing.

  NOTE: *If you type* CTRL-S *by accident, you may think the system is dead because nothing you do will have any effect. When this happens, type* CTRL-Q. *If* CTRL-S *was the cause of the problem,* CTRL-Q *will undo its iffect.*

The CTRL-key is echoed as an up-arrow.

There are also control *sequences* you can use. All control sequences consist of CTRL-C followed by another control character. For example,

- CTRL-C CTRL-A interrupts the current program if the program in question allows interrupts. If it doesn't, the control sequence has no effect.

- CTRL-C CTRL-B immediately terminates the program that is running. If it is the CLI, the CLI is refreshed and re-invoked.

Command Formats

Just as languages have syntactical rules which must be observed to make communication possible, so does the CLI. In other words, you must observe certain rules when entering commands so that you can communicate with the CLI.

Delimiters and Terminators

CLI command lines consist of a command alone or of a command followed by one or more arguments. An argument may consist of a filename, program name, etc. Some commands require arguments, others allow them, and still others do not accept arguments. The command line can be either in upper or lower case since the CLI does not distinguish between them. If the command line has one or more arguments, you must delimit the command and each argument. Delimiters can be:

- One or more spaces (except at the beginning or end of a command line).

- One or more tabs (except at the beginning or end of a command line).

- A single comma.

- Any combination of spaces, tabs, and single commas.

Furthermore, a command is not transmitted to the CLI until you terminate it properly. You can use the following as terminators:

- New line
- Form feed
- Carriage return.

So, for example, depending on which delimiters you used, a command line could look like one of the following three options:

Option one: the space delimiter with the new-line terminator:

```
) CREATE FLOWERS )
)
```

Option two: the tab delimiter with the new-line terminator:

```
) CREATE        FLOWERS )
)
```

Option three: the comma delimiter with the new-line terminator:

```
) CREATE,FLOWERS ↓
)
```

Note that in the three options above, the " ↓ " symbol indicates a new line. The right parenthesis is the CLI prompt.

More examples of command lines are given below. The following three options all perform the same function of renaming the ORIGINAL file to BACKUP.

Option 1 uses the space delimiters and new-line terminator:

```
) RENAME ORIGINAL BACKUP ↓
)
```

Option two uses tab delimiters and the new-line terminator:

```
) RENAME        ORIGINAL        BACKUP↓
)
```

Option three uses the comma delimiter and new-line terminator:

```
) RENAME,ORIGINAL,BACKUP ↓
)
```

Switches

You can modify certain CLI commands by using switches. A switch has the format /SWITCHNAME and may consist of one or more characters. Switches can either be simple or keyword. A simple switch has the format:

/SWITCH

For example, you can modify the COPY command so that the contents of one file are copied and then appended to another file. To do so, you would use the /A switch:

```
) COPY/A TRIG COSINE )
)
```

The DISKSTATUS command can also be modified by appending the byte-length switch /B:

```
) DISKSTATUS/B )
)
```

Note that, in this case, no argument was specified so the system assumes a default argument.

A keyword switch has the format:

/KEYWORD=value

For example, you can create a file and set the element size with a keyword switch /ELEMENTSIZE = 10, as shown below:

```
) CREATE/ELEMENTSIZE=10 ZORP↓
)
```

Be very careful in your spacing of this command. For example, if you insert spaces before and after the equals sign, you will get:

```
) CREATE/ELEMENTSIZE = 10 ZORP↓
Error: Unknown switch specified
CREATE/ELEMENTSIZE,=,10,ZORP
)
```

A commonly used keyword switch is the /L or listing switch. The example below sends the listing to a disc file named BIND.LS as shown:

```
) XEQ BIND/L=BIND.LS  PROG.OB  MSL.LB !
```

Multiple Commands

You can enter more than one CLI command on a line by separating the commands with a semi-colon (;). For example:

```
) DIR FUNDS;DEL ACCT !
)
```

The commands are executed from left to right, so if any command is invalid, neither it nor anything to the right of it will be executed.

In the multiple command example below, you first ask for the rev number of the executable program PROG.PR and then execute the program.

```
) REVISION PROG.PR;XEQ PROG.PR !
```

Continuation Lines

You can continue a command line to the next line by typing an ampersand (&) before the New Line. The CLI issues the prompt "&)" on the continuation line. For example,

```
) DEL MUX SYNCH &↓
&) ASYNCH COMM↓
)
```

There is no limit to the number of continuation lines you can have. Short lines and long lines may be continued.

The continuation line below shows the TYPE command for six files:

```
) TYPE/L PROG_ONE.SR  FORTSUM.SR  TESTFILE.ONE &↓
&) ANOTHER BACKUP.TWO  MORE.SAME ↓
```

Again, spacing is extremely important. For example, if you do not put a space either before or after the continuation, you will get:

```
) DEL MUX SYNCH&
&) ASYNCH COMM
SYNCHASYNCH, Error: File does not exist
)
```

In this case, CLI read the filenames as one, "SYNCASYNCH", thus causing an error.

The key is to space your command lines either before or after the continuation character.

Abbreviations

The CLI allows you to abbreviate all commands and switches. The shortest abbreviation you can use consists of the smallest number of characters, beginning with the first character, which results in a unique specification for a command or switch. So, for example, you can abbreviate FILESTATUS to F since it is the only CLI command which begins with an F. You cannot, however, abbreviate DELETE to D because several commands begin with D. Even DE would be insufficient because of the command DEBUG. In fact, the shortest acceptable abbreviation for DELETE is DEL.

The following is a list of CLI commands and their minimal abbreviations:

COMMAND	ABBREVIATION
attributes	A
boot	BO
bye	BY
chain	CHAI
characteristics	CHAR
copy	CO
create	CR
Date	DA
Debug	DEB
delete	DEL
directory	DIR
diskstatus	DISK
dismount	DISM
execute	E
filestatus	F
help	H
information	I
message	ME
mount	MO
pathname	P
rename	REN
revision	REV
searchlist	S
time	TI
type	TY
write	W
XEQ	X

Parentheses

Parentheses in a command line result in command repetition. If you enter a command followed by an argument list in parentheses, the CLI executes the arguments in the list as if each argument were entered on a separate line. For example, the command lines in option one and two below are equivalent.

Option 1:

```
> TYPE (X Y Z)↓
```

Option 2:

```
> TYPE X↓
> TYPE Y↓
> TYPE Z↓
```

Both options result in a display of the contents of files "X", "Y", and "Z".

Another example of parentheses is given below. The two options perform the same function:

```
) DELETE (TESTFILE  BACKUP ONE  ORIGINAL)↓
```

```
) DELETE TESTFILE↓
) DELETE BACKUP.ONE↓
) DELETE ORIGINAL↓
)
```

If a subset of the entire argument list is in parentheses, the command is repeated for each argument in the subset in conjunction with the remaining argument(s). For example, the command lines in options one and two below are equivalent. They both result in files APPLES and DOUGH being copied to file PIE.

Option 1:

```
) COPY PIE (APPLES DOUGH)↓
)
```

Option 2:

```
) COPY PIE APPLES↓
) COPY PIE DOUGH↓
)
```

If you enter a command line with two or more argument groups in parentheses, the CLI executes the command for the first argument in each group, then for the second, and so on. For example, the command lines in options one and two are equivalent. They both result in TAX being copied to SALARY and GLOP being copied to ZORP.

Option 1:

```
) COPY (SALARY ZORP) (TAX GLOP)↓
)
```

Option 2:

```
) COPY SALARY TAX↓
) COPY ZORP  GLOP↓
)
```

If you enter a command line with two or more commands in parentheses, followed by an argument, each command in the parentheses is executed with the argument. For example, the command lines in options one and two below are equivalent. File **TEST.BU** will be displayed and then deleted.

Option 1:

```
) (TYPE DELETE) TEST.BU↓
```

Option 2:

```
) TYPE TEST.BU↓

) DELETE TEST.BU↓
```

Another example of two commands in parentheses is given below. Both options perform the same functions:

Option 1:

```
) (REVISION XEQ ) FORTSUM.PR )
```

Option 2:

```
) REVISION  FORTSUM.PR
) XEQ FORTSUM.PR
```

The REVISION command requests the rev number of the FORTSUM.PR program. XEQ requests the execution of FORTSUM.

Angle Brackets

Angle brackets in a command line result in *argument expansion.* They can help you code arguments that contain the same character or character combination. The CLI forms arguments by joining each character enclosed within angle brackets with the characters that appear immediately before the left angle bracket and immediately after the right angle bracket. For example, the command lines in options one and two below are equivalent. They both result in the deletion of the three files named X.1, X.2, and X.3.

Option 1:

```
> DELETE X.<1 2 3>↓
>
```

Option 2:

```
> DELETE X.1↓
> DELETE X.2↓
> DELETE X.3↓
>
```

Another example of angle brackets is shown below. Options 1 and 2 below perform the same functions:

Option 1

```
) TYPE  PROG.<SR  FR  BU> ↓
```

Option 2

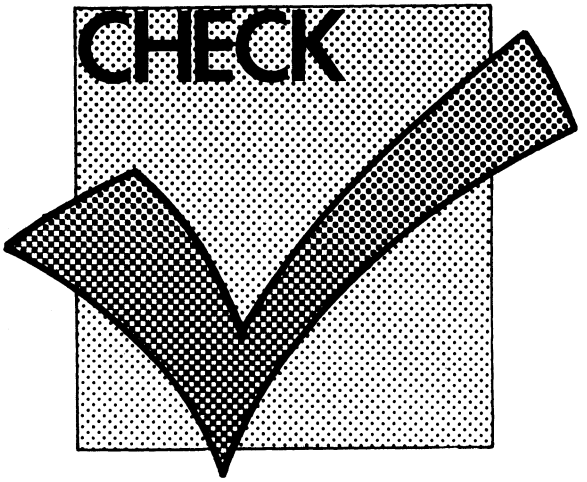
```
) TYPE PROG.SR ↓
) TYPE PROG.FR ↓
) TYPE  PROG.BU ↓
```

The result is the display of the three PROG files.

Order of Evaluation

The CLI allows you to nest angle brackets within angle brackets, parentheses within parentheses, angle brackets within parentheses, and parentheses within angle brackets. The CLI first expands the argument list by processing angle brackets from left to right and from inner to outer. When no angle brackets are left in the command line, the CLI processes parentheses from left to right in pairs.

Check your progress by trying the Quiz on the next page.



CLI COMMAND ENTRY

QUIZ

Directions:

Answer the following questions by writing in the appropriate response in the space provided.

1. You have entered the command line shown below, without the terminator:

```
) DELETE X.1
```

State two methods for deleting the command without executing it.

A. _____

B. _____

2. You have just entered the command line to display a very long file, as shown below.

```
) TYPE LONGFILE )
)
```

How would you stop the display scroll so that you can examine the output?

A. _____

How would you re-start the display after you have stopped it?

B. _____

3. What effect does the control sequence ↑C↑B have on programs?

   A. _____

   What effect does ↑C↑B have on CLI?

   B. _____

4. Given the command line below:

```
) TYPE/L ORDER.1 ↓
)
```

   A. What is the delimiter? _____
   B. What is the terminator? _____
   C. What is the switch? _____
   D. What is the argument? _____

5. Given the three commands below:

```
) TYPE PROD ↓

) COPY MASTER PROD ↓
) DELETE PROD ↓
)
```

   Combine the three commands on one command line:

) _____

6. Given the three commands below:

```
) TYPE PAGE ⍎
) TYPE BOOK ⍎
) TYPE CHAPTER ⍎
```

Use the continuation symbol and combine the commands into one line with a continuation. Write your answer in the blank screen below:

```
) _____
  _____
```

7. Given the three commands below:

```
) TYPE/L A1 ⍎
) TYPE/L B2 ⍎
) TYPE/L A3 ⍎
)
```

Use parentheses to combine the operation into one command. Use the blank screen below:

```
) _____
```

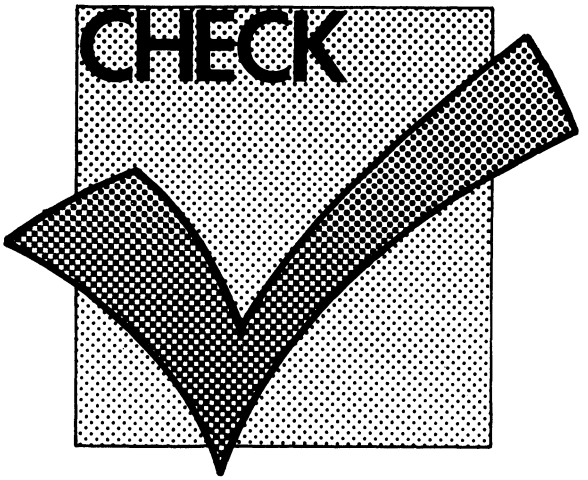
8. Given the commands below:

```
) TYPE/L TEST.1 ↓
) TYPE/L TEST.BU ↓
) TYPE/L TEST.SR ↓
)
```

Use angle brackets to combine the operations into one command. Use the blank screen below for your answer.

```
) _____
```

Check your answers against those on the following pages.



CLI COMMAND ENTRY

QUIZ ANSWERS

1. The command line may be deleted by:
   A. repeatedly pressing the DELETE/RUBOUT key until the command is erased from the screen.
   B. pressing the CONTROL key and U-key simultaneously. (CTRL–U)

2. A. Press CONTROL and S to freeze the screen display.
   B. Press CONTROL and Q to restart the frozen display.

3. A ↑C↑B terminates programs and returns control to CLI.
   B. ↑C↑B refreshes CLI. That is, the disc copy of CLI is written over the copy currently in memory. The working directory is re-set to the root.

4. A. The delimiter is a space (between "/L" and ORDER.1).
   B. The terminator is " ⏎ " (the new-line)
   C. The switch is "/L" (list to the LPT)
   D. The argument is the filename "ORDER.1".

5. The three commands combined on one line are:

```
> TYPE PROG; COPY MASTER PROD; DELETE PROD ⏎
```

The key to this answer is the semicolon command separator which allows multiple commands on one line. Other combinations are possible.

6. Combining the three TYPE commands and adding the continuation symbol yield:

```
) TYPE  PAGE  BOOK &)
&) CHAPTER)
```

Note that the system provides the & continuation symbol.

Remember that MP/OS prompts with an ampersand and right parentheses on the continued line.

7. Combining three TYPE commands with parentheses yields:

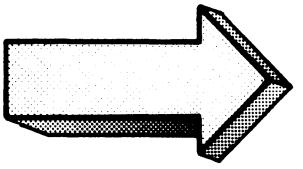
```
) TYPE/L (A1  B2  A3))
```

8. Combining the TYPE command with angle brackets produces the following

```
) TYPE/L  TEST.<1  BU  SR>)
```

If you are satisfied with your progress then continue with the next segment of Module Two.

A SCORE OF 7 CORRECT OUT OF THE 8 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN YOU UNDERSTAND THE CORRECT ANSWER. THEN CONTINUE WITH THE NEXT SEGMENT OF MODULE TWO.



CLI COMMANDS

Abstract

This segment of Module Two covers fifteen basic CLI commands which allow the manipulation of the directory structure, file management, and system monitoring.

Objectives

Upon completion of this segment, you will be able to:

Given the list of CLI commands below,

1. State the function of each command

2. Use each command to solve an appropriate processing problem.

3. Solve an error situation involving selected commands. The commands covered in this segment are:

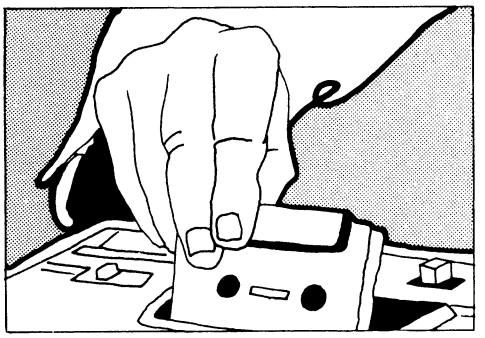
attributes	date	filestatus	time
bye	delete	mount	type
copy	directory	pathname	XEQ
create	dismount	rename	

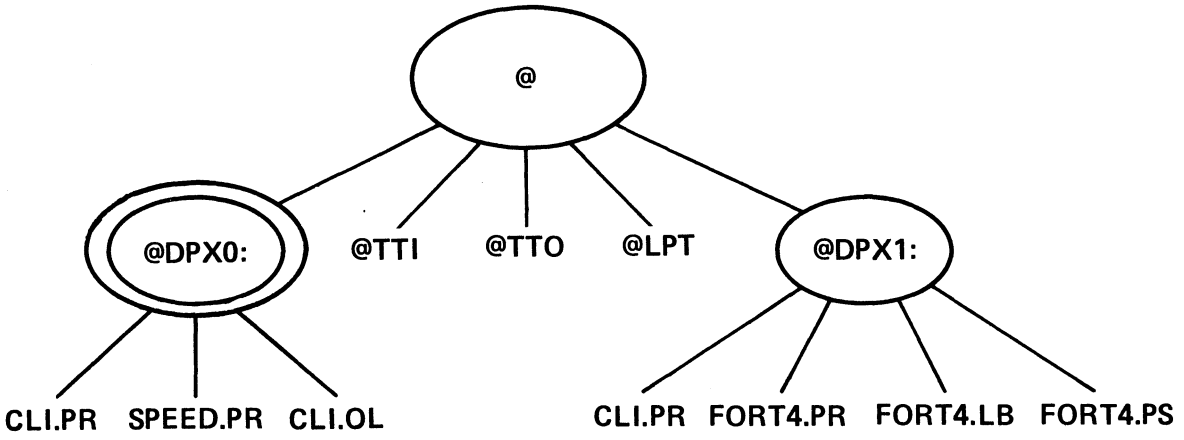
Directions

1. Listen to the tape for this segment of Module Two.

2. Try the CLI COMMANDS QUIZ

3. Do the CLI COMMANDS Lab Exercise.

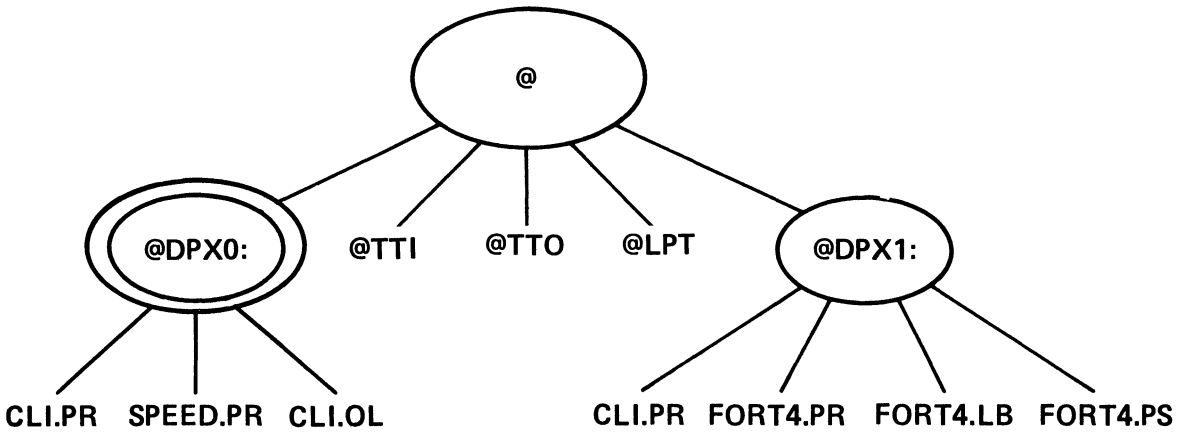





```
MP/OS   CLI   REV 1.0
)
```

@ = device directory
@DPX0 = system master device
@DPX0. root directory for primary diskette
@DPX1 = secondary diskette
@DPX1. = root directory for secondary diskette

Figure 2-5

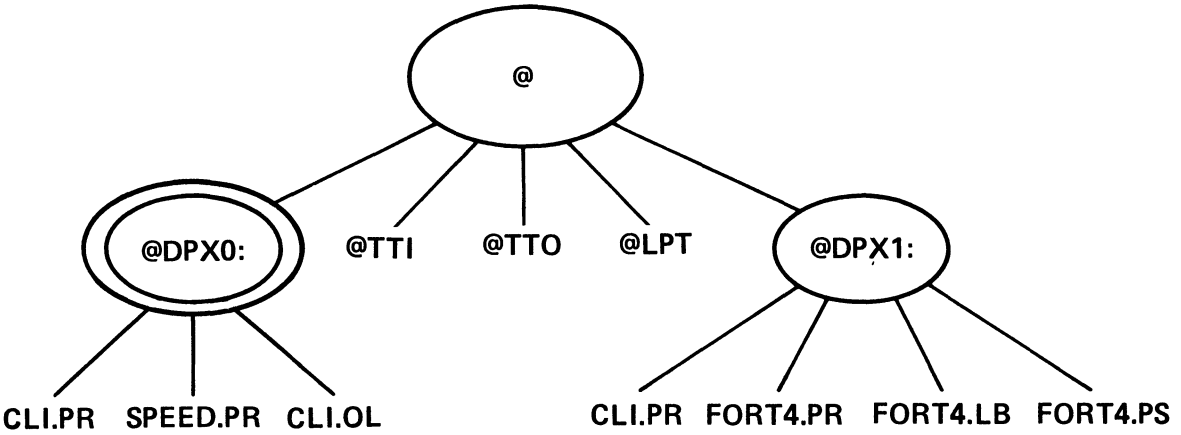


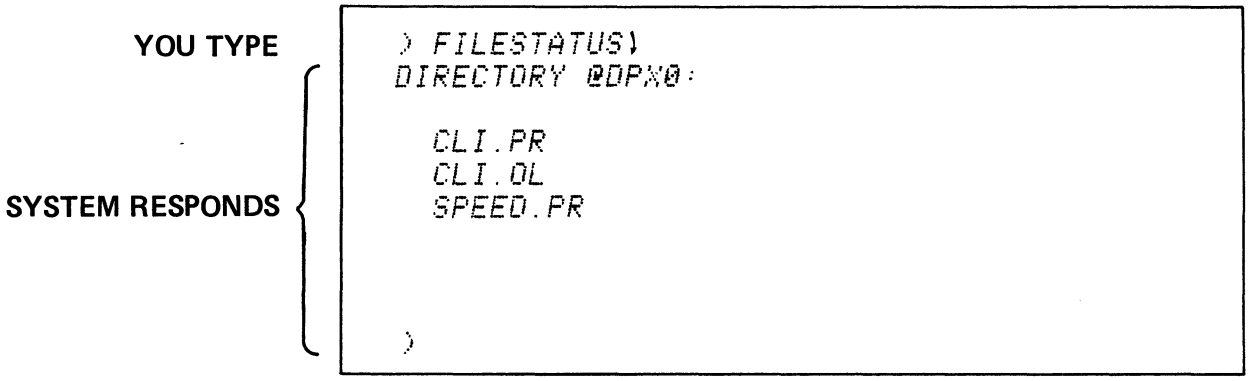
YOU TYPE
SYSTEM RESPONDS

```
) DIRECTORY )
@DPX0:
)
```

DIRECTORY COMMAND — DISPLAYS THE CURRENT WORKING DIRECTORY.

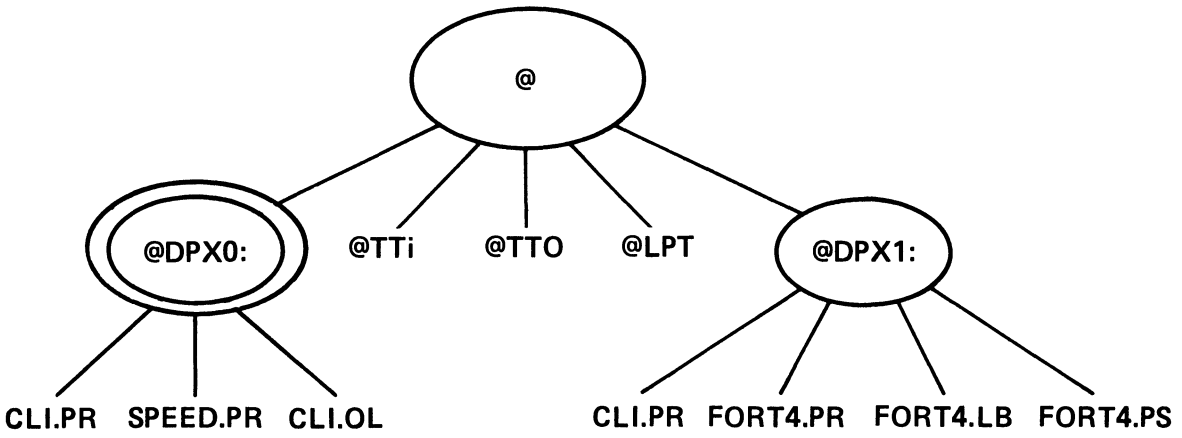
Figure 2-6





FILESTATUS COMMAND – DISPLAY THE STATUS OF FILES

Figure 2-7



YOU TYPE

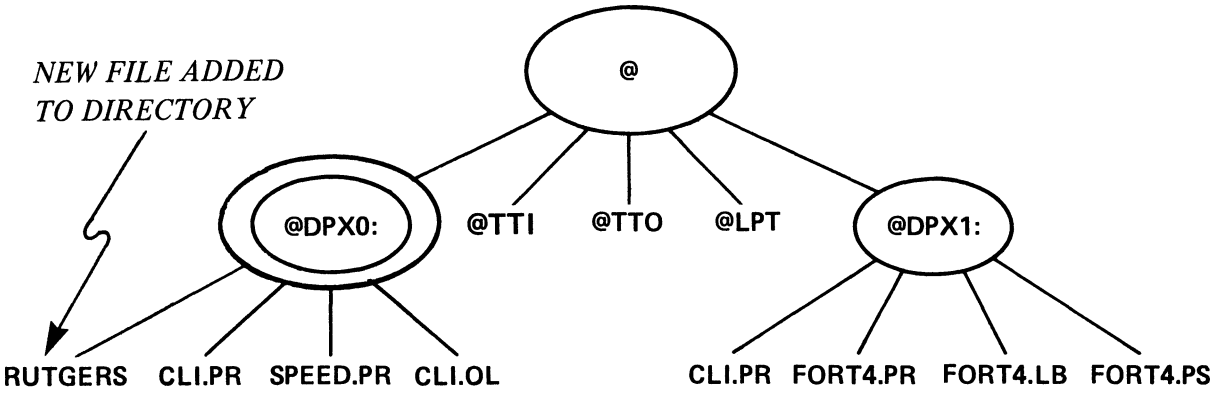
SYSTEM RESPONDS

```
) FILESTATUS/ASSORTMENT)
DIRECTORY @DPX0:

  CLI.PR          PRG  24-JAN-79   5:34:13      34304
  CLI.OL          OLF  24-JAN-79   5:34:13      34816
  SPEED.PR        PRG  24-JAN-79   5:34:13      13312
)
```

FILESTATUS/SWITCH [OPTIONAL PATHNAME]

ASSORTMENT -- file's Type, Time & Date of last modification, length in bytes.

Figure 2-8

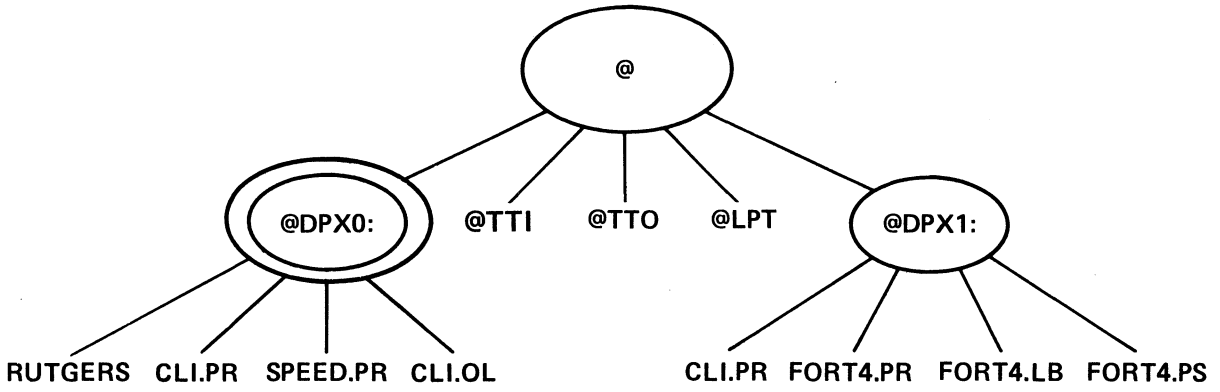


YOU TYPE

```
) CREATE RUTGERS )
)
```

CREATE COMMAND -- CREATE A FILE IN THE CURRENT WORKING DIRECTORY.

Figure 2-9

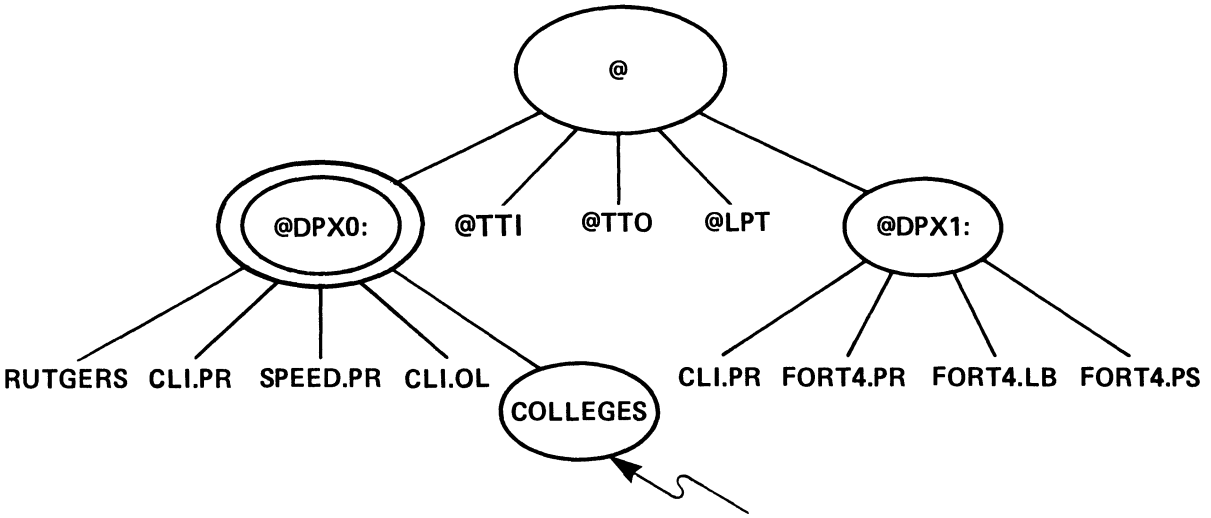


YOU TYPE

SYSTEM RESPONDS

```
) FI/AS RUTGERS)
DIRECTORY @DPX0:

  RUTGERS              TXT    1-JAN-00    0:31:30        0
)
```

Figure 2-10



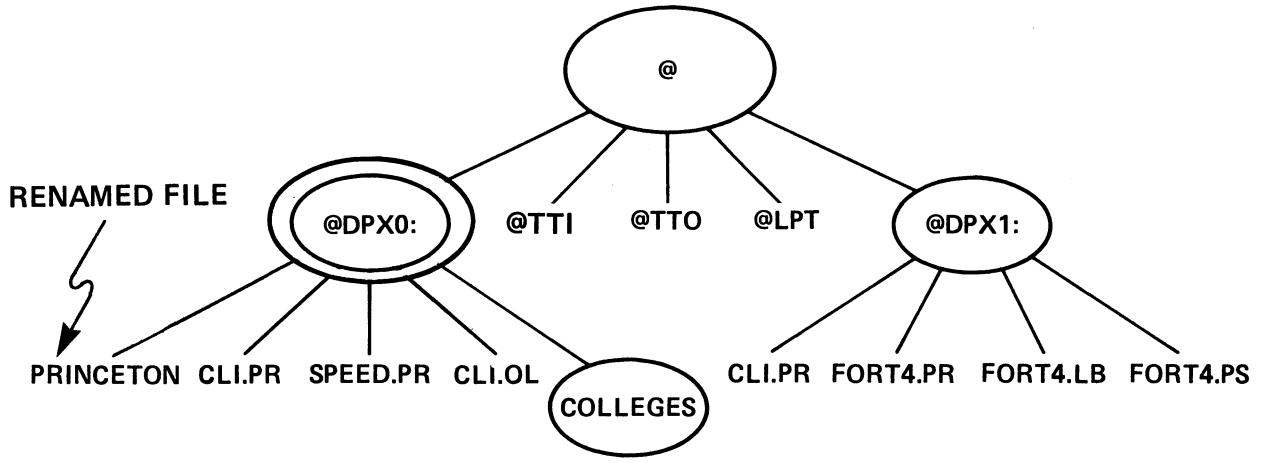
YOU TYPE

```
) CRE/DIR COLLEGES)
)
```

CREATE/SWITCH

/DIR – CREATE A DIRECTORY FILE AS A SUBDIRECTORY OF THE CURRENT WORKING DIRECTORY.

Figure 2-11

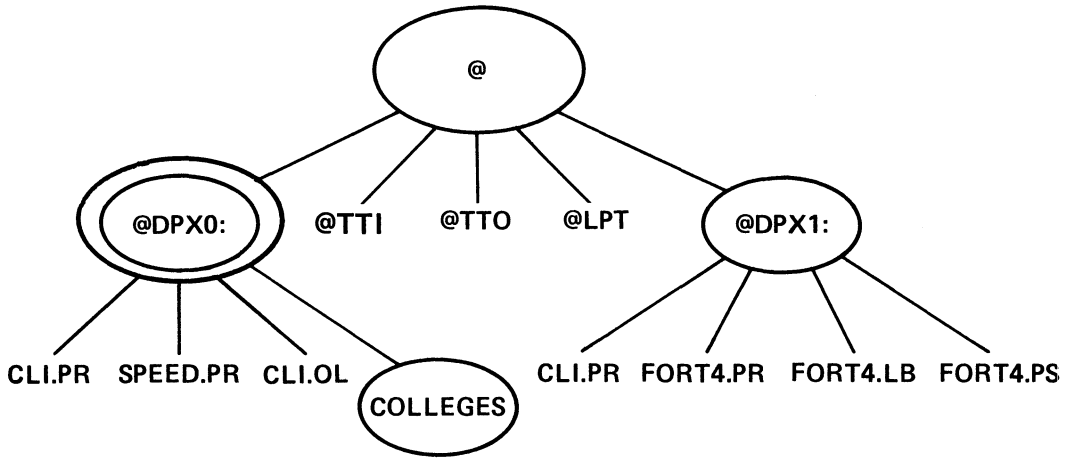


YOU TYPE
YOU TYPE
SYSTEM RESPONDS

```
) RENAME RUTGERS PRINCETON )
) FI/SORT )
DIRECTORY @DPX0:
   CLI.OL
   CLI.PR
   COLLEGES
   PRINCETON
   SPEED.PR
)
```

RENAME COMMAND – CHANGE A FILE'S NAME.

Figure 2-12



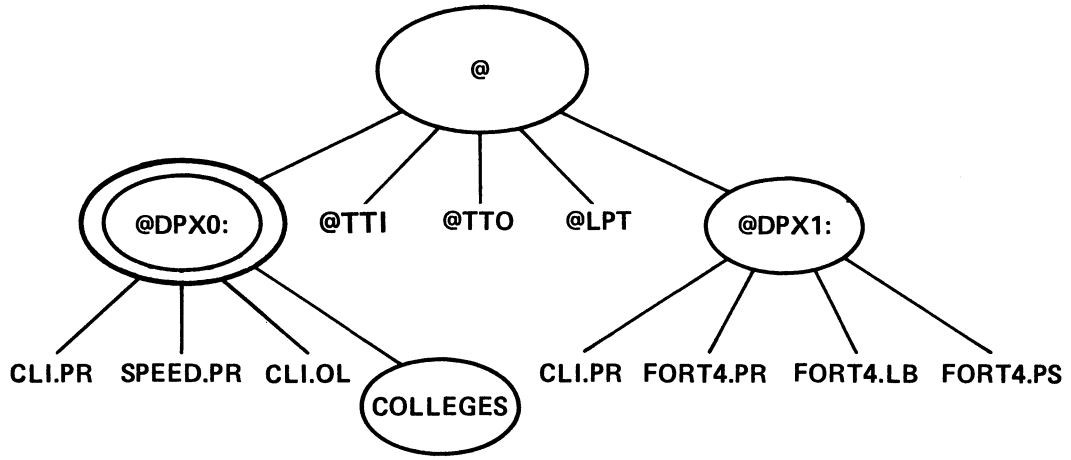
YOU TYPE

SYSTEM RESPONDS

```
) DELETE/V PRINCETON↓
Deleted PRINCETON
)
```

DELETE COMMAND -- REMOVE A FILE.

Figure 2-13



YOU TYPE
SYSTEM ASKS
YOU TYPE
SYSTEM RESPONDS

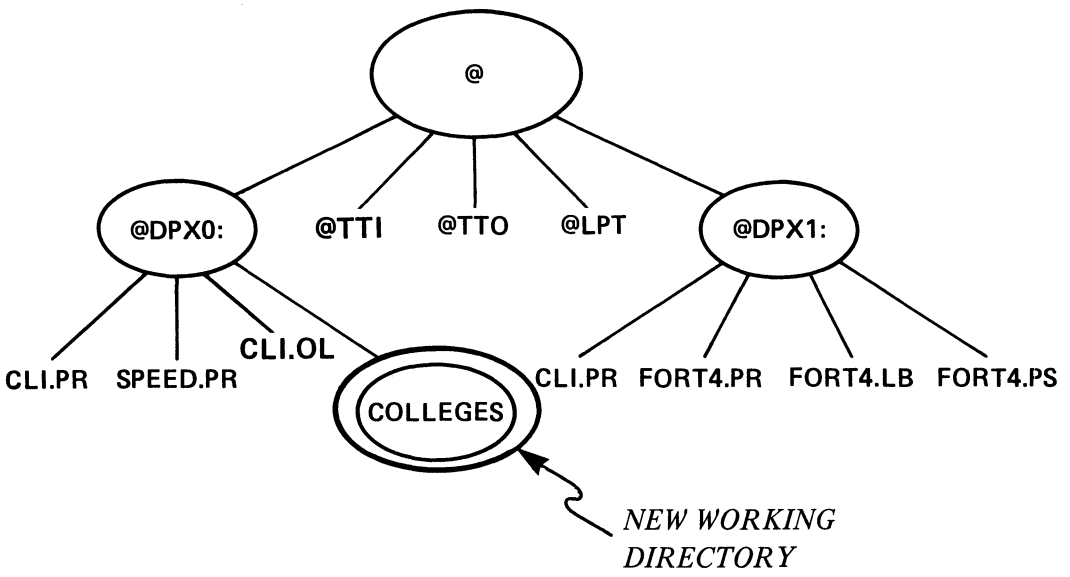
```
) DELETE/V/C PRINCETON )
PRINCETON ?YES )
Deleted PRINCETON
)
```

DELETE/SWITCH

/V = VERIFY WHEN FILE IS DELETED

/C = CONFIRM THAT FILE IS TO BE DELETED.

Figure 2-14



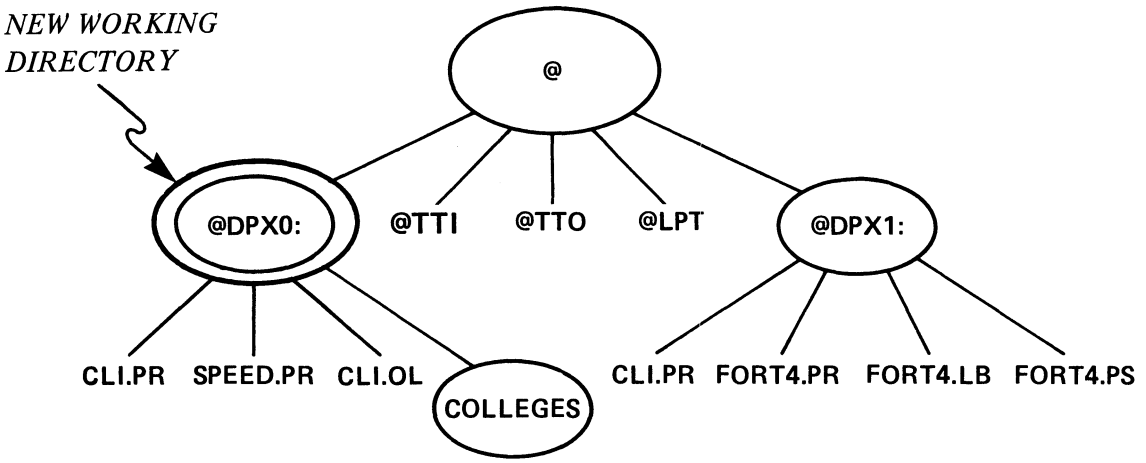
YOU TYPE
YOU TYPE
SYSTEM RESPONDS

```
) DIR COLLEGES )
) DIRECTORY )
@DPX0:COLLEGES
)
```

DIRECTORY PATHNAME ARGUMENT

MAKE SPECIFIED PATHNAME THE NEW WORKING DIRECTORY

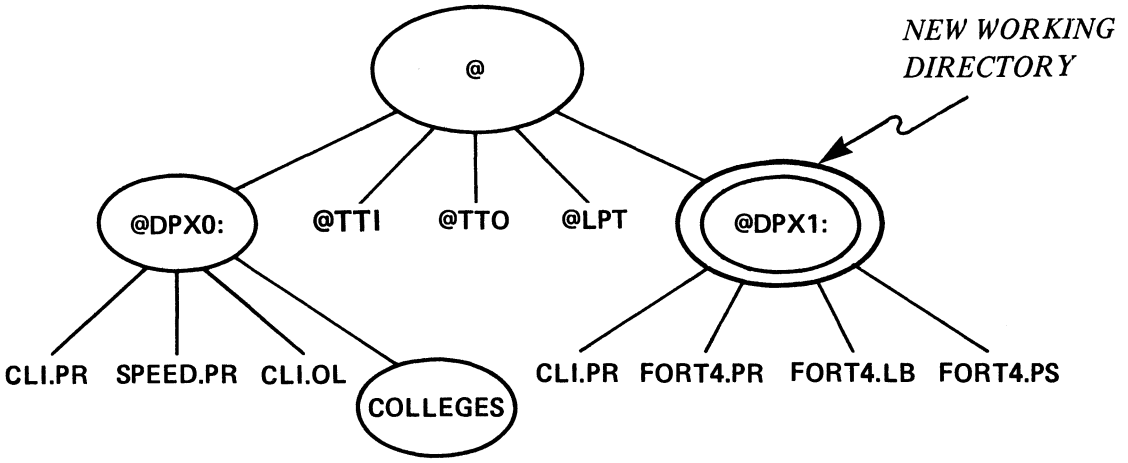
Figure 2-15

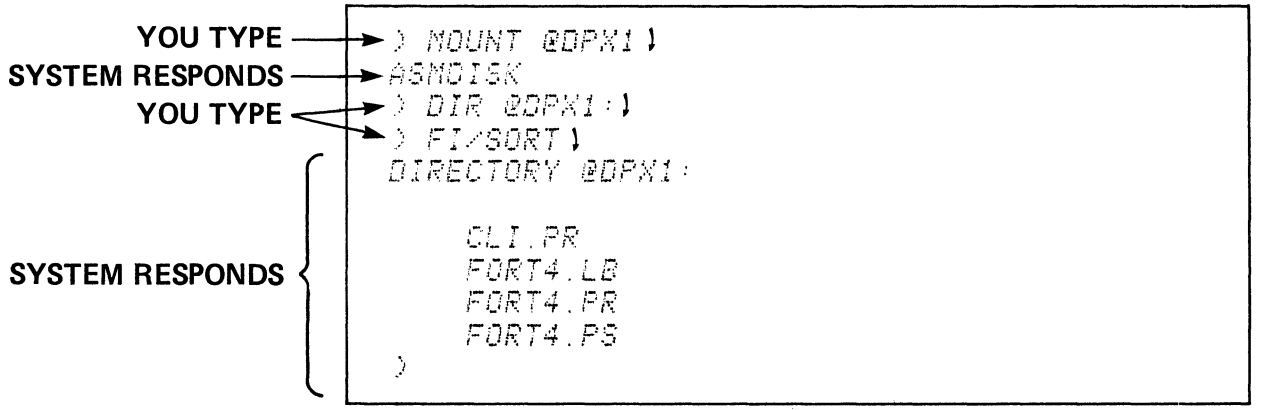


SYSTEM RESPONDS
YOU TYPE

```
) DIR ^)
) DIR )
@DPX0:
)
```

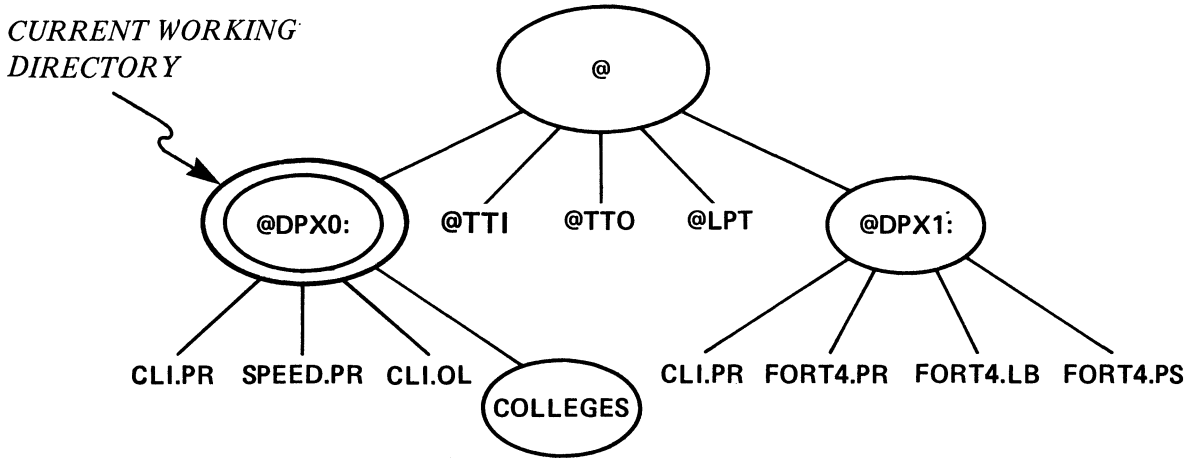
Figure 2-16





MOUNT DEVICENAME – MAKE DIRECTORY DEVICE (DISC, DISKETTE) ACCESSIBLE FOR I/O.

Figure 2-17

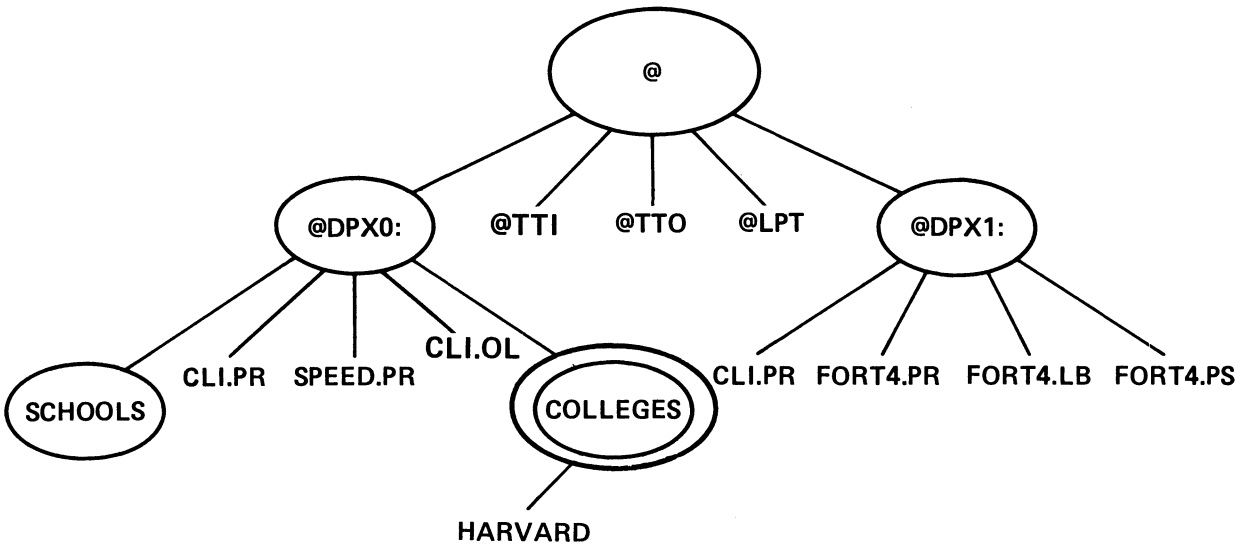


YOU TYPE

```
) DIR @DPX0:)
) DISMOUNT @DPX1)
)
```

DISMOUNT DEVICENAME – PREPARE DISK OR DISKETTE FOR REMOVAL, BRING IT OFF-LINE.

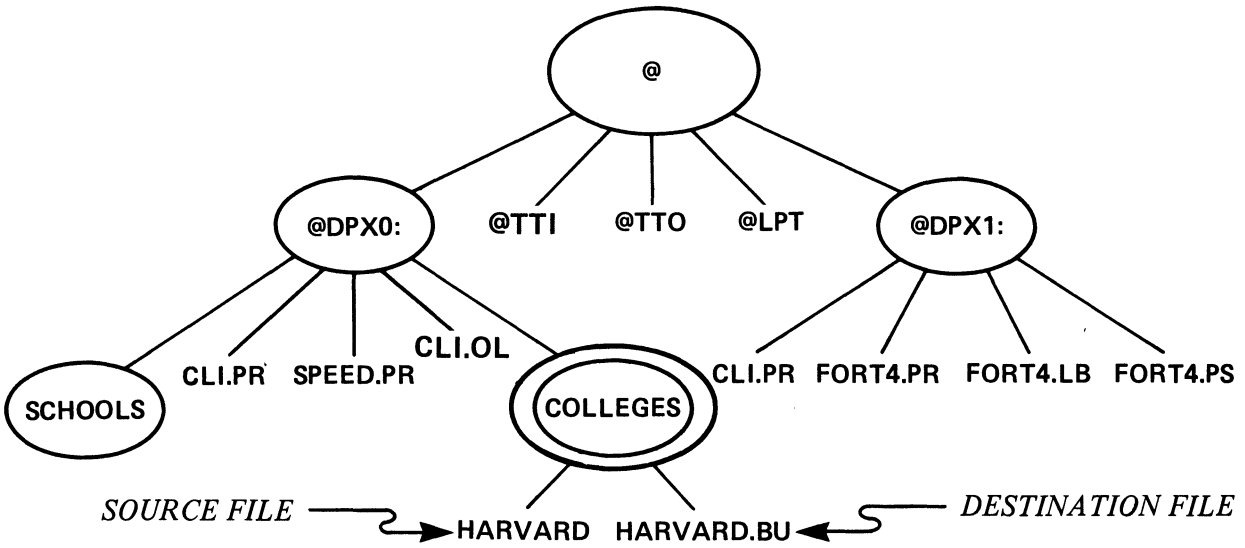
Figure 2-18



YOU TYPE

```
) CREATE/DIR SCHOOLS ↓
) DIR COLLEGES ↓
) CREATE HARVARD ↓
)
```

Figure 2-19



YOU TYPE

```
) COPY HARVARD.BU HARVARD )
) TYPE/L HARVARD HARVARD.BU )
)
```

COPY TOFILE FROMFILE – COPY ONE OR MORE FILES TO A DESTINATION FILE.

TYPE – DISPLAY CONTENTS OF FILE(S).

Figure 2-20

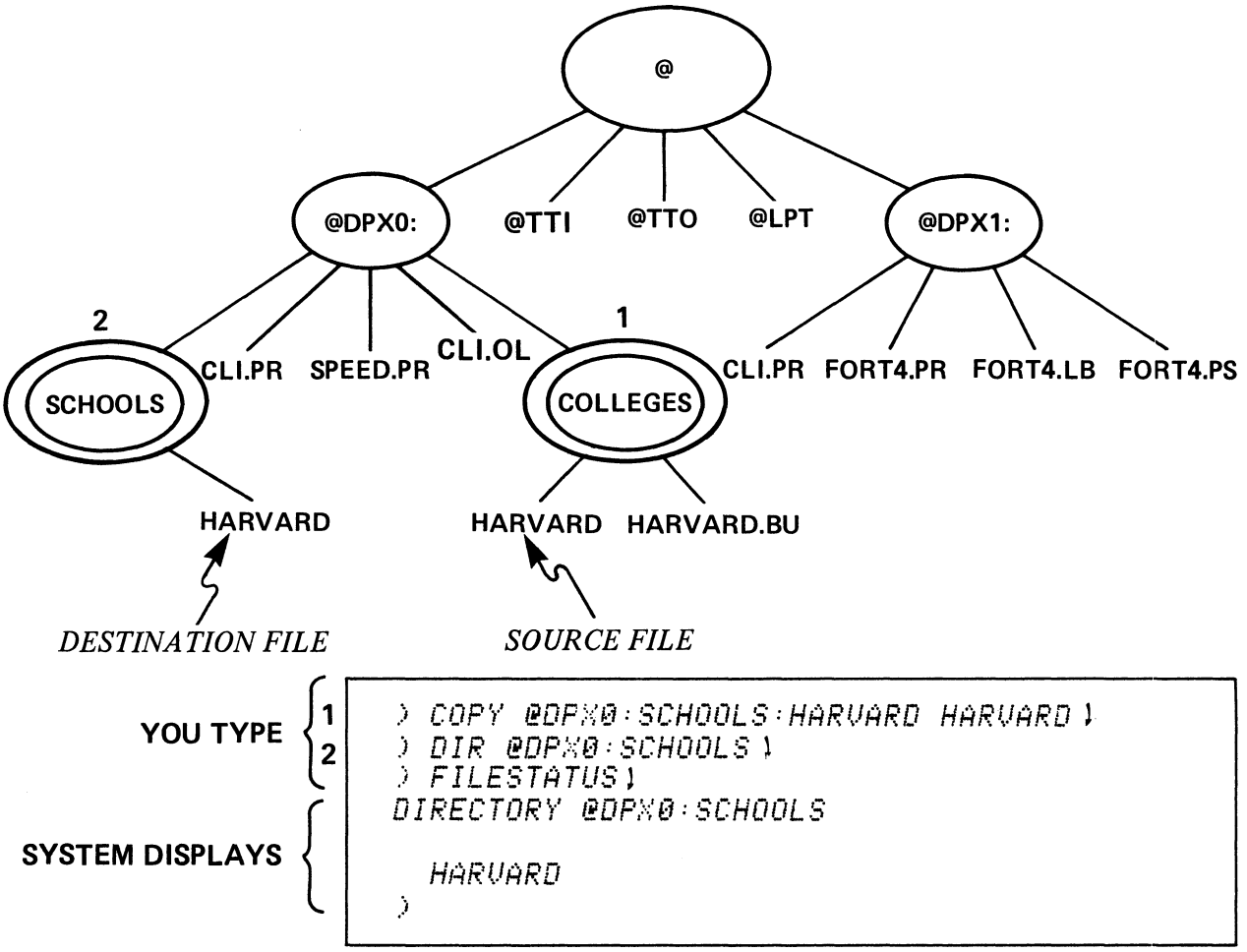
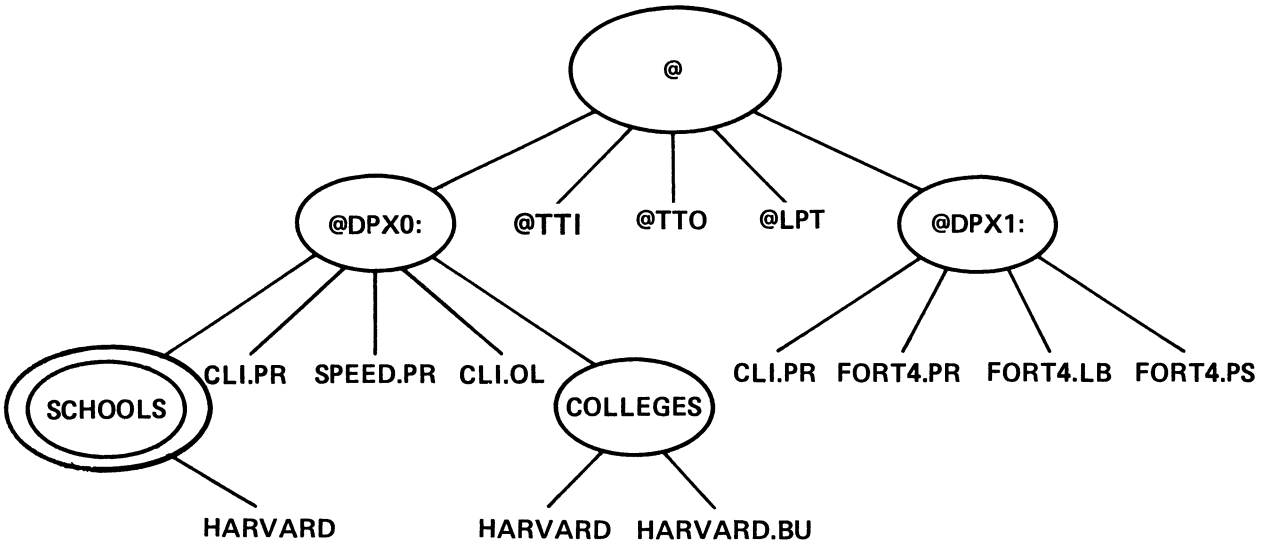

@
@DPX0:
@TTI
@TTO
@LPT
@DPX1:
2
1
SCHOOLS
CLI.PR
SPEED.PR
CLI.OL
COLLEGES
CLI.PR
FORT4.PR
FORT4.LB
FORT4.PS
HARVARD
HARVARD
HARVARD.BU
DESTINATION FILE
SOURCE FILE
YOU TYPE
1
2
SYSTEM DISPLAYS
) COPY @DPX0:SCHOOLS:HARVARD HARVARD↓
) DIR @DPX0:SCHOOLS↓
) FILESTATUS↓
DIRECTORY @DPX0:SCHOOLS
HARVARD
)

Figure 2-21



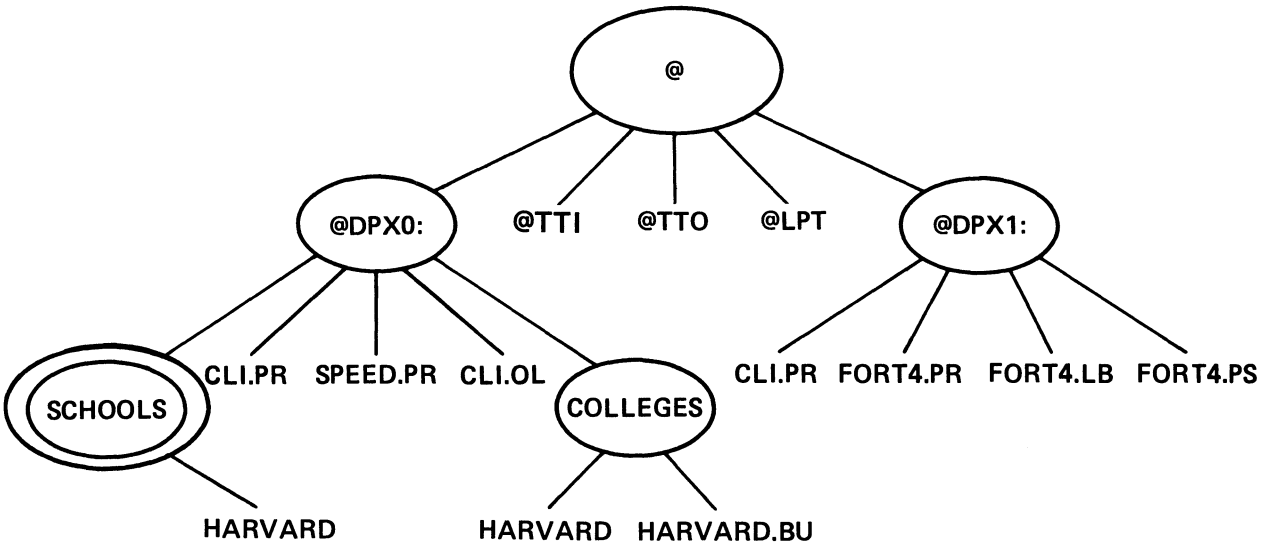
YOU TYPE
SYSTEM DISPLAYS
YOU TYPE
SYSTEM DISPLAYS

```
) ATTRIBUTES HARVARD↓
HARVARD No attribute protection
) A HARVARD W R↓
) A HARVARD↓
HARVARD R W
)
```

ATTRIBUTES FILENAME – DISPLAY A FILE'S ATTRIBUTES

ATTRIBUTES FILENAME W R P A – ASSIGN ATTRIBUTES TO FILENAME

Figure 2-22



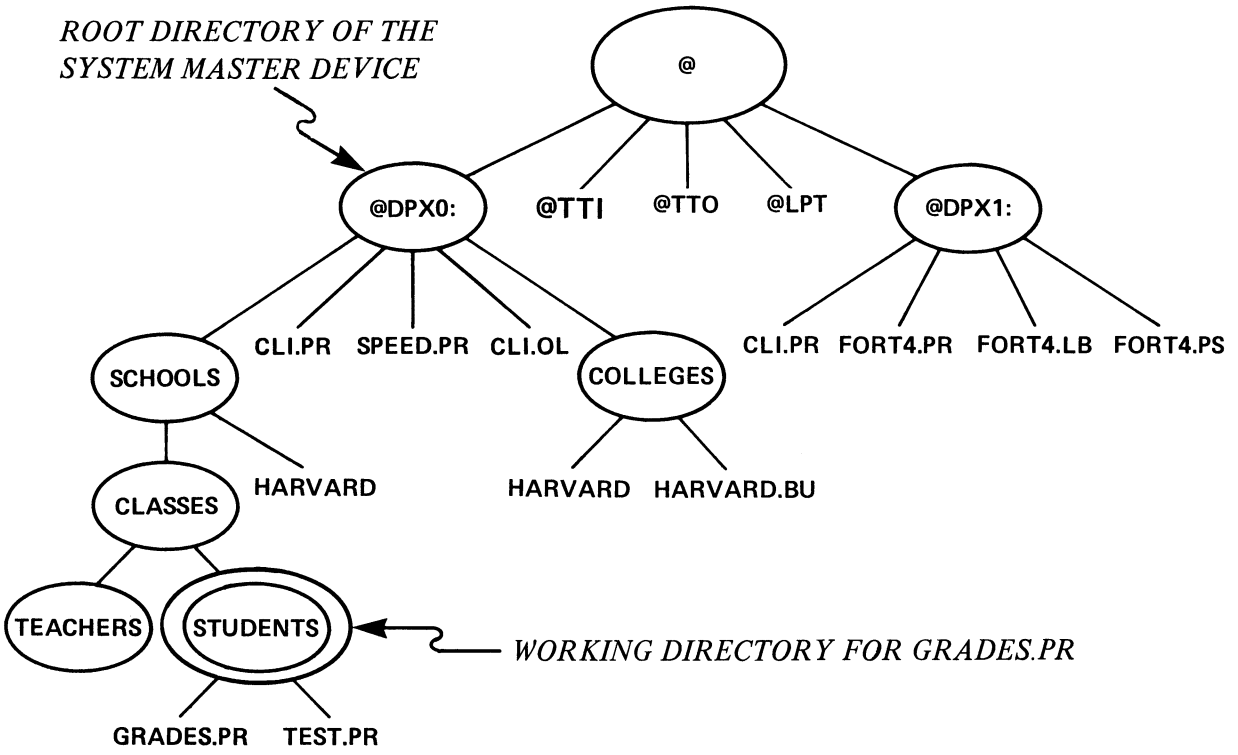
YOU TYPE

```
) ATTRIBUTES HARVARD P↓
) ATTRIBUTES HARVARD↓
```

SYSTEM DISPLAYS

```
HARVARD P
)
```

Figure 2-23



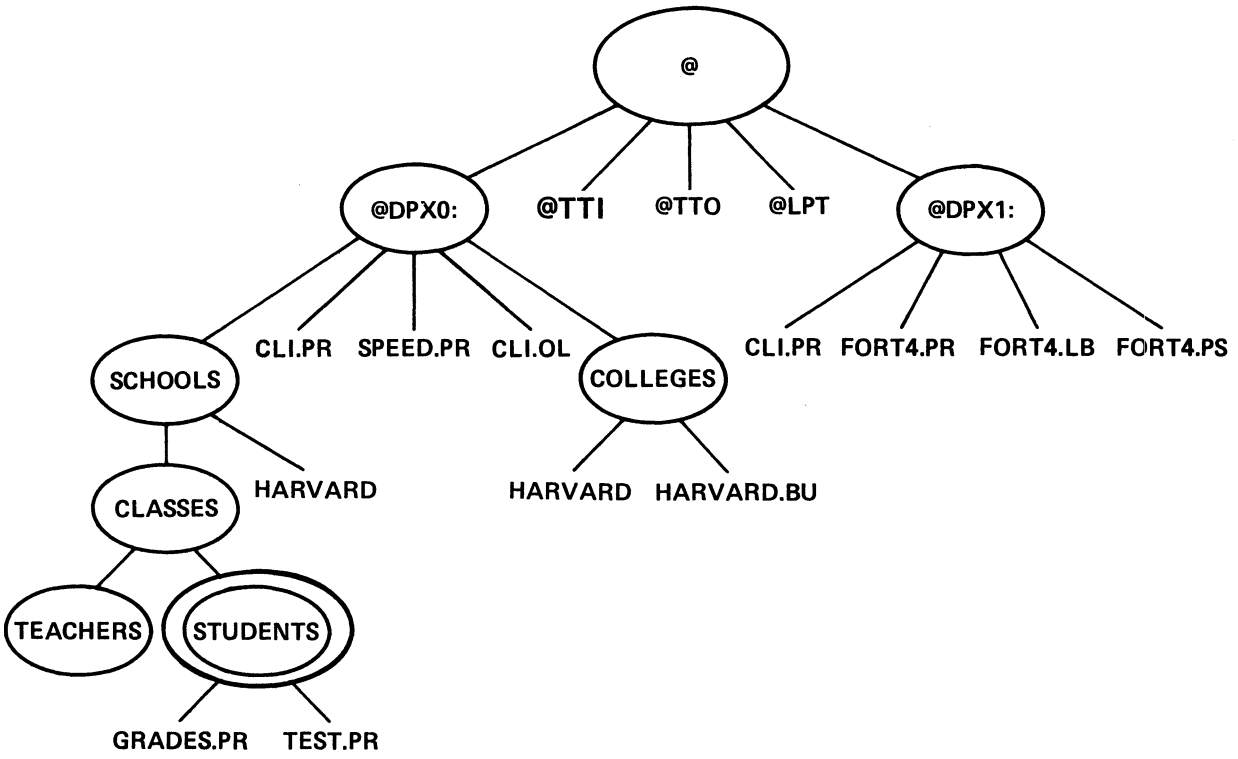
YOU TYPE

```
) DIR @DPX0:SCHOOLS:CLASSES:STUDENTS ↓
) XEQ GRADES.PR ↓
```

YOU TYPE

```
) DIR :SCHOOLS:CLASSES:STUDENTS ↓
) XEQ GRADES ↓
```

Figure 2-24

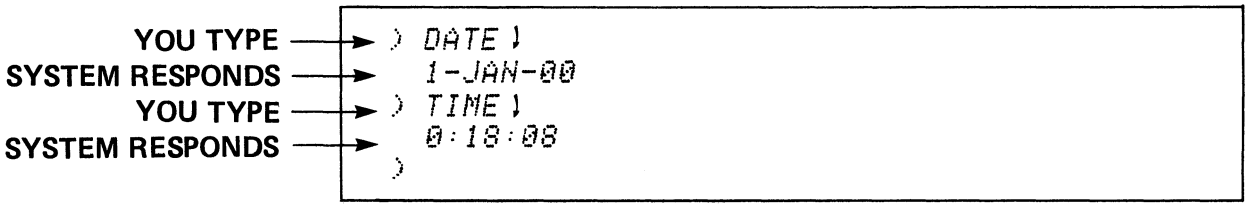


YOU TYPE
SYSTEM DISPLAYS

```
) PATHNAME STUDENTS )
@DPX0:SCHOOLS:CLASSES:STUDENTS
)
```

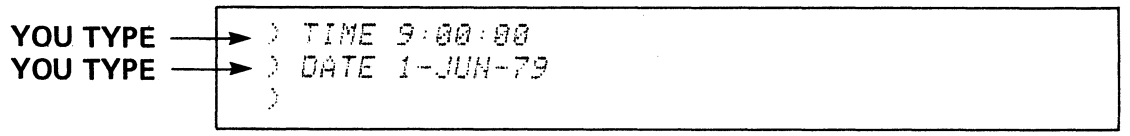
PATHNAME FILENAME – DISPLAY A COMPLETE PATHNAME

Figure 2-25



DATE – SET OR DISPLAY THE SYSTEM DATE.
TIME – SET OR DISPLAY THE SYSTEM TIME.

Figure 2-26



NOTE THE 24 HOUR CLOCK.

Figure 2-27

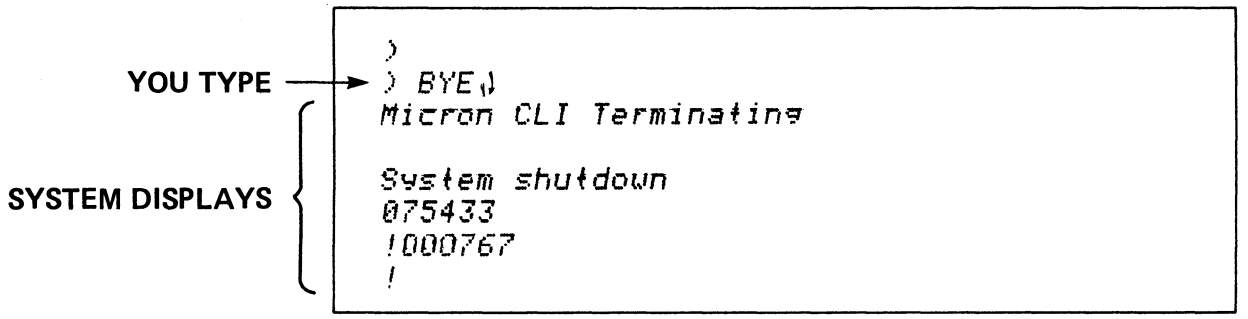
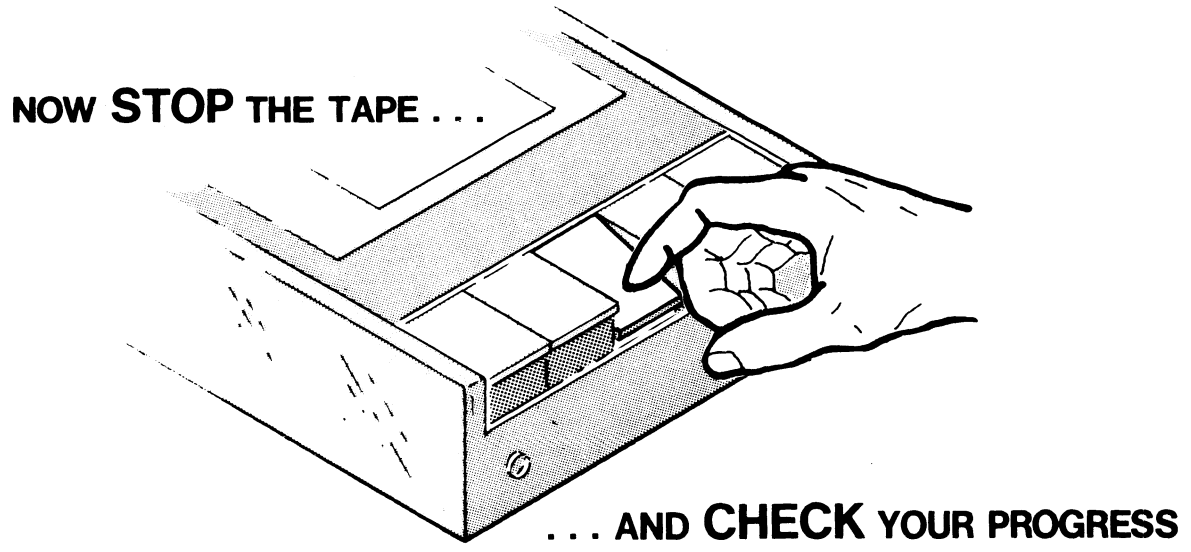


Figure 2-28

COMMAND	COMMON ABBREVIATION	FUNCTION
ATTRIBUTES	AT	set or display a file's attributes
BYE	BY	terminate CLI. Shutdown system.
COPY	CO	copy one or more files to a destination
CREATE	CRE	create file a directory file or non-directory file
DATE	DA	set or display the current system date
DELETE	DEL	delete one or more files.
DIRECTORY	DIR	set or display the current working directory.
DISMOUNT	DIS	prepare a specified disk for removal.
FILESTATUS	Fi	list file status information
MOUNT	MO	make directory device accessible for I/O.
PATHNAME	Path	display a complete, fully qualified, pathname.
RENAME	REN	change a file's name.
TIME	Ti	set or display the current system time.
TYPE	Ty	type the contents of one or more files.
XEQ	X	execute a program.



CLI COMMANDS

QUIZ

Circle the letter that best answers the question.

1. The entry that displays the length, type, and date of last modification of all files in the working directory is:

   A. ATTRIBUTES ⤵

   B. FILESTATUS ⤵

   C. FILESTATUS/ALL ⤵

   D. Fi/AS ⤵

   E. ATT/ALL ⤵

2. The command for erasing the file named OVERTIME from the working directory is:

   A. DELETE/DIR OVERTIME ⤵

   B. D OVERTIME ⤵

   C. DEL OVERTIME ⤵

   D. ERASE OVERTIME ⤵

   E. EXECUTE OVERTIME ⤵

3. The command for displaying the contents of a source file named GENIUS is:

   A. DISPLAY GENIUS ⤵

   B. PRINT GENIUS ⤵

   C. TYPE GENIUS ⤵

   D. LIST GENIUS ⤵

   E. T GENIUS ⤵

4. The command for gracefully shutting down the operating system is:

   A. OFF ↵

   B. DOWN ↵

   C. BYE ↵

   D. CRASH ↵

   E. HALT ↵

5. The command for displaying the current working directory is:

   A. CURRENT ↵

   B. WORKING ↵

   C. WORKDIR ↵

   D. DIRECTORY ↵

   E. WHERE ↵

6. The command sequence for displaying the current system time and date is:

   A. T; D ↵

   B. TIME; DATE ↵

   C. TD ↵

   D. WHEN ↵

   E. CLOCK ↵

7. The command for bringing the disc device DPD0 on-line and accessible for I/O is:

   A. DIR @DPD0 ↵

   B. ON @DPD0 ↵

   C. MOUNT @DPD0 ↵

   D. MOUNT DPD0 ↵

   E. XEQ @DPD0 ↵

8, The command for changing the name of the file PLANT to TREE is:

   A. NAME PLANT TREE ↵

   B. CHANGE PLANT TREE ↵

   C. REN PLANT TREE ↵

   D. RENAME TREE PLANT ↵

   E. MAKE PLANT TREE ↵

9. The command for adding a new non-directory file named TOWER to the working directory is:

   A. ADD TOWER ↵

   B. NEW TOWER ↵

   C. CREATE TOWER ↵

   D. CREATE/DIR TOWER ↵

   E. MOUNT TOWER ↵

10. The command for displaying the fully qualified pathname of the file SECRET is:

    A. DISPLAY SECRET ↵

    B. FILE SECRET ↵

    C. FILENAME SECRET ↵

    D. PATH SECRET ↵

    E. Fi/NAME SECRET ↵

11. The command for making the file BOOPSIE a duplicate of the file LOGON.CLI is:

    A. DUP BOOPSIE LOGON.CLI ↵

    B. DUP LOGON.CLI BOOPSIE ↵

    C. COPY BOOPSIE LOGON.CLI ↵

    D. COPY BOOPSIE.BU LOGON.CLI ↵

    E. COPY LOGON.CLI BOOPSIE ↵

12. The following command will tell you whether the file PAYROLL can be deleted from the working directory:

    A. FILESTATUS PAYROLL ↵

    B. Fi/AS PAYROLL ↵

    C. ATTRIB PAYROLL ↵

    D. ATT PAY ↵

    E. ATTRIBUTES ↵

13. The command for running the program file **BROADCAST** is:

    A. RUN BROADCAST ↵

    B. X BROADCAST ↵

    C. BROADCAST ↵

    D. DO BROADCAST ↵

    E. XEQ BROAD ↵

14. The command for taking the secondary diskette DPX1 off-line and ready for removal is:

    A. RELEASE @DPX1 ↵

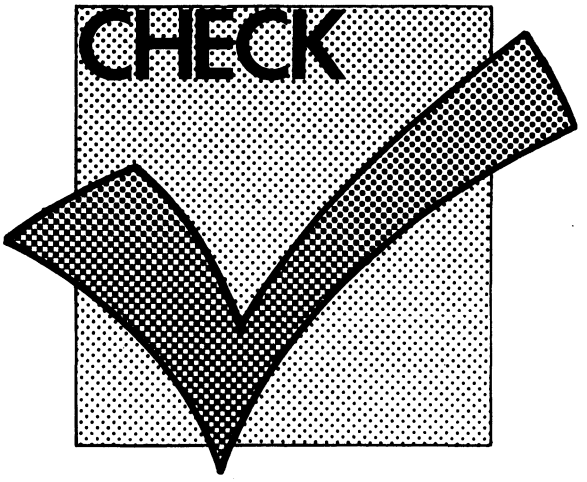
    B. DELETE @DPX1 ↵

    C. REMOVE @DPX1 ↵

    D. DISMOUNT @DPX1 ↵

    E. DISM DPX1 ↵

Check your answers on the following pages.



CLI COMMANDS

QUIZ ANSWERS

Answers and comments are as follows: correct selection is *circled.*

1. The entry that displays the length, type, and date of last modification of all files in the working directory is:

   A. ATTRIBUTES ↓
      Displays Error message: No Argument Specified.
   B. FILESTATUS ↓
      Only lists the filenames and working directory.
   C. FILESTATUS/ALL ↓
      Incorrect switch
   (D.) Fi/AS ↓
      Appropriate abbreviations.
   E. ATT/ALL ↓
      Error on switch and incorrect command.

2. The command for erasing the non-directory file named OVERTIME from the working directory is:

   A. DELETE/DIR OVERTIME ↓
      Overtime is not a directory file.
   B. D OVERTIME ↓
      Command is abbreviated too far.
   (C.) DEL OVERTIME ↓
      Success!
   D. ERASE OVERTIME ↓
      Unknown command or macro will be displayed as an error message.
   E. EXECUTE OVERTIME
      This attempts to run a program called OVERTIME.PR.

3. The command for displaying the contents of a source file named GENIUS is:

   A. DISPLAY GENIUS↓
      Error: Unknown command or macro.
   B. PRINT GENIUS↓
      Error: unknown command or macro.
   (C.) TYPE GENIUS↓
      Yup!
   D. LIST GENIUS↓
      No . . . error message
   E. T GENIUS↓
      Error: abbreviation not unique.

4. The command for gracefully shutting down the operating system is:

   A. OFF↓
      Error: unknown command or macro
   B. DOWN↓
      Same error as A.
   (C.) BYE↓
      Successful system shutdown.
   D. CRASH↓
      Error: unknown command or macro.
   E. HALT↓
      Same error as D.

5. The command for displaying the current working directory is:

   A. CURRENT↓
      Error.
   B. WORKING↓
      No.
   C. WORKDIR↓
      No such command.
   (D.) DIRECTORY↓
      Yup.
   E. WHERE↓
      Error: unknown command or macro.

6. The command sequence for displaying the current system time and date is:

   A. T; D↓
      Error: abbreviation not unique.
   (B.) TIME; DATE↓
      This will do it.
   C. TD↓
      Error: unknown command or macro.
   D. WHEN↓
      Not unless you create a macro to do it.
   E. CLOCK↓
      Same error as C.

7. The command for bringing the disc device DPD0 on-line and ready for I/O is:

   A. DIR @DPD0:↓
      No, this changes the working directory.
   B. ON @DPD0↓
      Error: Unknown command or macro.
   (C.) MOUNT @DPD0↓
      This does it.
   D. MOUNT DPD0↓
      Close, but you need the "@" prefix.
   E. XEQ @DPD0↓
      No, error.

8. The command for changing the name of the file PLANT to TREE is:

   A. NAME PLANT TREE↓
      Error: unknown command or macro.
   B. CHANGE PLANT TREE↓
      Same error as A.
   (C.) REN PLANT TREE↓
      This is the one.
   D. RENAME TREE PLANT↓
      Reverse order, Error: file does not exist.
   E. MAKE PLANT TREE↓
      Error same as A.

9. The command for adding a new file named TOWER to the working directory is:

   A. ADD TOWER↵
      Error: unknown command or macro.
   B. NEW TOWER↵
      Same as A.
   (C.) CREATE TOWER↵
      Yes, TOWER gets 0 length.
   D. CREATE/DIR TOWER↵
      This works, but makes TOWER a file of type "directory".
   E. MOUNT TOWER↵
      Error: argument not a directory device.

10. The command for displaying the fully qualified pathname of the file SECRET is:

   A. DISPLAY SECRET↵
      Error: unknown command or macro.
   B. FILE SECRET↵
      No, this displays the presence of a file named SECRET in the working directory by showing only its filename.
   C. FILENAME SECRET↵
      Same as B.
   (D.) PATH SECRET↵
      Yes, displays the fully qualified pathname.
   E. Fi/NAME SECRET↵
      Error: unknown switch.

11. The command for making the file BOOPSIE a duplicate of the file LOGON.CLI is:

   A. DUP BOOPSIE LOGON.CLI↵
      Error: unknown command or macro.
   B. DUP LOGON.CLI BOOPSIE↵
      Same as A.
   (C.) COPY BOOPSIE LOGON.CLI↵
      This will do it. Note the reverse direction of the copy.
   D. COPY BOOPSIE.BU LOGON.CLI↵
      This works, but not to the filename we requested in the question.
   E. COPY LOGON.CLI BOOPSIE↵
      Error: file does not exist (BOOPSIE)

12. The following command will tell you whether the file PAYROLL can be deleted from the working directory:

    A. FILESTATUS PAYROLL↓
       Displays only whether PAYROLL exists in the working directory by printing the filename.
    B. Fi/AS PAYROLL↓
       No. Displays the length, type, etc. of PAYROLL, but not its permanence attribute.
    (C.) ATTRIB PAYROLL↓
       Yes. Displays the attributes for PAYROLL.
    D. ATT PAY↓
       No. Error: file does not exist; filename is too short.
    E. ATTRIBUTES↓
       Error: command requires arguments.

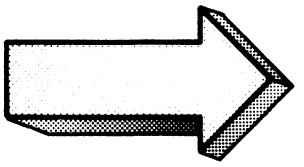
13. The command for running the program file BROADCAST is:

    A. RUN BROADCAST↓
       Error: unknown command or macro.
    (B.) X BROADCAST↓
       Got it.
    C. BROADCAST↓
       Error: unknown command or macro.
    D. DO BROADCAST↓
       Same error as A.
    E. XEQ BROAD↓
       Error: file does not exist.

14. The command for taking the secondary diskette DPX1 off-line and ready for removal is:

    A. RELEASE @DPX1↓
       Error: unknown command or macro.
    B. DELETE @DPX1↓
       No, delete expects a filename.
    C. REMOVE @DPX1↓
       Same error as A.
    (D.) DISMOUNT @DPX1↓
       This will do it.
    E. DISM DPX1↓
       Close, but the device name requires the "@" prefix.

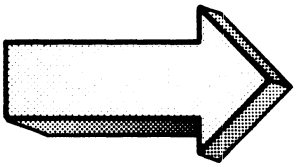
A SCORE OF 12 CORRECT QUESTIONS OUT OF THE 14 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE SUTDENT GUIDE.



Now prepare your system for the lab session.

First, your computer system must be powered up, loaded, and on-line. Step-by-step instructions for doing so are provided in Appendix A of this course. When your system is ready to boot then continue with the next step.

Second, your operating system must be read into memory. Step-by-step instructions for doing so are provided in Appendix A of this Course. When your system is ready for execution then begin the lab exercise on the next page. The lab exercise assumes that you are up and running.



CLI COMMANDS

LAB EXERCISE

Abstract

In this exercise you will use the CLI commands for managing files, monitoring the system, and manipulating the system environment.

Directions

This exercise is designed for use with or without a functioning computer system. For maximum benefit follow the steps below. Failure to do so diminishes the value of the exercise. Take your time, keep your head down, and follow through.

1. Cover the "screen" answer we have provided. Answers are separated by a line of asterisks.

2. Read the operation you are to perform.

3. If necessary, reference the CLI command or command sequence in the appropriate documentation. References include this course and the MP/OS Utilities Reference Manual 093-40000 2

4. Write down the appropriate command and anticipated response in the space provided. Be sure to use the complete answer. Be precise and specific.

5. Check your response with the correct answer(s) following the blank screen, only after you have filled in *your* response.

6. If your response is accurate, then continue with the next step. If not, try to figure out the source of your error and make the adjustment.

7. *If you have a computer,* enter the command sequence and re-check your results. Use the console control characters and sequences as necessary. Note that most CLI errors are caused by incorrect typing. Fear not: your MP/OS is a stable, mature product. It will not go away if you misspell a word. MP/OS will issue an error message if it cannot handle your entry. Just use the error message to diagnose the problem and make the correction.

COVER THE ANSWER

1. Write the command for determining the working directory. Show the system's response.

_____

_____

_____

****

```
) DIR ↓
@DPD0:
)
```

The DIRECTORY command displays the current working directory.

Do it on your system.

Several Notes:

(1) We use the down-arrow symbol (↓) to indicate a new line. In reality, this is *not* echoed on your terminal.

(2) We use the 10 MB disc as the system master device (DPD0). If your system master device is a diskette, substitute DPX0 for the filename where required. If it is a 12. MB, use DPH0. The following table outlines the mass storage (disc) devices currently available with MP/OS.

MASS STORAGE DEVICE CHARACTERISTICS CURRENTLY AVAILABLE WITH MP/OS:

MODEL	Fixed Disk Drive	6095	6038/9
NAME	DPH0	DPD0	DPX0, DPX1
Capacity/ Unit	12 Mb	10 Mb	315 Kb, 630 Kb bytes
Surfaces/ Unit	2	4	1
Heads/ Surface	2	1	1
Tracks/ Head	192	408	77
Sectors/ Track	32	12	8
Bytes/ Sector	512	512	512
Rotational Speed	49.4 rev/sec	80 rev/sec	6 rev/sec

2. Determine the files in your working directory. Show the system's response. (This could be lengthy so briefly summarize the anticipated response):

_____

_____

_____

_____

_____

_____

_____

****

```
) FILESTATUS }
DIRECTORY @DPD0:

   CLI.PR
   CLI.OL
   MASM.PR
   MASMXR.PR
   SPEED.PR
   ERMES
   BIND.PR
   BIND.OL
   MASM.PS
   MSL.LB
   FORT4.PR
   FORT4.LB
   FORT4.PS
   DOCUMENTS
   DISKSTUFF
   PSOURCES
   ?MSG
   ?SWAP_1
)
```

The FILESTATUS command displays the files in the working directory.

Do it on your system.

Your system may show several different files. Note the repetition of the directory.

3. Write the command to create a directory file called PRACTISE. Show the system's response:

_____

_____

_____

****

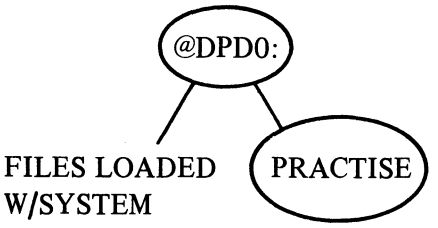
```
) CREATE/DIR PRACTISE)
)
```

CREATE sets up a disc file.

Now do it on your system.

Remember to use the /DIR switch on the CREATE command to specify the directory file type.

A simplified illustration of your file structure would look like:



Recall our convention of ovalling directory files and underlining non-directory files. We did not elaborate on the files loaded with your system.

4. Make PRACTISE your working directory. Show the system's response:

_____

_____

_____

_____

****

```
) DIR PRACTISE
)
```

The DIRECTORY command with an argument makes the specified file the working directory. Obviously the file must be a directory file.

Do it on your system.

If you type PRACTISE incorrectly, the system will tell you that the file does not exist. No harm is done, just try again. Remember to space between DIR and PRACTISE or else the system will tell you that DIRPRACTISE does not exist as a command or macro. Keep trying until you get it right.

5. Determine the working directory. Show the full pathname that the system provides:

_____

_____

_____

_____

_____

****

```
) DIR)
@DPD0:PRACTISE
)
```

Do it on your system.

Note that the system provides the full pathname starting from the root directory (in this case @DPD0:). As you recall, this does not alter the file structure.

6. Now create a file called ZORP in the PRACTISE directory. Write the command and anticipated response:

_____

_____

_____

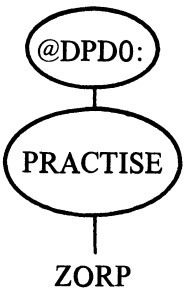
****

```
) CREATE ZORP ↓
)
```

Now do it on your system.

Since no switches are appended to the CREATE command, ZORP will be a text type file with a length of zero.

The file structure looks like this (we left off the files loaded with your system).



7. Determine the working directory and its contents with one command. Show the system's response:

_____

_____

_____

_____

****

```
) FILESTATUS/ASSORTMENT↓
DIRECTORY @DP00:PRACTISE

  ZORP                TXT    1-JAN-00    0:09:50          0
)
```

Do it on your system

FILESTATUS could be shortened to FI/AS. The information accompanying the ZORP file is file type (TXT), date and time of last modification, and length in bytes.

Note that the PRACTISE directory only displays the files in its list. PRACTISE has no "knowledge" of other files in the system. You cannot "see" the other files in the system from PRACTISE

8. Assign read-protect and write-protect attributes to ZORP . Check the success of the operation. Show the system's response:

_____

_____

_____

_____

_____

****

```
) ATTRIBUTES ZORP R W
) ATTRIBUTES ZORP
ZORP R W
)
```

The ATTRIBUTES command sets and displays the attributes of a specified file.

Now enter the command on your system.

The ATTRIBUTES command may be shortened to "A". You need only specify the simple filename ZORP and not specify the entire pathname, because the working directory contains ZORP.

9. Change the name of ZORP to TEST . Show the system's response:

_____

_____

****

```
) RENAME ZORP TEST ↓
)
```

The RENAME command changes the name of a file as specified.

Now do it on your system.

The old filename is specified first. The new filename is specified second. If the new filename is already in use, you may get:

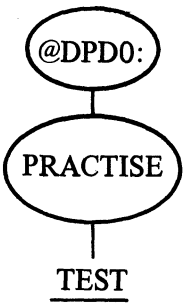
```
) RENAME ZORP TEST ↓
Error: file already exists
RENAME, @DP00:PRACTISE:TEST
)
```

Try another filename or use the "/D" switch, as shown in the following illustration.

```
) RENAME/D ZORP TEST
)
```

The "/D" switch deletes the original version of TEST and replaces it with the data and name from ZORP.

The file structure is now:



You will note that the only difference is that the filename TEST has replaced ZORP in the directory.

10. Verify the success of the rename operation from question 9:

_____

_____

****

```
) FI/AS )
DIRECTORY @DPD0:PRACTISE

  TEST                TXT    1-JAN-00    0:02:03          0
)
```

Do it on your system.

You can use the generalized FILESTATUS command or you can specify a filename argument with the command:

```
) FI/AS TEST )
DIRECTORY @DPD0:PRACTISE

  TEST                TXT    1-JAN-00    0:02:03          0
)
```

This becomes very useful when the file structure expands.

11. Make a duplicate copy of TEST in the same directory. Use TEST.2 as the filename for the second copy. Write the commands:

_____

_____

_____

****

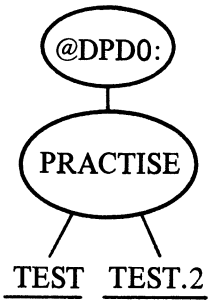
COPY duplicates the specified file.

```
) COPY TEST.2 TEST ↓
)
```

Now do it on your system.

Note that the destination file is specified first and the sending file is second. Simple filenames suffice for copies within the same directory.

The file structure is now:



The contents, type, and length of TEST and TEST.2 should be identical.

12. Verify the copy operation. Show the system's response:

_____

_____

_____

_____

_____

_____

****

```
) FI/AS/SORT↓
DIRECTORY @DPD0:PRACTISE

  TEST                TXT    1-JAN-00    0:02:03          0
  TEST.2              TXT    1-JAN-00    0:14:06          0
)
```

The "/ASSORTMENT" switch displays the file's type, modification times, and byte lengths.

The "/SORT" switch lists the filenames in alphabetical order.

The command line may be shortened to FI/AS/S.

Note that TEST and TEST.2 are identical in type and length. Their contents (currently empty) are exactly the same.

13. Create another directory, within the working directory, called FLOOR. Verify the operation's success. Show all commands and responses below:

_____

_____

_____

_____

_____

****

```
) CREATE/DIR FLOOR↓
) FI/AS↓
DIRECTORY @DPD0:PRACTISE

  TEST.2              TXT    1-JAN-00    0:14:06          0
  TEST                TXT    1-JAN-00    0:02:03          0
  FLOOR               DIR    1-JAN-00    0:15:11          0
)
```

Now do it on your system.

Notice the DIR type assigned to FLOOR. Also, since we did not specifically set the time and date parameters, we are still operating with the default settings of the year 1900 and the time as just after midnight.

14. Make FLOOR the working directory. Verify the success of the operation:

_____

_____

_____

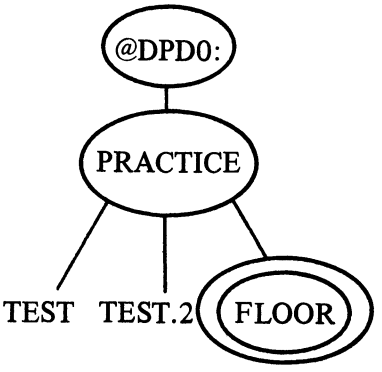
****

```
) DIR FLOOR )
) DIR )
@DPD0:PRACTISE:FLOOR
)
```

You can now add files to the FLOOR directory with simple filenames.

15. Draw a diagram of the file structure. Include only the system directory and the files you have created in this session. Circle the directories and underline the files. Double circle the working directory.

****



16. Create a non-directory file named BOTTOM in the working directory. Verify the success of the operation:

_____

_____

_____

_____

_____

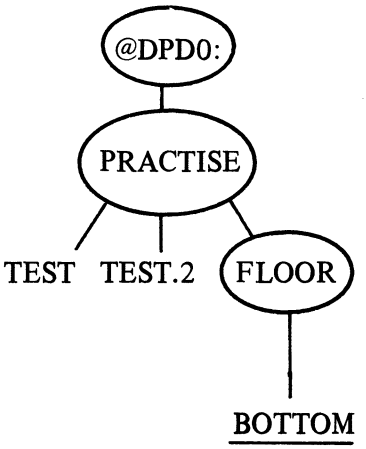
****

```
) CREATE BOTTOM )
) FILESTATUS )
DIRECTORY @DPD0:PRACTISE:FLOOR

  BOTTOM
)
```

CREATE opens and names a file in the working directory (by default).

Now do it on your system.

Your file structure would now look like this:



Note the addition of BOTTOM to the FLOOR directory.

17. Determine the fully qualified pathname to "BOTTOM". Show the commands and responses:

_____

_____

_____

****

```
) PATHNAME BOTTOM )
@DPD0:PRACTISE:FLOOR:BOTTOM
)
```

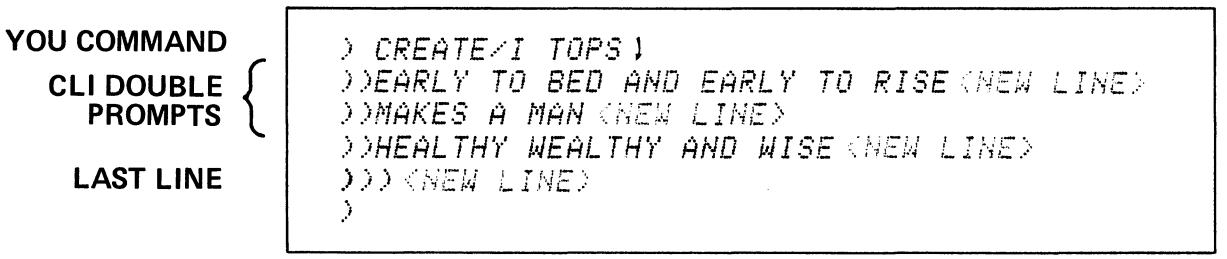
PATHNAME displays the fully qualified pathname of the specified file.

The simple filename suffices here. If you try to use a partial pathname as the argument, you get:

```
) PATHNAME :BOTTOM )
Error: File does not exist
PATHNAME.:BOTTOM
)
```

No harm is done. Just try again.

18. *This is new.* By appending the /I switch to the CREATE command, you can create a new file and insert text into it. The /I switch says that text will be inserted through @TTI the system console. An example follows:



Note that CLI prompts with a *double* right parenthesis during the insertion sequence. Each line of text is terminated by a NEW-LINE character. The last line of the inserted text is signalled by a single right parenthesis followed by a NEW-LINE.

Enter the text illustrated above into the TOPS file. Watch your spacing and follow the rules carefully.

19. Verify the success of the TOPS operation. Show the command and briefly describe the anticipated result:

_____

_____

_____

_____

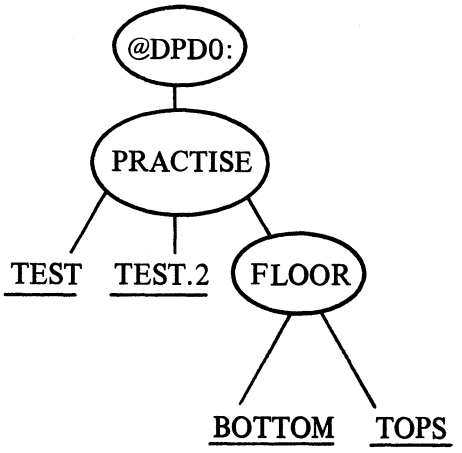
****

```
) TYPE TOPS )
EARLY TO BED AND EARLY TO RISE
MAKES A MAN
HEALTHY WEALTHY AND WISE

)
```

The TYPE command displays the contents of the specified file.

Now do it on your system.

Use TYPE/L to get a hard copy printout of the file. The FILESTATUS command (FI/AS) would show how many bytes TOPS had covered. The file structure is now:



The CREATE/I sequence operates in the working directory.

20. Make a duplicate copy of TOPS in the parent directory PRACTISE . Name the new copy NEWFILE . Show the command(s) and responses:

_____

_____

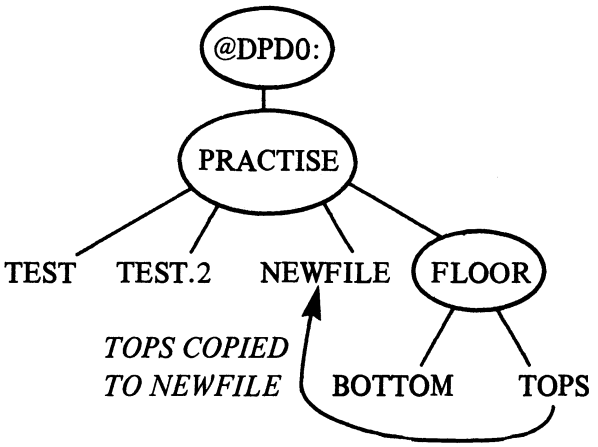
_____

****

```
) COPY :PRACTISE:NEWFILE TOPS )
)
```

Enter the command on your system.

The destination file is specified first and the source file second. This destination requires the full pathname. The file structure is now:



Notice how ":" was substituted for @DPD0: This is possible because @DPD0: is the root directory of the system master device.

21. Now make PRACTISE the working directory and verify that NEWFILE is a duplicate of TOPS . Show the commands and responses:

_____

_____

_____

_____

****

```
) DIR ^ ↓
) TYPE NEWFILE ↓
EARLY TO BED AND EARLY TO RISE
MAKES A MAN
HEALTHY WEALTHY AND WISE

)
```

DIR ^ makes the parent directory the working directory.

Now do it on your system.

There are various ways of accomplishing this operation. You could put both commands on one line:

DIR^; TYPE NEWFILE ↓

or you could skip changing the working directory and use the full pathname as:

TYPE @DPD0:PRACTISE:NEWFILE ↓

or the shorter version:

TYPE :PRACTISE:NEWFILE ↓ "

22. Make the system directory your working directory. Specify three commands to accomplish this. You are now in the PRACTISE directory.

1. _____

_____

2. _____

_____

3. _____

****

1.

```
) DIR ^ )
)
```

2.

```
) DIR @DPD0: )
)
```

3.

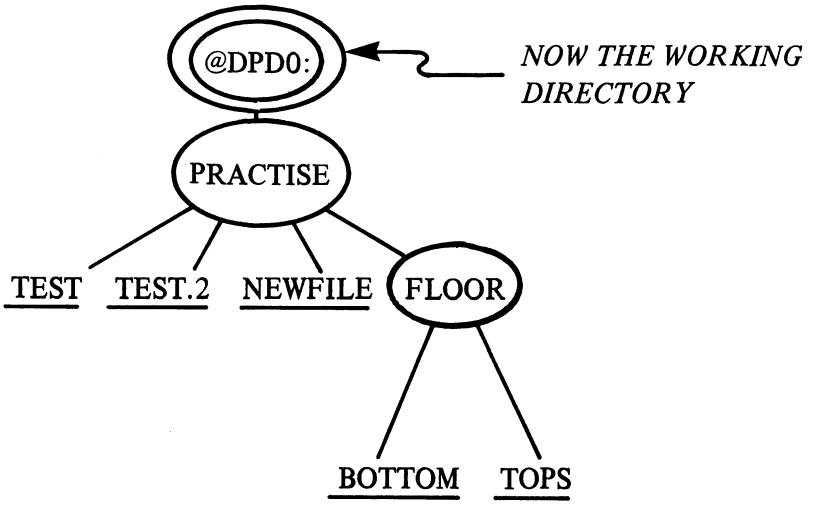
```
) DIR : )
)
```

Perform the operation on your system.

Note that the third option allows you to specify a colon as an abbreviated pathname for @DPD0:. This option is available only on the system's master device.

23. Draw the file structure in the box provided. Indicate your current position in the file structure with a double circle. Include only the files you created in this lab.

****



24. Delete the PRACTISE directory, files, and sub-directory. @DPD0: is your working directory. Verify your deletions:

_____

_____

_____

****

```
) DELETE/DIR/V/C :PRACTISE:FLOOR )
:PRACTISE:FLOOR ?YES )
Deleted :PRACTISE:FLOOR
) DIR :)
) DELETE/DIR/V/C :PRACTISE )
:PRACTISE ?YES )
Deleted :PRACTISE
)
```

The first delete removes the FLOOR directory and files. The second delete erases the PRACTISE directory and files.

Now perform the operation on your system.

If you specify a directory it will only be deleted if it is empty, unless you use the /DIR switch. If you use the /DIR switch, the subtree (if any) of the specified directory will be deleted.

The /V switch directs the system to verify the deletions. We also used the /C switch to check or confirm our desired deletion.

25. Check the system's date. Then re-assign it to May 6, 1979. Verify the re-assignment.

_____

_____

_____

_____

****

```
) DATE↓
  1-JAN-00
) DATE 6-MAY-79↓
) DATE↓
  6-MAY-79
)
```

DATE sets or displays the current system date.
TIME sets or displays the current system time.

Now do it on your system.

Note the proper sequence for entering the date parameters: dd-MMM-YY.

The date parameter defaults to 1-JAN-00 each time CLI is refreshed. A refresh occurs after system panics (attempts to POP from level 0) and system initialization.

26. Check the system time. Re-set it to 12:45 P. M. Verify the success of the operation.

****

```
) TIME )
  0:25:35
) TIME 12:45:00 )
) TIME )
 12:45:03
)
```

Now perform the operation on your system.

Time is a cumulative quantity. The system time shown here indicates that twenty-five minutes have passed since the system was booted into operation.

If you do not re-set the system time, it will revert to 0:00:00 with each boot or initialization.

27. Determine the revision number of CLI. Show the system's response.

_____

_____

_____

_____

****

```
) REVISION CLI.PR )
00.01
)
```

The REVISION command displays the rev number of the specified file.

Your CLI should show a different rev number. Remember that the "Revision" command works only on executable files.

28. Terminate your session on the system. Show the system's response.

_____

_____

_____

_____

****

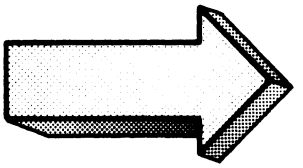
```
) BYE )
MP/OS  CLI Termin
                    System shutdown
075424
!
```

The BYE command safely shuts down the system.

Now do it on your system.

If you have a Dasher hard-copy terminal, the system will display the truncated message. The Dasher CRT will display the full message. The console debug is indicated by the "!" prompt.

YOUR NEXT STEP
IS TO LEARN
HOW TO USE CLI
MACROS WHICH
BEGIN ON THE
NEXT PAGE . . .



CLI MACROS

Abstract

This segment of Module Two describes the procedures for creating and executing CLI Macro files.

Objectives

Upon completion of this segment you will be able to:

1. State the purpose of CLI macro files.

2. Write, enter, and execute CLI macros to solve a processing problem.

3. Given a CLI macro error situation,
   a) diagnose the error;
   b) reference the solution;
   c) write the solution.

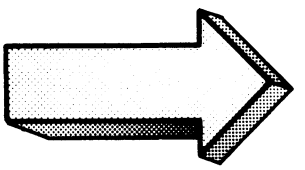
Directions

1. Turn to figure 2-29 on the next page of this Guide.

2. Listen to the tape for this segment of Module Two.

3. Try the CLI Macros Quiz.

4. Work the CLI Macros Lab exercise.



YOU TYPE →
SYSTEM RESPONDS →

```
) CREATE/I CRUNCH.CLI ↵
))
```

Figure 2-29

```
) CREATE/I CRUNCH.CLI ;
)) MOD_ONE,MOD_TWO,BACKUP.1,BACKUP.2 ;
)))
)
```

Figure 2-30

YOU TYPE

```
) TYPE/L [CRUNCH.CLI] ↵
)


) TYPE/L CRUNCH.CLI ↵
)
```

CLI MACRO FILES & BRACKETS

Figure 2-31

```
) CREATE/I BARRY_ONE ↓
)) THIS IS THE BARRY_ONE FILE ↓
)) CREATED FOR THE .CLI MACRO TEST ↓
)) LAST LINE ↓
))) ↓
)
```

```
) CRE/I BARRY_TWO ↓
)) THIS IS THE BARRY_TWO FILE ↓
)) LINE TWO OF BARRY_TWO ↓
)) LAST OF BARRY_TWO ↓
))) ↓
)
```

Figure 2-32

```
) CREATE/I BARRY.1 ↓
)) THIS IS THE BARRY.1 FILE ↓
)) ?????????????????????????????????? ↓
)) >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> ↓
)) ,,.,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,, ↓
)) LAST LINE OF BARRY.1 ↓
))) ↓
)
```

```
) CRE/I BARRY.2 ↓
)) THIS IS THE BARRY.2 FILE ↓
)) @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ ↓
)) 22222222222222222222222222222222 ↓
)) 4444444444444444444444444 ↓
)) LAST OF BARRY.2 ↓
))) ↓
)
```

Figure 2-33

```
) CREATE/I TEST.CLI ↵
)) BARRY_ONE,BARRY_TWO,BARRY.1,BARRY.2 ↵
))) ↵
)
```

Figure 2-34

YOU TYPE
SYSTEM RESPONSES

```
) TYPE TEST.CLI ↵
BARRY_ONE,BARRY_TWO,BARRY.1,BARRY.2

)
```

YOU TYPE
SYSTEM RESPONSES

```
) TYPE [TEST CLI] ↵
THIS IS THE BARRY_ONE FILE
CREATED FOR THE .CLI MACRO TEST
LAST LINE

THIS IS THE BARRY_TWO FILE
LINE TWO OF BARRY_TWO
LAST OF BARRY_TWO

THIS IS THE BARRY.1 FILE
????????????????????????????
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
LAST LINE OF BARRY.1

THIS IS BARRY.2 FILE
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
22222222222222222222222222
4444444444444444444
LAST OF BARRY.2
)
```

Figure 2-35

```
1.  ) TYPE CRUNCH
2.  CRUNCH Error: File does not exist
3.  )
```

Figure 2-36

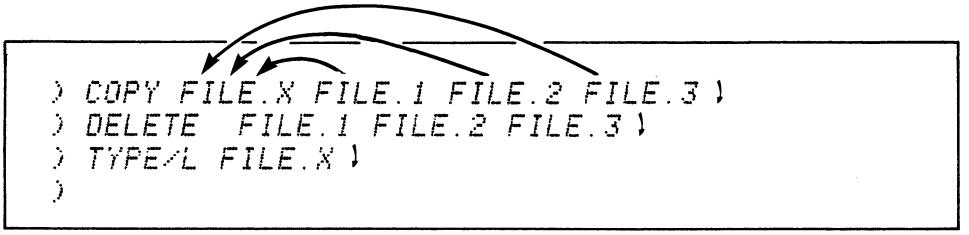


Figure 2-37

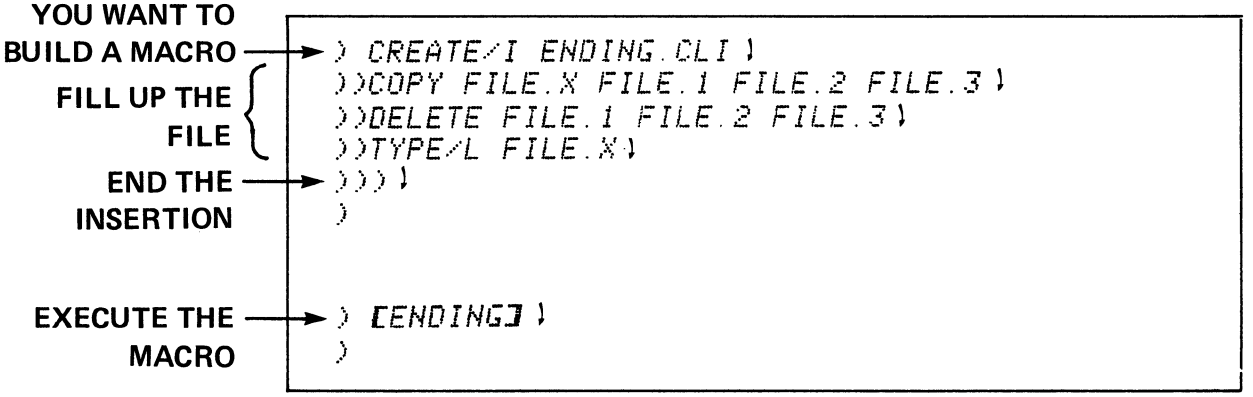


Figure 2-38

YOU BUILD THE MACRO FILE

```
) CREATE/I SAMPLE.CLI↓
))TYPE %1%↓
))DELETE %1%↓
)))↓
)
```

EXECUTE MACRO

```
) SAMPLE BARRY_ONE↓
THIS IS THE BARRY_ONE FILE
CREATED FOR THE .CLI MACRO TEST
LAST LINE

)
```

Figure 2-39

```
) [SAMPLE BARRY_ONE]↓
THIS IS THE BARRY_ONE FILE
CREATED FOR THE .CLI MACRO TEST
LAST LINE

)
```

EXECUTE MACRO

SAMPLE IS TYPED

SAMPLE DELETED

```
  ) SAMPLE SAMPLE↓
TYPE %1%
DELETE %1%

)
```

Figure 2-40

BUILD THE MACRO FILE

```
) CREATE/I SWITCH CLI)
))RENAME %1% TEMP)
))RENAME %2% %1%)
))RENAME TEMP %2%)
)))/
)
```

EXECUTE THE MACRO FILE

```
) SWITCH APPLES ORANGES)
)
```

Figure 2-41

TOPICS

- MACROS
- .CLI EXTENSION
- CREATE/I
- EXECUTION

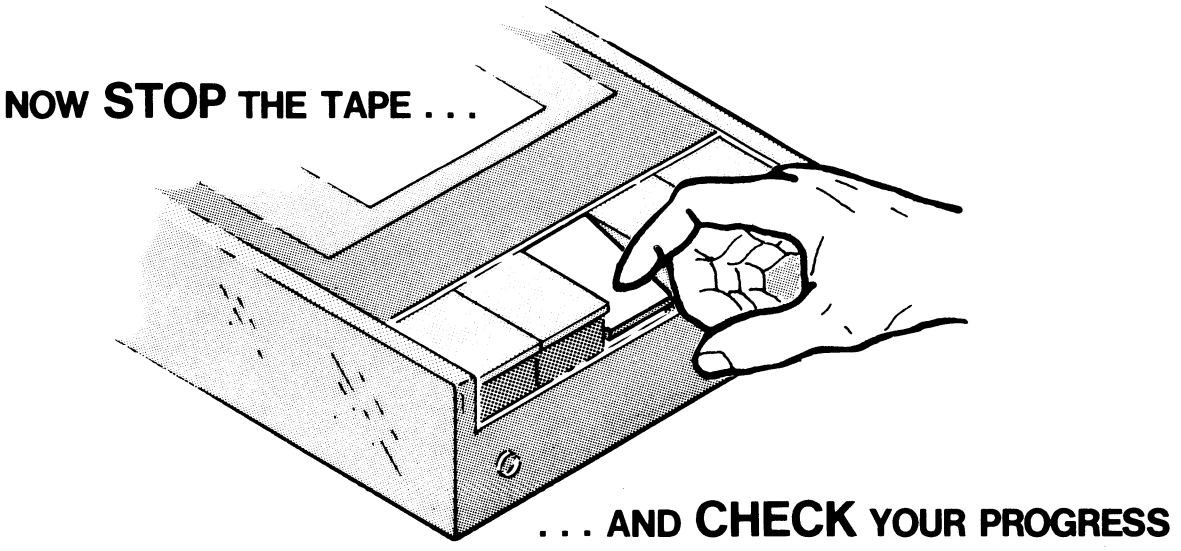


Figure 2-42

CLI MACROS

QUIZ

Circle the letter(s) of the correct answer(s) to the following questions. Note that a question may have more than one correct answer.

1. The command line for creating a macro named QUIZ is:

   A. CLI QUIZ ↵

   B. CLI/I QUIZ.CLI ↵

   C. CRE/I QUIZ.CLI ↵

   D. CREATE QUIZ ↵

2. The command line for executing a macro file named QUIZ.CLI is:

   A. QUIZ ↵

   B. CLI QUIZ ↵

   C. [QUIZ] ↵

   D. [QUIZ.CLI] ↵

3. The command line for displaying the contents of the macro file QUIZ.CLI is:

   A. TYPE QUIZ ↵

   B. TYPE QUIZ.CLI ↵

   C. [TYPE QUIZ.CLI] ↵

   D. [TYPE QUIZ] ↵

4. Given the following macro file sequence, what will be the result:

```
) CREATE/I HOWS.CLI ↓
)) FILESTATUS %1% ↓
)) ATTRIBUTES %1% ↓
)) PATHNAME %1% ↓
))) ↓
)  HOWS  R ↓
```

A. The file "HOWS.CLI" is given the 'R' attribute.

B. The FILESTATUS, ATTRIBUTES, and fully qualified PATHNAME of the file "HOWS" are displayed.

C. The FILESTATUS, ATTRIBUTES, and PATHNAME of "HOWS.CLI" are displayed.

D. The FILESTATUS, ATTRIBUTES, and PATHNAME of the file "R" are displayed.

5. Given the following sequence, what is the result:

```
) CREATE/I  RUN.CLI ↓
)) XEQ  MASM  %1% ↓
)) XEQ  BIND  %1%       MSL.LB ↓
)) XEQ %1% ↓
))) ↓
) RUN PROGASM ↓
```

A. "PROGASM" is assembled, bound, and executed.

B. "RUN.CLI" is assembled, bound, and executed.

C. "RUN" is assembled, bound, and executed.

D. "MSL.LB" is assembled, bound, and executed.

6. Given the following sequence, what is the result:

```
) CREATE/I  KILL.CLI ↓
)) ATTRIBUTES %1% ↓
)) ATTRIBUTES %1%  P ↓
)) DELETE %1% ↓
))) ↓
) KILL  QUIZ.CLI ↓
```

A. "QUIZ.CLI" is deleted.

B. "KILL.CLI" is deleted.

C. "QUIZ.CLI" is not deleted.

D. "KILL.CLI" is not deleted.

7. Given the following sequence, what is the result:

```
) CREATE/I  ERASE.CLI ↓
)) ATTRIBUTES %1%  W ↓
)) DELETE %1% ↓
)) FILESTATUS/AS  %1% ↓
))) ↓
) ERASE  ERASE.CLI ↓
```

A. "ERASE.CLI" is deleted.

B. "ERASE.CLI" is not deleted.

C. "1" is deleted.

D. "1" is not deleted.

CHECK YOUR ANSWERS ON THE FOLLOWING PAGES.

CLI MACROS

QUIZ ANSWERS

1. The command line for creating a macro named QUIZ is:

   A. CLI QUIZ ↵
      The system displays the error message: Error: unknown command or macro.
   B. CLI/I QUIZ.CLI ↵
      Same as A.
   (C.) CRE/I QUIZ.CLI ↵
      Yes. This will work. The system responds with a double prompt "))".
   D. CREATE QUIZ ↵
      This creates a file named Quiz, but not a macro file.

2. The command line for executing a macro file named QUIZ.CLI is:

   (A.) QUIZ ↵
      Yes, this works. It is not necessary to specify the entire name.
   B. CLI QUIZ ↵
      Error: unknown command or macro.
   (C.) [QUIZ] ↵
      Yes, this works also.
   (D.) [QUIZ.CLI] ↵
      Yes, this works.

3. The command line for displaying the contents of the macro file QUIZ.CLI is:

   A. TYPE QUIZ ↵
      No. The system will display the error message: "Error: file does not exist."
   (B.) TYPE QUIZ.CLI ↵
      Yes. The complete filename is necessary here.
   (C.) [TYPE QUIZ.CLI] ↵
      Yes, this works.
   D. [TYPE QUIZ] ↵
      No, same as A.

4. Given the following macro file sequence, what will be the result:

```
) CREATE/I  HOWS.CLI ↓
)) FILESTATUS %1% ↓
)) ATTRIBUTES %1% ↓
)) PATHNAME %1% ↓
))) ↓
) HOWS   R ↓
```

A. The file "HOWS.CLI" is given the "R" attribute.
No. "R" is a filename in this case.

B. The FILESTATUS, ATTRIBUTES, and fully qualified PATHNAME of the file "HOWS" are displayed.
No, in this case "HOWS" is the macro command.

C. The FILESTATUS, ATTRIBUTES, and PATHNAME of "HOWS.CLI" are displayed
No.

(D.) The FILESTATUS, ATTRIBUTES, and PATHNAME of the file "R" are displayed.
Yes. "R" replaces the dummy argument %1%. "R", of course, must exist as a file in the working directory.

5. Given the following sequence, what is the result:

```
) CREATE/I RUN.CLI ↓
)) XEQ  MASM  %1% ↓
)) XEQ BIND %1% MSL.LB ↓
)) XEQ %1% ↓
))) ↓
) RUN PROGASM ↓
```

(A.) "PROGASM" is assembled, bound, and executed.
Yes, MASM is the Macroassembler, BIND is the binder. This macro may be useful for your development work.

B. "RUN.CLI" is assembled, bound, and executed.
No, RUN.CLI is not specified as the dummy argument.

C. "RUN" is assembled, bound, and executed.
No, see B.

D. "MSL.LB" is assembled, bound, and executed.
No, this is the Macroassembler library used in assembly.

6. Given the following sequence, what is the result:

```
) CREATE/I KILL.CLI ↓
)) ATTRIBUTES %1% ↓
)) ATTRIBUTES %1%  P ↓
)) DELETE %1% ↓
))) ↓
) KILL QUIZ.CLI ↓
```

A. "QUIZ.CLI" is deleted.
No, the permanent attribute prevents deletion.

B. "KILL.CLI" is deleted.
No, this is not specified as the argument.

(C.) "QUIZ.CLI" is not deleted.
Yes, see A.

(D.) "KILL.CLI" is not deleted.
This is true, although indirectly.

7. Given the following sequence, what is the result:

```
) CREATE/I ERASE.CLI ↓
)) ATTRIBUTES %1%  W ↓
)) DELETE %1% ↓
)) FILESTATUS/AS  %1% ↓
))) ↓
) ERASE  ERASE.CLI ↓
```

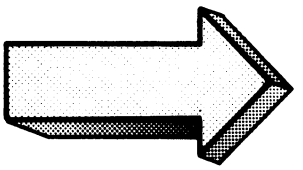
(A.) "ERASE.CLI" is deleted.
Yes, "ERASE.CLI" replaces the dummy argument and is deleted.

B. "ERASE.CLI" is not deleted.
No. The "W" attribute does not prevent deletion of the file.

C. "1" is deleted.
No. "1" indicates the position of the argument. That is, the first argument in the command line substitutes for the %1%.

(D.) "1" is not deleted.
"1" is not specified as the filename argument.

A SCORE OF 6 CORRECT ANSWERS OUT OF THE 7 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.



CLI MACROS

LAB EXERCISE

Abstract

In this exercise you will use the CLI commands in macro format.

Directions

This exercise is similar to the previous one. Follow the same step-by-step procedure. Remember: if you have an MP/100, MP/200 or microNOVA, use it to perform this lab exercise.

Briefly:

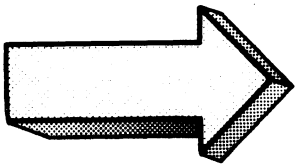
1. Cover the answers.
2. Read the required operation.
3. Write down the command or answer.
4. Uncover the answer and compare it to yours.
5. Perform the operation on your system.
6. Check the system response.

Again, we assume that your system is up and running. If not, perform the system boot. Follow the procedure detailed in Appendix A.

Your system should display the CLI initial message and prompt:

```
MICRON  CLI     Rev 1.0
)
```

Your revision number may be different.



1. Create a text file called ENIAC with the following:

   "Today's minicomputer,
   at a cost of $300,
   has more computing
   capacity than the ENIAC".

   Write in the commands and anticipated response:

_____

_____

_____

_____

_____

_____

_____

****

```
) CREATE/I ENIAC
))TODAY'S MINICOMPUTER,<NEW LINE>
))AT A COST OF $300<NEW LINE>
))HAS MORE COMPUTING<NEW LINE>
))CAPACITY THAN THE ENIAC.<NEW LINE>
)))<NEW LINE>
)
```

CREATE/I allows you to input text from your console and create a new file.

Now do it on your system.

Note: remember to terminate each line of text with a New-line or line feed (depending on your terminal).

2. Build a macro file that will (1) make a copy of your original file, (2) type the original, and (3) delete the original. Name the macro file TICKLER. Write the sequence here:

_____

_____

_____

_____

_____

****

```
) CREATE/I  TICKLER.CLI
))COPY %2% %1% <NEW LINE>
))TYPE %1% <NEW LINE>
))DELETE %1% <NEW LINE>
)))<NEW LINE>
)↓
```

Do it on your system.

Note: the macro file must have the .CLI extension to be recognized as a macro.

3. Now use the TICKLER macro file to backup, type, and delete ENIAC. Name the backup file ENIAC.BU. Show the command and system response:

****

```
) TICKLER ENIAC ENIAC.BU )
TODAY'S MINICOMPUTER,
AT A COST OF $300
HAS MORE COMPUTING
CAPACITY THAN THE ENIAC.

)
```

Now enter the sequence on your system.

Note that ENIAC replaces the dummy argument %1% and ENIAC.BU replaces %2%. You can use the square brackets or leave them off.

4. Create a macro file named PICKLE that will (1) make two disc copies of your original, (2) print both copies on the line printer, (3) delete the original and (4) verify the deletion. Write the complete sequence here:

_____

_____

_____

_____

_____

****

```
) CREATE/I PICKLE.CLI )
))COPY (%2% %3%) %1%<NEW LINE>
))TYPE/L (%2% %3%)<NEW LINE>
))DELETE/V %1%<NEW LINE>
)))
)
```

Do it on your system.

Note the use of parenthesis in the command lines. This eliminates the necessity of typing additional command lines.

5. Use PICKLE.CLI to make two copies of ENIAC.BU. Name the copies ENIAC.1 and ENIAC.2. Type both copies on the line printer. Delete the original and verify the deletion. Show the system's response:

_____

_____

_____

_____

****

```
) PICKLE ENIAC.BU  ENIAC.<1 2>
DELETED ENIAC.BU
)
```

Try it on your system.

Note that dummy argument %1% is replaced by ENIAC.BU, %2% by ENIAC.1 and %3% by ENIAC.2. The angle brackets ease coding. The square brackets are not essential.

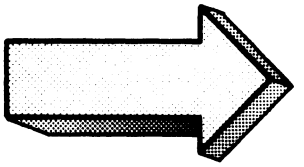
6. In this question we reverse the tables. Given the macro file below:

```
) CREATE/I FINAL.CLI )
))TYPE %0%<NEW LINE>
))TYPE %1%<NEW LINE>
))DELETE/V %1%<NEW LINE>
))TYPE %2%)DEL/V %2%<NEW LINE>
)))).
)
```

What will happen with the following:

```
) FINAL.CLI  TICKLER.CLI  PICKLE.CLI )
```

Briefly describe what will occur (it's too long to write out):



_____

_____

_____

_____

****

1. FINAL.CLI will be displayed on the console.
2. TICKLER.CLI will be displayed on the console.
3. TICKLER.CLI will be deleted and the deletion will be verified on the console.
4. PICKLE.CLI will be displayed on the console.
5. PICKLE.CLI will be deleted and the deletion will be verified on the console.
6. CLI returns the prompt ")".

The actual response is shown below:

```
) FINAL.CLI TICKLER.CLI  PICKLE.CLI !
TYPE %0%
TYPE %1%
DELETE/V %1%
TYPE %2%;DEL/V %2%

COPY %2% %1%
TYPE %1%
DELETE %1%

DELETED TICKLER.CLI
COPY (%2% %3%) %1%
TYPE/L (%2% %3%)
DELETE/V %1%

DELETED PICKLE.CLI
)
```

Try it on your system.

Make sure that you specify FINAL.CLI or else the macro may misfire.

7. *This is a new one.* The WRITE command displays its arguments on the line following the command. For example:

If you append the /L switch to the WRITE command, the argument string is displayed on the line printer.

Another example for the console:

```
) WR GOOD MORNING!  MP/OS IR READY
GOOD MORNING! MP/OS IR READY
)
```

Here we use the WRITE abbreviation WR.

The WRITE command is not all that exciting by itself. However, the next question incorporates it into a macro command.

Create a macro file named QUERY that 1) displays the message: "the status of your file is:", and 2) displays the filestatus of the requested file. Show the sequence:

_____

_____

_____

_____

_____

_____

_____

****

```
) CREATE/I QUERY.CLI )
))WRITE THE STATUS OF YOUR FILE IS: )
))FILESTATUS/AS %1% )
)))
)
```

The WRITE command carries the message as its arguments.

Now enter the sequence on your system.

8. Execute the QUERY macro on itself. Show the command and anticipated results:

****

```
) QUERY QUERY.CLI )
THE STATUS OF YOUR FILE IS:

DIRECTORY @DP00:

  QUERY.CLI              TXT    18-JUL-79  15:51:54      52
)
```

QUERY.CLI replaces the dummy argument %1%. The message, pathname, and files status are displayed.

Now try it on your system.

Watch the spelling of this one.

9. *More new information.* When CLI is invoked, the macro file LOGON.CLI is automatically executed (if you have one). Create a LOGON macro that displays a welcome message and the initial directory name:

_____

_____

_____

_____

_____

_____

****

```
) CREATE/I LOGON.CLI )
))WRITE  HELLO!  YOUR WORKING DIRECTORY IS )
))DIRECTORY )
)))
)
```

LOGON.CLI now has a WRITE message and DIRECTORY command.

Enter the sequence on your system.

10. Execute the LOGON macro. Show the command and anticipated response:

_____

_____

_____

_____

****

```
) LOGON↵
HELLO!  YOUR WORKING DIRECTORY IS
@DP0@:
)
```

It is not necessary to use square brackets or LOGON.CLI.

Try it on your system.

11. *Warning: this sequence is only for the strong-hearted.*
Now lets see if LOGON really works! Shutdown your system and bring it back up. List the sequences and expected responses:

_____

_____

_____

_____

_____

_____

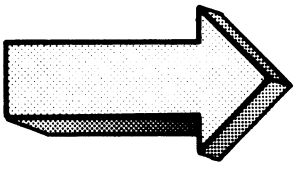
****

```
) BYE )
MP/OS CLI TERMINATING
SYSTEM SHUTDOWN
074524
!100027L
MP/OS    Rev 1.0

MP/OS  CLI  Rev 1.0
HELLO YOUR WORKING DIRECTORY IS
@0P00:
)
```

BYE shuts down the system.
1000 27L initiates the boot sequence from the 10MB disc.
The LOGON macro executes just prior to CLI's initial prompt.

Do it on your system.

THIS CONCLUDES THE CLI MACROS LAB EXERCISE. AT THIS POINT YOU SHOULD BE ABLE TO CREATE AND EXECUTE A MACRO FILE. NOW CONTINUE TO THE MODULE TWO QUIZ.



MODULE TWO

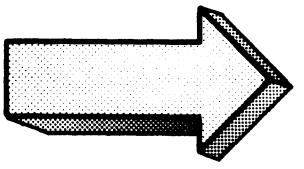
QUIZ

Directions

1. Answer each question by writing the appropriate answer in the space provided. The answers are cumulative (that is: each question must account for previous questions and answers).

2. At the end of the quiz, check your answers against the Answer Guide.

3. As an added verification, try the commands in squence (unless noted) on your system. Note: we assume that your system is up and running. We have used the 10 MB . disc as our system master device (@DPD0).



REMEMBER: FILL IN THE COMMANDS *AND* THE ANTICIPATED SYSTEM RESPONSES.

1. Set the system time to 10:00 A. M.

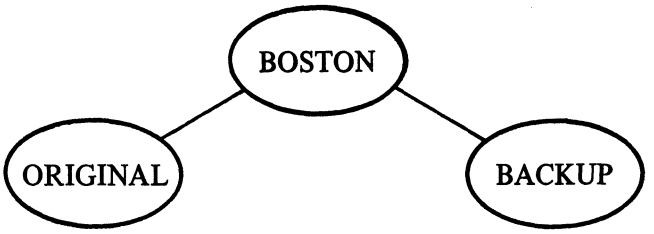
2. Set the system date to May 6, 1979.

3. Verify the system time and date. Use one command line. (Remember that these questions are cumulative. That is, your answer should take the previous questions into account.)

4. Build a macro file called WHEN that displays messages such as: "The time is now" and "today's date is" followed by the system time and date.

5. Execute the WHEN macro.

6. Given the following diagram:



Create this file structure on your system. (BOSTON will be a sub-directory of your root directory).

7. Given the following text:

"This is the Jersey file
Don't lose your jersey in Jersey
Third and last line of Jersey."

Write this text into a file named JERSEY in the ORIGINAL directory:

8. Verify the contents *and* status of the JERSEY file.

9. Determine the fully qualified pathname of JERSEY .

10. Use one command to make a duplicate of JERSEY called JERSEY.BU in the BACKUP directory.

11. Make BOSTON the working directory.

12. Use one command line to compare the status of JERSEY and JERSEY.BU.

13. Create a macro file called "?" that will display file status information on *separate* lines. Display should include: date the file was last accessed;
date the file was last modified;
byte length
time of last access
time of last modification
type
full pathname.

Check the MP/OS Utilities Reference Manual for info on the switches.

14. Execute the "?" file for JERSEY.

15. Delete the files and directories you created in this session. Confirm, then verify.

16. Shutdown the system.

CHECK YOUR ANSWERS ON
THE FOLLOWING PAGES

MODULE TWO

QUIZ ANSWERS

1.

```
) TIME 10:00:00 ↵
)
```

2.

```
) DATE 6-MAY-79 ↵
)
```

3.

```
) TIME;DATE ↵
10:00:34
 6-MAY-79
)
```

4.

```
) CREATE/I WHEN.CLI ↵
))WRITE *****  THE TIME IS NOW   *** ↵
))TIME ↵
))WRITE  ***   TODAY'S DATE   *** ↵
))DATE ↵
))) ↵

)
```

5.

```
) WHEN ↵
***** THE TIME IS NOW ***
10:02:44
*** TODAY'S DATE ***
 6-MAY-79
)
```

6.

```
) CREATE/DIR BOSTON ↵
) DIR BOSTON ↵
) CREATE/DIR BACKUP ↵
) CREATE/DIR ORIGINAL ↵
)
```

7.

```
) DIR ORIGINAL ↵
) CREATE/I JERSEY ↵
))THIS IS THE JERSEY FILE ↵
))DON'T LOSE YOUR JERSEY IN JERSEY ↵
))THIRD AND LAST LINE OF JERSEY ↵
))) ↵

)
```

8.

```
) TYPE JERSEY ↵
THIS IS THE JERSEY FILE
DON'T LOSE YOUR JERSEY IN JERSEY
THIRD AND LAST LINE OF JERSEY

) FI/AS JERSEY ↵
DIRECTORY @DPX0:BOSTON:ORIGINAL

  JERSEY              TXT     6-MAY-79  10:05:22          87
)
```

9.

```
) PATHNAME JERSEY↓
@DPX0:BOSTON:ORIGINAL:JERSEY
)
```

10.

```
) COPY @DPX0:BOSTON:BACKUP:JERSEY.BU  JERSEY ↓
)
```

11.

```
) DIR BOSTON ↓
)
```

12.

```
) FI/AS ORIGINAL:JERSEY  BACKUP:JERSEY.BU↓
DIRECTORY @DPX0:BOSTON:ORIGINAL

  JERSEY              TXT     6-MAY-79  10:05:22          87
DIRECTORY @DPX0:BOSTON:BACKUP

  JERSEY.BU           TXT     6-MAY-79  10:10:34          87
)
```

13.

```
) CREATE/I ?.CLI↓
))FI/DLA %1%
))FI/DLM %1%
))FI/ELEM %1%
))FI/LENGTH %1%
))FI/TLA %1%
))FI/TLM %1%
))FI/TYP %1%
))PATHNAME %1%
)))↓

)
```

14.

```
) ? JERSEY↓
DIRECTORY @DPX0:BOSTON:ORIGINAL

  JERSEY                    6-MAY-79
DIRECTORY @DPX0:BOSTON:ORIGINAL

  JERSEY                    6-MAY-79
DIRECTORY @DPX0:BOSTON:ORIGINAL

  JERSEY                    1
DIRECTORY @DPX0:BOSTON:ORIGINAL

  JERSEY                                87
DIRECTORY @DPX0:BOSTON:ORIGINAL

  JERSEY                   10:10:33
DIRECTORY @DPX0:BOSTON:ORIGINAL

  JERSEY                   10:05:22
DIRECTORY @DPX0:BOSTON:ORIGINAL

  JERSEY                  TXT
@DPX0:BOSTON:ORIGINAL:JERSEY
)
```

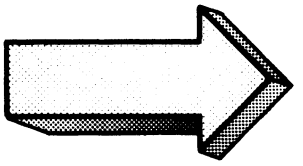
15.

```
) DIR↓
@DPX0:
) DELETE/V/C/DIR BOSTON↓
BOSTON? YES↓
Deleted BOSTON
)
```

16.

```
) BYE↓
Micron CLI Terminating

System shutdown
056301
!
```

A SCORE OF 13 CORRECT ANSWERS OUT OF THE 16 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.



MODULE THREE
SPEED

MODULE THREE
SPEED

Abstract

This module instructs in the use of the SPEED text editor to create, modify, and copy ASCII source text. The module is divided into the following sections:

- Concepts, terminology, and console control.
- Commands for managing files.
- Commands for manipulating text.

Objectives

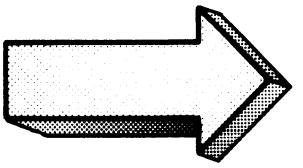
Upon completion of this module, you will be able to:

1. Use CLI to invoke SPEED;

2. Use SPEED to:
   a) open and close files for I/O.
   b) input and output source text from files to the edit buffer;
   c) Move the character pointer to specified locations within the text;
   d) Insert, modify, and delete text in the edit buffer;
   e) Display text in the current edit buffer in various lengths;
   f) Call C. L. I.

Directions

Turn to the next page of the Student Guide and read the introduction to the first segment of Module Three.



SPEED CONCEPTS

Abstract

This segment of Module Three discusses basic SPEED terminology, console control operations, and processing concepts.

Objectives

Upon completion of this segment you will be able to:

1. Define, in your own words, the following SPEED terms:
   a) command terminator
   b) command delimiter
   c) character
   d) string
   e) line
   f) page
   g) window
   h) edit buffer

2. List, in order, the steps in a typical edit cycle.

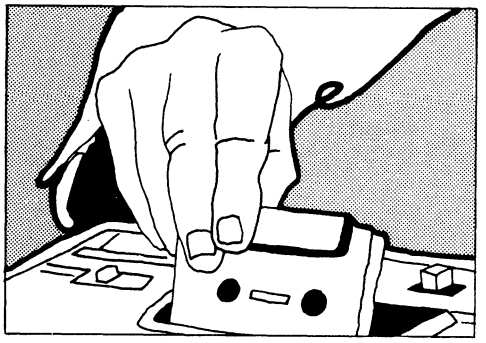
3. State the three methods for erasing command line characters prior to execution. Describe the situations for using each method.

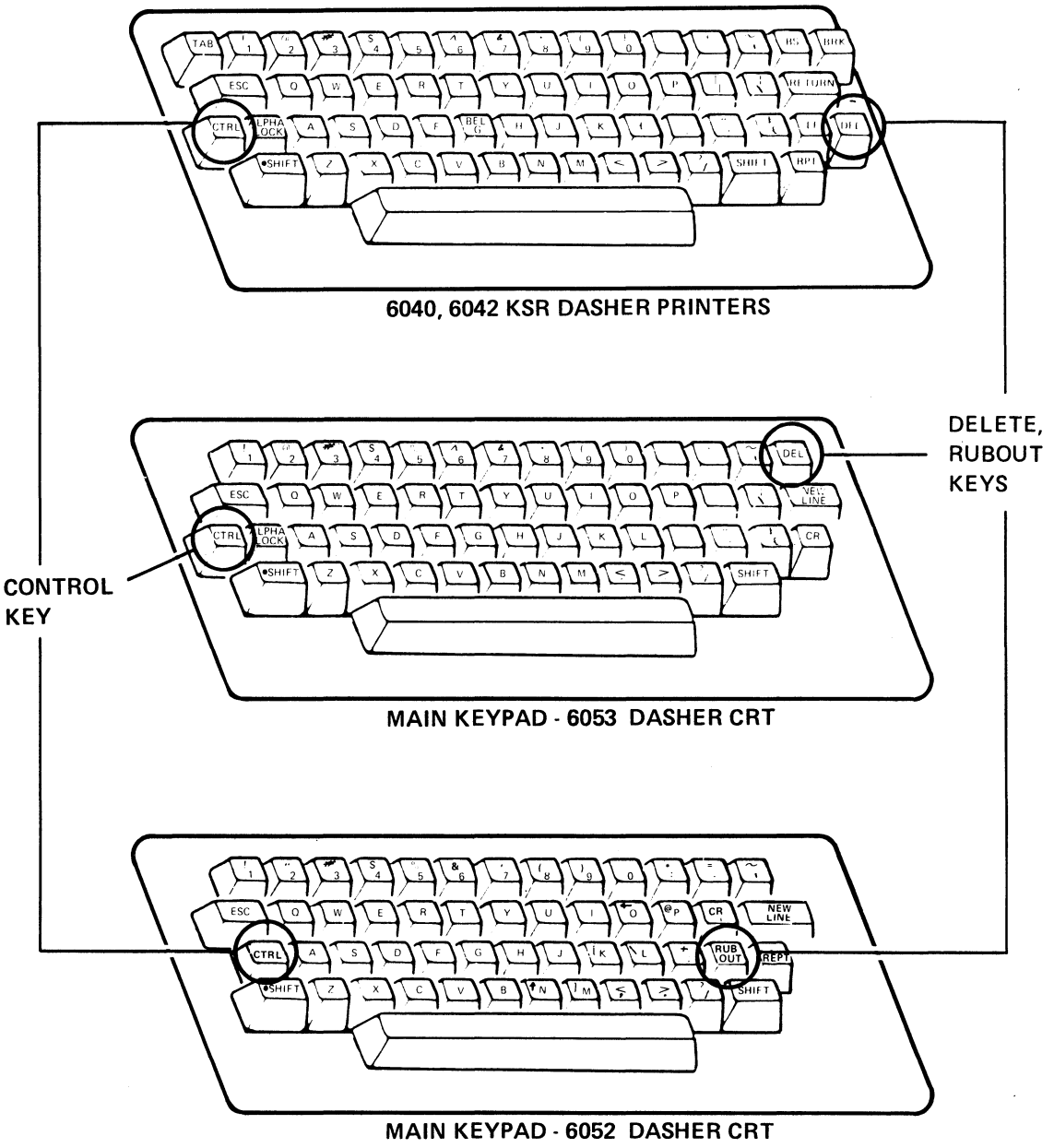
4. State the methods for delimiting and terminating command lines.

5. Given an editing situation, draw and label the simple memory configuration.

Directions

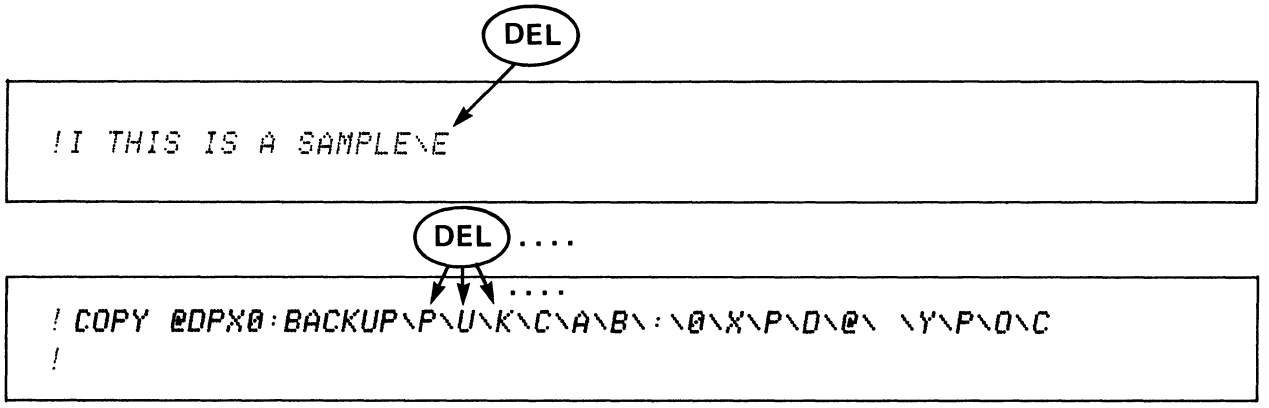
Turn to figure 3-1 in your *Student Guide* and listen to the tape for this segment of Module Three.





CONTROL & DELETE KEYS

Figure 3-1



HARD-COPY DASHER TERMINAL DELETES

Figure 3-2

```
$$
!I THIS IS AN EXAMPLE OF CTRL-U ^U
```

```
$$
!I HELLO, THIS IS A TEST^U
```

CTRL–U SPEED LINE DELETION

Figure 3-3

```
$$
!I THIS IS AN  EXAMPLE OF THE↓
CONTROL-C, CONTROL-A COMBINATION ^C^A

!
```

```
 $$
!I THIS IS AN EXAMPLE OF THE↓
CONTROL-C, CONTROL-A COMBINATION↓
THIS TEST IS SPONSORED BY↓
DATA GENERAL CORPORATION'S↓
INSTRUCTIONAL DEVELOPMENT GROUP.↓
BARRY SMITH PRESIDING OVER THE↓
POPULUS.HERE COMES THE TEST ^C^A

!#T$$

!
```

CTRL–C, CTRL–A DELETION OF A MULTIPLE-LINE SPEED COMMAND.

LINE – STRING OF CHARACTERS ENDING IN A NEW LINE, FORM-FEED, CARRIAGE RETURN OR NULL CHARACTER.

Figure 3-4

TEXT

- One or more ASCII characters
- Upper or lower case

1. CHARACTER

- Occupies one position in Edit Buffer.
- Any single ASCII alphanumeric character.
- TAB, Form-feed, New-line

2. STRING

- Sequence of any ASCII characters

3. SPECIAL CHARACTERS

- Command Terminators ↑D = $$
- String delimiter
  Command Separator ESCape = $

4. LINE

- Maximum length limited by console width.
- Minimum length one character.
- Sequence of characters ending with new-line, carriage return, form-feed, or null character.

5. PAGE

- Sequence of characters ending with a form-feed (CTRL-L)
- Minimum length one character.
- Maximum length limited by memory.

6. WINDOW

- Sequence of lines ending at the window length limit
- Set by Window Command.
- 20 lines for CRT
- 60 lines for printer page.

7. EDIT BUFFER

- Area of memory where Speed manipulates your text.
- Limited by memory size
- 36 Buffers available (A-Z, 0-9)

SPEED TERMINOLOGY

Figure 3-5

1. Invoke SPEED

2. Open files for input and output.

3. Read text from the input file into the edit buffer.

4. Edit text in the edit buffer.

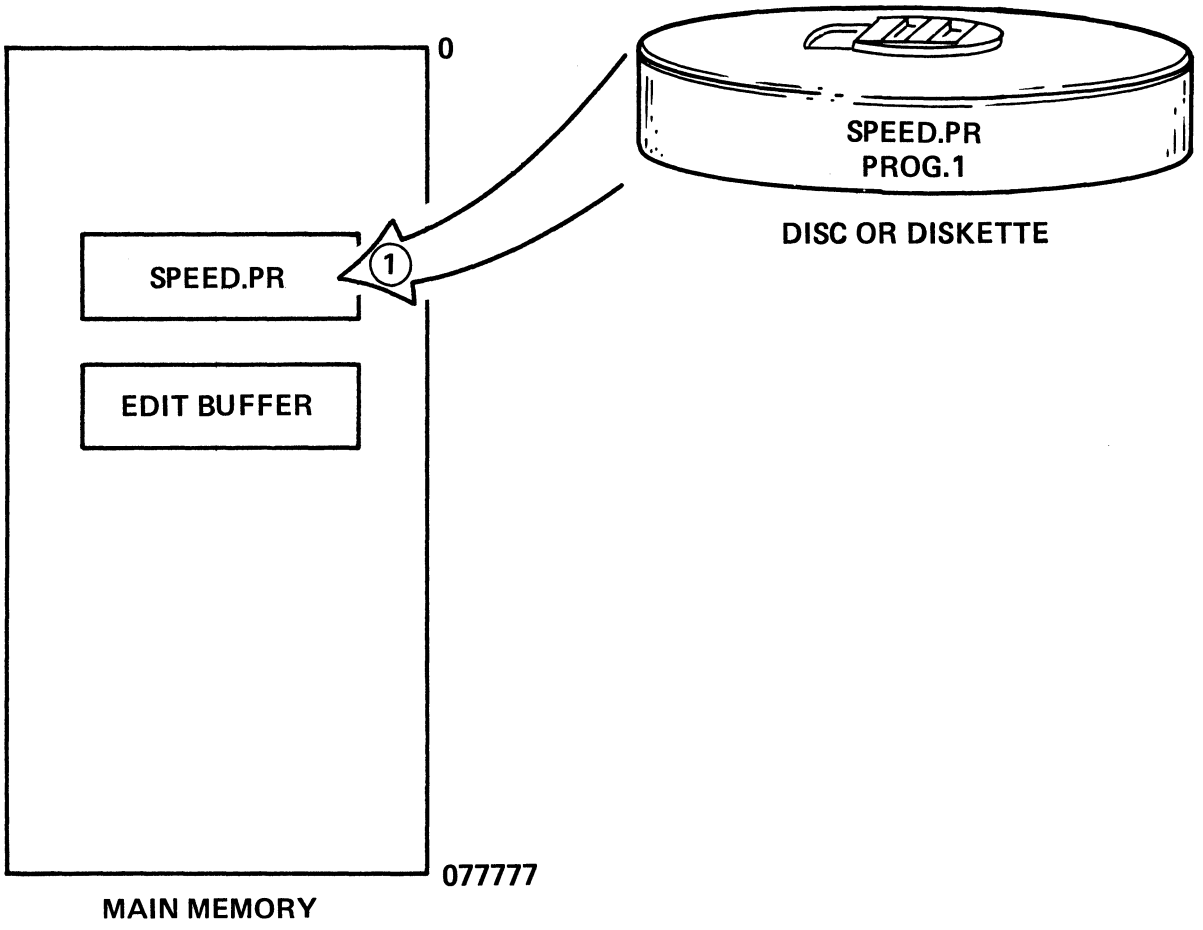
5. Write text from the edit buffer to the output file.

6. Close the input and output files.

7. Exit from SPEED.

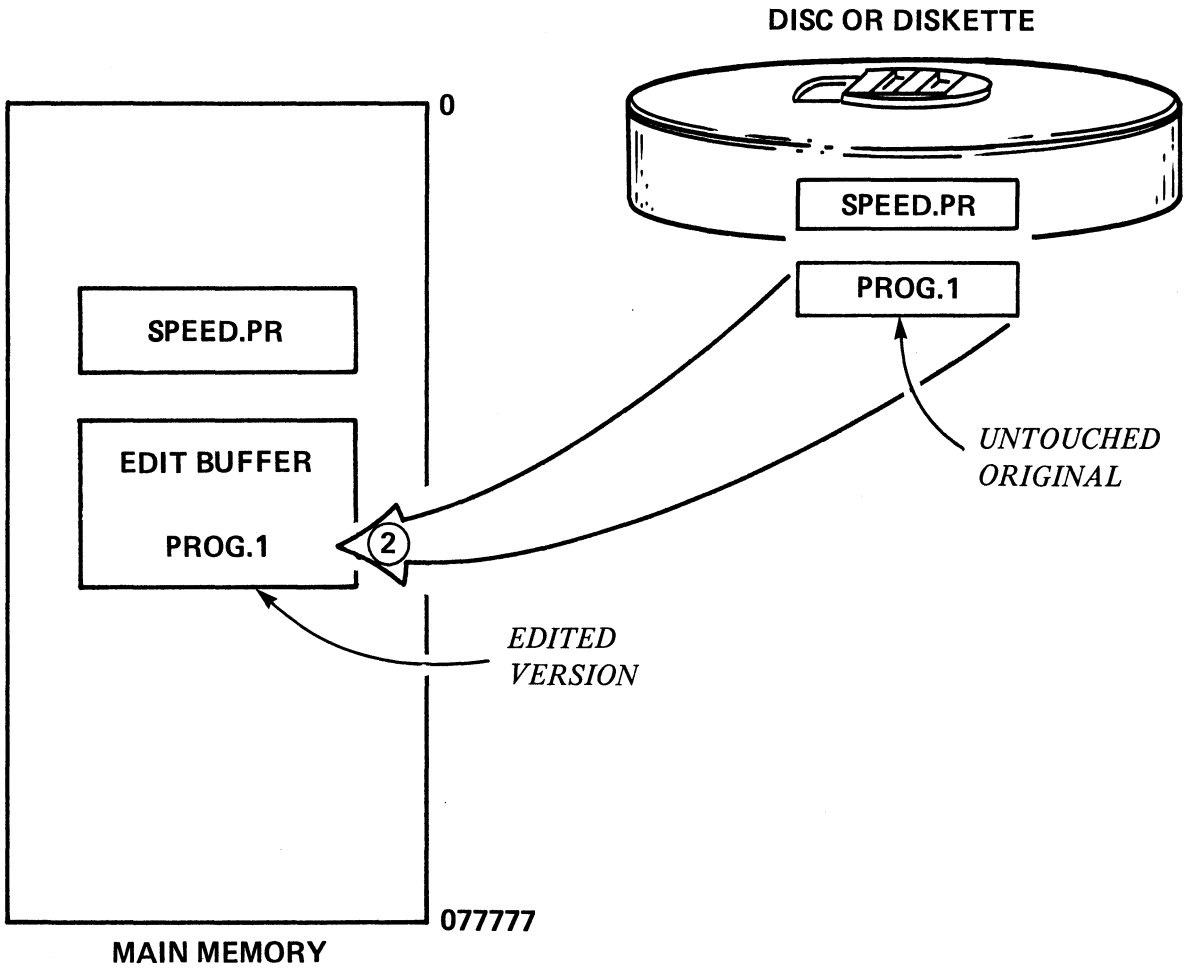
EDIT CYCLE

Figure 3-6



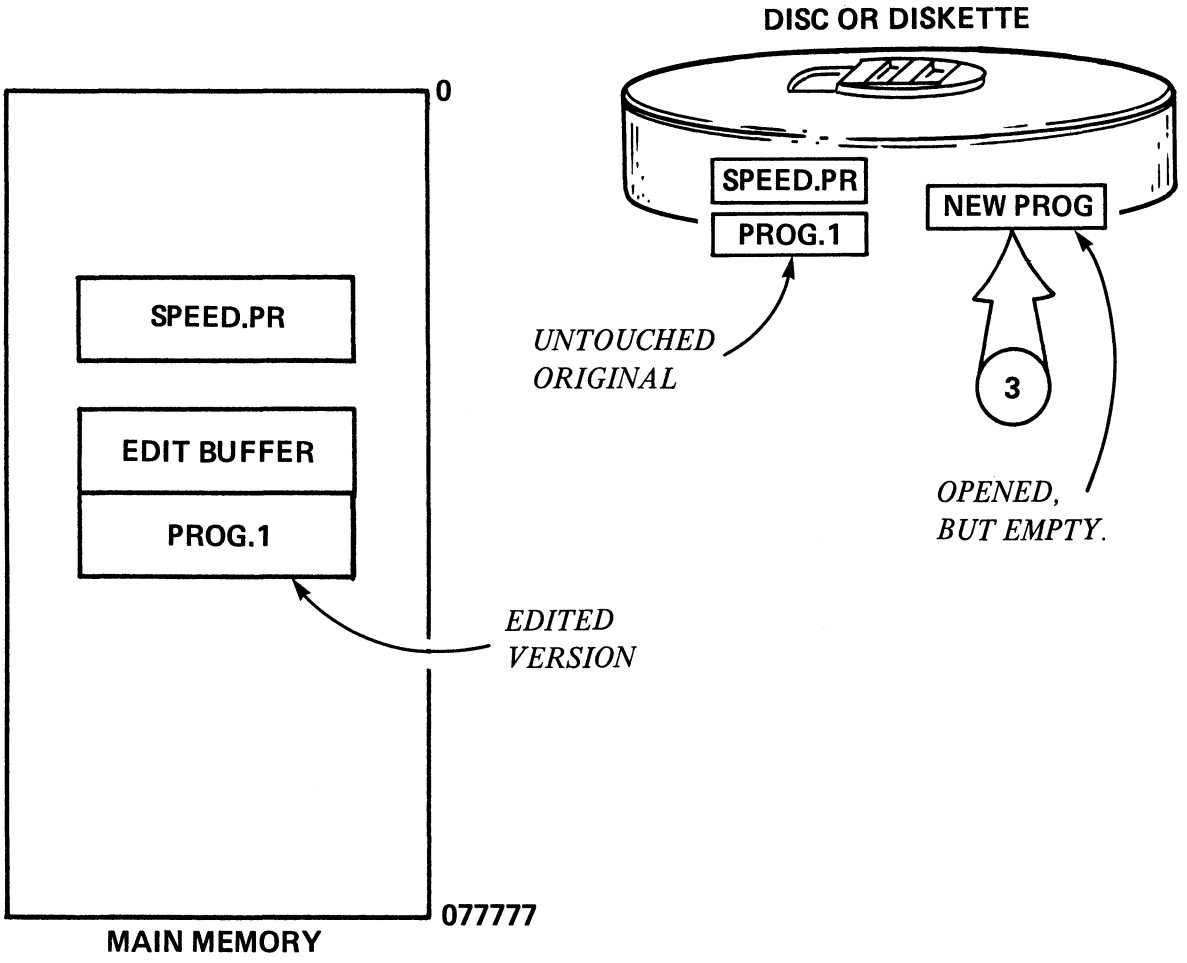
INVOKING SPEED

Figure 3-7



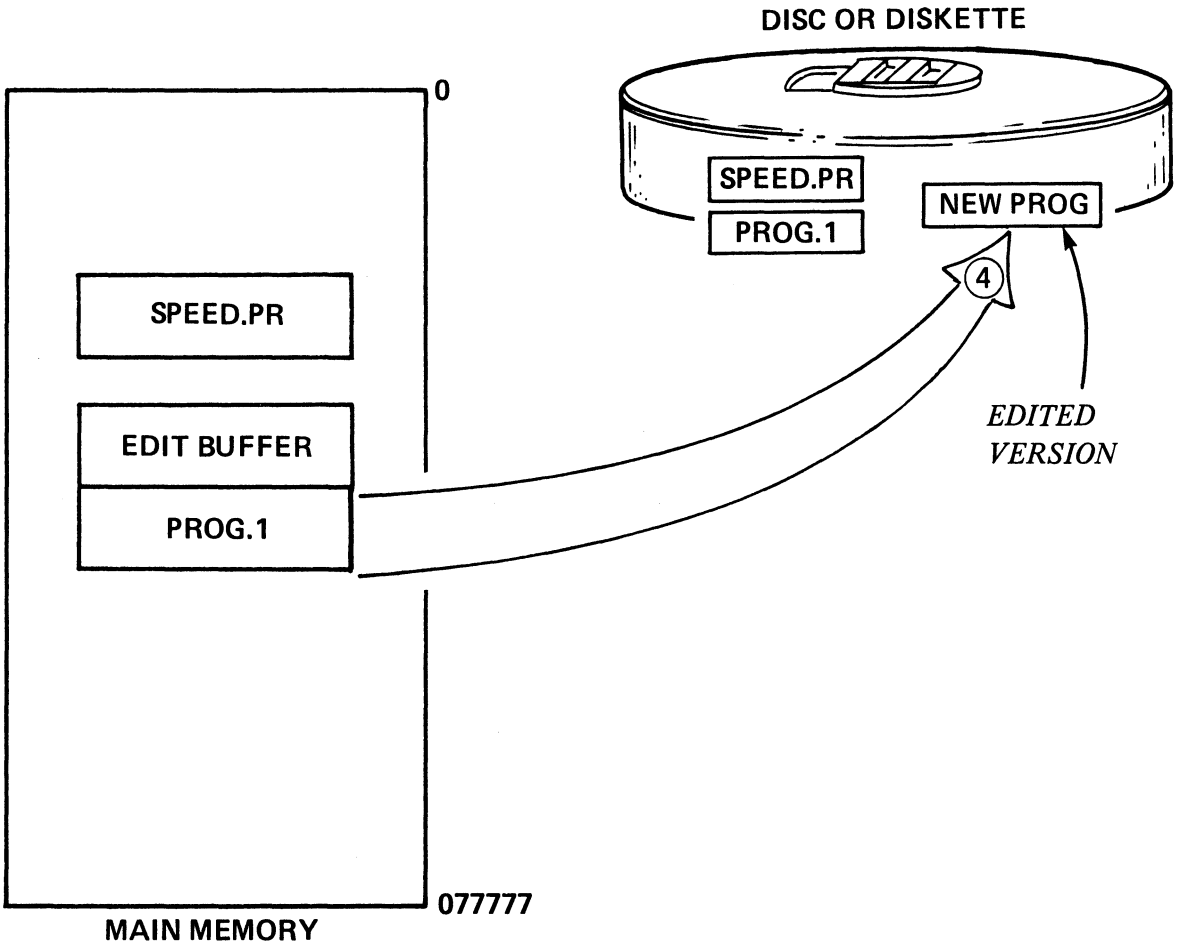
READ A PAGE OF PROG.1 FOR EDITING

Figure 3-8



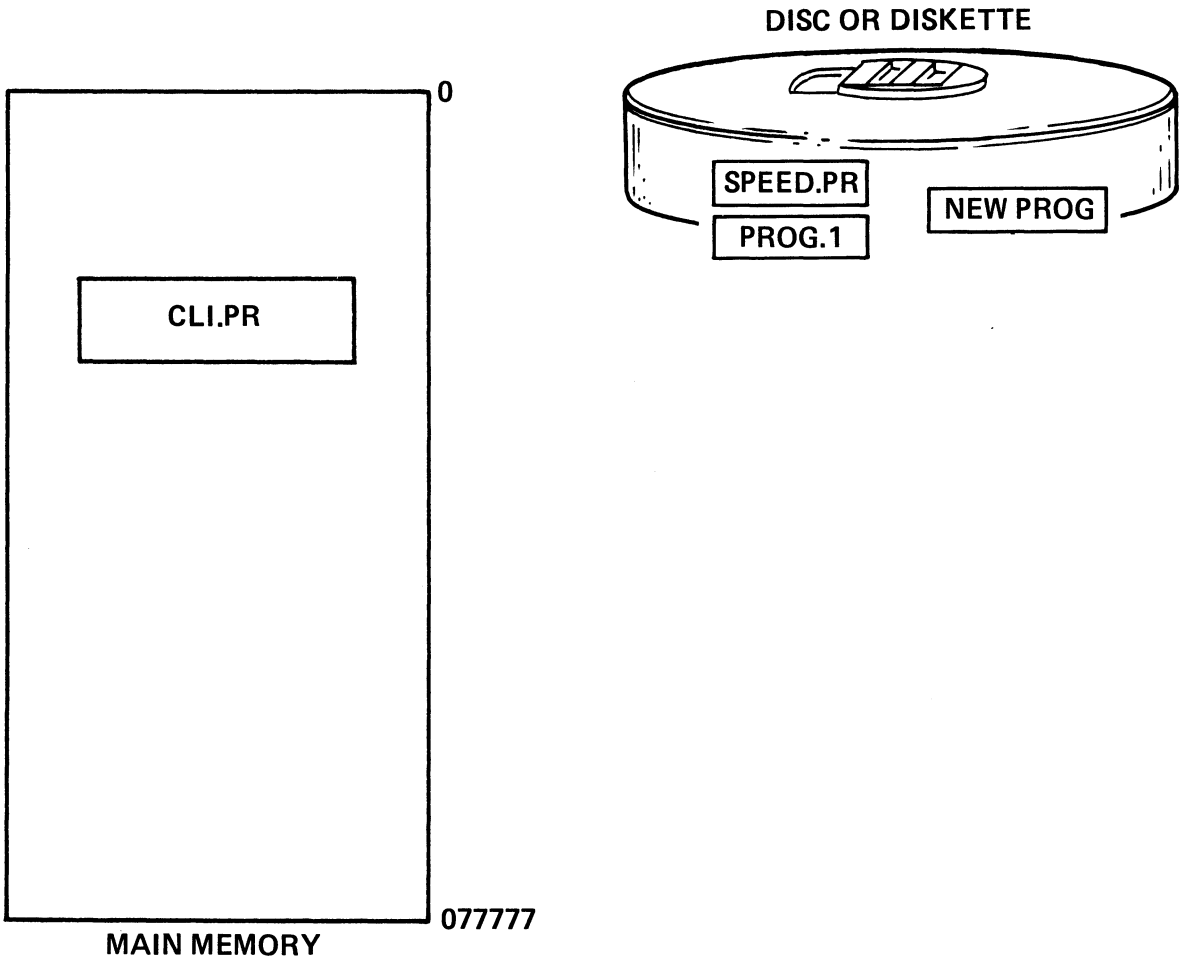
OPEN A FILE FOR OUTPUT

Figure 3-9



WRITE THE EDITED VERSION OUT TO DISC

Figure 3-10



TERMINATE SPEED, REINVOKE CLI

Figure 3-11

TOPICS

- CONSOLE CONTROL
- EDIT CYCLE
- MEMORY CONFIGURATION
- TERMINOLOGY
- CONCEPTS

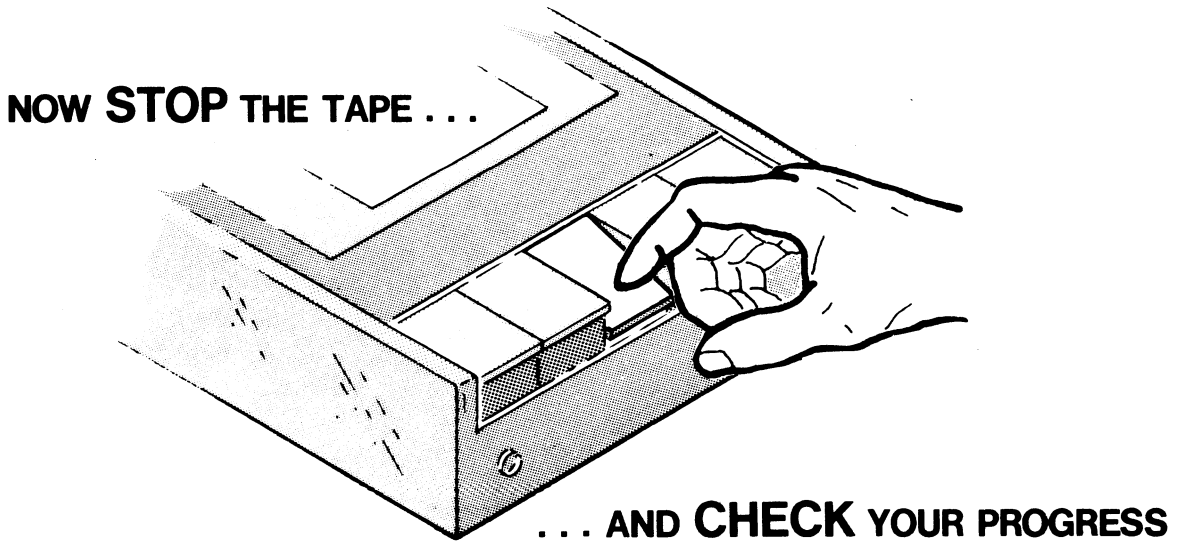


Figure 3-12

SPEED CONCEPTS

QUIZ

Abstract

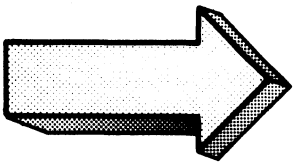
In this exercise you will review basic SPEED terminology, console control, and SPEED concepts.

Directions

Part One

1. Answer each question by circling the letter(s) of the correct answer(s). Note that a question may have more than one correct answer.

2. Check your answers against the Answer Guide following the questions. Make sure you understand the cause of any discrepancies.



1. To *delete* the last character typed, you press:

    A. RUBOUT or DELETE
    B. CONTROL-U
    C. CONTROL-D
    D. ESCAPE
    E. CONTROL-C, CONTROL-A

2. To *delete* one command line (that does not contain any new-line characters), press:

    A. (ONE) RUBOUT or DELETE
    B. CONTROL-U
    C. CONTROL-D
    D. ESCAPE
    E. REPEATED RUBOUTS or DELETES
    F. CONTROL-C, CONTROL-A.

3. To *terminate* a command line and allow SPEED to execute it, you press:

    A. RUBOUT or DELETE
    B. CONTROL-U
    C. CONTROL-D
    D. ESCAPE
    E. REPEATED RUBOUTS or DELETES

4. To *delimit* a command, press:

    A. RUBOUT or DELETE
    B. CONTROL-U
    C. CONTROL-D
    D. ESCAPE

5. SPEED command termination is echoed as:

    A. $$
    B. $
    C. !
    D. #

6. In SPEED, a *character* is:

   A. any ASCII character
   B. Tab
   C. form-feed
   D. new-line

7. A *line* is:

   A. any ASCII character followed by a new-line.
   B. a new-line.
   C. a sequence of characters followed by a new-line.
   D. a sequence of characters followed by a form-feed.

8. A *page* is:

   A. a sequence of characters followed by a new-line.
   B. a sequence of lines followed by a form-feed.
   C. a form-feed.
   D. one line followed by a form-feed.

9. A *window* is

   A. a sequence of characters followed by a new-line.
   B. a sequence of lines followed by a form-feed.
   C. a sequence of lines delimited by the window length limit.
   D. a sequence of lines delimited by a CONTROL-D.

10. The *edit* buffer is

    A. an area of disc or diskette space used for manipulating text.
    B. an area of memory used for manipulating text.
    C. an area of memory used for overlays.
    D. an area of disc or diskette space used for holding files opened for I/O.

11. The RUBOUT or DELETE key is echoed on a hard-copy terminal by:

    A. $$
    B. $
    C. backslash and the last character entered.
    D. ∧U
    E. ∧C∧A.

12. To delete several lines of command text prior to execution, you type:

    A. RUBOUT or DELETE
    B. CONTROL-U
    C. CONTROL-D
    D. CONTROL-C, CONTROL-A.
    E. REPEATED RUBOUTS or DELETES.

Part Two

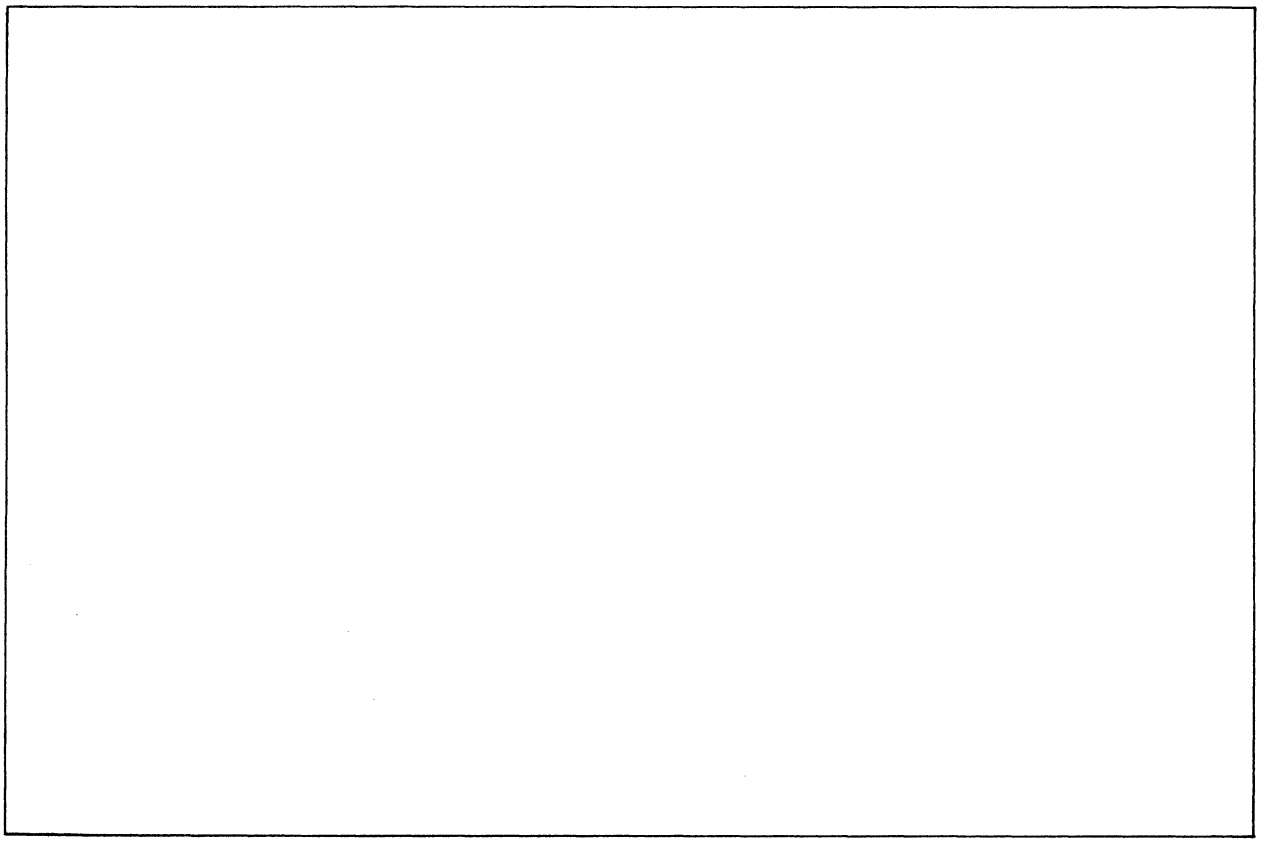
13. Given the following steps in an edit cycle, *list* them in their usual order:

    A. edit text in the edit buffer.
    B. open files for input and output.
    C. close the input and output files.
    D. invoke Speed.
    E. exit from Speed.
    F. Write text from the edit buffer to the output file.
    G. Read text from the input file to the edit buffer.

    1._____ 2._____ 3._____ 4._____ 5._____ 6._____ 7._____

14. Given the following edit situation, draw and label a simple device and memory configuration to illustrate the situation.

    A. SPEED has been invoked.
    B. File EXAMPLE has been opened and a page of text read into Memory.
    C. File EXAMPOUT has been opened for output.



Check your answers
on the following page

SPEED CONCEPTS

QUIZ ANSWERS

1. To *delete* the last character typed, you press:

(A.)	RUBOUT or DELETE		
B.	CONTROL-U	∧U	deletes the entire line.
C.	CONTROL-D	∧D	terminates a SPEED command
D.	ESCAPE	$	delimits SPEED commands
E.	CONTROL-C, CONTROL-A	∧C,∧A	deletes a series of lines.

2. To *delete* one command line (that does not contain any new-line characters), press:

A.	(ONE) RUBOUT or DELETE	DEL	only gets one character.
(B.)	CONTROL-U	∧U	erases an entire line.
C.	CONTROL-D	∧D	terminates a SPEED command.
D.	ESCAPE	ESC.	delimits SPEED commands
(E.)	REPEATED RUBOUTS or DELETES		tedious, but O.K.
F.	CONTROL-C, CONTROL-A.	∧C ∧A	Yes, works here, too.

3. To *terminate* a command line and allow SPEED to execute it, you press:

A.	RUBOUT or DELETE	Refer to 1 and 2, above.
B.	CONTROL-U	
(C.)	CONTROL-D	
D.	ESCAPE	
E.	REPEATED RUBOUTS or DELETES	

4. To *delimit* a command, press:

A.	RUBOUT or DELETE	DEL	erases one character.
B.	CONTROL-U	∧U	erases a line with no new-lines
C.	CONTROL-D	∧D	terminates the command.
(D.)	ESCAPE	$	Yes!

5. SPEED command termination is echoed as:

(A.)	$$	$$ CONTROL-D.
B.	$	$ ESCAPE echo.
C.	!	! SPEED prompt
D.	#	# command modifier meaning "all".

6. In SPEED, a *character* is:

(A.)	any ASCII character	Each choice fits the question. Note that each one occupies one space in the buffer.
(B.)	Tab	
(C.)	form-feed	
(D.)	new-line	

7. A *line* is:

(A.)	any ASCII character followed by a new-line.	Only D fails this criterion.
(B.)	a new-line.	
(C.)	a sequence of characters followed by a new-line.	
D.	a sequence of characters followed by a form-feed.	No, this defines a page.

8. A *page* is:

A.	a sequence of characters followed by a new-line.	Defines a line.
(B.)	a sequence of lines followed by a form-feed.	The formal definition
(C.)	a form-feed	Creates a blank page.
(D.)	one-line followed by a form-feed	Creates a one-line page.

9. A *window* is

A.	a sequence of characters followed by a new-line.	Defines a line.
B.	a sequence of lines followed by a form-feed.	Defines a page.
(C.)	a sequence of lines delimited by the window length limit.	
D.	CONTROL-D	No, a multiple-line command.

10. The *edit buffer* is

A.	an area of disc or diskette space used for manipulating text.	No
(B.)	an area of memory used for manipulating text.	
C.	an area of memory used for overlays.	No, this defines a node.
D.	an area of disc or diskette space used for holding files opened for I/O.	No

11. The RUBOUT or DELETE key is echoed on a hard-copy terminal by:

A.	$$	$$ CONTROL-D echo.
B.	$	$ ESCAPE echo.
(C.)	backslash and the last character entered.	Yes
D.	∧U	One-line eraser: CONTROL-U.
E.	∧C∧A	Multiple-line eraser: CONTROL-C, CONTROL-D.

12. To delete several lines of command text prior to execution you type:

A.	RUBOUT or DELETE	Deletes one character.
B.	CONTROL-U	∧U Deletes one line.
C.	CONTROL-D	∧D Terminates a SPEED command.
(D.)	CONTROL-C, CONTROL-A	Yes
E.	REPEATED RUBOUTS or DELETES	Only works for one line.

Part Two

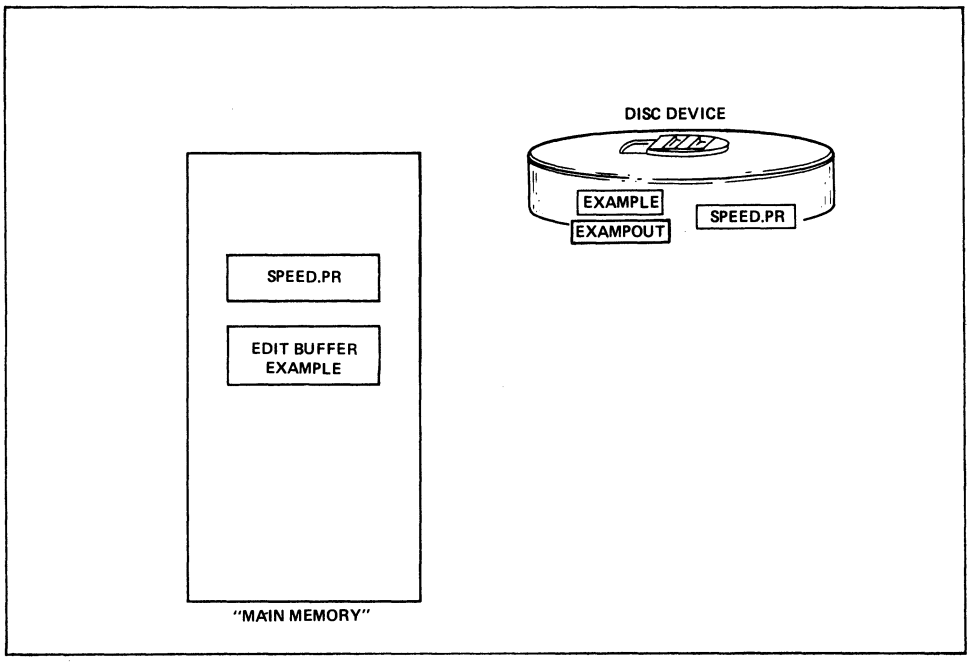
13. Given the following steps in an edit cycle, *list* them in their usual order:

    A. edit text in the edit buffer.
    B. open files for input and output.
    C. close the input and output files.
    D. invoke Speed.
    E. exit from Speed.
    F. Write text from the edit buffer to the output file.
    G. Read text from the input file to the edit buffer.

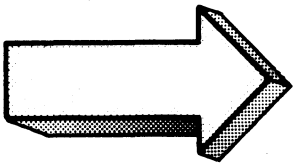
    1. D 2. B 3. G 4. A 5. F 6. C 7. E

14. Given the following edit situation, draw and label a simple device and memory configuration to illustrate the situation.

    A. SPEED has been invoked.
    B. File EXAMPLE has been opened and a page of text read into Memory.
    C. File EXAMPOUT has been opened for output.



A SCORE OF 12 CORRECT ANSWERS OUT OF THE 14 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.



FILE COMMANDS

Abstract

This segment of Module Three describes SPEED commands used for opening, closing, reading, and writing files during an edit session.

Objectives

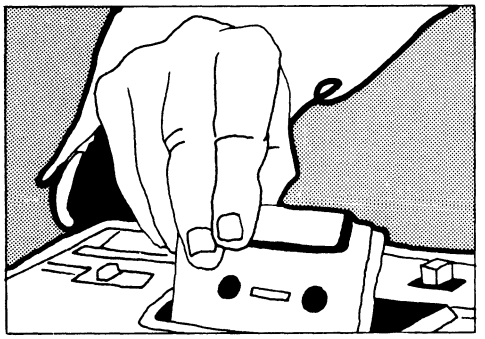
Upon completion of this segment you will be able to:

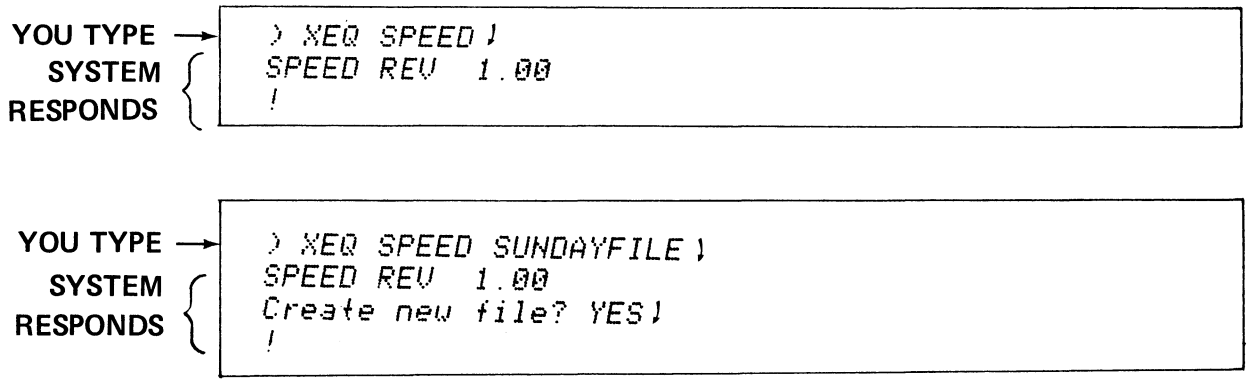
1. State and describe the function of commands for:
   a) invoking SPEED
   b) exiting SPEED
   c) opening files
   d) closing files
   e) reading files
   f) writing files
   g) displaying file status
   h) window and page mode
   i) display mode
   j) update mode

2. Given a simple editing situation, enter the appropriate file handling command for that situation.

Directions

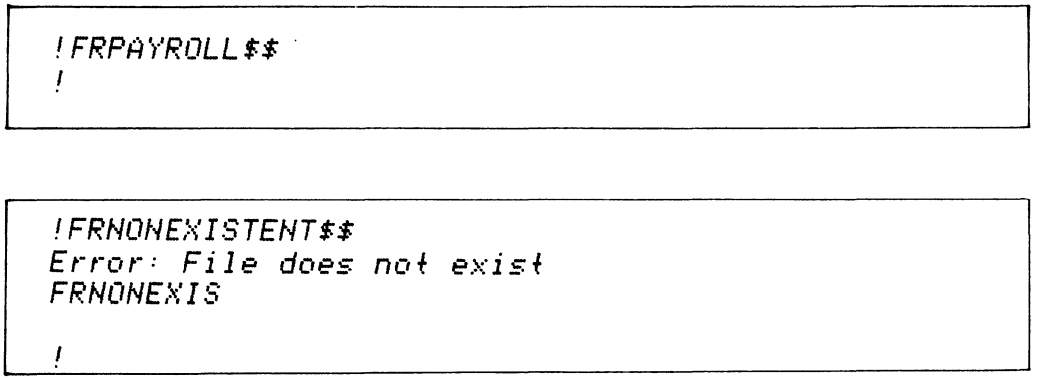
Turn to Figure 3-13 in the *Student Guide* and listen to the tape for Module Three, segment two.





INVOKING SPEED

Figure 3-13



FRpathname CTRL-D **OPEN a file for READING**

FR COMMAND

Figure 3-14

```
!FWPAYROLL.NEW$$
!
```

```
!FWOLDFILE$$
Error: File already exists
FWOLDFILE

!
```

FWpathname$$ open a file for writing.

FW COMMAND

Figure 3-15

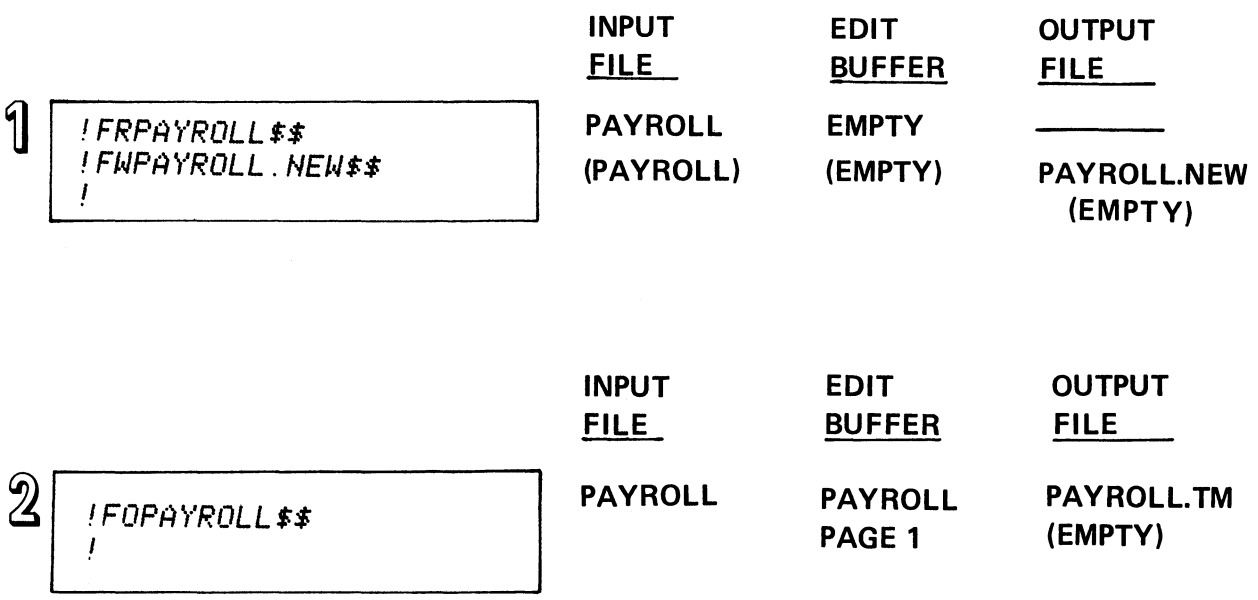
```
!FOPAYROLL$$
!
```

```
$$
!FONONEXISTENT$$
Error: FILE DOES NOT EXIST
FONONEXIS
```

FOpathname$$ **OPEN pathname for Reading**
OPEN pathname for Writing
READ one page of Pathname.

FO COMMAND

Figure 3-16



FR open for Read

FW open for Write

FO open for Read
open for Write (.TM)
read a page
update mode ON.

FR, FW, FO COMMANDS

Figure 3-17

```
!FU$$
!
```

```
!FUFOLLY$$
Error: File does not exist
FOLLY

!
```

FC$$ CLOSE ALL FILES.

THE FC COMMAND

Figure 3-18

```
!FC$$
!
```

```
!FCFOLLY$$
Error: File does not exist
FOLLY

!
```

FU$$ WRITE current buffer to output file

WRITE remainder of input file to output file
CLOSE all global files.
CLEARS current buffer.

THE FU COMMAND

Figure 3-19

```
!FB$$
```

```
!FBINCOME$$
!
```

FB filename$$ — **WRITE current buffer to output file**
WRITE remainder of input file to output file.
CLOSE all global files.
CLEAR current buffer
NAME output file to original file.

Figure 3-20

		INPUT FILE	EDIT BUFFER	OUTPUT FILE
1	!FC$$!	CLOSED	UNCHANGED	CLOSED
2	!FU$$!	CLOSED & DELETED	EMPTY	EDITED TEXT & REST OF INPUT FILE & RENAMED
3	!FB$$!	CLOSED & RENAMED AS .BU	EMPTY	EDITED TEXT & REST OF INPUT FILE & RENAMED

CLOSE COMMANDS

Figure 3-21

```
!Y$$


!Y$$
Confirm (Y-command) ? YES
!
```

Y$$ YANK (read) a page or window into the buffer.

```
) XEQ SPEED/D)
.
 .
  .
   .
```

/D...switch turns on display mode READS a page or window of text.

TEXT READ COMMANDS

Figure 3-22

```
!5WD$$
!

!0WD$$
!
```

```
) XEQ SPEED/D )
.
 .
  .
   .
```

WD Set or display the display mode.
5 WD display 10 lines of text.

DISPLAY MODE

Figure 3-23

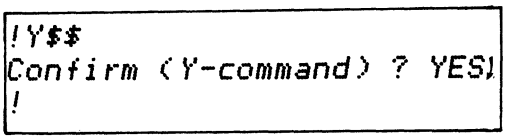
```
!A$$
!
```

```
!A$$
Error: No more characters in input file
A$$

!
```

A$$ READ in a page or window of text
APPEND the page or window to the current buffer.

APPEND COMMAND

Figure 3-24

		INPUT FILE	EDIT BUFFER	OUTPUT FILE
1		ALREADY OPEN & UNCHANGED	INPUT ONE PAGE	UNCHANGED

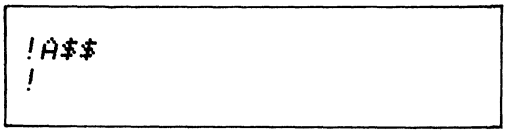
		INPUT FILE	EDIT BUFFER	OUTPUT FILE
2		OPEN, UNCHANGED	PAGE 1 & PAGE 2	UNCHANGED

Figure 3-25

```
!P$$
!P$$
Error: No open file
P$$
!
```

```
!5P$$
!
!PW$$
!
```

P$$ WRITE current buffer to the output file.
nP$$ WRITE n lines of current buffer to output file.

WRITE COMMANDS

Figure 3-26

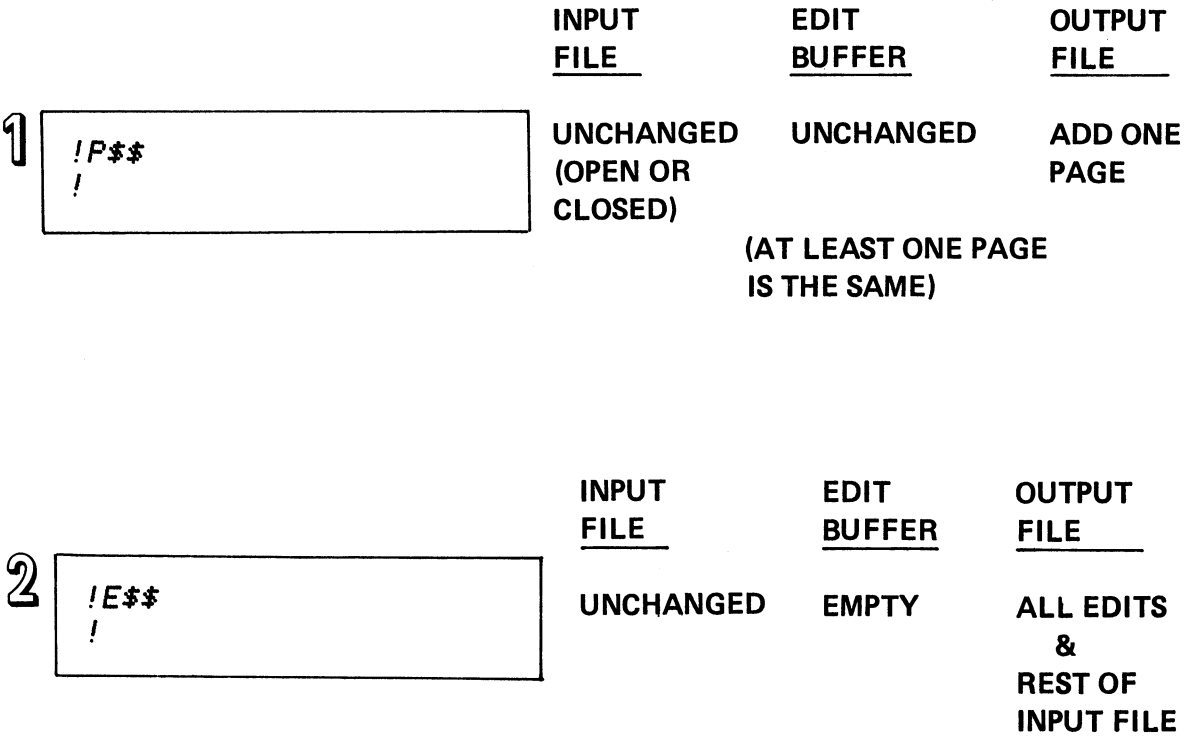
```
!E$$
!
```

```
!E$$
Error: No open file
E$$

!
```

E$$ WRITE current buffer to the output file.
WRITE remainder of input file to output file.

EJECT COMMAND

Figure 3-27



WRITE COMMANDS

Figure 3-28

1

```
!R$$
!
```

2

```
!R$$
Error: No open file
R$$

!
```

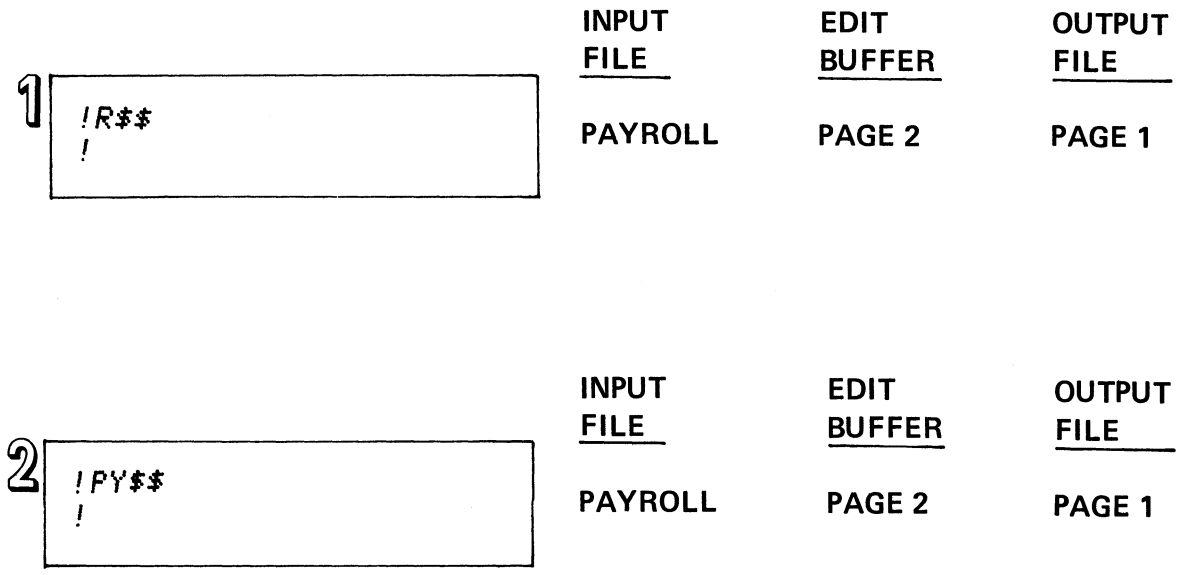
3

```
!R$$
Error: No more characters in input file
R$$

!
```

R$$ WRITE current buffer to output file.
READ the next page from the input file.

R COMMAND

Figure 3-29



R, PY COMMANDS

Figure 3-30

```
!F?$$
Global:
        Input  File - @DPX1:SUNDAYFILE
        Output File - None
Local:
        Input  File - None
        Output File - None
!
```

F?$$ **List open, global and local, input and output files.**

FILESTATUS COMMAND

Figure 3-31

```
!H$$
Confirm? YES
)
```

EXIT SPEED

Figure 3-32

INVOKE & EXIT

XEQ SPEED	invoke SPEED.
XEQ SPEED filename	invoke and open file
XEQ SPEED/D	invoke with display-on
H$$	exit SPEED

OPEN & CLOSE FILES

FR	file read
FW	file write
FC	files close
FO	file open
FU	file update
FB	file backup
F?	file status

READ & WRITE FILES

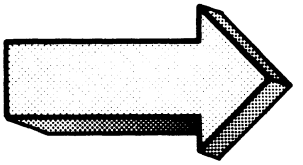
Y	yank
P	put
A	append
E	eject
R	put and yank

MODES

WD	window/page mode
WM	display mode

MODULE 3, FILE COMMANDS SEGMENT

Figure 3-33



FILE COMMANDS

QUIZ

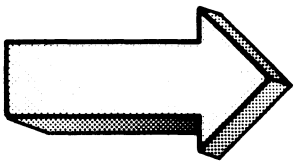
Abstract

This Quiz reviews the CLI and Speed commands used for managing input and output files during a Speed edit session.

Directions

1. Circle the letter or letters of the correct answer(s) for the following questions. A question may have more than one correct answer.

2. Check your answers against the Answer Guide following the Quiz.



Part I. Multiple Choice

1. A command for invoking SPEED from CLI is:

   A. XEQ SPEED ↵
   B. X SPEED ↵
   C. SPEED ↵
   D. EXECUTE SPEED ↵

2. The command line for opening the INVENTORY file for editing is:

   A. (from CLI) XEQ SPEED INVENTORY ↵
   B. (from SPEED) FRINVENTORY ↵
   C. (from SPEED) FRINVENTORY$$
   D. (from SPEED) FWINVENTORY$$

3. A command for opening INVENTORY for outputting edited text is:

   A. (from CLI) XEQ SPEED INVENTORY ↵
   B. (from SPEED) FRINVENTORY ↵
   C. (from SPEED) FRINVENTORY$$
   D. (from SPEED) FWINVENTORY$$

4. A command for closing input and output files is:

   A. FR$$
   B. FW$$
   C. FC$$
   D. E$$

5. The command(s) for writing data to an output file is(are):

   A. FW$$
   B. W$$
   C. P$$
   D. E$$

6. The command(s) for reading data from an input file to the edit buffer is(are):

   A. ER$$
   B. R$$
   C. Y$$
   D. P$$

7. The command(s) for outputting the current contents of the edit buffer and reading in the next page or window length is(are):

   A. FWFR$$
   B. PY$$
   C. R$$
   D. IO$$

8. The command for displaying the status of files in SPEED is:

   A. FILESTATUS ⌇
   B. F?$$
   C. WHA?$$
   D. HUH?$$

9. The command for terminating SPEED and returning to the calling program is:

   A. X$$
   B. H$$
   C. E$$
   D. BYE$$

10. A command that writes out the edit buffer and rest of the input file, clears the edit buffer, closes files, renames the original file as a .BU, and renames the edited copy as the original is:

    A. FO$$
    B. FN$$
    C. FB$$
    D. FU$$

11. A command that writes out the edit buffer and the rest of the input file, clears the edit buffer, closes files, deletes the original, and renames the edited copy as the original is:

    A. FO$$
    B. FN$$
    C. FB$$
    D. FU$$

12. A command to open DOOR, yank in a page, and open DOOR.TM for output is:

    A. FRDOOR$$
    B. FWDOOR$$
    C. FODOOR$$
    D. FODOOR.TM$$

13. A command for reading in a page of text to the edit buffer without writing over the current contents of the buffer:

    A. S$$
    B. P$$
    C. K$$
    D. A$$

14. Display mode of 10 may be set with:

    A. (from CLI)) X SPEED/D
    B. (from SPEED) 10WD$$
    C. (from SPEED) 5WD$$
    D. (from SPEED) WD=10$$

15. Page mode may be set with:

    A. (from SPEED) PW$$
    B. (from SPEED) ØWM$$
    C. (from SPEED) PM$$
    D. (from CLI) XEQ SPEED

Part II. Matching Columns

Match the command in the left column with the function in the right column.

1._____WD	A.	Exit SPEED.
2._____WM	B.	open a file for input.
3._____FR	C.	open a file for output.
4._____FU	D.	close all files.
5._____A	E.	Open input files, yank a page, create file .TM and open for output.
6._____H	F.	Copy buffer and rest of input to output file, close files, delete original, rename output to original.
7._____FW		
8._____E	G.	Copy buffer and rest of input to output file, close files, rename input to .BU, rename output to original name.
9._____FC		
10._____R	H.	Display pathnames of open files.
11._____FO	I.	Read in one page or window of text.
12._____P	J.	Write buffer to output file with a form-feed.
13._____FB	K.	Read in one page or window and add it to current buffer.
14._____Y	L.	Write out current buffer and rest of input file to output file.
15._____F?	M	Write out current buffer with a form feed, read in one page or window.
	N.	Set display mode.
	O.	Set page or window mode.

NOW CHECK YOUR ANSWERS
ON THE FOLLOWING PAGES

FILE COMMANDS

QUIZ ANSWERS

1. A command for invoking SPEED from CLI is:

   (A.) XEQ SPEED ↵
   Yes

   (B.) X SPEED ↵
   Yes, abbreviate XEQ.

   C. SPEED ↵
   Will work only if SPEED.CLI exists.

   (D.) EXECUTE SPEED ↵
   Yes.

2. The command for opening the INVENTORY file for editing is:

   (A.) (from CLI) XEQ SPEED INVENTORY ↵
   Inventory is the input file.

   B. (from SPEED) FRINVENTORY ↵
   Wrong terminator.

   (C.) (from SPEED) FRINVENTORY$$
   Same as A.

   D. (from SPEED) FWINVENTORY$$
   Opens for output.

3. A command for opening INVENTORY for outputting edited text is:

   (A.) (from CLI) XEQ SPEED INVENTORY ↵
   Opens INVENTORY for input and output.

   B. (from SPEED) FRINVENTORY ↵
   No, only input and wrong terminator.

   C. (from SPEED) FRINVENTORY$$
   No, only input.

   (D.) (from SPEED) FWINVENTORY$$
   Yes!

4. A command for closing input and output files is:

A. FR$$
No, command error.

B. FW$$
Error command requires arguments.

(C.) FC$$
Yes, closes input and output files.

D. E$$
Outputs, but does not close.

5. The command(s) for writing data to an output file is (are):

A. FW$$
Error: Invalid character in pathname; also, wrong one.

B. W$$
Error: Illegal command

(C.) P$$
Yes, entire buffer with a Form Feed.

(D.) E$$
Yes, buffer plus rest of input file.

6. The command(s) for reading data from an input file to the edit buffer is (are):

A. ER$$ Too much: E will read to end of input file and put everything out; R will have nothing to read.

(B.) R$$
Puts out current buffer, reads one page.

(C.) Y$$
Reads in one page or window (over current buffer).

D. P$$
Puts out the current buffer.

7. The command(s) for outputting the current contents of the edit buffer and reading in the next page or window length is (are):

A. FWFR$$

No, sets up file "FR" for output.

(B.) PY$$

Puts buffer out, yanks in a page or window.

(C.) R$$

Same as B.

D. IO$$

No, inserts the character "O" into buffer.

8. The command for displaying the status of files in SPEED is:

A. FILESTATUS ⤵

No, only in CLI.

(B.) F?$$

Yes: input, output, and Update mode are displayed.

C. WHA?$$

No, illegal character, then exits SPEED.

D. HUH?$$

Exits SPEED (if buffer is empty, if not, fails to exit).

9. The command for terminating SPEED and returning to the calling program is:

A. X$$

"X" may be used to execute CLI commands and Programs from SPEED.

(B.) H$$

Yes, back to parent, usually CLI.

C. E$$

No, writes out buffer and rest of input file.

D. BYE$$

B is an illegal command, Y and E are not executed.

10. A command that writes out the edit buffer and rest of the input file, clears the edit buffer, closes files, renames the original file as a .BU, and renames the edited copy as the original is:

A. FO$$

No, opens files, yanks a page and turns on update mode.

B. FN$$

No, illegal command.

(C.) FB$$

Yes.

D. FU$$

Almost, except original input is deleted and output is given original's name.

11. A command that writes out the edit buffer and the rest of the input file, clears the edit buffer, closes files, deletes the original, and renames the edited copy as the original is:

A. FO$$

No, opens files; yanks; turns on update mode.

B. FN$$

Error: Illegal command.

C. FB$$

No, see question number 10 above.

(D.) FU$$

Yes!

12. A command to open DOOR, yank in a page, and open DOOR.TM for output is:

A. FRDOOR$$

No, only opens DOOR for input.

B. FWDOOR$$

No, only opens DOOR for output.

(C.) FODOOR$$

Yes.

D. FODOOR.TM$$

No, .TM changes filenames.

13. A command for reading in a page of text to the edit buffer without writing over the current contents of the buffer:

A. S$$
No, a search command error.

B. P$$
No, puts out the buffer.

C. K$$
No, kills one line in the buffer up to the C. P.

(D.) A$$
Yes, append

14. Display mode of 10 may be set with:

(A.) (from CLI)) X SPEED/D ↓
Yes.

(B.) (from SPEED) 10WD$$
Yes, 20 lines are displayed

C. (from SPEED) 5WD$$
No, mode = 5, here. (10 lines displayed)

D. (from SPEED) WD=10$$
No, will return the current mode setting.

15. Page mode may be set with:

A. (from SPEED) PW$$
No, puts out the buffer.

(B.) (from SPEED) 0WM$$
Yes.

C. (from SPEED) PM$$
No, puts out the buffer, then moves C.P.

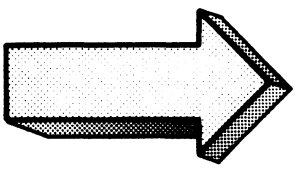
(D.) (from CLI) XEQ SPEED ↓
Yes, defaults to page mode.

Part II Matching Columns

Match the command in the left column with the function in the right column.

1. N WD	A.	Exit SPEED.
2. O WM	B.	Open a file for input.
3. B FR	C.	open a file for output.
4. F FU	D.	close all files.
5. K A	E.	Open input files, yank a page, create file .TM and open for output.
6. A H	F.	Copy buffer and rest of input to output file, close files, delete original, rename output to original.
7. C FW		
8. L E	G.	Copy buffer and rest of input to output file, close files, rename input to .BU, rename output to original name.
9. D FC		
10. M R	H.	Display pathnames of open files.
11. E FO	I.	Read in one page or window of Text.
12. J P	J.	Write buffer to output file with a form-feed.
13. G FB	K.	Read in one page or window and add it to current buffer.
14. I Y	L.	Write out current buffer and rest of input file to output file.
15. H F?	M.	Write out current buffer with a form feed, read in one page or window.
	N.	Set display mode.
	O.	Set page or window mode.

A SCORE OF 26 CORRECT ANSWERS OUT OF THE 30 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.



FILE COMMANDS

LAB EXERCISE

Abstract

This exercise reviews some of the commands covered in the File Commands segment of Module Three.

Directions

This exercise is similar to the previous labs. You may complete it with or without access to an MP/OS system. The steps are:

1. Cover the answer;
2. Read the operation;
3. Reference a solution;
4. Write down the solution;
5. Check your answer against the answer provided;
6. Try the commands on your system;
7. Resolve any discrepancies.

To complete this lab you must have the SPEED text editor on your system. The CLI FILESTATUS command will help here. (Fi SPEED.PR).

You will also need the file LABTEST for this exercise. Use the CREATE command with the /i switch as shown:

```
) CREATE/I LABTEST↓
))THIS IS THE FILE LABTEST↓
))SECOND LINE OF LABTEST↓
))333333333333333333↓
))444444444444444444↓
))555555555555555555↓
))SIXTH AND LAST LINE↓
)))↓

)
```

You are now ready for the lab.

1. Invoke SPEED. Show the commands and anticipated results in the space below:

_____

_____

_____

****

```
) XEQ SPEED)
SPEED REV  1.00
!
```

XEQ SPEED is the short, simple Speed invocation.
XEQ may be abbreviated to XE or X. You may append the IO switch (XEQ SPEED/D) to your command to turn on display mode. You may also specify a filename. Use the simple version for now.

Try it on your system.

SPEED.PR is now ready for execution. The exclamation point is the prompt character.

2. Open the file LABTEST for editing. Show the commands and responses.

_____

_____

_____

****

```
!FRLABTEST$$
!
```

The FR command (file read) opens a file for reading. This assumes that LABTEST exists and does not have the W (write-protected) attribute.

Do it on your system.

Remember that the terminator is CONTROL-D. Note that SPEED did not type out the LABTEST source in this case. (Display mode is NOT on.)

We used the basic open command. You may have specified others. If you did, close everything (FC$$) and use the FR command. This avoids confusion with later questions.

3. Is update mode on? Which files are open? Show the entries and anticipated responses.

_____

_____

_____

_____

_____

_____

_____

_____

****

```
!F?$$
Global:
        Input  File - @DPX1:LABTEST
        Output File - None
Local:
        Input  File - None
        Output File - None
!
```

The F? command reports the filestatus of open files. Note the full pathname of LABTEST, located on a secondary diskette. Also, note the absence of the UPDATE MODE message. Update mode is OFF.

Execute the command on your system.

Your pathname may be different, depending on your system.

You can now read text from LABTEST, but you cannot write it out to an output file.

4. Set up LAB.TWO as the output file. Show the command and anticipated response:

_____

_____

_____

****

```
!FWLABTWO$$
!
```

We used the FW command to open the output file. Do the same on your system. Watch your spacing and new-lines.

Stick to the simple FW command for now.

5. Now what is the status of your edit files? Show all entries *before* you go to your system.

****

```
!F?$$
Global:
        Input  File - @DPX1:LABTEST
        Output File - LABTWO
Local:
        Input  File - None
        Output File - None
!
```

Again, the F?$$ command displays the Speed filestatus information.

Do it on your system.

Note that LABTWO was shown as the output file, but in which directory? The X command helps this, as shown. Look for our LAB files:

```
!XFILESTATUS$$
DIRECTORY @DPX1:

  SUNDAYFILE      PROG.LS         PROGDEBMAP      PROGDEB.PR
  LOGON.CLI       MASM.PS         PROG_ONE.OB     PROGDEB.OB
  PROG_ONE        MESSAGE.PR      PROG_ONE.PR     SPEED.PR
  PROG_ONE.BU     PROGDEB.LS      ?MASM.ST.TMP    LABTEST
                                                  LABTWO
!
```

Try the command on your system.

6. Bring a page of LABTEST into the edit buffer. Show how:

_____

_____

_____

****

```
!Y$$
!
```

The Y$$ command yanks, or reads, a page or window of text from the input file to the edit buffer. Anything in the input buffer is destroyed or written over.

Try it on your system.

Note that nothing was displayed after the Y command. That is, no text was shown on our screen during the yank. There are several ways of handling this, as shown on the next page.

7. Turn on display mode. Set it so that 8 lines will be displayed.

```
!4WD$$
(^)THIS IS THE FILE LABTEST
SECOND LINE OF LABTEST
33333333333333333333
44444444444444444444
55555555555555555555
SIXTH AND LAST LINE


!
```

The nWD$$ command initiates display mode. (n = number of lines to display before and after the C.P.)

Wow! Turning on the display mode causes the display to begin immediately. Our file is less than eight lines, so it is displayed in its entirety. (The ( ) is the character pointer. All edits are made at the current location of the C.P. or character pointer.

The command for determining the current display mode setting is as follows:

```
!WD=$$
4
!
```

Try it.

Note that there are no spaces in the command line. Always multiply the WD value by 2 to get the number of lines to be displayed.

8. We will detail the commands for editing text in your edit buffer in the next segment of Module Three. For now, turn off display mode and verify that it is off:

_____

_____

_____

_____

****

```
!0WD$$
!WD=$$
0
!
```

The zero modifier for the WD command turns off display mode. You then use the WD as an argument for the "=" command to display the new display mode setting.

Do it on your system.

9. For now, write the command for writing the entire buffer to the output file:

_____

_____

_____

****

```
!P$$
!
```

The P$$ command writes the contents of the current buffer to the output file and appends a form-feed.

Try it on your system.

Does the P command clear (release) the edit buffer? Use the following to find out:

```
!#T$$
THIS IS THE FILE LABTEST
SECOND LINE OF LABTEST
33333333333333333333
44444444444444444444
55555555555555555555
SIXTH AND LAST LINE
!
```

"#T" is the command for displaying the contents of the entire edit buffer. It is still there. The P$$ command only copies the buffer out. Repeated P$$ commands will make multiple copies of the buffer in the output file.

10. Now try this. Read in another page from the input file, but do not over-write the edit buffer's current contents. Write the command and anticipated response:

_____

_____

_____

_____

****

```
!A$$
Error: No more characters in input file
A$$

!
```

The A$$ command reads in a page of text from the input file and adds it to the current edit buffer.

The original input file was only one page long. That page was brought in with the Y command.

No harm is done – do it on your system.

Another Y command would yield the same error. The P command, however, would put the buffer out, making the output file two pages long. Both pages would show the same six lines. You can try it if you want.

11. You have used the basic I/O commands. Now close the files. Show all entries:

_____

_____

_____

****

```
!FC$$
!
```

The FC$$ command closes the input and output files.

Note that FC does not anticipate filename arguments.

Do it on your system.

12. What is the status of your edit files now? Show the commands and anticipated responses.

_____

_____

_____

_____

_____

_____

****

```
!F?$$
Global:
        Input  File - None
        Output File - None
Local:
        Input  File - None
        Output File - None
!
```

Once again, the F?$$ command displays the Speed file status.

The FC command closed the input and output files.

Do it on your system.

13. We closed the files. Is there anything left in the edit buffer? How do you find out?

****

```
!#T$$
THIS IS THE FILE LABTEST
SECOND LINE OF LABTEST
33333333333333333333
44444444444444444444
55555555555555555
SIXTH AND LAST LINE
!
```

The #T command displays the contents of the edit buffer. FC does not erase the buffer. You could open another output file (FW) and put the buffer out (P) for another copy.

Try the #T command. Remember that the #T command is not necessary if you set the display mode ON. There is a difference between the two displays. #T displays the entire buffer from beginning to end. Display mode only shows the lines surrounding the character pointer in a specified range.

14. Terminate SPEED.

_____

_____

_____

_____

****

```
!H$$
Confirm? YES)
)
```

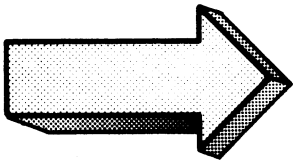
The H$$ command begins the Speed termination process.

SPEED prompts for confirmation only if there is text in the edit buffer.

Do it on your system.

We returned to CLI, as evidenced by the right parenthesis ")" prompt.

THIS CONCLUDES THE SPEED FILE COMMANDS LAB EXERCISE. YOU SHOULD BE ABLE TO MANIPULATE FILES WITH SPEED COMMANDS. NOW CONTINUE TO THE NEXT SEGMENT OF MODULE THREE.



EDIT COMMANDS

Abstract

In this segment of Module Three we discuss various commands for editing text in the edit buffer. Included are commands for searching, displaying, inserting and deleting text, and moving the character pointer.

Objectives

Upon completion of this segment, you will be able to:

A. Given a text editing situation, state an appropriate command or series of commands for solving the editing situation.

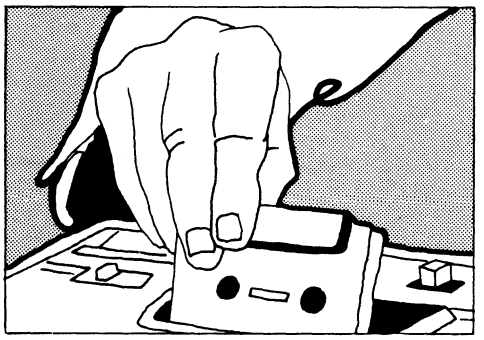
B. State, and describe the function of, the commands for:
   a) moving the character pointer
   b) searching text
   c) inserting text
   d) deleting text

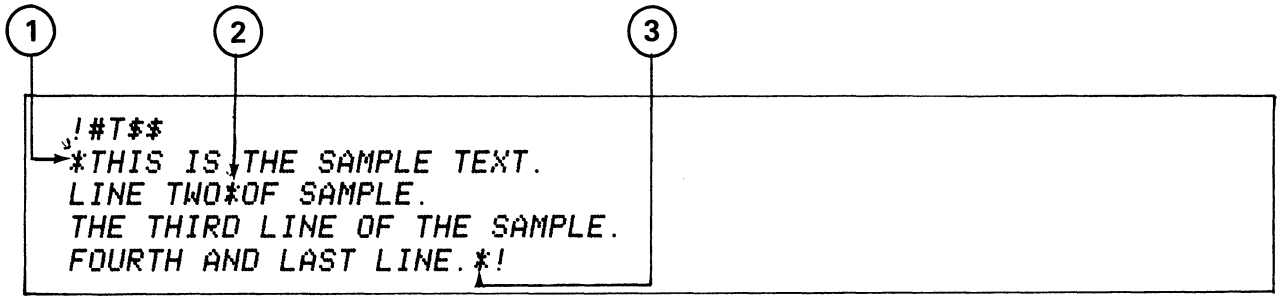
Directions

1. Turn to figure 3-52 on the next page of your *Student Guide* and listen to the tape for this segment of Module Three.

2. Take the Edit Commands Quiz.

3. Try the Text Editor Lab.



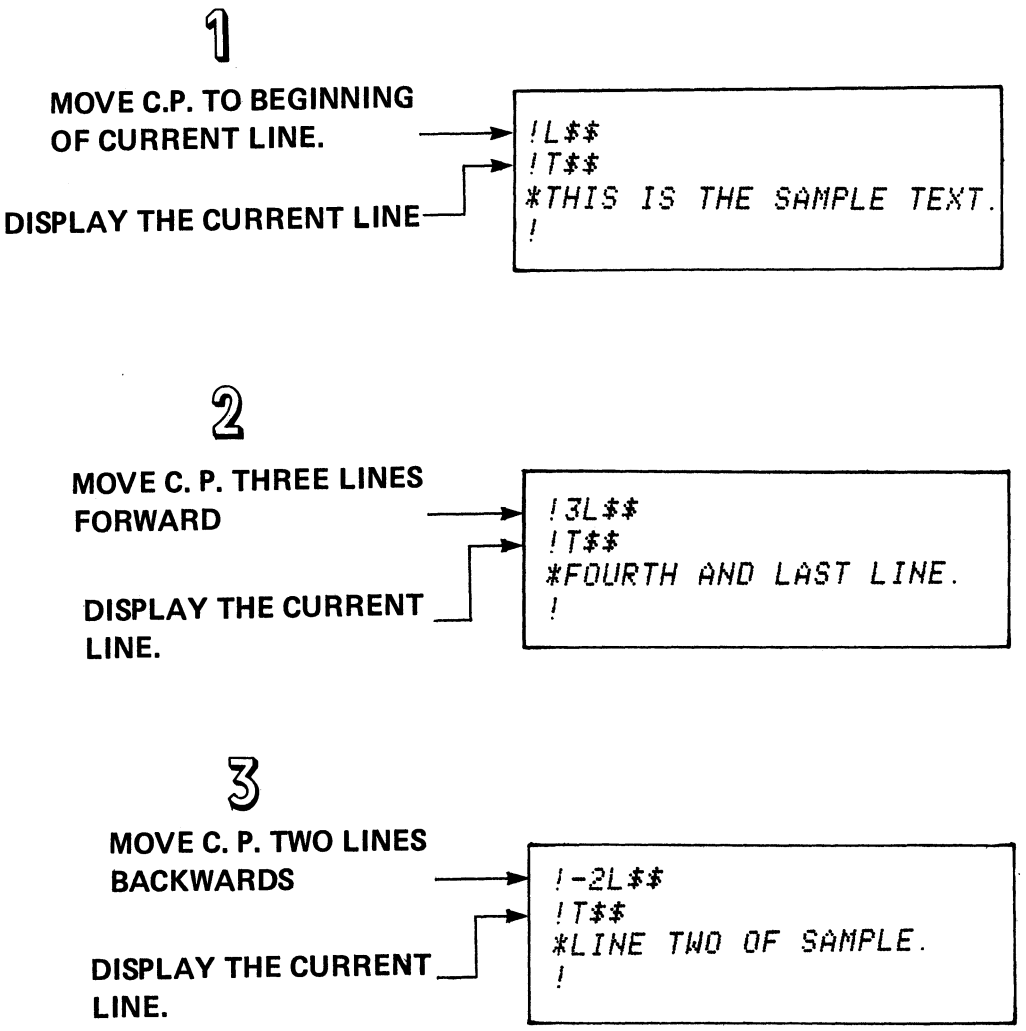


C. P. on DASHER CRT

```
(^)THIS IS A SAMPLE TEXT
LINE TWO OF SAMPLE
THE THIRD LINE OF THE SAMPLE
FOURTH AND LAST LINE
```

C. P. on DASHER HARD–COPY TERMINAL

Figure 3-52



nL$$ move the C.P. n lines from current position.

C.P. MOVER — THE L COMMAND

Figure 3-53

1

MOVE THE C.P. TO LINE TWO

DISPLAY THE CURRENT LINE

```
!2J$$
!T$$
*LINE TWO OF SAMPLE.
!
```

2

MOVE THE C.P. TO THE BEGINNING OF THE BUFFER

DISPLAY THE CURRENT LINE.

```
!J$$
!T$$
*THIS IS THE SAMPLE TEXT.
!
```

3

MOVE THE C.P. TO THE END OF THE BUFFER

DISPLAY THE CURRENT LINE.

```
!ZJ$$
!T$$
*!
```

nJ$$ Move the C.P. to the nth line in the edit buffer.

C.P. MOVER — THE J COMMAND

Figure 3-54

1

MOVE THE C.P. FIVE CHARACTERS TO THE RIGHT.

DISPLAY THE CURRENT LINE

```
!5M$$
!T$$
LINE *TWO OF SAMPLE.
!
```

2

MOVE THE C.P. THREE CHARACTERS TO THE LEFT.

```
!-3M$$
!T$$
LI*NE TWO OF SAMPLE.
!
```

nM$$ Move the C.P. n character positions from the current location.

MOVE COMMAND

Figure 3-55

1

DISPLAY THE CURRENT LINE →

```
!T$$
*LINE TWO OF SAMPLE.
!
```

2

DISPLAY TWO LINES STARTING FROM CURRENT LINE →

```
!2T$$
LINE TWO OF SAMPLE.
THE THIRD LINE OF THE SAMPLE.
!
```

3

DISPLAY ENTIRE BUFFER →

```
!#T$$.
THIS IS THE SAMPLE TEXT.
LINE TWO OF SAMPLE.
THE THIRD LINE OF THE SAMPLE.
FOURTH AND LAST LINE.
```

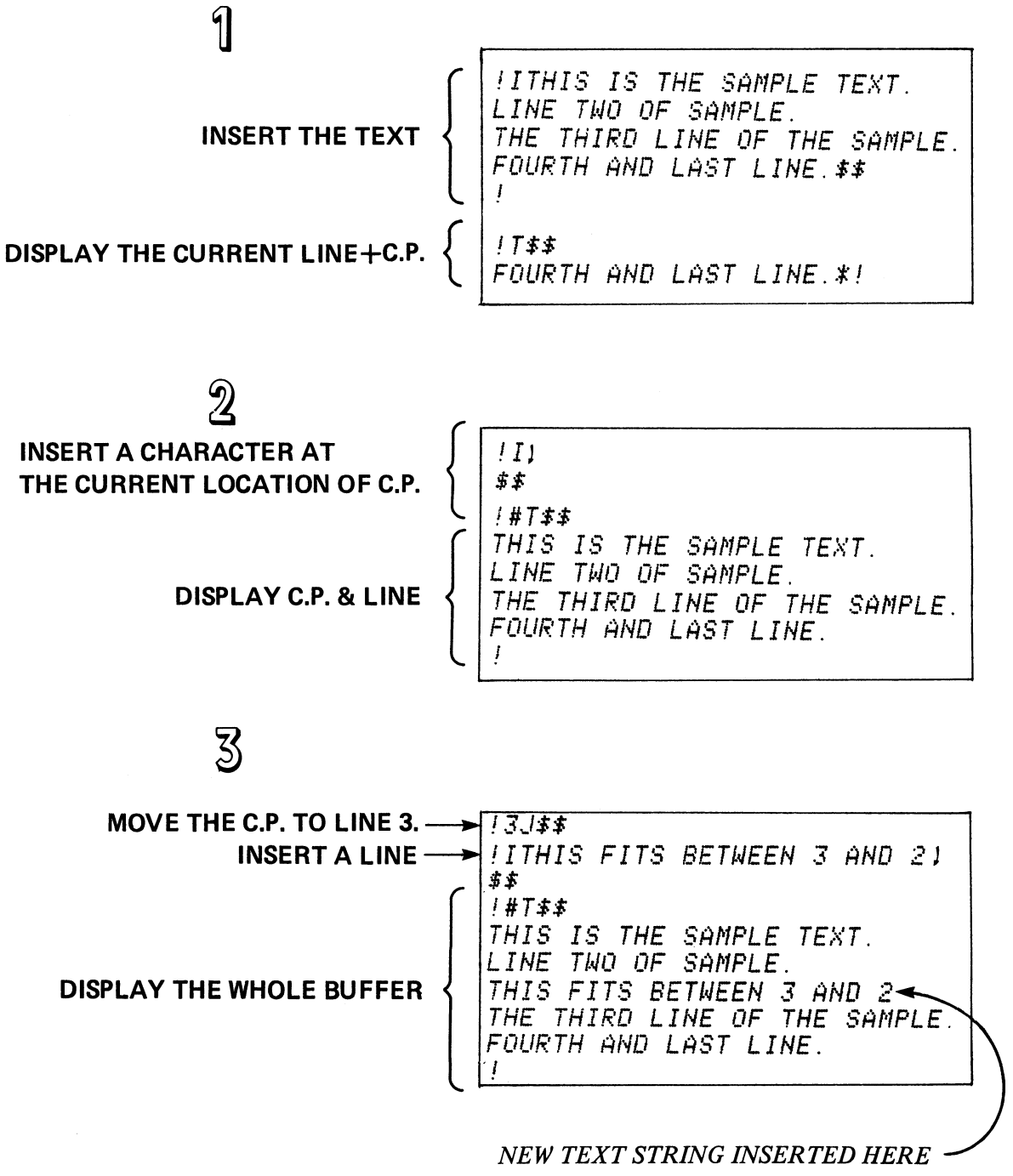
T$$ Display the line with the character pointer.

nT$$ Display n lines, starting from the C.P. line.

#T$$ Display all lines.

THE T COMMAND

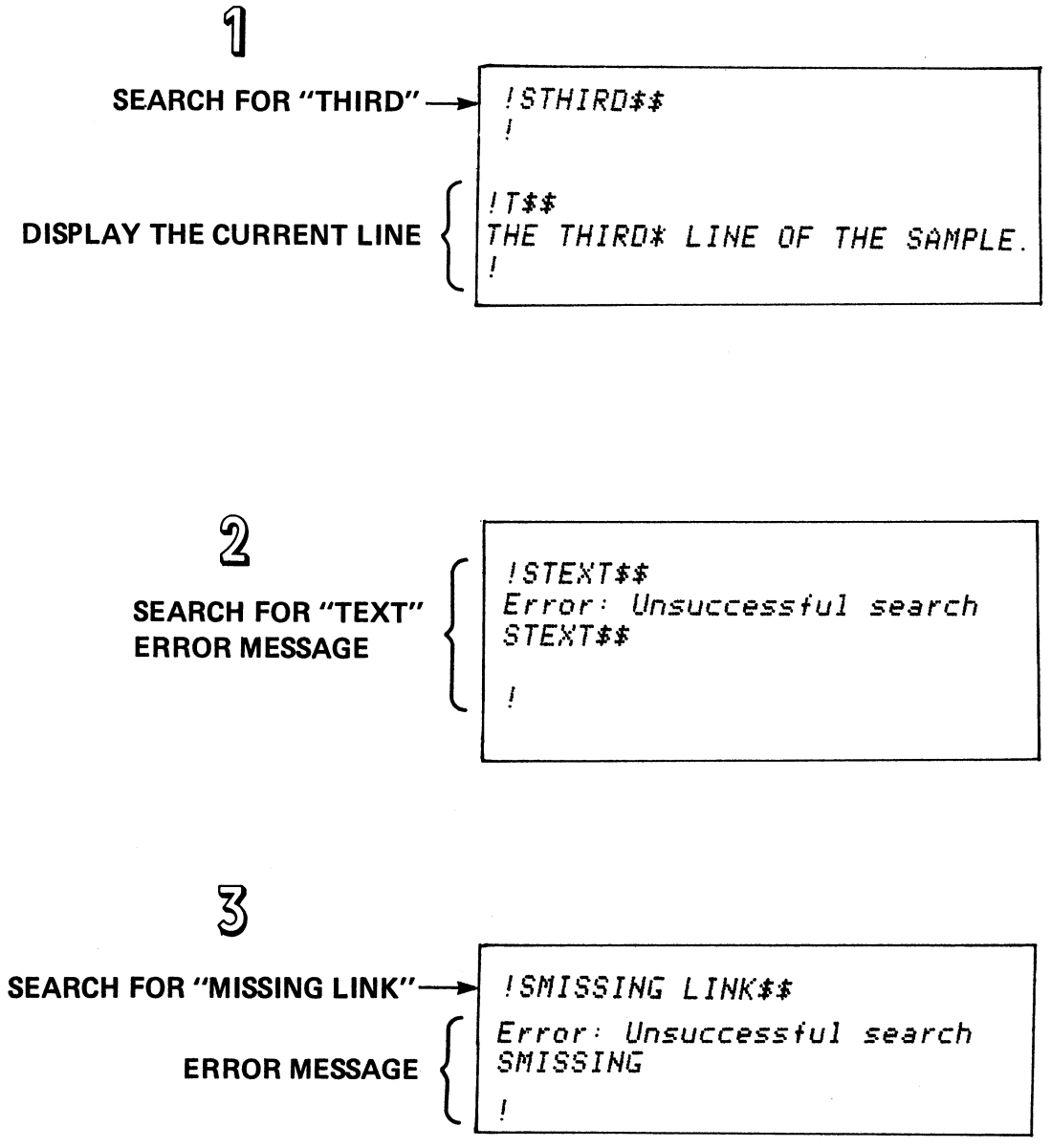
Figure 3-56



I text-string $$ *Insert text-string at current location of the C.P.*

INSERT (I) COMMAND

Figure 3-57



S text-string $$ *Search for text-string*

SEARCH COMMANDS

Figure 3-58

1

Move to line 3.
Display the C. P.
Search backwards 3 lines
Display the new C. P.

```
!3JT$$
(^)THIS FITS BETWEEN 3 AND 2
!-3SSAMPLE TEXT$$
!T$$
THIS IS A SAMPLE TEXT(^).
!
```

2

Search backward
Display the C. P.

```
$$
!0STHIS$$
!T$$
THIS(^) IS A SAMPLE TEXT.
!
```

3

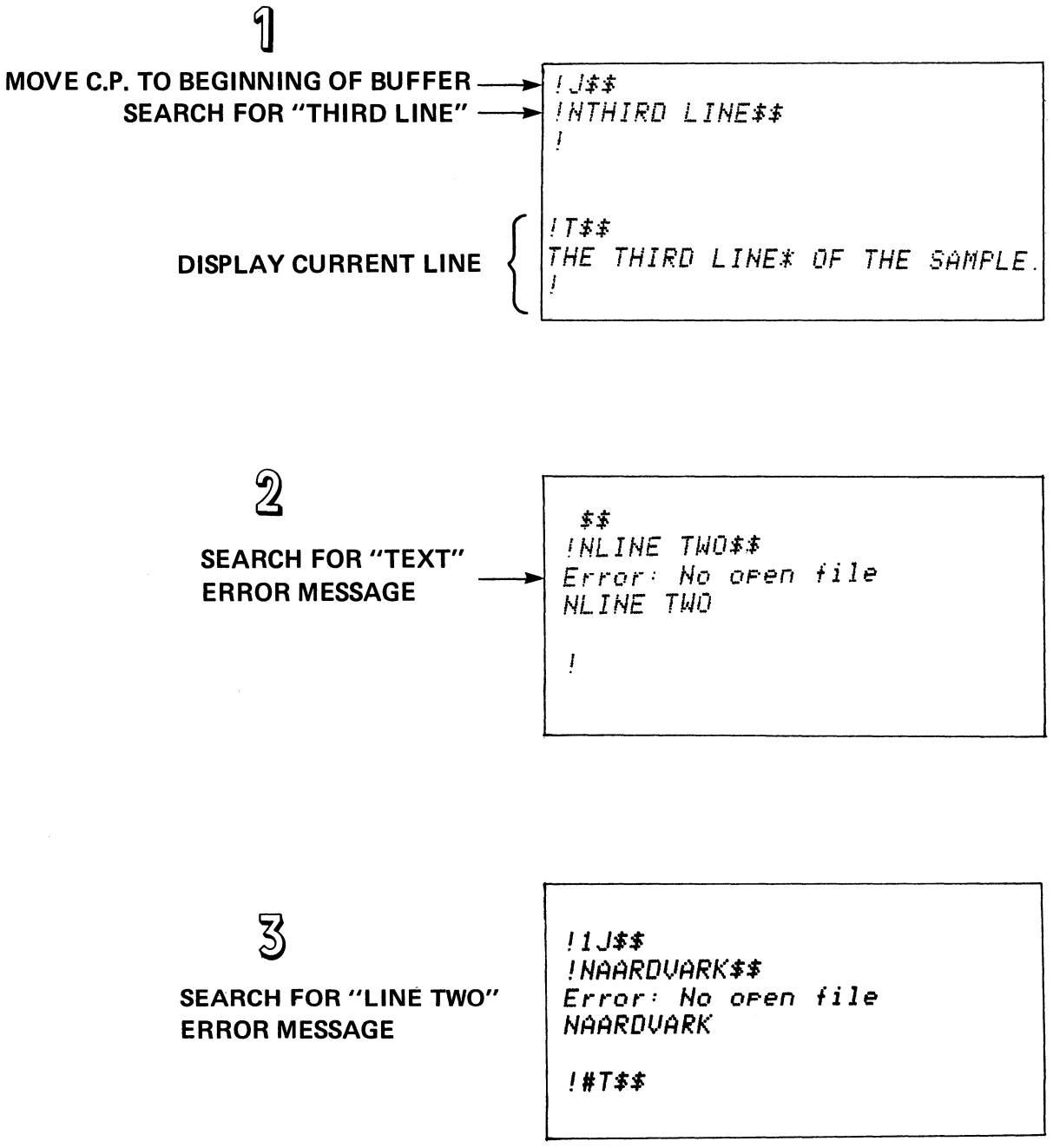
Display the C. P.
Search for "Fits"
Display the new C. P.

```
!T$$
THIS(^) IS A SAMPLE TEXT.
!1,100SFITS$$
!T$$
THIS FITS(^) BETWEEN 3 AND 2
!
```

nS *text-string* $$	search for *text-string* starting from C.P. and going n lines back toward the beginning of the buffer.
0 S *text-string* $$	search for *text-string*, starting from C.P. and going back to the beginning of the current line.
n, z S *text-string* $$	search for text-string, starting from character position n + 1 and continuing to character position Z.

SEARCH OPTIONS

Figure 3-59



N *text-string*$$ **Search for** *text-string* **throughout the entire input file.**

NON-STOP N SEARCH COMMAND

Figure 3-60

1

MOVE C.P. TO BEGINNING OF BUFFER →
CHANGE "SAMPLE" TO "PRACTICE" →

DISPLAY CURRENT LINE

```
!J$$
!CSAMPLE$PRACTICE$$
!
!T$$
THIS IS THE PRACTICE$ TEXT.
!
```

2

CHANGE "THIS IS" TO "LINE ONE OF" →
ERROR MESSAGE →

```
!CTHIS IS$LINE ONE OF$$
Error: Unsuccessful search
CTHIS IS$
!
```

3

MOVE C. P. TO LINE ONE
CHANGE "NUCLEAR" TO "SOLAR" →

```
!J$$
!CNUCLEAR$SOLAR$$
Error: Unsuccessful search
CNUCLEAR$
!
```

C *old-text* **$** *new-text***$$**

Search for *old-text.*
Delete *old-text*
Insert *new-text*
Leave C.P. after *new-text.*

CHANGE (C) COMMAND

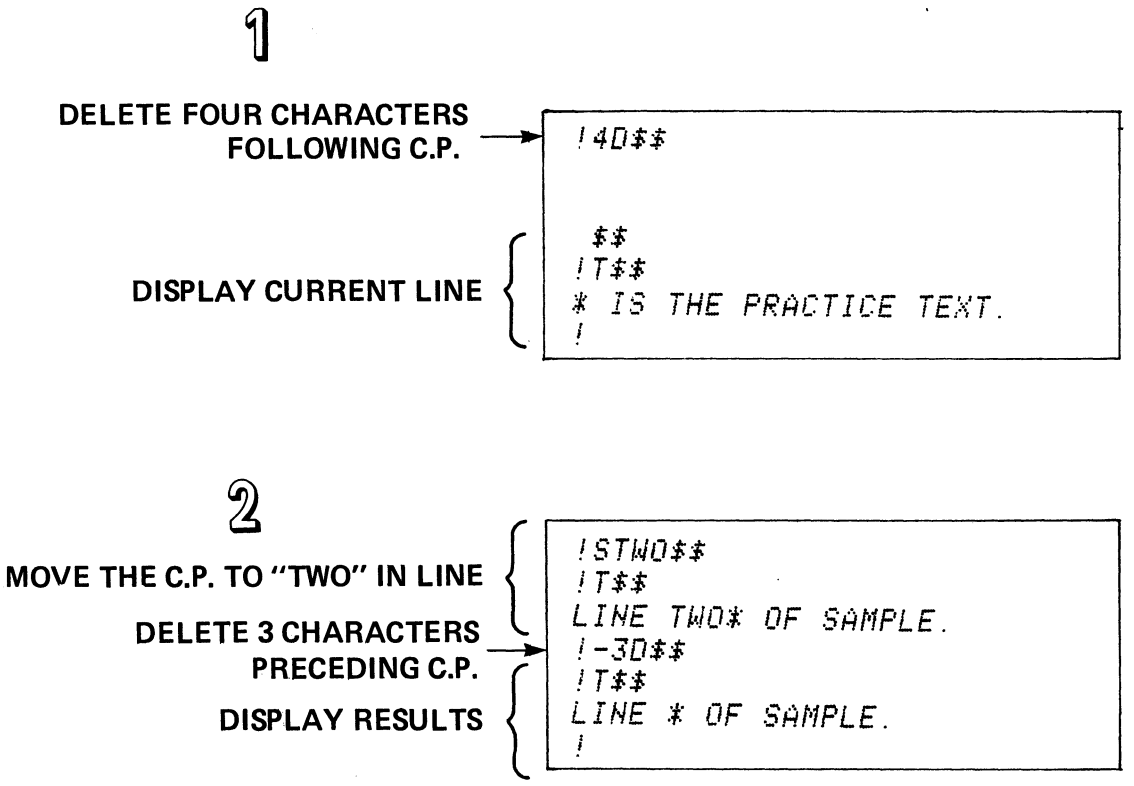
Figure 3-61

```
!C3 AND 2$$↖
!T$$
THIS EITS BETWEEN (^)
!
```

C *text-string* **$$** **Search for** *text-string*
Delete *text-string*
Leave C.P. after *text-string.*

CHANGE (C) COMMAND

Figure 3-62



n D$$ Delete n characters to the right of the C.P.

–nD$$ Delete n characters to the left of the C.P.

DELETE CHARACTERS (D) COMMAND

Figure 3-63

1

MOVE C.P. TO LINE TWO →
DELETE (KILL) ONE LINE →
FOLLOWING C. P.
DISPLAY ENTIRE BUFFER

```
!2J$$
!1K$$
!#T$$
 IS THE PRACTICE TEXT.
THIS FITS BETWEEN 3 AND 2
THE THIRD LINE OF THE SAMPLE.
FOURTH AND LAST LINE.
!
```

2

MOVE C. P. TO LINE TWO →
DELETE ONE LINE PRECEDING C.P. →
DISPLAY BUFFER

```
!2J$$
!-1K$$
!#T$$
'THIS FITS BETWEEN 3 AND 2
THE THIRD LINE OF THE SAMPLE.
FOURTH AND LAST LINE.
!
```

3

MOVE C. P. TO THE MIDDLE OF A LINE →
DELETE THE CHARACTERS IN THE FRONT OF THE LINE

```
!STHIRD$$
!T$$
THE THIRD* LINE OF THE SAMPLE.
!K$$
!T$$
* LINE OF THE SAMPLE.
!
```

nK$$ DELETE n LINES FOLLOWING C.P.

–nK$$ DELETE n LINES PRECEEDING C.P.

DELETE LINES (K) COMMAND

Figure 3-64

TOPICS

CP MOVERS

L	line move
J	jump to a line
M	move character positions

SEARCHERS

S	search
C	change
N	non-stop search

INSERTS

I	Insert
C	change

DELETES

C	change
D	delete characters
K	kill lines

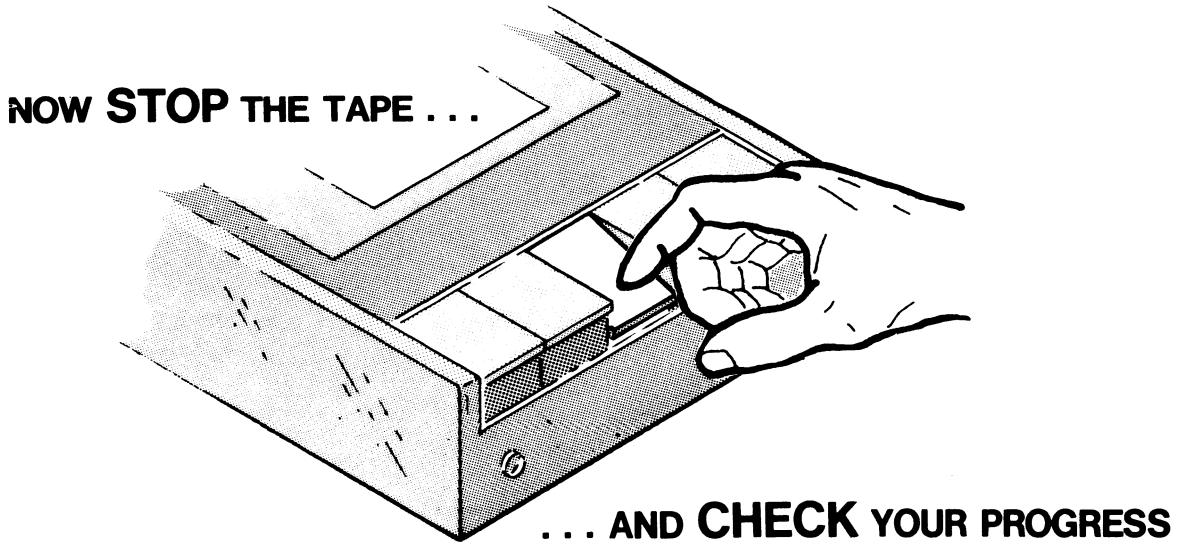


Figure 3-65

TEXT EDITOR

QUIZ

Abstract

This quiz tests your comprehension of the text manipulation commands.

Directions

Part I Multiple Choice

1. Answer the following questions by circling the letter of the correct answer. Questions may have more than one correct answer and, therefore, you should circle more than one letter.

2. The contents of the screen shown below are assumed to be in your edit buffer. Reference this figure for the questions in Part I of this Quiz.

```
MODULE THREE OF THE MP/OS COURSE.
THIS SEGMENT COVERS TEXT
EDITING COMMANDS.
WRITTEN APRIL 3,1979
```

1. The C. P. is at line one. The command for displaying the entire buffer is:

   A. D$$
   B. T$$
   C. #T$$
   D. P$$

2. The C.P. is at line one. A command sequence for displaying only the third line is:

   A. I$$
      T$$
   B. 3J$$
      T$$
   C. 3T$$
   D. 2K$$
      T$$

3. The command for moving the C. P. from line one to line four is:

   A. 4J$$
   B. 4L$$
   C. 3L$$
   D. 4M$$

4. The C. P. is at line four: move it to the end of the buffer:

   A. ZJ$$
   B. 1L$$
   C. 23M$$
   D. 5J$$

5. The C. P. is at the end of the buffer. Move it to the beginning of the buffer and verify the move:

   A. 1J$$
      T$$
   B. −4L$$
      T$$
   C. −5L$$
      T$$
   D. 0J$$
      T$$

6. Position the C. P. after "segment" in line two:

   A. 2J$$
      12M$$
   B. 1L$$
      12M$$
   C. 3J$$
      −13M$$

7. The C. P. is at line one. Insert the character string "DATA GENERAL CORPORATION" as line 5:

   A. IDATA GENERAL CORPORATION<NEW LINE>
      $$
   B. 5J$$
      TDATA GENERAL CORPORATION<NEW-LINE>
      $$
   C. 5J$$
      IDATA GENERAL CORPORATION <NEW LINE>
      $$

   D. 4L$$
      CDATA GENERAL CORPORATION <NEW LINE>

8. You have just inserted line 5. The command *and response* for verifying the location of the C. P. are:

   A. T$$
      DATA GENERAL CORPORATION (↑)
   B. T$$
      (↑)DATA GENERAL CORPORATION
   C. T$$
      (↑)

   D. T$$
      DATA GENERAL CORPORATION (↑)

9. The C.P. has been moved to the end of line five. Delete the fifth line that you just inserted.

   A. 5K$$
   B. 5J$$
      K$$
   C. L$$
      1K$$
   D. 5J$$
      1K$$

10. The C. P. is at line one. Change "1979" to "1980".

   A. 4J$$
      S1979$$
      I1980$$
   B. 4J$$
      S1979$$
      −4M$$
      I1980$$
   C. 4J$$
      SAPRIL 3, $$
      I1980$$
   D. C1979$1980$$

11. You just changed "1979" to "1980". Verify the change.

   A. T$$
   B. L$$
      T$$
   C. 1L$$
      T$$
   D. 0T$$

12. The C. P. is at line one. Delete "WRITTEN" from line 4:

   A. 4J$$
      7D$$
   B. CWRITTEN$$
   C. SWRITTEN$$
      −7D$$
   D. DWRITTEN$$

Part II Matching Columns

Match the command on the left with the functional description on the right.

COMMANDS		FUNCTIONS
_____C	1.	Delete lines of text.
_____J	2.	Search for string, delete it, insert another string.
_____D	3.	Insert string at CP location.
_____L	4.	Delete characters.
_____S.	5.	Search for string in buffer, put, yank (if there are open files).
_____K	6.	Search for string in buffer.
_____T	7.	Move CP character positions.
_____M	8.	Move CP to beginning of specified line, relative to start of buffer.
_____N	9.	Move CP to beginning of a line, relative to current line.
_____#T	10.	Display contents of entire buffer.
_____I	11.	Display contents of current line with CP.

CHECK YOUR ANSWERS
ON THE FOLLOWING PAGES

TEXT EDITING
QUIZ ANSWERS

1. The C. P. is at line one. The command for displaying the entire buffer is:

A.	D$$	Deletes the first character in the buffer.
B.	T$$	Only displays one line
(C.)	#T$$	The "#" indicates the entire buffer.
D.	P$$	Tries to output one page.

2. The C.P. is at line one. A command sequence for displaying only the third line is:

A.	I$$ T$$	Inserts nothing and displays line one.
(B.)	3J$$ T$$	C.P. moves to line 3 which is then displayed.
C.	3T$$	This displays lines 1, 2, and 3.
(D.)	2K$$ T$$	This will work: lines 1 and 2 are deleted and line 3 is displayed.

3. The command for moving the C. P. from line one to line four is:

(A.)	4J$$	
B.	4L$$	Moves the C.P. to line 5.
(C.)	3L$$	
D.	4M$$	Moves the C.P. four character positions.

4. The C.P. is at line four: move it to the end of the buffer:

(A.)	ZJ$$	2 means end of buffer.
(B.)	1L$$	The beginning of line five represents the end of the buffer.
(C.)	23M$$	23 character positions get you to the end of the buffer.
(D.)	5J$$	Same as B.

5. The C. P. is at the end of the buffer. Move it to the beginning of the buffer and verify the move:

(A.)	1J$$ T$$	1J says jump to line 1.
(B.)	−4L$$ T$$	Line 5 minus 4 equals line 1.
(C.)	−5L$$ T$$	This works even though you subtracted too much.
(D.)	0J$$ T$$	0J says jump to line 1 of buffer.

6. Position the C.P. after "segment" in line two:

(A.)	2J$$ 12M$$	"segment" is 5 character positions into line two.
(B.)	1L$$ 12M$$	Line one plus one line (1L) equals line two.
(C.)	SSEGMENT$$	Search for "segment".
(D.)	3J$$ −13M$$	Include the NEW-LINE character and spaces when you count back.

7. The C.P. is at line one. Insert the character string "DATA GENERAL CORPORATION" as line 5:

A. IDATA GENERAL CORPORATION<NEW-LINE>
$$ Inserts the string as line one, the current C.P. location.

B. 5J$$
TDATA GENERAL CORPORATION<NEW LINE>
$$ CP moves to line 5, nothing typed, error!

(C.) 5J$$
IDATA GENERAL CORPORATION<NEW LINE>
$$

D. 4L$$
CDATA GENERAL CORPORATION<NEW LINE>$$
CP moves to line 5, then error!

8. You have just inserted line 5. The command *and response* for verifying the C. P. location are:

A. T$$
DATA GENERAL CORPORATION (↑) WRONG DISPLAY.

B. T$$
(↑)DATA GENERAL CORPORATION C.P. IN WRONG PLACE.

(C.) T$$ Yes, C.P. is positioned after NEW-LINE,
(↑) now at an empty line 6.

D. T$$
DATA GENERAL CORPORATION (↑) WRONG DISPLAY.

9. The C.P. has been moved to the end of line five. Delete the fifth line that you just inserted.

A.	5K$$	Deletes nothing.
B.	5J$$ K$$	Almost, but without a number specified with K, nothing is deleted.
(C.)	1L$$ 1K$$	Moves C.P. to front of line 5 and 1 with K deletes one line.
(D.)	5J$$ 1K$$	Similar to B, only this time the required "1" is specified.

10. The C. P. is at line one. Change "1979" to "1980".

A.	4J$$ S1979$$ I1980$$	Inserts 1980 after 1979.
B.	4J$$ S1979$$ –4M$$ I1980$$	Inserts 1980 before 1979.
C.	4J$$ SAPRIL 3,$$ I1980$$	Inserts 1980 before 1979.
(D.)	C1979$1980$$	Changes 1979 to 1980.

11. You just changed "1979" to "1980". Verify the change:

(A.)	T$$	Displays current line.
(B.)	L$$ T$$	Moves C.P. to front of line and displays whole line.
C.	1L$$ T$$	No. Moves C.P. to next line and displays it.
(D.)	0T$$	Yes. "0" requests display of line from beginning to C. P. location.

12. The C.P. is at line one. Delete "WRITTEN" from line four:

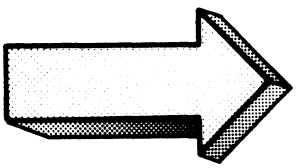
(A.)	4J$$ 7D$$	Move C.P. to line 4 and delete 7 characters
(B.)	CWRITTEN$	Deletes "WRITTEN"
(C.)	SWRITTEN$$ −7D$$	Yes. The search moves C.P. after "WRITTEN". Negative 7 deletes seven characters.
D.	DWRITTEN	No, error.

PART II ANSWERS FOLLOW ON THE NEXT PAGE

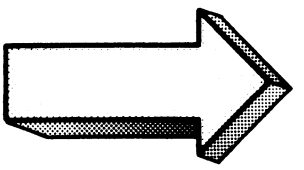
Part II Matching Columns

Match the command on the left with the functional description on the right.

COMMANDS		FUNCTIONS
2	C	1. Delete lines of text.
8	J	2. Search for string, delete it, insert another string.
4	D	3. Insert string at C P location.
9	L	4. Delete characters.
6	S	5. Search for string in buffer, put, yank (if there are open files).
1	K	6. Search for string in buffer.
11	T	7. Move C.P. character positions.
7	M	8. Move CP to beginning of specified line, relative to start of buffer.
5	N	9 Move CP to beginning of a line, relative to current line.
10	#T	10. Display contents of entire buffer.
3	I	11. Display contents of current line with CP.



A SCORE OF 19 CORRECT QUESTIONS OUT OF THE 23 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.



TEXT EDITOR
LAB

Abstract

This Lab exercise covers the commands discussed in Module Three.

Directions

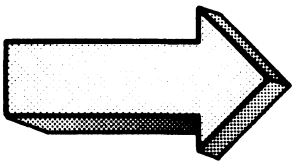
This lab is applicable whether or not you have access to a functional system. The sequence of steps is the same as for previous labs:

1. *Cover the answers.*

2. Write your answer in the blank space.

3. Check your answer against the answer provided.

4. Try the sequence on your system.



It is assumed that your system is functioning and the console is displaying the CLI prompt. If not, follow the directions in Module Two for bringing up your system.

1. Bring SPEED in for execution. Show the system's response:

_____

_____

_____

****

```
) XEQ SPEED)
SPEED REV  1.00
!
```

XEQ may be abbreviated to XE or X.
You may append the "/D" switch to SPEED to turn on display mode (=10). You also may have a later revision of SPEED.

Use the simple version as shown.

Do it on your system.

2. Open a file named TEST for output and verify the operation (i.e., show which files are open). Show the system's response:

****

```
!FWTEST$$
!F?$$
Global:
        Input  File - None
        Output File - TEST
Local:
        Input  File - None
        Output File - None
!
```

The FW command opens a file for output.
The F? command displays the open files.

Remember to terminate each command with the CONTROL–D sequence.

Do it on your system.

Our lab environment is a dual diskette with a hard-copy Dasher terminal (6042).

3. Insert the following text as five lines in your edit buffer: (Show the command and response.)

"Key factors in selecting candidates are:
1. positive professional attitude;
2. concise communications;
3. solid technical background;
4. independent motivated thinking."

_____

_____

_____

_____

_____

_____

_____

****

```
!IKEY FACTORS IN SELECTING CANDIDATES ARE:↓
1. POSITIVE PROFESSIONAL ATTITUDE;↓
2. CONCISE COMMUNICATIONS;↓
3. SOLID TECHNICAL BACKGROUND;↓
4. INDEPENDENT MOTIVATED THINKING.↓
$$
!
```

The I command inserts text into the buffer at the current location of the C.P.

Do it on your system.

Note: terminate each line with the new line or line feed. Terminate the command string with the CONTROL–D.

4. Display the newly inserted text. Show the C.P. location.

_____

_____

_____

_____

_____

_____

****

```
!#T$$
KEY FACTORS IN SELECTING CANDIDATES ARE:
1. POSITIVE PROFESSIONAL ATTITUDE;
2. CONCISE COMMUNICATIONS;
3. SOLID TECHNICAL BACKGROUND;
4. INDEPENDENT MOTIVATED THINKING.
```

The T command, with the # prefix, displays the entire buffer (not the C.P.).

Do it on your system.

Note that the C.P. is positioned after the last character (line-feed) inserted into the text. This requires two commands. If your C.P. shows up at the end of the fifth line, then you failed to insert the last new-line. Use the insert command (I) to get it in.

5. Move the C.P. to the beginning of the buffer and verify the results by displaying the new C.P. location.

REMEMBER: TO GET THE MAXIMUM BENEFIT OUT OF THIS EXERCISE, YOU SHOULD NOT LOOK AT THE ANSWER UNTIL YOU HAVE WRITTEN OUT YOUR COMPLETE ANSWER.

_____

_____

_____

_____

****

```
!1J$$
!T$$
(^)KEY FACTORS IN SELECTING CANDIDATES ARE:
!
```

1J moves the C.P. to line one in the buffer.

The T command displays the current line and C.P. location.

Do it on your system.

This could have been entered as one command line (1JT$$).

6. Find TECHNICAL . Verify the operation by displaying the line and C.P.

_____

_____

_____

_____

****

```
!STECHNICAL$$
!T$$
3. SOLID TECHNICAL(^) BACKGROUND;
!
```

The S command searches for the specified text. S uses the C.P. as its reference point, unless specifically stated otherwise.

Try it.

Note that the C.P. is positioned after TECHNICAL . If you received an error message then check your typing of the command and your text.

You might also use the N or Q commands for the search.

7. Use the L command to move to line two. Verify the move by showing the new location of the C.P.

_____

_____

_____

_____

****

```
!-2L$$
!T$$
(^)1. POSITIVE PROFESSIONAL ATTITUDE:
!
```

You left the C.P. in the fourth line (after TECHNICAL).
−2L moves the C.P. backwards 2 lines.
T displays the line with the C.P.

Try it on your system.

The minus L. command moves the C.P. backwards (upwards) in your text from the C.P.'s current line position. The operation is a simple subtraction.

This may be entered as one line (−2LT$$).

8. Delete the third line. Verify the operation by displaying the new buffer contents.

****

```
!3J$$
!1K$$
!#T$$
KEY FACTORS IN SELECTING CANDIDATES ARE:
1. POSITIVE PROFESSIONAL ATTITUDE;
3. SOLID TECHNICAL BACKGROUND;
4. INDEPENDENT MOTIVATED THINKING;
!
```

3J forces the C.P. to line three in the buffer.
1K deletes one line
½
#T displays the entire buffer, minus the deleted lines.

Note: The 1 must precede the K to delete one line.

Try it on your system.

This may be entered as one line (3J1KT$$).

9. Change BACKGROUND to FOREGROUND . Verify the result by displaying the new line.

_____

_____

_____

_____

****

```
!CBACKGROUND$FOREGROUND$$
!T$$
3. SOLID TECHNICAL FOREGROUND(^);
!
```

The C command searches for the first string, deletes it, and inserts the second string.

Try it on your system.

Note: be very careful with spaces in the change command. The character strings are separated by the ESCAPE character, which is echoed as a single dollar sign ($).

10. Use the M and D commands to delete PROFESSIONAL from line two. Show the results:

_____

_____

_____

_____

_____

_____

****

```
!2J$$
!12M$$
!12D$$
!T$$
1. POSITIVE (^) ATTITUDE;
!
```

Once again we used the J command to move the C.P. to the specified line (2).

12M moves the C.P. 12 characters to the right.
12D deletes 12 characters following the C.P.

The M and D commands require accurate counting of characters. Include the space and period as one character each.

Try it on your system.

11. Write your text to the output file. Determine what remains in your buffer:

_____

_____

_____

_____

_____

_____

_____

****

```
!P$$
!#T$$
KEY FACTORS IN SELECTING CANDIDATES ARE:
1. POSITIVE  ATTITUDE;
3. SOLID TECHNICAL FOREGROUND;
4. INDEPENDENT MOTIVATED THINKING;
!
```

The P command writes the buffer to the opened output file.

The P ' command copies out the buffer, but does not clear it. The buffer remains intact.

Try it on your system.

If you receive an error, check your file status with the F? command.

12. Close the output file. Return to C.L.I. Display the status of TEST.

****

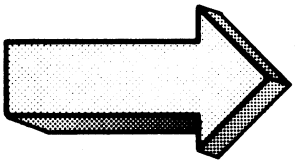
```
!FC$$
!H$$
Confirm? YES )

) FI/AS TEST )
DIRECTORY @DPX1:

  TEST                TXT    24-JUL-79  10:46:30          131
)
```

FC closes all open files.
H returns to the parent program.
The CLI FILESTATUS command displays the status of TEST.

Try it on your system.

The FC closes all files. Speed prompts for an exit confirmation because there is text in the edit buffer.

THIS CONCLUDES THE FINAL LAB EXERCISE FOR SPEED. SHUTDOWN YOUR SYSTEM AND CONTINUE TO MODULE FOUR.



MODULE FOUR
MP/OS PROGRAM DEVELOPMENT

MODULE FOUR

MP/OS PROGRAM DEVELOPMENT

Abstract

This module instructs in the concepts and procedures involved in getting your source language programs into executable form. It is assumed that you already know how to flowchart and code at least one programming language, and these topics will not be covered here. MP/OS currently handles MP/Fortran, MP/Pascal, and assembly languages. This module discusses all three of these languages.

The module is divided into the following segments:

* The program development cycle on MP/OS systems
* Macroassembler Concepts.
* Macroassembler Procedures.
* MP/Fortran Compilation
* MP/Fortran Assembly
* MP/Pascal Compilation
* Binding and execution of assembly, MP/Pascal, and MP/Fortran files

You have the option of completing only those sections associated with your language, or you may complete all of the sections.

Goal

Upon completion of this module you will be able to develop a given source language program for execution under the MP/OS operating system.

Objectives

Upon completion of this module, you will be able to:

1. State, in your own words, the purpose, input, and output of the Macroassembler and compiler utilities;

2. Given a CLI compiler and Macroassembler command line, identify and state the purpose of switches and arguments;

3. List the reference material available on compiler and Macroassembler commands (text and system sources);

4. Given a CLI compiler and Macroassembler command line, describe the results.

5. Write an appropriate CLI compiler or Macroassembler command for compiling or assembling a given source program. State the output names with extensions.

6. Given a compiler or Macroassembler error message:
   A) identify the possible cause of the error,
   B) reference a solution to the error.
   C) write a CLI command and/or compiler command to correct the error.

7. State, in your own words, the purpose, input and output of the binder utility.

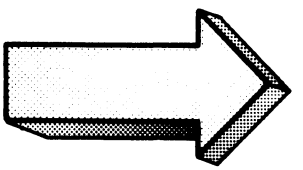
8. Given a CLI binder command line, identify and state the purpose of the switches and arguments;

9. List the reference material available on binder commands (text and system sources);

10. Given a CLI binder command line, describe the results;

11. Write a CLI binder command line to bind a given object file into executable state. State the full output names with extensions;

12. Given a binder error message:
   A) identify possible causes.
   B) reference a possible solution,
   C) write a solution with a CLI command line and/or CLI series, and/or binder command line.



PROGRAM DEVELOPMENT

Abstract

This section introduces you to the sequence of utilities required for developing executable programs under the MP/OS Operating System.

Objectives

Upon completion of this section you will be able to:

1. State, from memory, the sequence of steps required for developing a given program into executable form.

2. State the purpose of:
   A) Macroassembler
   B) compilers
   C) binder
   D) debugger
   E) Text Editor

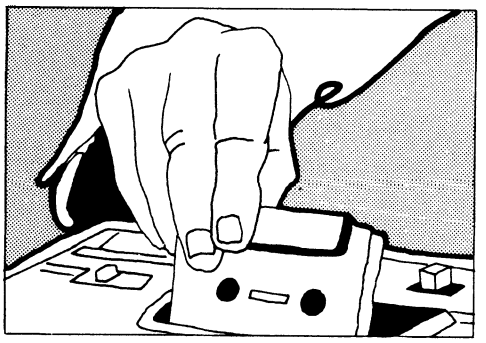
3. Describe the sequence of events occurring in the system with each step in Program Development.

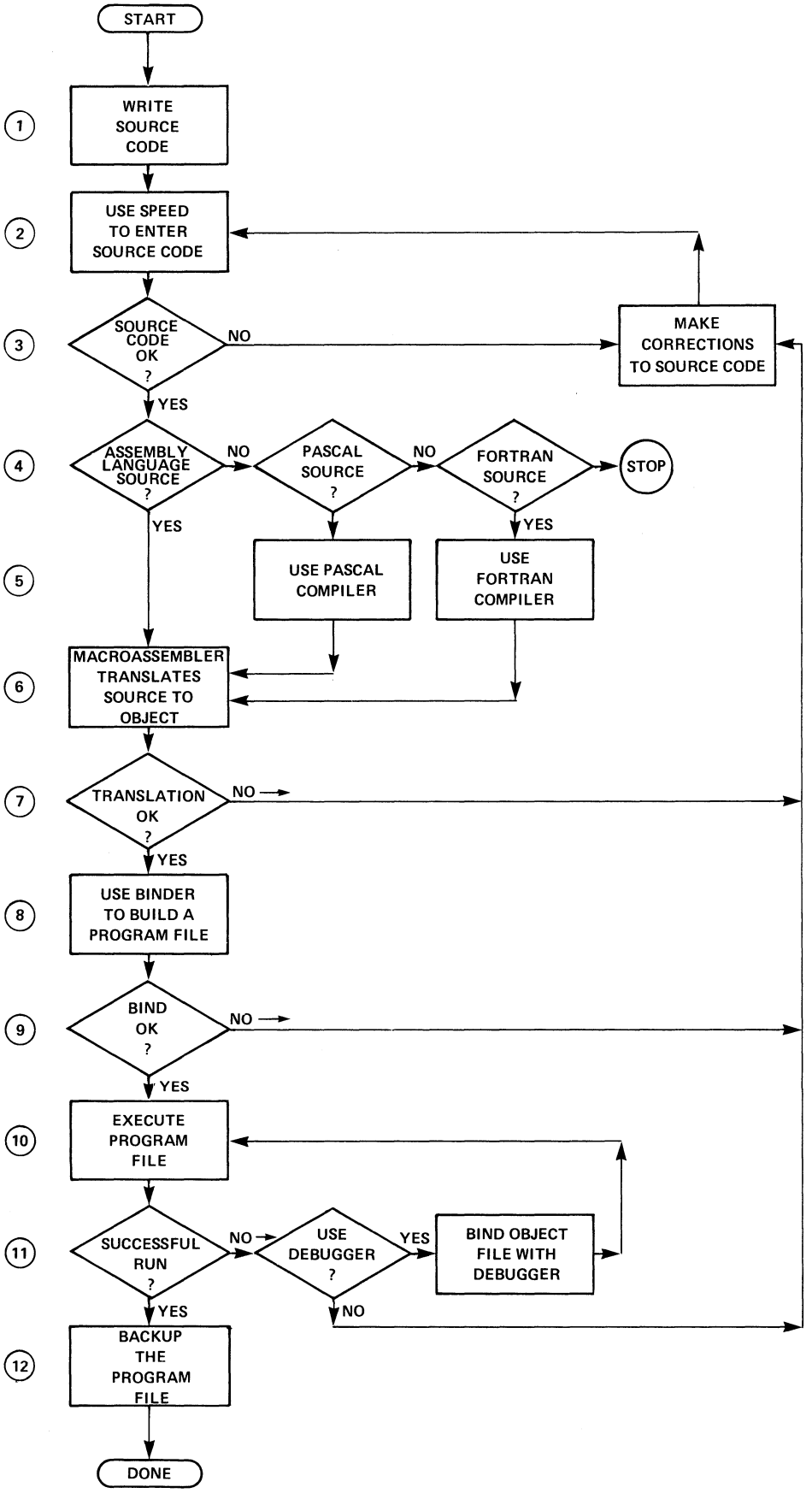
4. Define the input and output requirements of each step of Program Development.

Directions

1. Turn to figure 4-1 on the next page of your Student Guide.

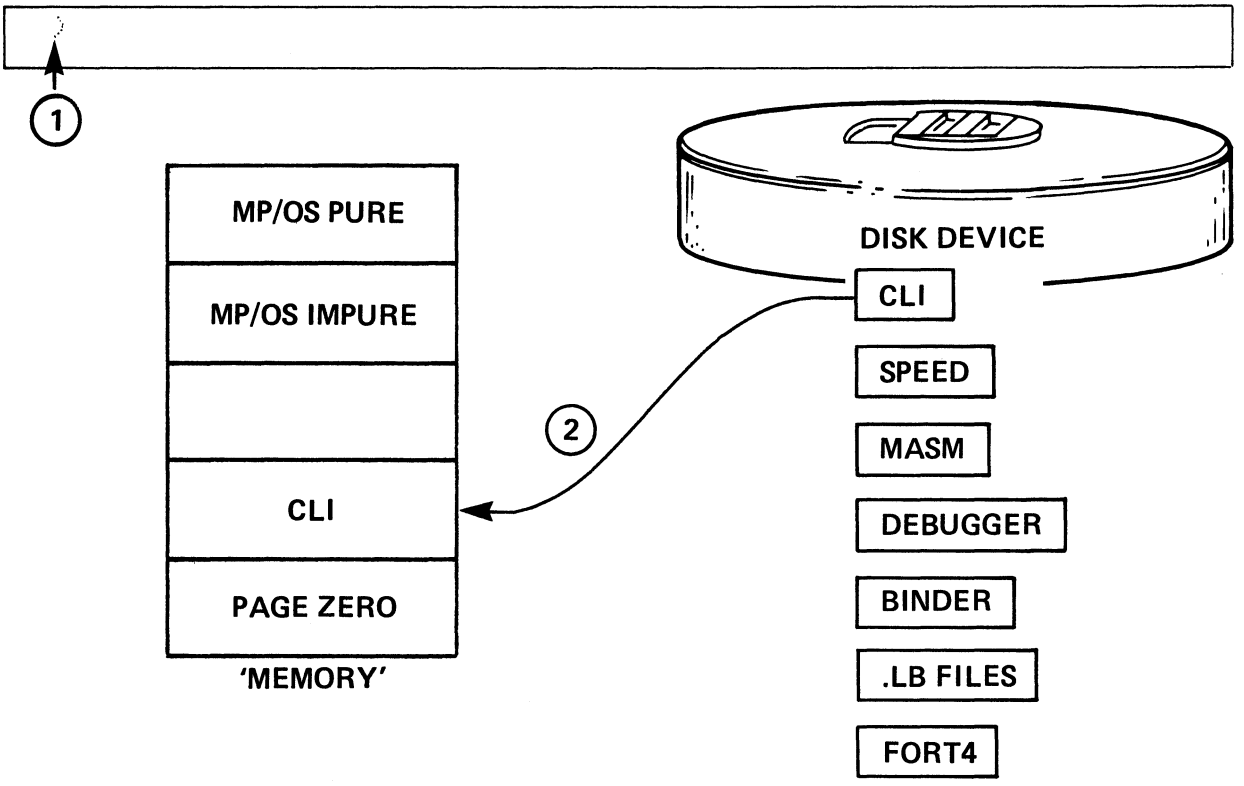
2. Listen to the audio-tape for this segment.





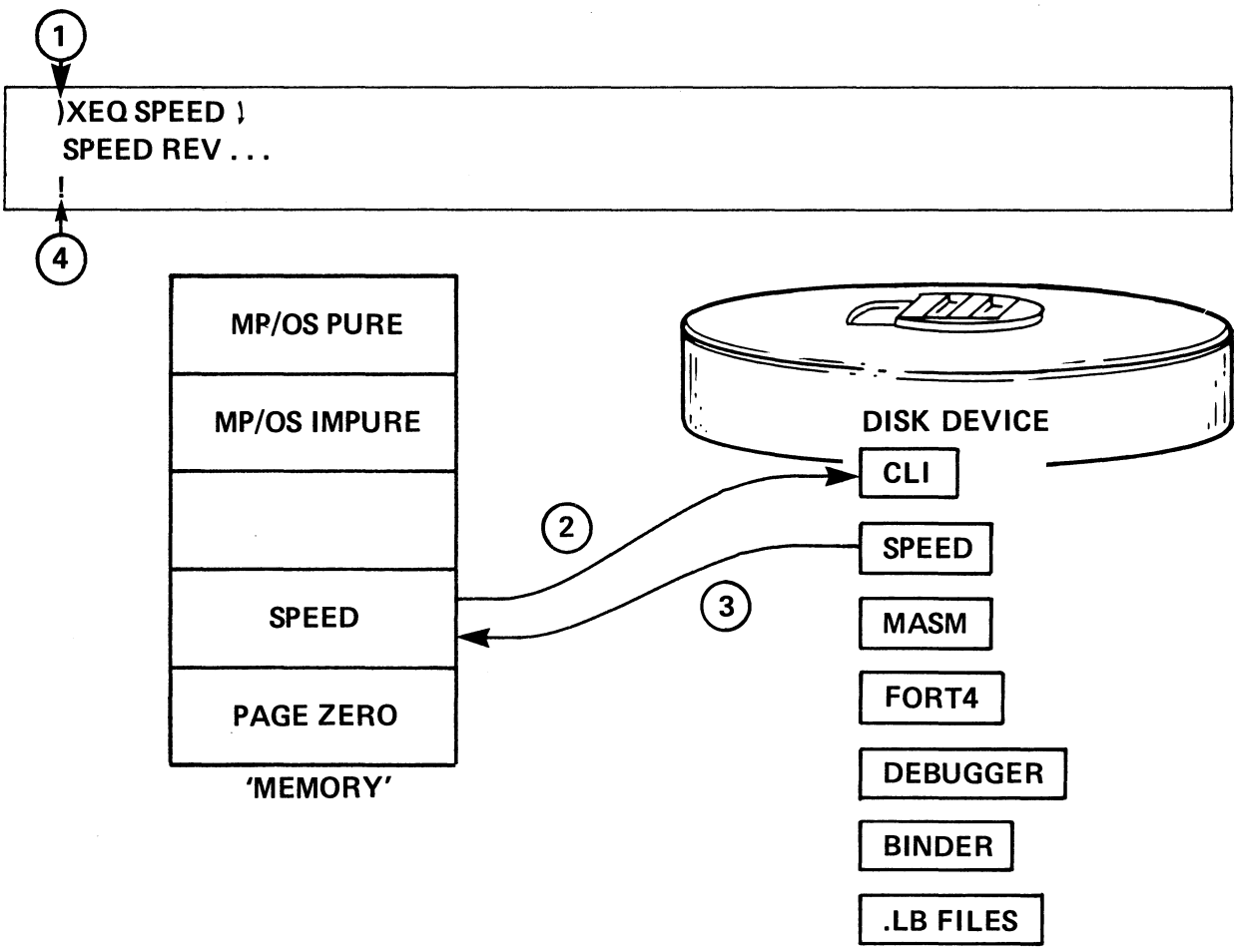
PROGRAM DEVELOPMENT STEPS

Figure 4-1



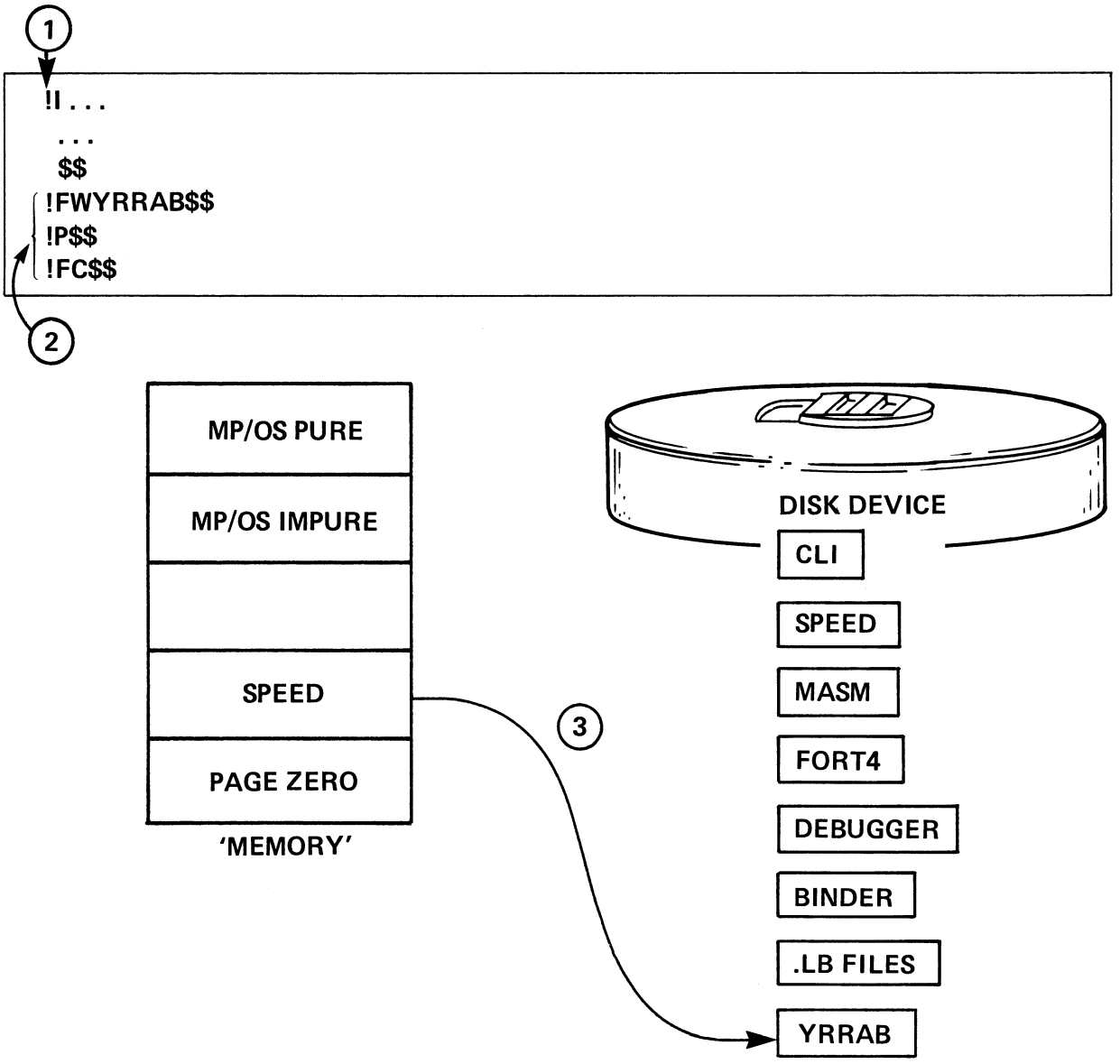
TYPICAL MEMORY CONFIGURATION AT SYSTEM INITIALIZATION

Figure 4-2



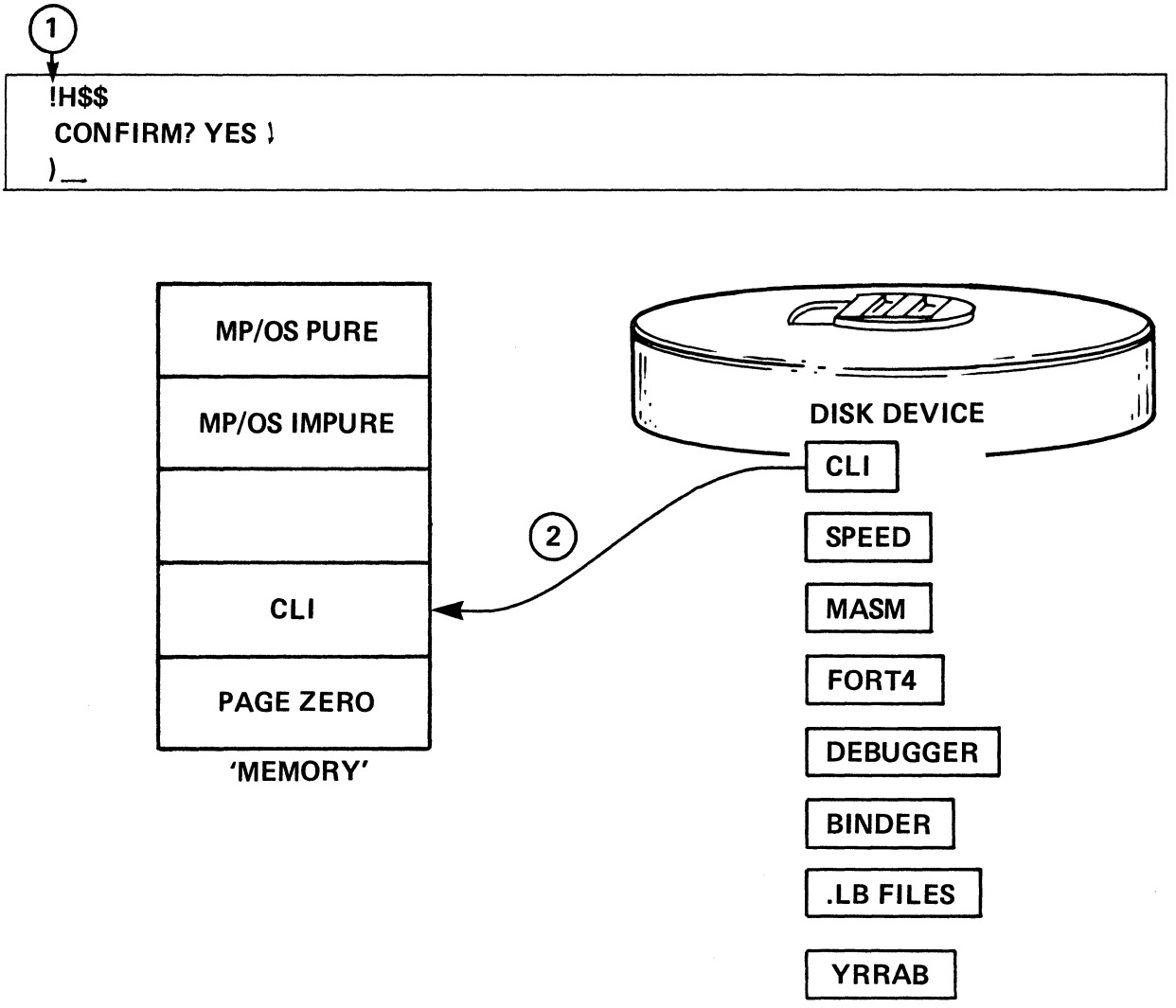
INVOKING SPEED

Figure 4-3



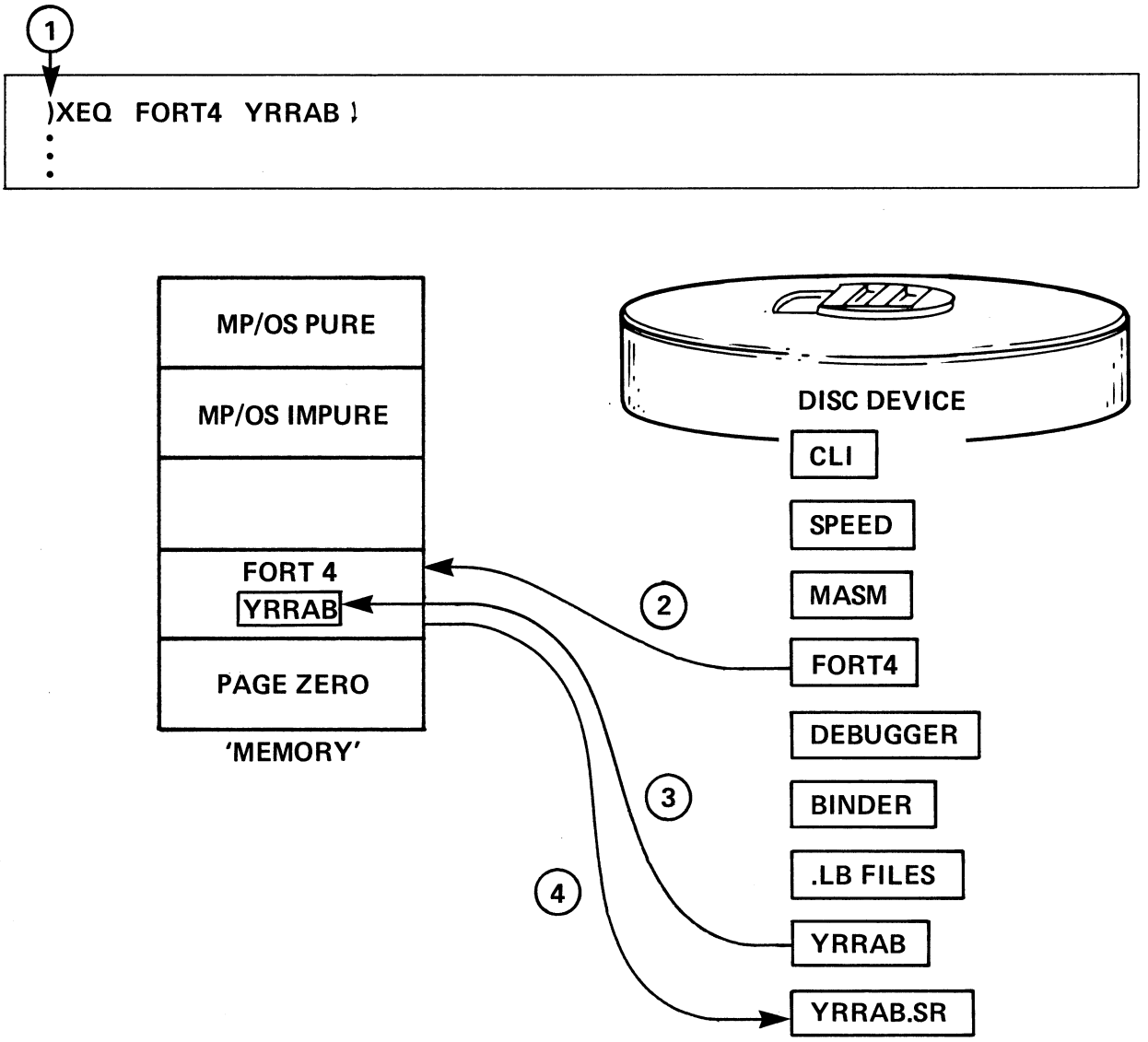
CREATING THE TEXT OUTPUT FILE

Figure 4-4



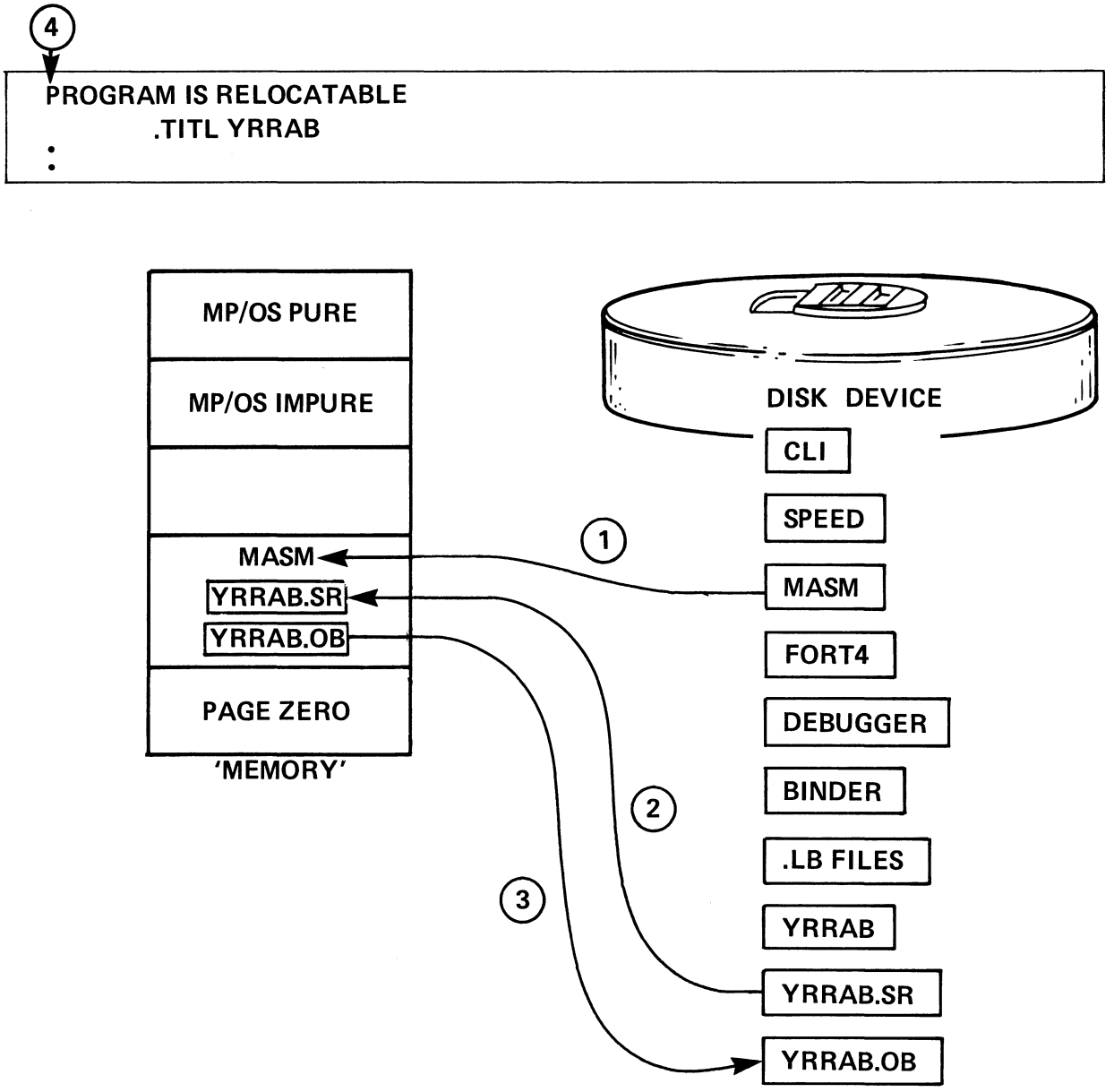
EXIT SPEED & RETURN TO CLI

Figure 4-5



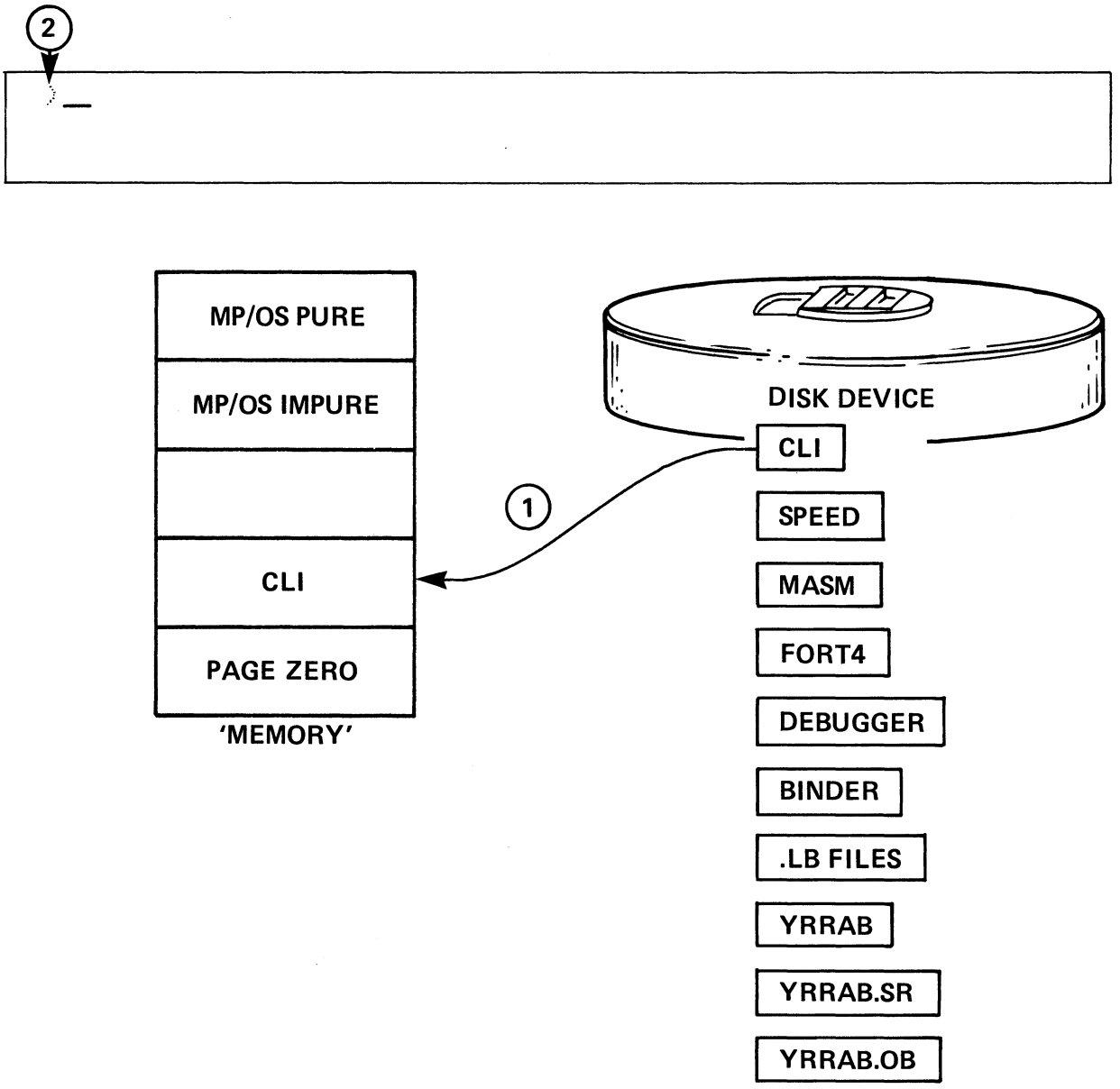
COMPILING THE SOURCE FILE

Figure 4-6



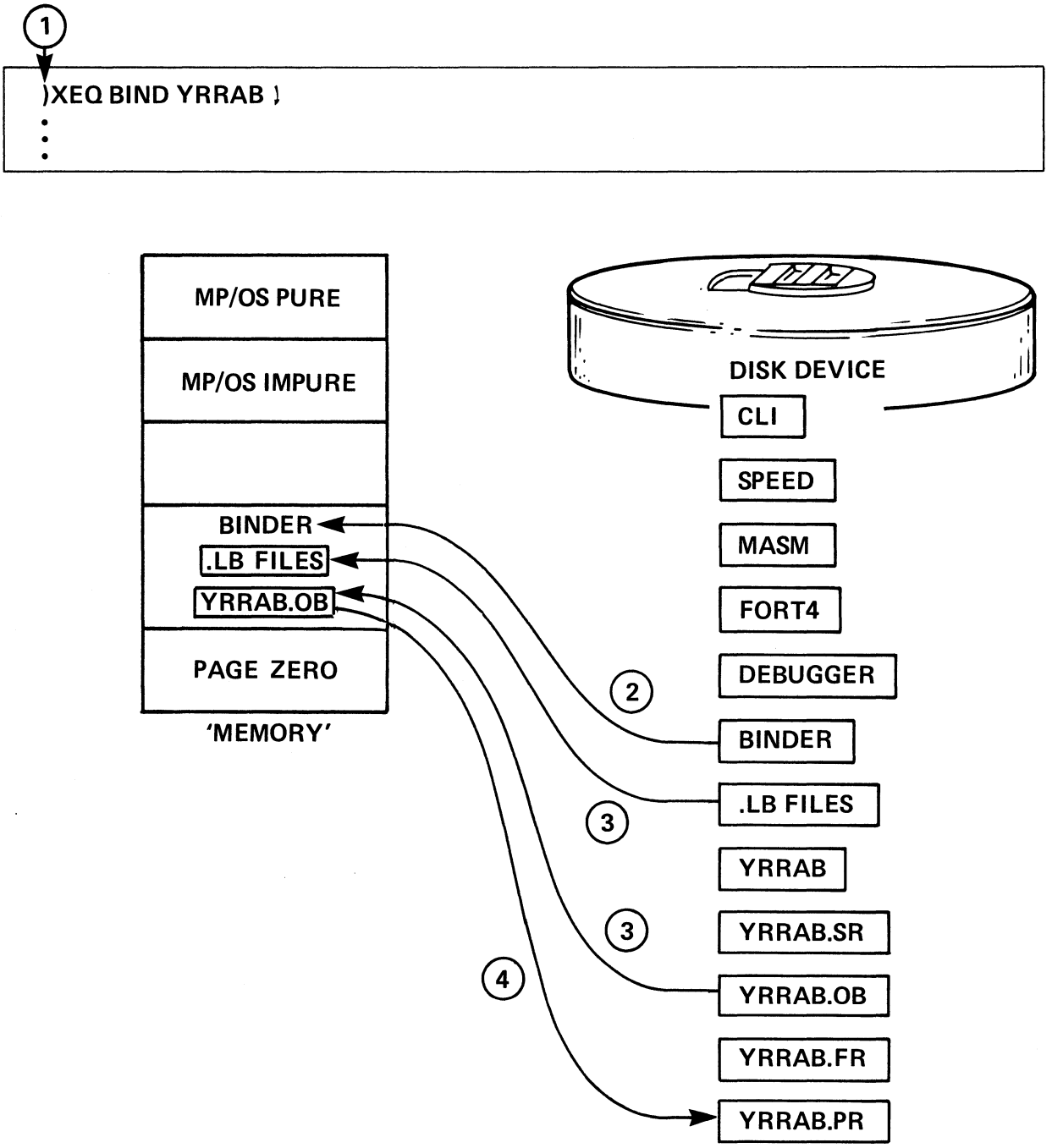
INVOKING THE MACROASSEMBLER

Figure 4-7



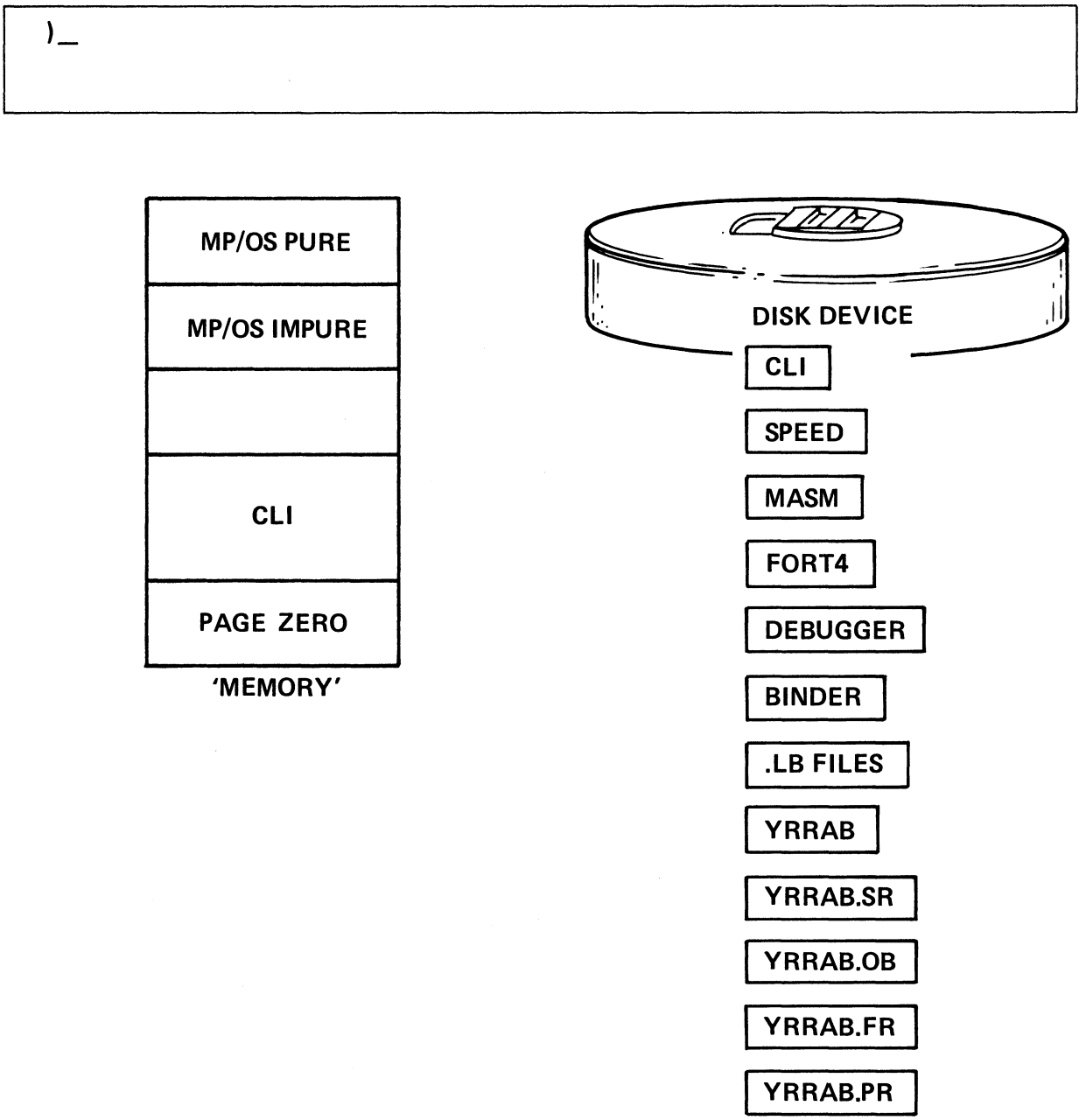
MACROASSEMBLER TERMINATES, CLI RETURNS

Figure 4-8



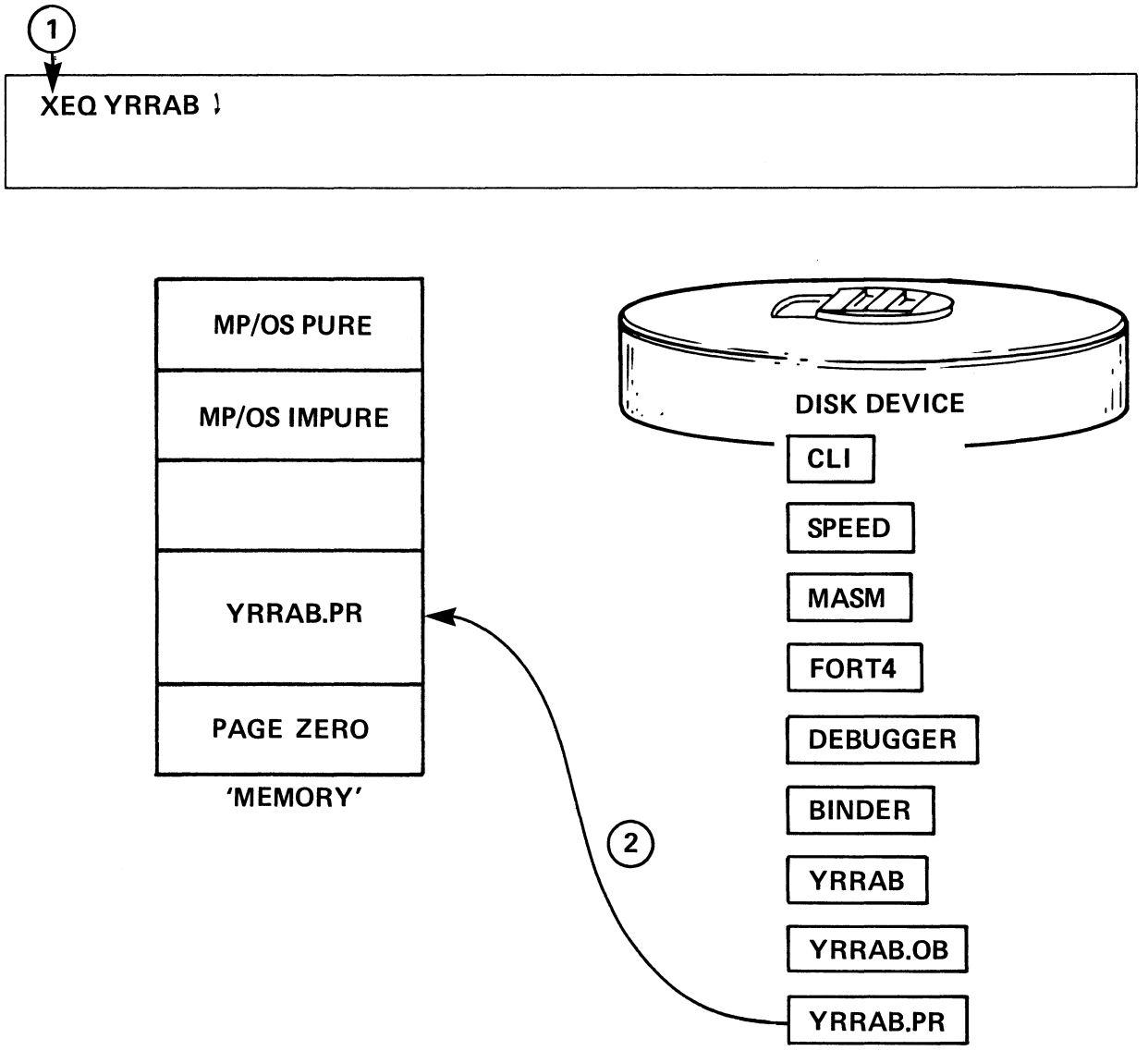
INVOKING THE BINDER

Figure 4-9



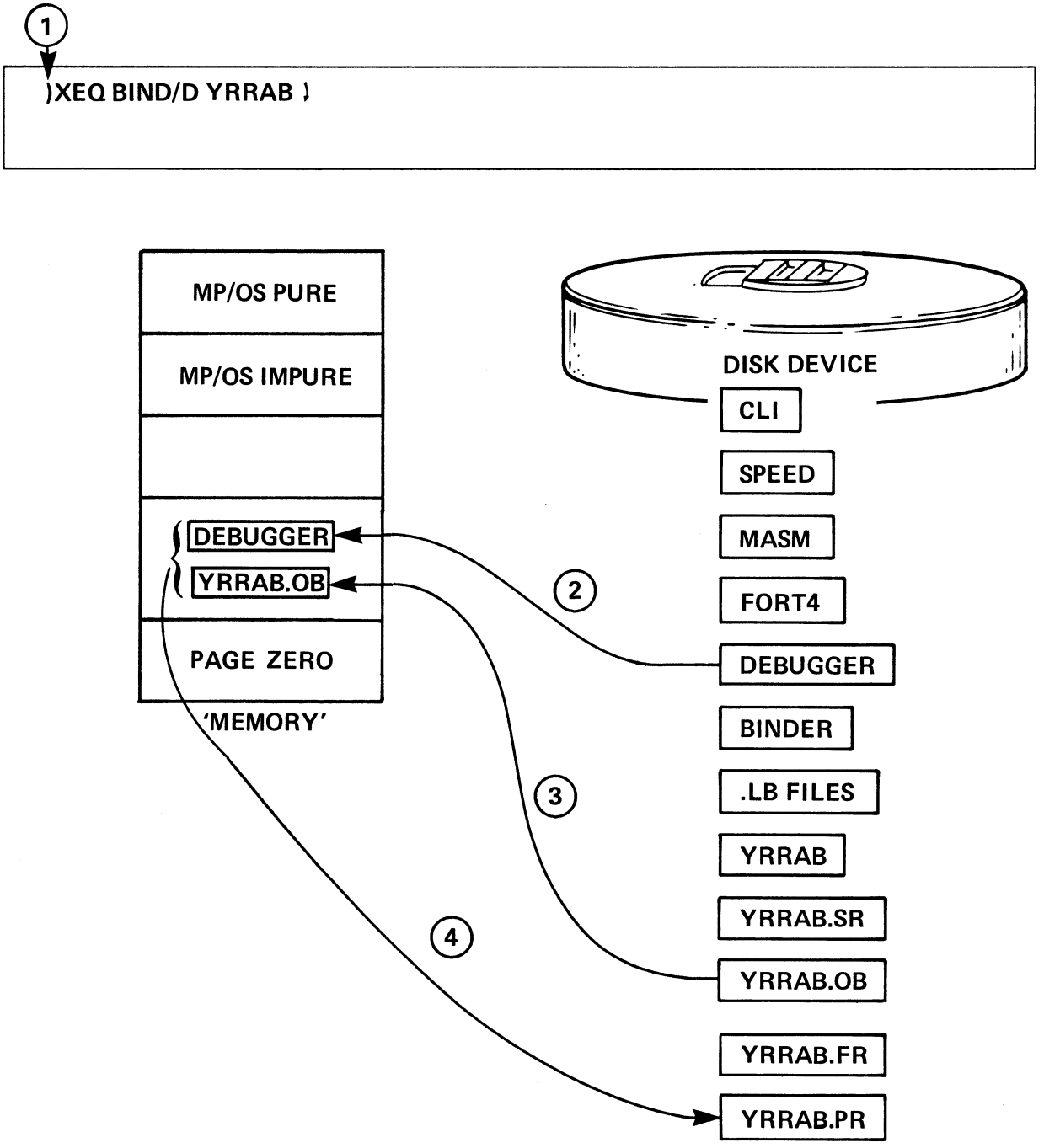
BINDER TERMINATES, CLI RETURNS

Figure 4-10



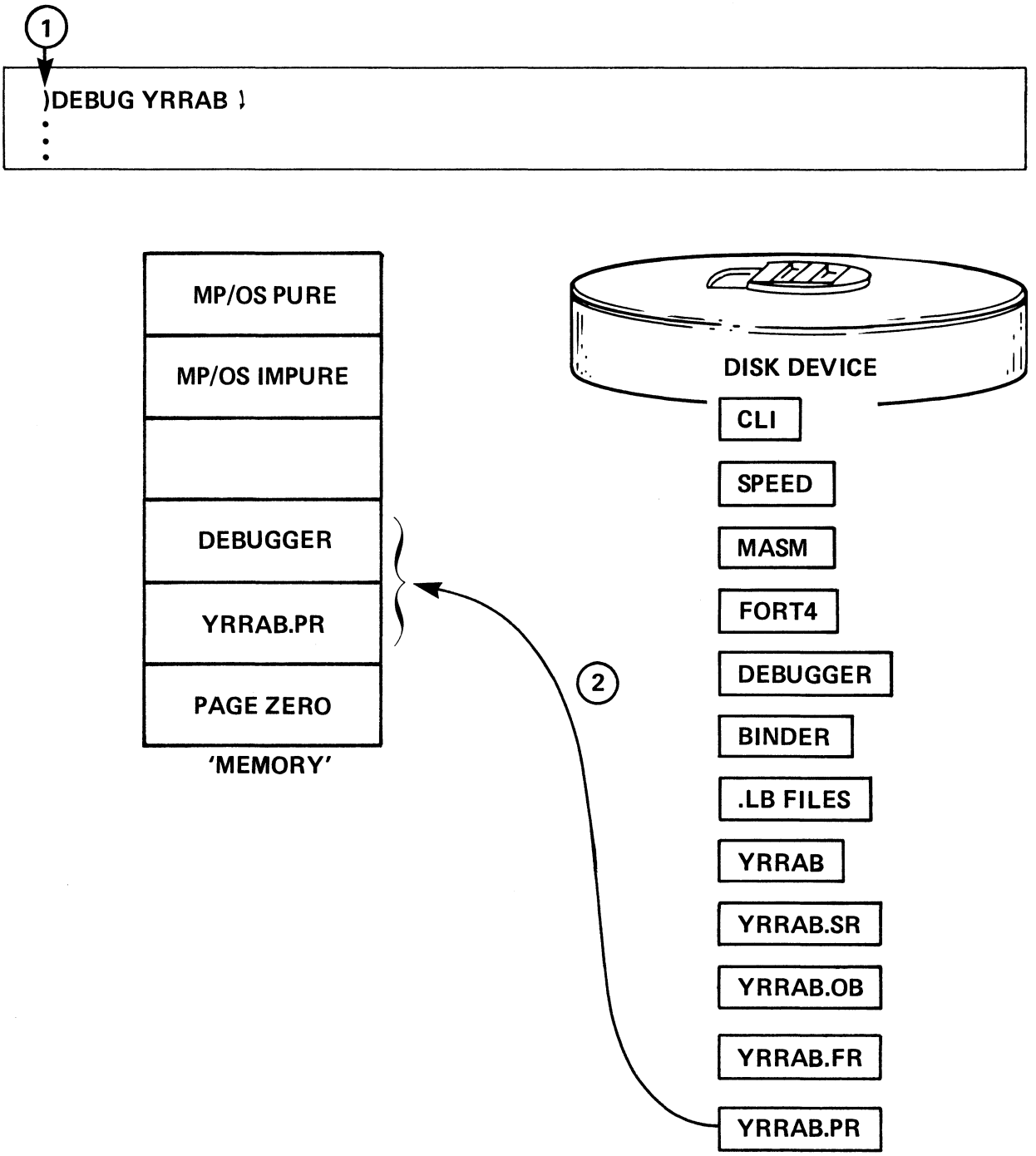
EXECUTING THE PROGRAM FILE

Figure 4-11



INVOKING THE DEBUGGER

Figure 4-12

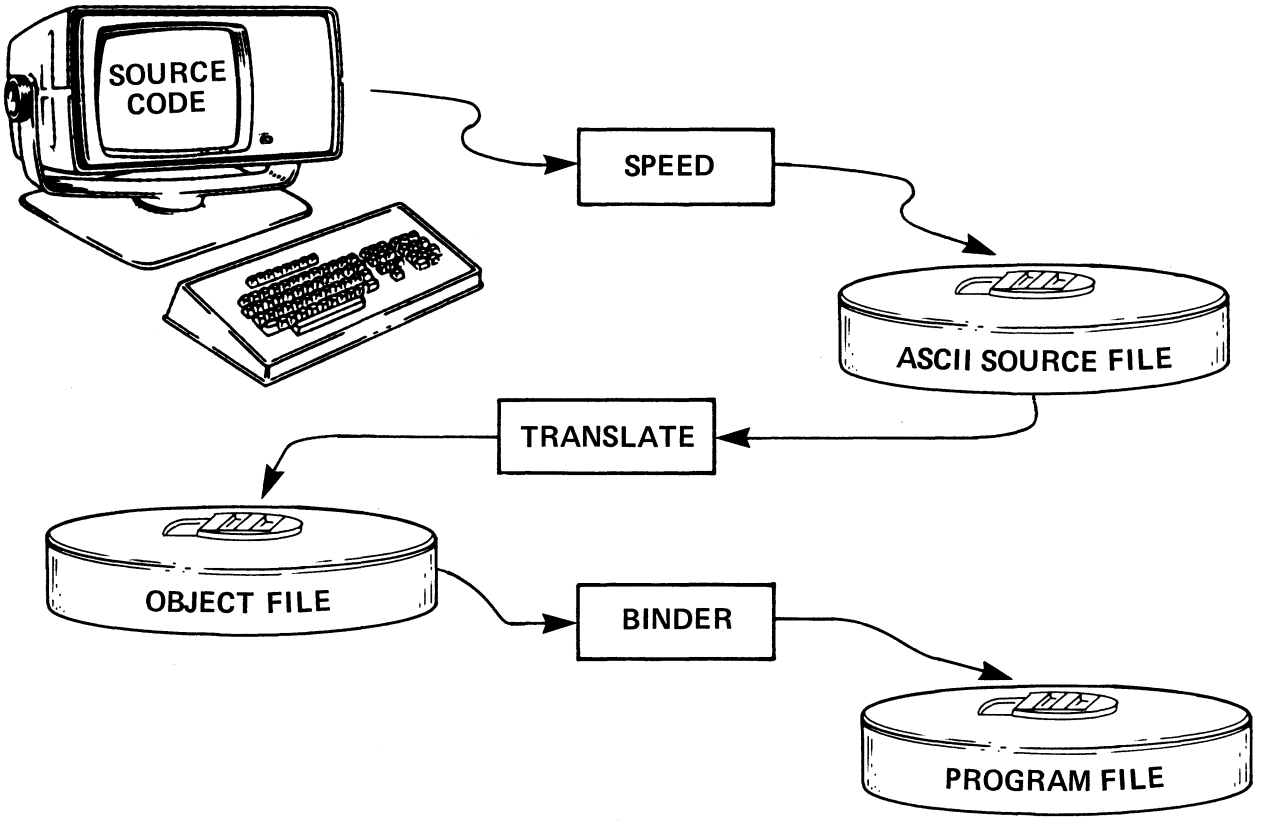


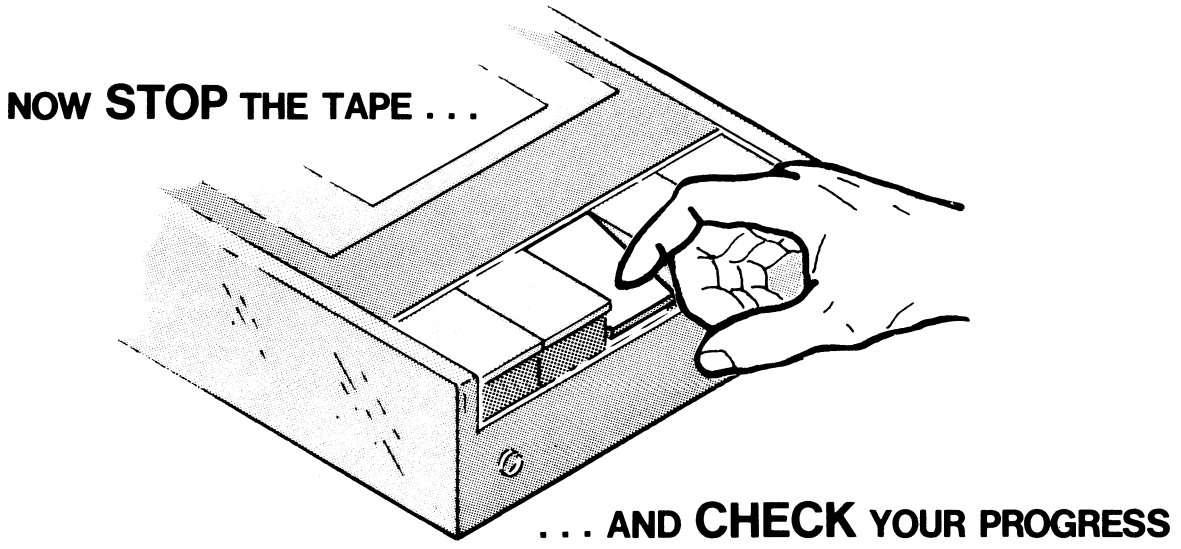
DEBUGGING THE PROGRAM FILE

Figure 4-13

TOPICS

PROGRAM DEVELOPMENT SIMPLIFIED





PROGRAM DEVELOPMENT
QUIZ

Write the answers to the following questions in the space provided.

1. State, in MP/OS terminology, the major steps in program development.

_____

_____

_____

_____

_____

2. State the purpose, input, and output of the following utilities:

Macroassembler: Purpose:_____

_____

Input:_____

Output:_____

Compiler: Purpose:_____

_____

Input:_____

Output:_____

Speed: Purpose:_____

_____

Input:_____

Output:_____

Debugger: Purpose:_____

_____

Input:_____

Output:_____

Binder Purpose:_____

_____

Input:_____

Output:_____

3. Each utility program is loaded into _____ space, which begins at address_____

4. Each utility program is invoked by a _____ command.

5. Each utility program, by default, stores its output on_____.

NOW CHECK YOUR ANSWERS

ON THE NEXT PAGE . . . .

PROGRAM DEVELOPMENT
QUIZ ANSWERS

1. The major steps in Program Development are:

Enter source code into the system through SPEED
Translate the source code into object code through the compiler
and/or Macroassembler;
Bind the object code into a program file;
Debug the program file (if necessary);
Execute the final program file.

2. The purpose, input, and output of the utilities:

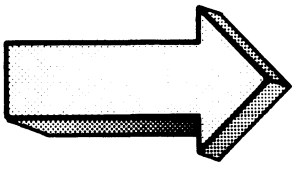
Macroassembler:	Purpose:	Translate MP/Assembly Language Source Code into Object Code.
	Input:	Source Code in ASCII format
	Output:	Object file, or Object code, or Object module
Compiler:	Purpose:	Translate MP/Pascal or MP/Fortran source code into object code (with the assistance of the Macroassembler).
	Input:	ASCII characters (source code).
	Output:	Object file or object code
Speed:	Purpose:	Create and modify source files.
	Input:	ASCII characters entered on the console.
	Output:	ASCII source code file.
Debugger:	Purpose:	Monitors program execution, allows stops, starts, displays, and alterations.
	Input:	Program file.
	Output:	Program file.
Binder:	Purpose:	Bind object code into a program file.
	Input:	Object code or object modules and Library Files.
	Output:	Program file.

3. Each utility is loaded into *user* space, which begins at address *400*.

4. Each utility program is invoked by a *CLI* command.

5. Each utility program stores its output on *the system disc or Disc Device.*

A SCORE OF 19 CORRECT ANSWERS OUT OF THE 23 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.



MACROASSEMBLER CONCEPTS

Abstract

This segment discusses the concepts involved in translating MP/Assembly Language Source Files into Object Files.

It is highly recommended that Pascal and Fortran programmers complete this segment.

Objectives

Upon completion of this segment you will be able to:

1. State, from memory, the possible Macroassembler outputs.

2. Name, define, and give examples of two Macroassembler input modes.

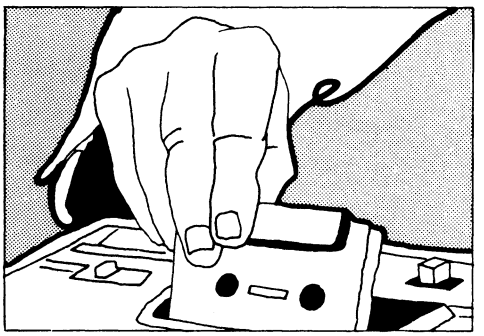
3. Given a Macroassembler program listing, identify examples of Macroassembler input modes.

4. Given a Macroassembler program listing, identify the following:

    A) line number
    B) error flag
    C) location counter
    D) relocation flag
    E) data field
    F) source line

5. Given an example of a cross-reference listing, identify:

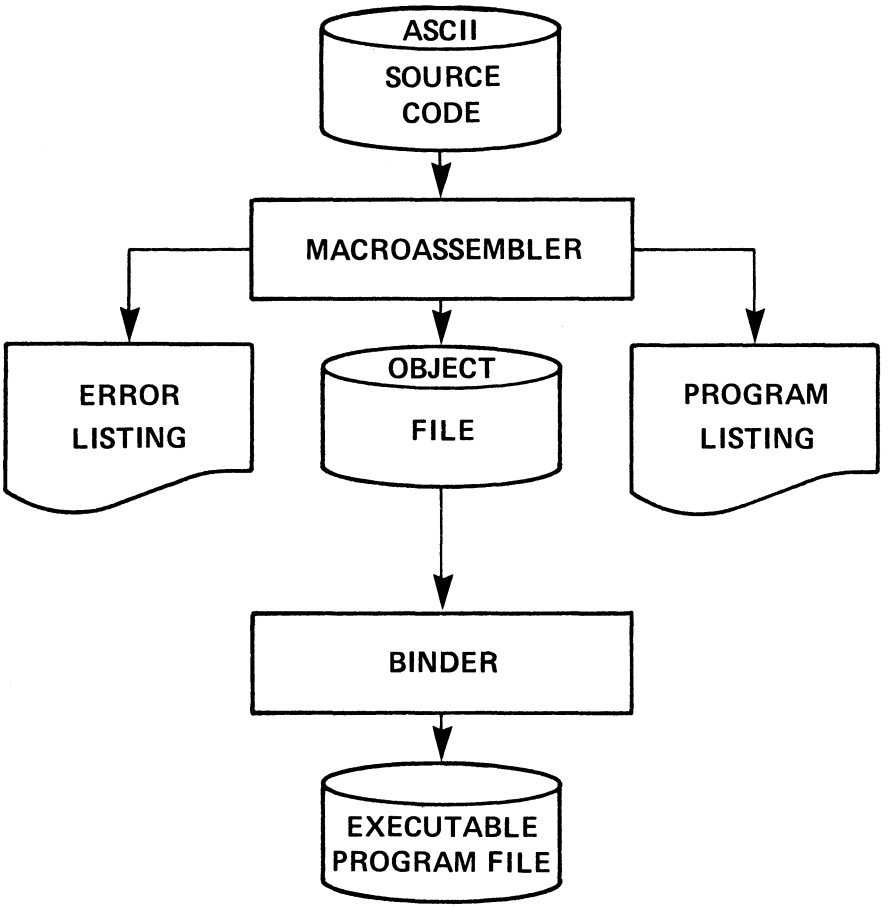
    A) symbol
    B) relative address
    C) reference page and line
    D) symbol type



Directions

1. Turn to the figure 4-14 in your Student Guide.

2. Listen to the tape for this segment of Module Four.



MACROASSEMBLER INPUT AND OUTPUT

Figure 4-14

STRING MODE

character strings accepted literally.

NORMAL MODE

character strings accepted as a series of atoms which may have symbolic interpretation.

MACROASSEMBLER INPUT MODES

Figure 4-15

STRING MODE

character strings accepted literally

```
1.  COMMENTS                ;THIS IS A COMMENT.

2.  MACRO DEFINITIONS       .MACRO      SAMPLE
                            LDA         0,LENGTH
                            LDA         1,WIDTH
                            MUL         0,1
                            STA         0,AREA

3.  TEXT STRINGS            .TXT        /THIS IS A SAMPLE TEXT STRING/
                            .TXTM       THE WIDTH IS AREA LENGTH
                            .TXTE       ATHE MULTIPLY SYMBOL IS A
```

MACROASSEMBLER STRING MODE INPUT

Figure 4-16

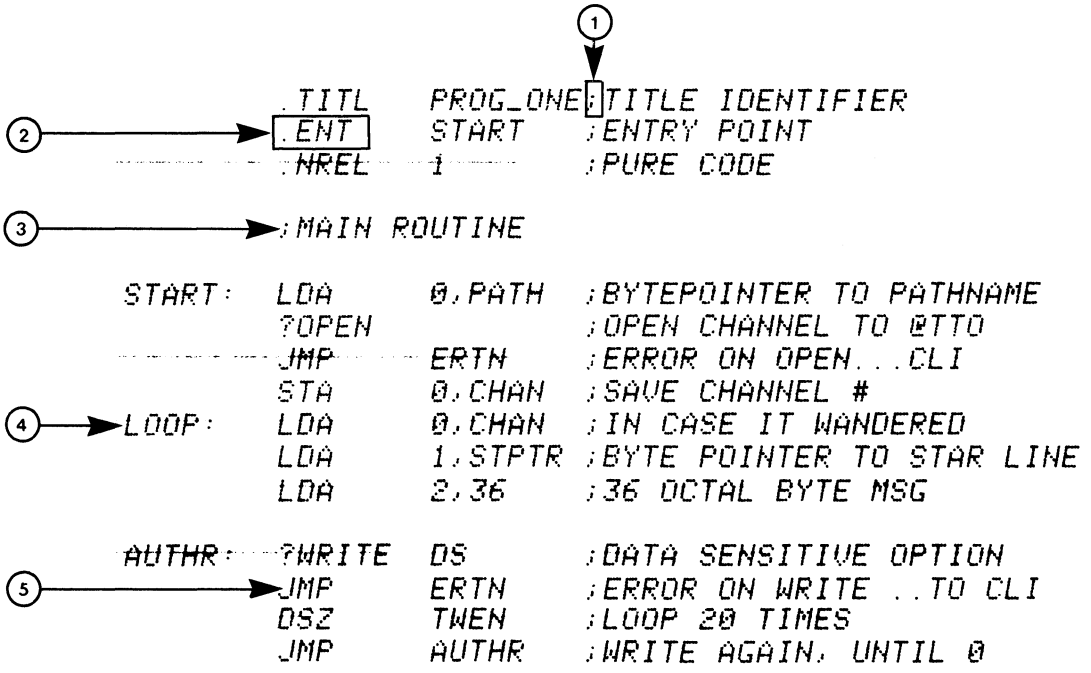
NORMAL MODE

Character strings accepted as ATOMS

1.	SYMBOLS		EXAMPLES
	PERMANENT SYMBOLS		
		Pseudo ops	.BLOCK .LOC
		Pseudo ops & values	.PASS, .MCALL
	SEMI-PERMANENT SYMBOLS		
		Instruction Mnemonics	JMP, MOV
	USER-DEFINED SYMBOLS		
		Location Name	TEMP1
		External Name	POST
		Global Name	BASE
2.	TERMINALS		
	OPERATOR TERMINALS		
		Shift	B
		Arithmetic	+, –
		Logical	&, !
		Relational	<, >
	BREAK TERMINALS		
		Spaces	△, □, ƀ
		Parentheses	( )
		Brackets	[ ]
		;	Comments
		<CR>	New-line
3.	NUMBERS		
	UNSIGNED	0 to 65535	
	SIGNED	–32, 768 to 32,767	
4.	SPECIALS		
	INDIRECT BIT	@	
	NUMBER SIGN	#	
	ASTERISKS	**	

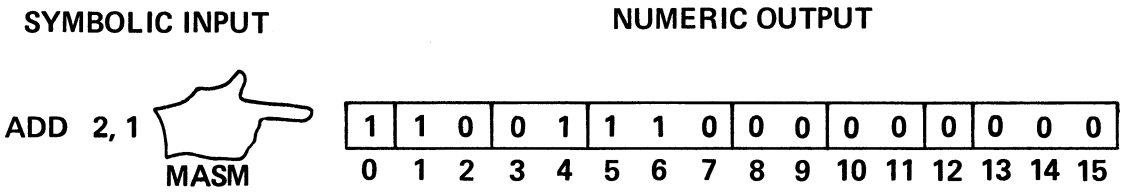
MACROASSEMBLER NORMAL MODE INPUT

Figure 4-17



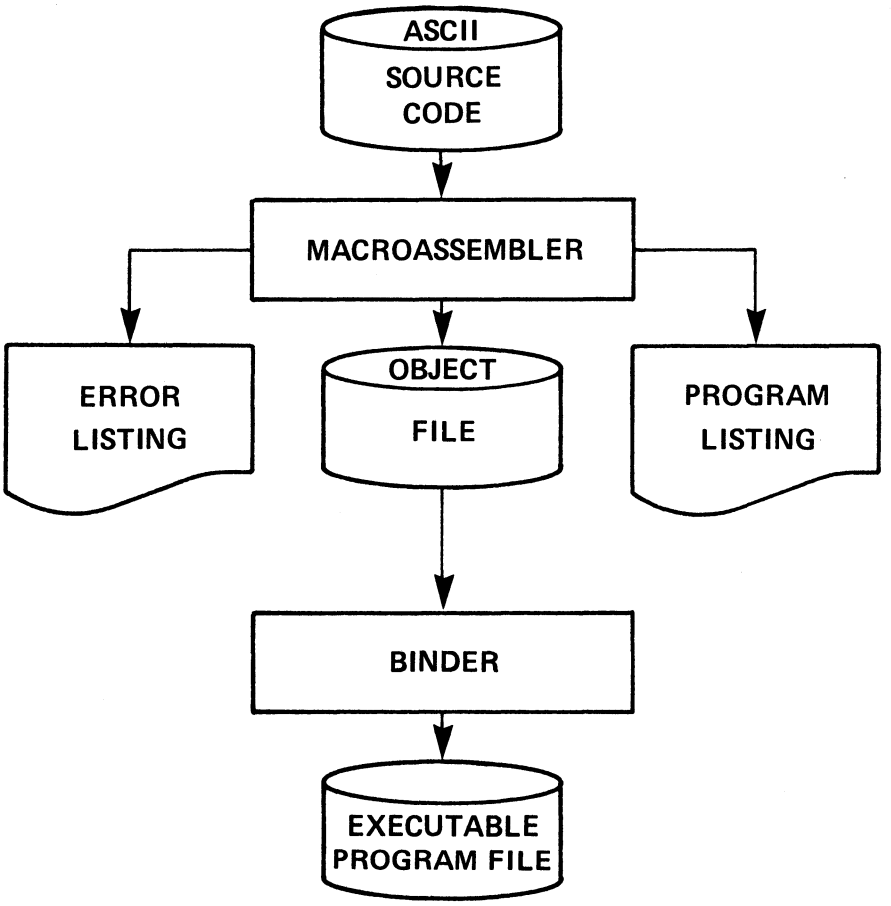
A MACROASSEMBLER INPUT STREAM

Figure 4-18



MACROASSEMBLER OUTPUT

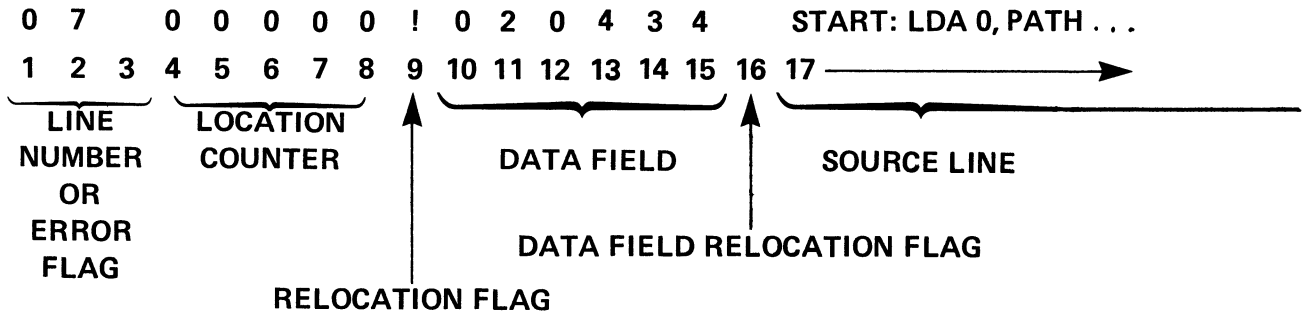
Figure 4-19



MACROASSEMBLER INPUT AND OUTPUT

Figure 4-20

```
 0001 PROG_  MP/OS  ASSEMBLER REV 1.00              06:18:79 11/51/42
                            .TITL   PROG_ONE;TITLE IDENTIFIER
 02                         .ENT    START   ;ENTRY POINT
 03         000001          .NREL   1       ;PURE CODE
 04
 05                         ;MAIN ROUTINE
 06
 07  00000!020434 START:    LDA     0,PATH  ;BYTEPOINTER TO PATHNAME
 08                         ?OPEN           ;OPEN CHANNEL TO @TTO
 09  00003!000426           JMP     ERTN    ;ERROR ON OPEN...CLI
 10  00004!040427           STA     0,CHAN  ;SAVE CHANNEL #




 20         000040          .LOC    40      ;LOC 40 HAS ...
 21  00040  000000'         STK             ;... THE STACK POINTER
 22  00041  000000'         STK             ;LOCATION 41
 23  00042  000045'         STK+45          ;LOC 42 HAS STACK LIMIT
 24
 25                         .END    START   ;END OF ASM INPUT
 26

 *#00000 TOTAL ERRORS, 00000 PASS 1 ERRORS  0003 PROG_
```



A MACROASSEMBLER PROGRAM LISTING

Figure 4-21

COLUMN NINE FLAG	MEANING
(space)	Absolute Address
–	Page zero relocatable Address
,	Impure code Address
!	Pure code Address

The location counter relocation flag indicates where the instruction or data field is located in memory.

INSTRUCTION ADDRESS RELOCATION MODE SYMBOLS

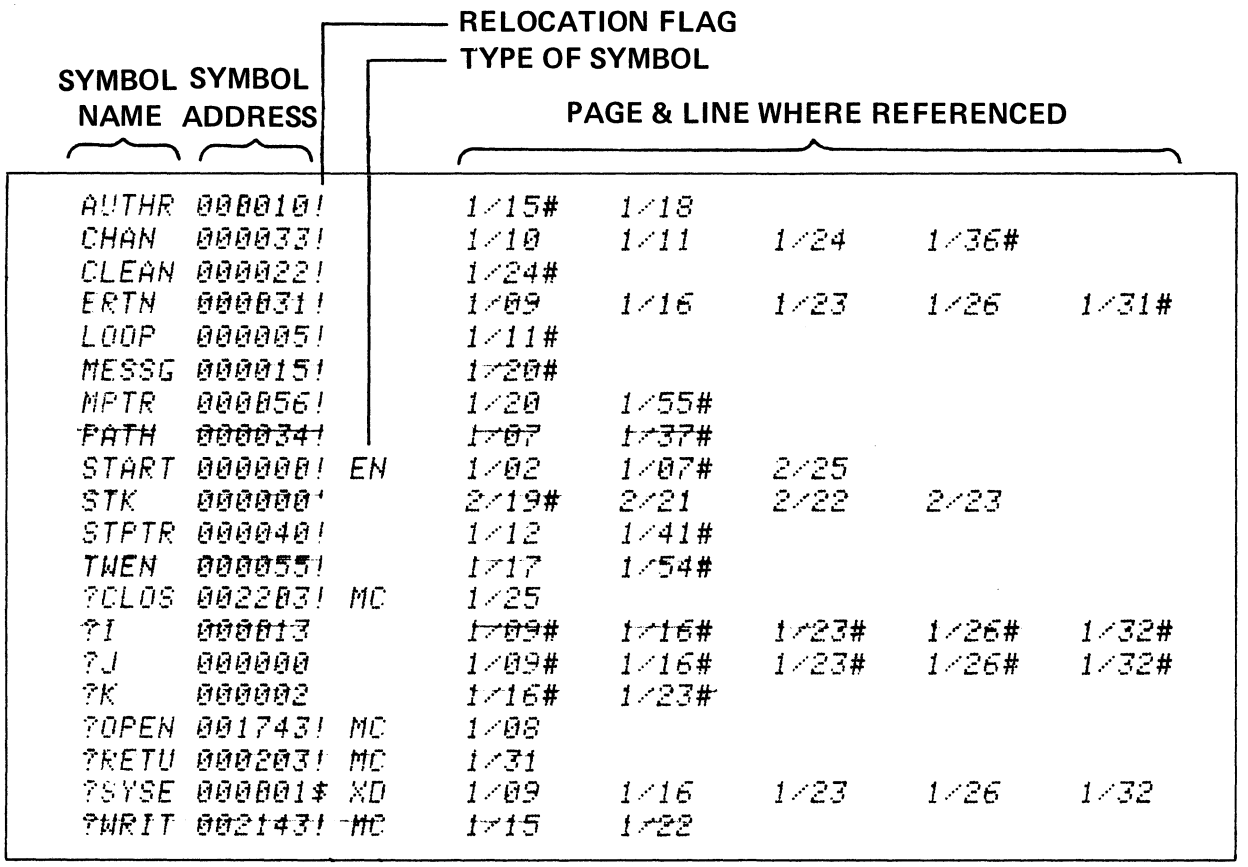
Figure 4-22

DATA FIELD RELOCATION FLAG	FLAG MEANING
(space)	Absolute
–	Page zero relocatable
'	Impure code, word relocatable
!	Pure code, word relocatable
"	Impure code, byte relocatable
&	Pure code, byte relocatable
$	Displacement field is externally defined

The data field relocation flag indicates where the instruction's data field is located in memory.

DATA FIELD RELOCATION FLAGS

Figure 4-23



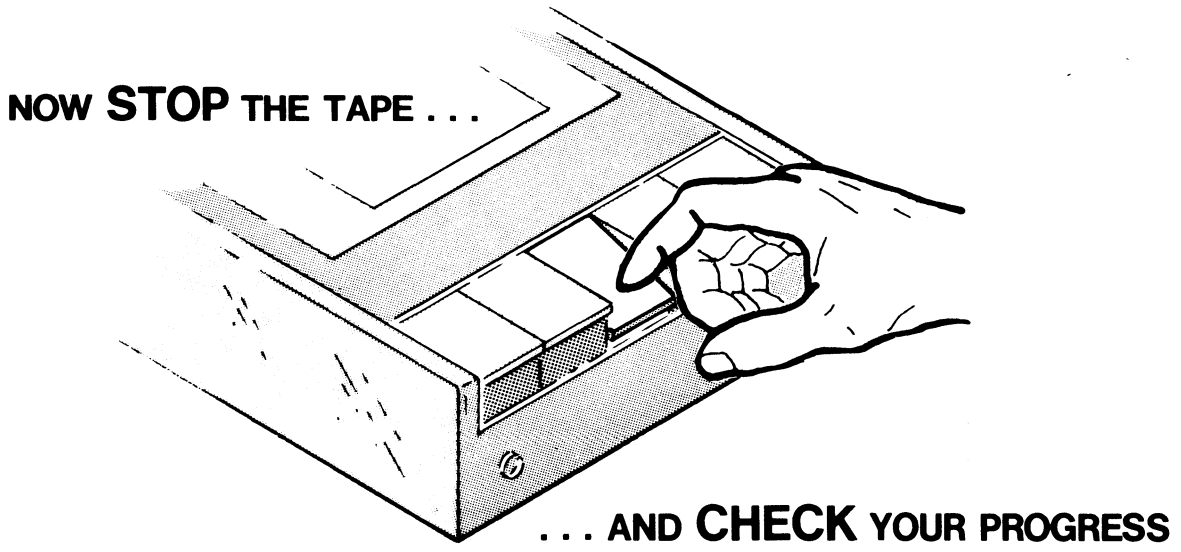
SYMBOL	MEANING
(spaces)	User symbol
EN	Entry (defined in .ENT statement)
EO	Overlay entry (defined in .ENTO statement)
XD	External displacement (defined in .EXTD statement)
XN	External normal (defined in .EXTN statement)
NC	Named common (defined in .COMM statement)

CROSS-REFERENCE SYMBOL TYPES
A MACROASSEMBLER CROSS-REFERENCE LISTING

Figure 4-24

TOPICS

- MASM CONCEPTS
- MASM INPUT (MODES, STRING & NORMAL)
- MASM OUTPUT OPTIONS (ERROR LIST, PROGRAM LIST)



MACROASSEMBLER CONCEPTS
QUIZ

Circle the appropriate answers to the following questions. Note that there may be more than one correct answer to a question.

PART I

1. Macroassembler output may include:
   - A. Error listing
   - B. Object file
   - C. Program listing
   - D. Cross-reference listing

2. The Macroassembler accepts input in the following modes:
   - A. String
   - B. Normal
   - C. Extended
   - D. Basic

3. Examples of *string mode* Macroassembler input include: (the question refers only to the circled area)

   A. LDA 1,BTPTR ;LOAD THE BYTEPOINTER

   B. .TXT /THE MICRON HAS LANDED/

   C. LDA 2,CHAN

   D. SUB # 2,2

4. Examples of *normal mode* Macroassembler input include: (The question refers to the circled area)

A.	START:	STA	3,SAV	;SAVE RETURN ACCUMULATOR
B.	START:	STA	3,SAV	;SAVE RETURN ACCUMULATOR
C.	START:	STA	3 , SAV	;SAVE RETURN ACCUMULATOR
D.	START:	STA	3,SAV	;SAVE RETURN ACCUMULATOR

PART II

Fill in the spaces with the answers to the following questions:

Following is a line from a Macroassembler program listing:

```
11 00005!020560 OPEN:      LDA      0,LPPTR      ;BYTE POINTER TO LPT
```

Identify the contents of the following fields:

5. Location counter:_____

6. Error flag:_____

7. Line number:_____

8. Source line:_____

9. Statement relocation flag:_____

10. Data field relocation flag:_____

Following is a line from a Macroassembler cross-reference listing:

```
START    000000    EN  1/02    1/07    3/39
```

Identify the following elements:

11. Symbol:_____

12. (Relative) address of symbol:_____

13. Pages where referenced: _____

14. Lines where referenced: _____

15. Type of symbol:_____

16. Relocation flag:_____

NOW CHECK YOUR ANSWERS

ON THE FOLLOWING PAGE.

MACROASSEMBLER CONCEPTS
QUIZ ANSWERS

1. Macroassembler output may include:

Ⓐ	Error listing	In print, if requested, else on screen
Ⓑ	Object file	as .OB, by default if error-free
Ⓒ	Program listing	in print, if requested, or on disc, if specified.
Ⓓ	Cross-reference	in print, only if requested, or on disc if specified.

2. The Macroassembler accepts input in the following modes:

Ⓐ	String	Examples: text strings and comments
Ⓑ	Normal	Examples: instruction mnemonics
C.	Extended	Not applicable
D.	Basic	Incorrect

3. Examples of string mode macroassembler input include:

Ⓐ	LDA	1,BTPTR	;LOAD THE BYTEPOINTER	(COMMENT)
Ⓑ	.TXT	/THE MICRON HAS LANDED/		(TEXT STRING)
Ⓒ	LDA	2,CHAN		(SEMI-PERMANENT SYMBOL)
D.	SUB#	2,2		(SPECIAL)

4. Examples of normal mode macroassembler input include:

A.	START:	STA	3,SAV	;SAVE RETURN ACC	(UDF SYMBOL)
B.	START:	STA	3,SAV	;SAVE RETURN ACC	(SEMIPERM SYMB)
C.	START:	STA	3,SAV	;SAVE RETURN ACC	(BREAK TERM)
D.	START:	STA	3,SAV	;SAVE RETURN ACC	(COMMENT)

Following is a line from a Macroassembler program listing:

```
11      00005!020560      OPEN:     LDA      0,LPPTR  ;BYTE POINTER TO LPT
```

Identify the following elements:

5. Location counter: 00005

6. Error flag: None, error-free line

7. Line number: 11

8. Source line: LDA 0,LPPTR

9. Relocation flag: !

10. Data field relocation flag: SPACE or BLANK

Following is a line from a Macroassembler cross-reference listing:

```
        START    000000    EN   1/02    1/07    3/39
```

Identify the following elements:

11. Symbol: START (user defined symbol)

12. (Relative) address of symbol 000000 (first address in program)

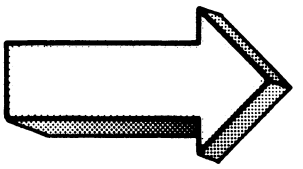
13. Pages where referenced: 1 (Twice) and 3.

14. Lines where referenced: 2 and 7 (page 1) and 39 (page 3)

15. Type of symbol: EN (Entry point)

16. Relocation flag: SPACE or BLANK

A SCORE OF 13 CORRECT ANSWERS OUT OF THE 16 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.



MACROASSEMBLER PROCEDURES

Abstract

This segment covers the procedures for translating assembly language programs from source code to object code.

Objectives

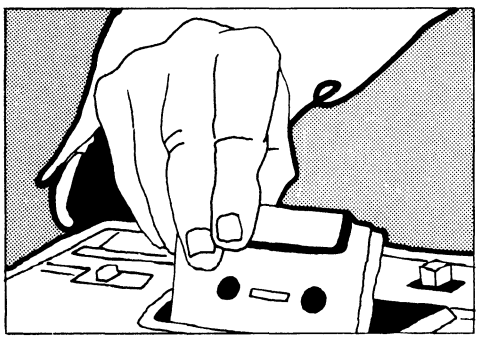
Upon completion of this segment you will be able to:

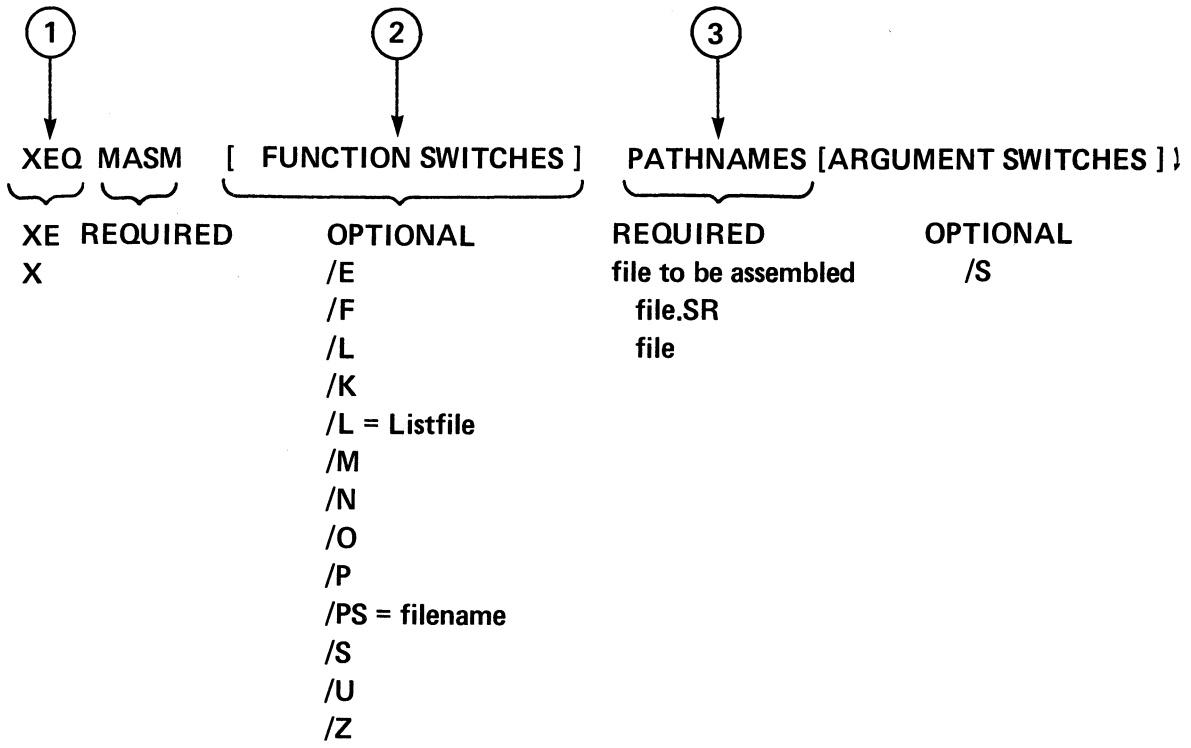
1. Given a CLI Macroassembler command line, identify and state the purpose of switches and arguments;
2. List the reference material available on Macroassembler commands (text and system);
3. Given a CLI Macroassembler command line, describe the results;
4. Write an appropriate CLI Macroassembler command line to assemble a given assembly language source program. State the output names with extensions;
5. Given an assembler error situation:
    A) Identify possible causes,
    B) Reference solutions,
    C) Write CLI and/or Macroassembler lines to correct the error.

Directions

1. Turn to Figure 4-25 on the next page of the Student Guide.

2. Listen to the tape for this segment.





CLI MACROASSEMBLER COMMAND LINE

Figure 4-25

SYMBOL	ACTION
/B = filename	Gives the name *filename* to the object file. Ordinarily the object file has the name of the first source file in the assembly command line, less the extension .SR (if any) and with a new extension .OB.
/E	Suppresses the error listing if there is a listing file. If there is no listing file, then errors will be displayed on the console despite the /E. See Appendix F for a description of error flags.
/F	Generates or suppresses form feeds as required to produce an even number of assembly pages. This keeps the first page of successive listings on the outsides of paper folds, making refolding unnecessary. By default, the macroassembler generates a form feed at the end of a listing, whether the number of pages is odd or even.
/K	Keeps the macroassembler's temporary symbol file at the end of assembly. Since virtually no programs require the use of this file, the macroassembler deletes it by default.
/L	Produces a listing file on the line printer. Listings always includes a cross reference of symbols in the program showing the page and line number where each symbol is used. If you use the /L switch, program MASMXR.PR must be present in the same directory as the macroassembler itself. MASM.PR itself is found, but, if it is missing, then an error message will be displayed at your console.
/L-*list-file*	Produces a listing file, but instead of sending it to the line printer, sends it to the file designated by *listfile.* If there is already a listing in this file, then the new listing allows it. *Listfile* can be any filename or pathname permitted by the operating system.
/M	Flags any redefinition of semipermanent symbols as multiple definition errors (M).
/N	Produces no object file.
/O	Overrides all listing control pseudo ops: .NOCON, .NOLOC, and .NOMAC. Also overrides listing suppression.

(CONTINUED)

/P	Adds semipermanent symbols to the cross-reference listing. By default they are not included.
/PS-*filename*	Uses *filename* instead of MASM.PS to build symbol table file.
/R	Produces a binary file even if there is an assembly error. By default, if there is an assembly error, the macroassembler does not produce a binary file.
/S	Skips the second assembly pass (produces no.OB file) and saves the macroassembler's symbol table, renaming it MASM.PS. (See below, "Macroassembler Symbol Table Files.")
/U	Includes user symbols in the object file. When the /U switch is also applied to the binder command line, then the debugger is able to find user symbols. This makes debugging easier.
/Z	Lists the DGC proprietory license heading at the top of each assembly and cross-reference page. By default this heading is not listed. Note that this switch is useful to DGC personnel only.

CLI MACROASSEMBLER COMMAND LINE SWITCHES

Figure 4-26

SYMBOL	ACTION
Source-file /S	Skips the file named *sourcefile* on the second pass of assembly. *Sourcefile* must not define any storage words. Typical files that might be skipped include parameter definition files and macro definition files. Skipping such a file on the second assembly pass does not hinder the assembly of other files in the command line. It merely decreases the size of the output listing and reduces assembly time.

MASM ARGUMENT SWITCH

Figure 4-27

	INPUT	OUTPUT
1		
`) XEQ MASM ZORP↓`	ZORP	ZORP.OB CONSOLE ERROR LIST
2		
`) XEQ MASM/L=@LPT ZORP↓`	ZORP	ZORP.OB ERROR LIST ON CONSOLE PROGRAM LISTING ON LPT

/L . . . produce a program listing

	INPUT	OUTPUT
3		
`) XEQ MASM/L=WEBO ZORP↓`	ZORP	ZORP.OB WEBO PROGRAM LISTING ERRORS ON CONSOLE

/L = filename . . . send program listing to disc.

Figure 4-28

	INPUT	OUTPUT
1		
) XEQ MASM/B=BARRY ZORP↓	ZORP	BARRY.OB ERRORS ON CONSOLE

/B = filename . . . name object file with specified filename.

2		
) XEQ MASM/E ZORP↓	ZORP	ZORP.OB ERRORS ON CONSOLE

/E . . . produce error list on console.

3		
) XEQ MASM/E/L=TALLY ZORP↓	ZORP	ZORP.OB TALLY PROGRAM LISTING WITHOUT ERRORS. ERRORS ON CONSOLE

/E/L = filename . . . Write error file to disc with specified filename.

Figure 4-29

	INPUT	OUTPUT
1		
`> XEQ MASM/N PLOP↓`	PLOP	ERRORS ON CONSOLE

/N . . . suppress object file.

2		
`> XEQ MASM/R PLOP↓`	PLOP	PLOP.OB ERRORS ON CONSOLE

/R . . . create object file despite errors.

3		
`> XEQ MASM/S PLOP↓`	PLOP	ERRORS ON CONSOLE

/S . . . skip pass 2, suppress object file, build MASM symbol table.

Figure 4-30

	INPUT	OUTPUT
1		
XEQ MASM/U ZORP BARRY ↵	ZORP BARRY	ZORP.OB BARRY.OB ERRORS ON CONSOLE
/U . . . add user-defined symbols to the object files.		
2		
) XEQ MASM/P ZORP PLOP ↵	ZORP PLOP	ZORP.OB PLOP.OB ERRORS ON CONSOLE
/P . . . include semi-permanent symbols in cross-reference listing.		
3		
) XEQ MASM/L=@LPT/Z BARRY.SR ↵	BARRY.SR	BARRY.OB ERRORS ON CONSOLE PROGRAM LISTING ON LPT WITH NOTICE
/Z . . . add DGC proprietary notice to program listing.		

Figure 4-31

TOPICS

- MASM PROCEDURES
- MASM COMMAND LINE
  - SWITCHES
  - ARGUMENT
- MASM INPUT
- MASM OUTPUT

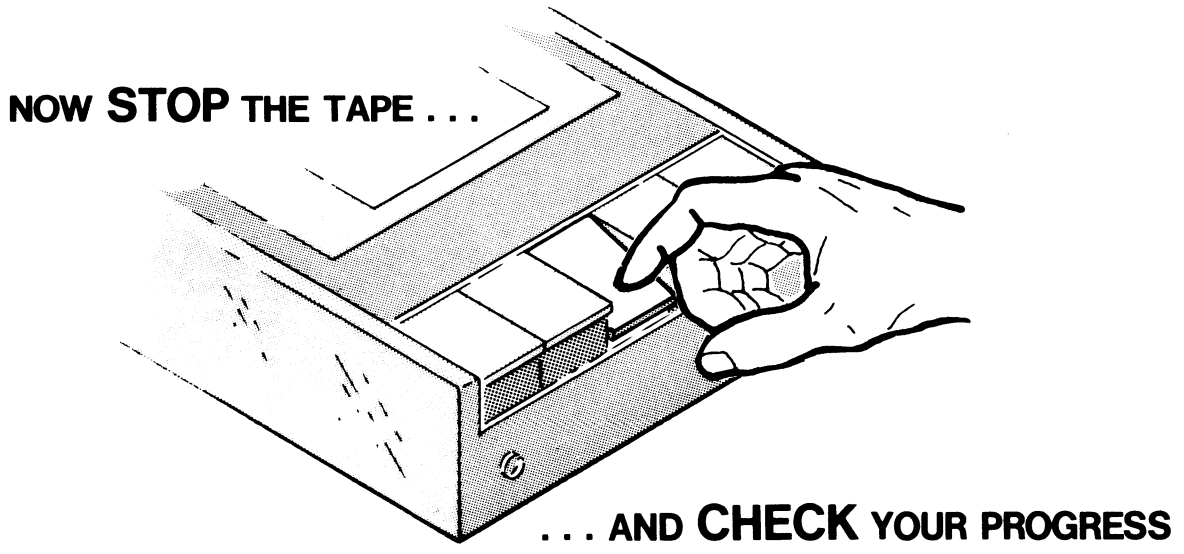


Figure 4-32

MACROASSEMBLER PROCEDURES
QUIZ

Directions. Fill in the space with the appropriate answers.

Given the following CLI Macroassembler command line, identify the switches and arguments and state the purpose of the switches and arguments.

XEQ MASM/L/P/U/Z MYPROG

1. Switches:_____

2. Arguments:_____

3. Purpose of the switches:_____

_____

_____

_____

4. Purpose of the argument:_____

_____

List three sources of information on Macroassembler commands:

5._____

6._____

7._____

Given the following CLI macroassembler command lines; briefly describe the anticipated results (assume a successful assembly):

) XEQ MASM ONEFILE

8._____

_____

) XEQ MASM/L=TWO.LS TWO

9._____

) XEQ MASM/B=THREE THREEFILE

10. _____

_____

) XEQ MASM/N FOURPROG

11. _____

_____

) XEQ MASM/U FIVEPROG

12. _____

_____

NOW CHECK YOUR ANSWERS

ON THE FOLLOWING PAGE.

MACROASSEMBLER

QUIZ ANSWERS

The switches, arguments, and purposes are as follows:

) XEQ MASM/L/P/U/Z MYPROG

1. Switches: /L /P /U /Z
2. Arguments: MYPROG
3. Purpose of the switches: "/L" produces a listing file on the line printer, if your system is so configured.

   "/P" adds semi-permanent (instruction mnemonics) symbols to the cross-reference listing. Note that some form of the "/L" switch must be included to produce a listing.

   "/U" includes user symbols (e.g., labels) in the object file.

   "/Z" adds the Data General proprietary license notice to the top of each listing page. Note that some form of the "/L" switch must be included for this to be worthwhile. Note also that this is useful to Data General employees only.

4. Purpose of the argument: "MYPROG" identifies the source language file that is to be assembled. The object file will be named "MYPROG.OB". Either MYPROG.SR or MYPROG will be input to MASM.

Sources of Macroassembler information include the following:

5. The CLI "HELP" command, if your system is configured with this facility on your media.
6. MP/OS Utilities Reference Manual (D.G. No. 093-400002)
7. This self-study course. Also, MP/OS Assembly Language Programmer's Reference (D.G. No. 093-400001).

The anticipated results of the CLI command lines are as follows: (assume a successful assembly).

) XEQ MASM ONEFILE

8. "ONEFILE" is assembled. "ONEFILE.OB" is created and saved on disc. No listing is produced. Errors are displayed on the console.

) XEQ MASM/L=TWO.LS TWO

9. "TWO" will be assembled. "TWO.OB" will be created as the object file and saved on disc. The listing will be written to "TWO.LS" and saved on disc. Errors will be directed to the system console (as well as the listing file).

) XEQ MASM/B=THREE THREEFILE

10. "THREEFILE" will be assembled. "THREE.OB" will be created as the object file and saved on disc. No listing will be created or printed. Errors will be displayed on the system console.

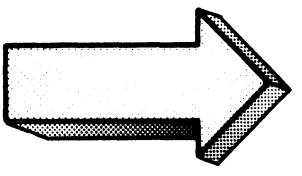
) XEQ MASM/N FOURPROG

11. "FOURPROG" will be input to the macroassembler, but no object file will be produced. Errors will be listed on the system console. No listing file will be produced.

) XEQ MASM/U FIVEPROG

12. "FIVEPROG" will be assembled. "FIVEPROG.OB" will be created as the object file and saved on disc. The user symbols (e.g., labels) will be included in the object file. Note that this makes debugging easier.

A SCORE OF 10 CORRECT ANSWERS OUT OF THE 12 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.



MACROASSEMBLER PROCEDURES

LAB EXERCISE

Directions

This lab exercise is similar to the labs in previous modules. That is, it may be completed with or without a functional system. For maximum benefit from the exercise it is imperative that you write in all answers before attempting to enter anything on the system.

First make sure that you have the following files on your media.

MASM.PR. . . The Macroassembler
MASMXR.PR. . . Produces the cross-reference listing. Not absolutely essential for assembly, but necessary for this lab.
MSL.LB. . . The macroassembler library routines.
MASM.PS. . . Assembler symbol file. Contains the following:
- NBID.SR. . . Nova basic instruction definitions
- NSKID.SR...Nova skip instruction definitions
- MP100ID.SR...MP/100 additional definitions
- SYSID.SR...System call definitions
- MPARU.SR... MP/OS user parameter file
- MP200ID.SR...MP/200 additional definitions (only required on MP/200 machines).

The CLI FILESTATUS command will assist you in your search. If you do not have MASM.PS you can create it with the following procedure:

TYPE the contents of the .SR files listed above. Use the CLI TYPE command to accomplish this. Read through the files to identify which macroassembler statments you need in the MASM.PS symbol file. For example in the lab that follows, we created a MASM.PS file from the NBID.SR and SYSID.SR files, and left out the other files. To do this we entered the following:

```
) XEQ MASM/S NBID.SR SYSID.SR ↓
```

The "/S" switch indicates that the macroassembler is producing a new symbol file, titled MASM.PS. This file is used during the assembly process.

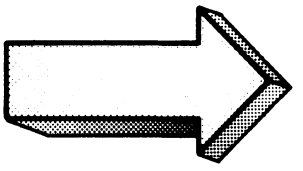
Now enter the file titled "PROG_ONE" into your system. Use SPEED to create the new file. Be very careful that the statements and statement labels are entered exactly as shown in PROG_ONE. It is not necessary to enter the comments, although they may prove helpful. You will use PROG_ONE as the subject of the following exercises. Once you are certain that PROG_ONE is clean, make a backup copy of it. Use the CLI COPY command as shown:

```
) COPY PROG_ONE.BU  PROG_ONE )
```

The FILESTATUS command showed the results on our system:

```
) FI/AS/S )
DIRECTORY @DPX1:

  MASM.PS            PST   18-JUN-79  11:09:27      143360
  PROG_ONE           TXT   15-JUN-79  16:52:56        1302
  PROG_ONE.BU        TXT   15-JUN-79  16:56:29        1302
)
```

You are now ready to try the questions. Remember to cover the answers until you have written in your answer.



```
) TYPE PROG_ONE)
        .TITL   PROG_ONE;TITLE IDENTIFIER
        .ENT    START   ;ENTRY POINT
        .NREL   1       ;PURE CODE

        ;MAIN ROUTINE

START:  LDA     0,PATH  ;BYTEPOINTER TO PATHNAME
        ?OPEN           ;OPEN CHANNEL TO @TTO
        JMP     ERTN    ;ERROR ON OPEN...CLI
        STA     0,CHAN  ;SAVE CHANNEL #
LOOP:   LDA     0,CHAN  ;IN CASE IT WANDERED
        LDA     1,STPTR ;BYTE POINTER TO STAR LINE
        LDA     2,36    ;36 OCTAL BYTE MSG

AUTHR:  ?WRITE  DS      ;DATA SENSITIVE OPTION
        JMP     ERTN    ;ERROR ON WRITE ..TO CLI
        DSZ     TWEN    ;LOOP 20 TIMES
        JMP     AUTHR   ;WRITE AGAIN, UNTIL 0

MESSG:  LDA     1,MPTR
        LDA     2,55    ;MSG IS 55 OCTAL BYTES
        ?WRITE  DS      ;WRITE 'MP/OS ..' MESSAGE
        JMP     ERTN    ;ERROR ON WRITE
CLEAN:  LDA     0,CHAN  ;PREP FOR CLOSE
        ?CLOSE          ;CLOSE CHANNEL TO @TTO
        JMP     ERTN    ;ERROR ON ?CLOSE
        SUB     0,0     ; CLEAR ACS FOR CLEAN RETURN
        SUB     1,1     ;... AC1 HAS ERROR LENGTH
        SUB     2,2     ;... AC2 HAS POINTER TO ERROR

ERTN:   ?RETURN         ;TO CLI, ERROR RETURNS KEEP
                        ;   CODES IN ACS

        ;DATA AREAS

CHAN:   0               ;CHANNEL # SAVE AREA
PATH:   .+1*2
        .TXT    /@TTO/  ;OUTPUT TO CONSOLE
STPTR:  .+1*2
        .TXT    /                       *<12>/
TWEN:   24              ;24 OCTAL = 20 DEC
MPTR:   .+1*2
        .TXT    /                   THE MP/OS HAS LANDED<12>/
        ;THE MP/OS STACK

        .NREL   0       ;IMPURE CODE
STK:    .BLK    50      ;RESERVE 50 WORDS
        .LOC    40      ;LOC 40 HAS ...
        STK             ;... THE STACK POINTER
        STK             ;LOCATION 41
        STK+45          ;LOC 42 HAS STACK LIMIT

        .END    START   ;END OF ASM INPUT

)
```

1. Assemble PROG_ONE without producing the object file. Write in the commands and the anticipated responses in the following spaces:

_____

_____

_____

_____

_____

****

```
) XEQ MASM/N PROG_ONE )
                                TITL   PROG_ONE;TITLE IDENTIFIER
```

The /N switch suppresses the creation of an object file.

Now try the command on your system. Give the system some time to respond, especially if you are working in a diskette environment.

If any errors are displayed on your console, it will be necessary to identify the cause of the error, reinvoke speed, and edit the source file so that it is error-free. Refer to the appendix of this Guide for a listing and explanation of the macroassembler error codes. Once the copy is error-free, back it up again, and go to the next step in this exercise.

2. List the files associated with PROG_ONE that now exist on your system. (This assumes that you have successfully completed question 1). Show the commands and anticipated response:

_____

_____

_____

_____

_____

_____

****

```
) FI/AS/S PROG+ )
DIRECTORY @DPX1:

  PROG_ONE          TXT    18-JUN-79  11:48:02        1293
  PROG_ONE.BU       TXT    18-JUN-79  11:49:51        1293
)
```

Do it on your system. The "/N" switch suppresses the object file creation and, therefore, no PROG_ONE.OB is produced.

Note that this example was produced in directory @DPX1., on diskette drive number 2 in a dual diskette system. Your directory may be different, as was explained in previous modules.

3. Assemble PROG_ONE and create an object file despite any errors. Show the anticipated results:

_____

_____

_____

_____

_____

****

```
) XEQ MASM/R PROG_ONE↓-
                        .TITL   PROG_ONE;TITLE IDENTIFIER
)
```

Now try the command on your system. The "/R" switch instructs the macroassembler to produce the object file despite the presence of any errors. No listing is produced.

4. Verify the existence of the object file created in question 3. Show the commands and anticipated results:

_____

_____

_____

_____

_____

****

```
) FI/AS/S PROG+↓
DIRECTORY @DPX1:

  PROG_ONE              TXT    18-JUN-79  11:48:02        1293
  PROG_ONE.BU           TXT    18-JUN-79  11:49:51        1293
  PROG_ONE.OB           OBF    18-JUN-79  12:29:37         312
)
```

Now do it on your system. Note that the object file was named "PROG_ONE.OB" as the default.

Delete this version of PROG_ONE.OB so that it does not confuse further exercises. Use the CLI command shown below:

```
) DELETE/V/C PROG_ONE.OB↓
PROG_ONE.OB? YES↓
Deleted PROG_ONE.OB
)
```

5. Time to get a listing. Assemble PROG_ONE and send the program listing to the line printer. Show the commands and anticipated results:

_____

_____

_____

_____

_____

****

```
) XEQ MASM/L PROG_ONE)
                              .TITL PROG_ONE;TITLE IDENTIFIER
)
```

The /L switch requests a program listing.

Do it on your system. Give the macroassembler a little time. By this time all errors should be cleaned up and, therefore, you should not receive any error messages.

Review the listing. Check the first three columns of each line for errors. Also check the cross-reference listing for strange user symbols.

6. List on your console the files associated with PROG_ONE that now exist on your system. Show the command and anticipated results.

> REMEMBER: TO GET MAXIMUM BENEFIT FROM THE EXERCISES, WRITE IN EACH ANSWER AS INSTRUCTED.

_____

_____

_____

_____

_____

****

```
) FI/SORT PROG+ )
DIRECTORY @DPX1:

  PROG_ONE
  PROG_ONE.BU
  PROG_ONE.OB
)
```

This version of PROG_ONE.OB was created by the MASM command which included the /L switch.

Do it on your system.

Delete PROG_ONE.OB again. Don't forget to delete only the .OB version.

7. The listing requires some time to print out. This time, assemble PROG_ONE and send the program listing to a file called "ONE.LIST" on your disc device. Show all commands and anticipated results:

_____

_____

_____

_____

_____

_____

_____

****

```
) XEQ MASM/L=ONE.LIST  PROG_ONE )

)
```

Now do it on your system.

The /L=ONE.LIST" keyword switch directs the listing, cross-reference table, and error messages to a disc file named ONE.LIST.

8. Verify the creation of ONE.LIST. Print it out and then delete it from the system. Show all commands and briefly describe the anticipated results:

_____

_____

_____

_____

_____

_____

****

THE ACTUAL LISTING IS ON THE NEXT TWO PAGES.

```
) FI/AS ONE.LIST
DIRECTORY @DPX1:

  ONE.LIST            TXT    18-JUN-79  12:35:21         3807
) TYPE ONE.LIST
 0001 PROG_ MP/OS ASSEMBLER REV 1.00          06:18:79 12/33/42
                          .TITL   PROG_ONE;TITLE IDENTIFIER
02                        .ENT    START   ;ENTRY POINT
03        000001          .NREL   1       ;PURE CODE
04
05                        ;MAIN ROUTINE
06
07 00000!020434 START:    LDA     0,PATH  ;BYTEPOINTER TO PATHNAME
08                        ?OPEN             ;OPEN CHANNEL TO @TTO
09 00003!000426           JMP     ERTN    ;ERROR ON OPEN...CLI
10 00004!040427           STA     0,CHAN  ;SAVE CHANNEL #
11 00005!020426 LOOP:     LDA     0,CHAN  ;IN CASE IT WANDERED
12 00006!024432           LDA     1,STPTR ;BYTE POINTER TO STAR LINE
13 00007!030036           LDA     2,36    ;36 OCTAL BYTE MSG
14
15                AUTHR:  ?WRITE  DS      ;DATA SENSITIVE OPTION
16 00012!000417           JMP     ERTN    ;ERROR ON WRITE ..TO CLI
17 00013!014442           DSZ     TWEN    ;LOOP 20 TIMES
18 00014!000774           JMP     AUTHR   ;WRITE AGAIN, UNTIL 0
19
20 00015!024441 MESSG:    LDA     1,MPTR
21 00016!030055           LDA     2,55    ;MSG IS 55 OCTAL BYTES
22                        ?WRITE  DS      ;WRITE 'MICRON...' MESSAGE
23 00021!000410           JMP     ERTN    ;ERROR ON WRITE
24 00022!020411 CLEAN:    LDA     0,CHAN  ;PREP FOR CLOSE
25                        ?CLOSE            ;CLOSE CHANNEL TO @TTO
26 00025!000404           JMP     ERTN    ;ERROR ON ?CLOSE
27 00026!102400           SUB     0,0     ; CLEAR ACS FOR CLEAN RETURN
28 00027!126400           SUB     1,1     ;... AC1 HAS ERROR LENGTH
29 00030!152400           SUB     2,2     ;... AC2 HAS POINTER TO ERROR
30
31                ERTN:   ?RETURN           ;TO CLI, ERROR RETURNS KEEP
32                                        ;   CODES IN ACS
33
34                        ;DATA AREAS
35
36 00033!000000 CHAN:     0               ;CHANNEL # SAVE AREA
37 00034!000072&PATH:     .+1*2
38 00035!040124           .TXT    /@TTO/  ;OUTPUT TO CONSOLE
39        052117
40        000000
41 00040!000102&STPTR:    .+1*2
42 00041!020040           .TXT    /                    *<12>/
43        020040
44        020040
45        020040
46        020040
47        020040
48        020040
49        020040
50        020040
51        020040
52        020052
53        005000
54 00055!000024 TWEN:     24              ;24 OCTAL = 20 DEC
55 00056!000136&MPTR:     .+1*2
56 00057!020040           .TXT    /              THE MP/OS HAS LANDED<12>/
57        020040
58        020040
59        020040
60        020040
```

```
 0002 PROG_
01        020040
02        020040
03        020040
04        052110
05        042440
06        046511
07        041522
08        047516
09        020110
10        040523
11        020114
12        040516
13        042105
14        042012
15        000000
16                          ;THE MP/OS STACK
17
18        000000            .NREL   0         ;IMPURE CODE
19 00000'000050 STK:        .BLK    50        ;RESERVE 50 WORDS
20        000040            .LOC    40        ;LOC 40 HAS ...
21 00040 000000'            STK               ;... THE STACK POINTER
22 00041 000000'            STK               ;LOCATION 41
23 00042 000045'            STK+45            ;LOC 42 HAS STACK LIMIT
24
25                          .END    START     ;END OF ASM INPUT
26

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS  0003 PROG_

AUTHR  000010!        1/15#   1/18
CHAN   000033!        1/10    1/11    1/24    1/36#
CLEAN  000022!        1/24#
ERTN   000031!        1/09    1/16    1/23    1/26    1/31#
LOOP   000005!        1/11#
MESSG  000015!        1/20#
MPTR   000056!        1/20    1/55#
PATH   000034!        1/07    1/37#
START  000000! EN     1/02    1/07#   2/25
STK    000000'        2/19#   2/21    2/22    2/23
STPTR  000040!        1/12    1/41#
TWEN   000055!        1/17    1/54#
?CLOS  002203! MC     1/25
?I     000013         1/09#   1/16#   1/23#   1/26#   1/32#
?J     000000         1/09#   1/16#   1/23#   1/26#   1/32#
?K     000002         1/16#   1/23#
?OPEN  001743! MC     1/08
?RETU  000203! MC     1/31
?SYSE  000001$ XD     1/09    1/16    1/23    1/26    1/32
?WRIT  002143! MC     1/15    1/22

) DELETE/V/C ONE.LIST↓
ONE.LIST? YES↓
Deleted ONE.LIST
) DELETE/V/C PROG_ONE.OB↓
PROG_ONE.OB? Y↓
Deleted PROG_ONE.OB
)
```

If your command sequence matches the answer then perform it on your system. Compare the ONE.LIST printout with the printout obtained from the previous question. Are they the same? (They should be, except for the name and the times.)

Save PROG_ONE.OB for a later question.

9. Assemble PROG_ONE again, but this time make a file named "OBJECT" the object file. Show the commands and anticipated results:

_____

_____

_____

_____

_____

_____

_____

****

```
) XEQ MASM/B=OBJECT PROG_ONE)
                          .TITL   PROG_ONE:TITLE IDENTIFIER
)
```

The keyword switch /B=OBJECT names the object file OBJECT.OB.

Now do it on your system. To avoid possible confusion, make sure "OBJECT" is a unique filename No listing is produced with this command.

10. Verify the existence of the new object file. Compare its size with PROG_ONE.OB. Show all commands and anticipated results:

****

```
) FI/LEN PROG_ONE.OB  OBJECT.OB)
DIRECTORY @DPX1:

  PROG_ONE.OB              312
  OBJECT.OB                312
)
```

The /LEN switch on the FILESTATUS command displays the byte length of the files.

Now do it on your system. Both OBJECT .OB and PROG_ONE.OB should be the same size. The only differences should be the filenames and time of last modification. Save OBJECT.OB and delete PROG_ONE.OB.

11. Assemble PROG_ONE and include your user symbols with the object file. Show all commands and anticipated results:

****

```
) XEQ MASM/U PROG_ONE)
                        .TITL   PROG_ONE;TITLE IDENTIFIER
)
```

The "/U" switch adds your user-defined symbols (labels) to the object file. This command switch is useful as part of the debugging sequence.

Do it.

12. Compare the size of the PROG_ONE.OB created in question 11 to the OBJECT.OB. Show all commands and responses:

_____

_____

_____

_____

****

```
) FI/LEN PROG_ONE.OB  OBJECT.OB)
DIRECTORY @DPX1:

  PROG_ONE.OB              502
  OBJECT.OB                312
)
```

Do it on your system.

PROG_ONE.OB should be larger than OBJECT.OB this time around since the user symbols were added in.

This concludes the macroassembler procedures lab exercises. You may continue to experiment with the macroassembler switches and results if you so desire. A suggested test is to get a cross-reference listing after assembling with the "/U" and "/P" switches. The listing should include the user symbols and semi-permanent symbols (instruction mnemonics).

If you are interested only in assembly language program development, then you should skip to the Binder segment.

If you are interested in Pascal program development, skip to the Pascal Compilation segment.

If you are interested in Fortran program development, go to the Fortran Compilation segment.

Now continue with
your next segment
in the Student Guide.

FORTRAN COMPILATION

Abstract

This segment discusses the concepts and procedures involved in compiling FORTRAN4 source files.

M.P. Fortran four programs require separate compilation and assembly steps. In this segment we cover the procedures involved in compiling Fortran Four programs under M.P.O.S. In the next segment, we cover the procedures for assembling the Fortran compiler's output. Both segments are required for the Fortran program developer. In addition, the topics covered in the proceding segments on Macroassembler Concepts and Macroassembler Procedures are highly beneficial for the Fortran programmer.

NOTE: Pascal Programmers may skip this segment and move on the Pascal compilation.

Objectives

Upon completion of this segment you will be able to:

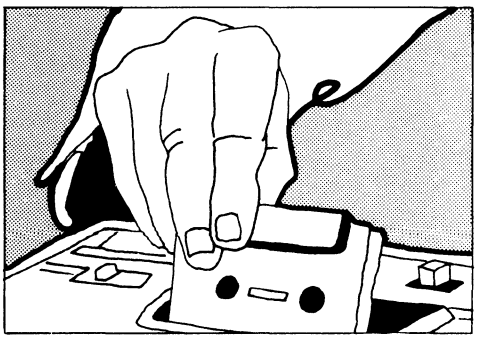
1. Given a CLI FORTRAN4 compiler command line, identify and state the purpose of the switches and arguments.

2. List the reference material available on FORTRAN4 compiler commands (text and system).

3. Given a CLI FORTRAN4 compiler command line, describe the results.

4. Write an appropriate CLI FORTRAN4 compiler command to compile a given Fortran program State the output names with extensions.

5. Given a FORTRAN4 compiler error situation:
   A) Identify possible causes;
   B) Reference solutions
   C) Write commands to correct the error.



Directions

1. Turn to Figure 4-50 on the next page of the Student Guide.

2. Listen to the tape for this segment.

XEQ	FORT4/SWITCHES	ARGUMENT
XEQ	/L	filename or
XE	/L=filename	fully qualified
X	/S=source file name	pathname
	/F	(.FR)
	/X	
	/E=errorfile	
	/P	

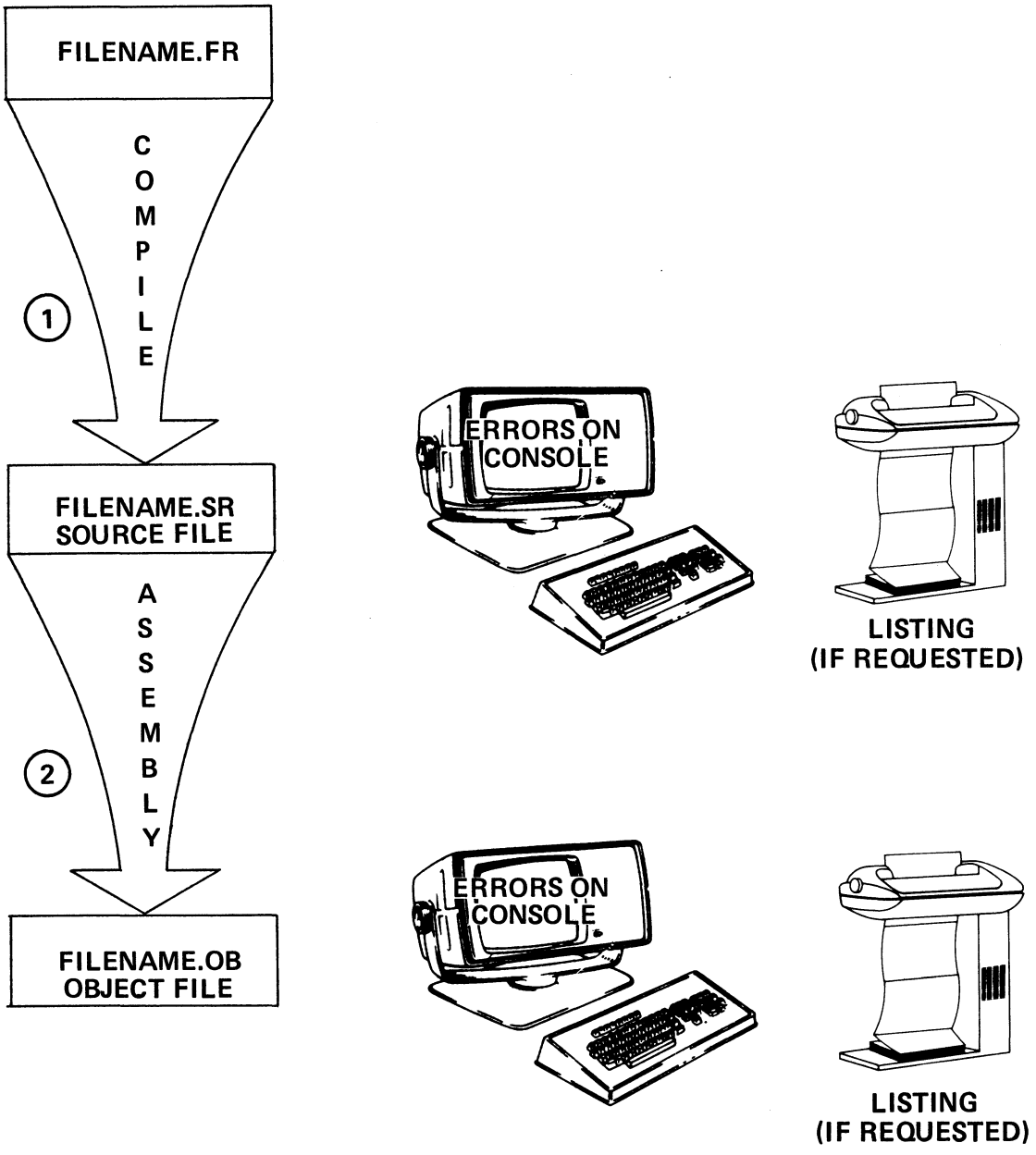
CLI FORTRAN COMPILER COMMAND LINE SYNTAX.

Figure 4-50

SWITCH	FUNCTION
/S = filename	Specifies the name of the assembler source file produced by the compiler.
/L	Produce a listing and send it to the lineprinter.
/L = filename	Produce a listing and send it to the named file on Disc.
/E = filename	Write error messages to the specified file. Default is to send error messages to the console.
/P	Compiler reads only the first 72 characters of each line of the input medium. Default is to read 135 characters, plus a new-line.
/X	Specifies that all lines with an X in column 1 should be compiled. Default is to skip "Xed" lines during compilation.
/F	Used in conjunction with the "/U" switch in assembly and binding. When assembly code is produced, all of the Fortran variables are internally equivalenced to identifiers recognizable by the Macro-assembler. The /F switch and /U switch cause all of these equivalences to be placed in assembler code. Aids debugging.

FORTRAN COMPILATION OPTIONAL SWITCHES

Figure 4-51



TWO STEPS IN FORTRAN OBJECT FILE CREATION

Figure 4-52

RESULT

```
) XEQ FORT4 MYFILE ↓
```

1

MYFILE
MYFILE.SR
} on Disc

Error Messages on Console

```
) XEQ FORT4 MYFILE.FR ↓
```

2

MYFILE.FR
MYFILE.SR
} on Disc

Error Messages on Console

```
) XEQ FORT4/X MYFILE ↓
```

3

MYFILE
MYFILE.SR
} on Disc

Error Messages on Console

/X . . . compile conditional compilation lines.

RESULT

```
) XEQ FORT4/L MYFILE↵
```

1

MYFILE
MYFILE.SR } on Disc

Errors Messages on Console
Listing on LPT

/L . . . produce a listing on the LPT.

```
) XEQ FORT4/L=MYFILE.LS  MYFILE↵
```

2

MYFILE
MYFILE.SR } on Disc

Errors Messages on Console
Listing to MYFILE.LS on Disc

/L = filename . . . store listing on disc.

```
) XEQ FORT4/E=ERRORFILE MYFILE↵
```

3

MYFILE
MYFILE.SR } on Disc

Errors to Errorfile on Disc
(No Errors creates an empty file).

/E = filename . . . create error file on disc.

Figure 4-54

RESULT

```
: XEQ FORT4/S=TEST MYFILE.FR↵
```

1

MYFILE.FR
TEST.SR
} on Disc

Error Messages on Console

/S = filename . . . name output "filename".

```
: XEQ FORT4/P MYFILE↵
```

2

MYFILE.SR
MYFILE.
} on Disc

Error Messages on Console

/P . . . read 1st 72 characters of input lines.

```
: XEQ FORT4/F MYFILE↵
```

3

MYFILE
MYFILE.SR
} on Disc

Error Messages on Console

/F . . . equivalence variables.

Figure 4-55

```
) TYPE MONEY.LS
; DGC FORTRAN IV REV 05.20IS

;       C       *** TITLE INCOME ***
;               ACCEPT "ENTER YOUR WEEKLY SALARY", SALARY
;               ACCEPT"ENTER YOUR AGE", AGE
;               YERLY = SALARY * 52
;               TOTAL = YERLY * (65 - AGE)
;               TYPE "TOTAL INCOME FROM" AGE, "TO RETIREMENT AT 65 IS $ ",TOTAL
;               TYPE "TOTAL INCOME FROM" AGE, "TO RETIREMENT AT 65 IS $ ",TOTAL
; *** 060 *** CHR 30
;               STOP
;               END

)
```

NOTE
ERROR
MESSAGE

ERRORS

- REPEATED ON NEXT LINE
- APPENDIX LIST
- ERROR CODES
- SNOWBALLING EFFECT
- ERROR MESSAGES

MONEY.LS OUTPUT LISTING FROM FORT4 COMPILATION.

Figure 4-56

```
) TYPE MONEY.SR )

; DGC FORTRAN IV REV 05.20IS


; C      *** TITLE INCOME ***

;       ACCEPT "ENTER YOUR WEEKLY SALARY", SALARY
        .TITL   .MAIN
        .ENT    .MAIN
        .NREL   1
        .TXTM   1
        .EXTU
        .EXTN   .I
        .CSIZ   2
        FS.
.MAIN:
        JMP     @.+1
        L1.
L1.:

        JSR     @.FREA
        .C1
        0
        6
        .TXT    "ENTER YOUR WEEKLY SALARY"
        0
        2
        V.+0            ;SALARY
        5

;       ACCEPT"ENTER YOUR AGE", AGE
        JSR     @.FREA
        .C1
        0
        6
        .TXT    "ENTER YOUR AGE"
        0
        2
        V.+2            ;AGE
        5

;       YERLY = SALARY * 52
        FXFL1
        .C2
        FFLD1
        V.+0            ;SALARY
        FML1
        FFST1
        V.+4            ;YERLY
```

(Continued on Next Page)

Figure 4-57

```
        TOTAL = YERLY * (65 - AGE)
        FXFL1
        .C3
        FFLD1
        U.+2            ;AGE
        FSB1
        FFLD1
        U.+4            ;YERLY
        FML1
        FFSI1
        U.+6            ;TOTAL

;       TYPE "TOTAL INCOME FROM" AGE, "TO RETIREMENT AT 65 IS $ ",TOTAL
        JSR     @.FWRI
        .C4
        0
        6
        .TXT    "TOTAL INCOME FROM"

;       TYPE "TOTAL INCOME FROM" AGE, "TO RETIREMENT AT 65 IS $ ",TOTAL
; *** 060 *** CHR 30
        JMP     @.+1
        L2.
L3.:    .TXT    "TO RETIREMENT AT 65 IS $ "
L2.:

;       STOP
        JSR     @.STOP
        .TXT    ""

;       END
        JSR     @.FRET
.C4:    000012
.C3:    000101
.C2:    000064
.C1:    000013

        FS.=10
        SFS.=0
        T.=-16Z
        U.=200+T.
        TS.=T.+7
        FTS.=T.+0
        US.=U.+7
        FUS.=U.+0
        .END
```

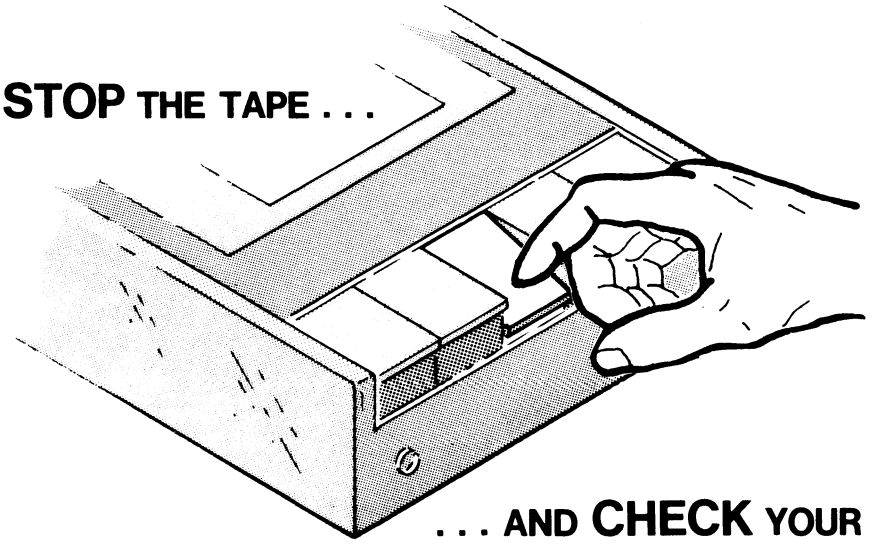
MASM OUTPUT LISTING OF ASSEMBLED .SR FILE.

Figure 4-57

TOPICS

- FORTRAN COMPILATION
- COMMAND LINE
- SWITCHES
- OUTPUT
- ERRORS

NOW STOP THE TAPE . . .



. . . AND CHECK YOUR PROGRESS

MP/FORTRAN4 COMPILATION

QUIZ

Write the answers in the space provided.

Given the following FORTRAN4 CLI Command line, identify the switches and arguments and state the purpose of the switches and arguments:

) XEQ FORT4/L=SUM.LS/E=SUM.ER/X SUM.FR

1. Switches: _____

2. Arguments: _____

3. Purpose of the switches _____

_____

_____

_____

4. Purpose of the argument. _____

List three sources of information on MP/FORTRAN4 compiler commands.

5. _____

6. _____

7. _____

Given the following CLI FORTRAN4 compiler command lines, briefly describe the anticipated result:

) XEQ FORT4 SUM

8. _____

_____

) XEQ FORT4/L=SUM.LS SUM.FR

9. _____

_____

) XEQ FORT4/S=NEW SUM

10. _____

_____

) XEQ FORT4/L/X SUM

11. _____

_____

) XEQ FORT4/F SUM

12. _____

_____

) XEQ FORT4/P SUM

13. _____

_____

NOW CHECK YOUR ANSWERS
ON THE FOLLOWING PAGES

MP/FORTRAN4 COMPILATION

QUIZ ANSWERS

The switches, arguments and purposes are as follows:

) XEQ FORT4/L=SUM.LS/E=SUM.ER/X SUM.FR

1. Switches: /L=SUM.LS /E=SUM.ER /X

2. Argument: SUM.FR

3. Purpose of switches: "/L=SUM.LS" produces a listing and stores it in a file called SUM.LS; the TYPE command will display this listing file.
   "/E=SUM.ER" directs error messages to the file called SUM.ER. Use TYPE here also. If there are no errors, SUM.ER is still created, but it is an empty file with a length of 0.
   "/X" will allow conditional compile lines (X in column 1) to be compiled.

4. Purpose of the argument: SUM.FR identifies the file to be compiled. Compiler output will be titled SUM.SR. SUM.SR is then input to the macroassembler.

Three sources of information on FORTRAN4 commands include the following:

5. CLI HELP command on systems so equipped;

6. MP/OS Utilities Reference Manual (093-400002)

7. This self-study course.
   Also: MP/FORTRAN IV Programmer's Reference (D.G. No. 093-400004).

The anticipated results of the CLI command lines are as follows:

) XEQ FORT4 SUM

8. SUM.FR (or SUM, if SUM.FR does not exist) is compiled. SUM.SR is the compiler's assembly language output and is stored on disc. No listing is produced. Errors are displayed on the system console.

) XEQ FORT4/L=SUM.LS SUM.FR

9. SUM.FR is compiled, producing SUM.SR as its assembly language translation. The listing is produced and stored in SUM.LS on the disc device. Errors are displayed on the console.

) XEQ FORT4/S=NEW SUM

10. SUM.FR is compiled (or SUM, if SUM.FR does not exist). The compiler's assembly language translation is stored in the file named NEW.SR, as directed by the /S switch. No listing is produced. Errors are displayed on the system console.

) XEQ FORT4/L/X SUM

11. SUM.FR is compiled (or SUM, if SUM.FR does not exist). SUM.SR is produced as the compiler's assembly language translation and is stored on disc. A listing is produced on the line printer. Errors are displayed on the console. The compilation includes all conditional compilation lines (marked with an X in column one) as directed by the /X switch.

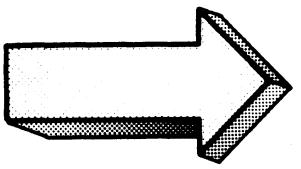
) XEQ FORT4/F SUM

12. Again, SUM.FR is compiled (or SUM, if SUM.FR does not exist). SUM.SR is produced as the assembly language translation and includes the equivalenced variables. SUM.SR is stored on disc. Errors are displayed on the system console. No listing is produced.

) XEQ FORT4/P SUM

13. SUM.FR (or SUM) is compiled. Only the first 72 characters of each FORTRAN source statement are read by the compiler. SUM.SR is produced as the compiler's assembly language translation and is stored on disc. Errors are displayed on the system console. No listing is produced.

A SCORE OF 11 CORRECT ANSWERS OUT OF THE 13 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.



MP/FORTRAN COMPILER

LAB EXERCISE

Directions

1. To perform the activities in this lab exercise, it is necessary that your system have the following file:

   FORT4.PR. . . . THE FORTRAN4 Compiler

   Verify the presence of this file with the FILESTATUS command. If it is on your system, but not in your directory, then modify your searchlist to make it accessible to you.

2. SPEED in the file titled MONEY.FR listed below:

```
C       *** TITLE INCOME ***
        ACCEPT "ENTER YOUR WEEKLY SALARY", SALARY
        ACCEPT"ENTER YOUR AGE", AGE
        YERLY = SALARY * 52
        TOTAL = YERLY * (65 - AGE)
        TYPE "TOTAL INCOME FROM" ,AGE, "TO RETIREMENT AT 65 IS $ ",TOTAL
        STOP
        END
```

3. Once you are certain that MONEY.FR is a clean copy, back it up with the CLI COPY command. Name the backup file INCOME.BU.

You are now ready for the lab exercise. Remember:

1. Cover the answer,
2. Read the question,
3. Write the solution,
4. Check the answer,
5. Perform the operation on your system (if you have access),
6. Resolve any discrepancies.

1. Compile MONEY.FR. Write in the required commands in full. Show all anticipated results in the space provided:

****

```
) XEQ FORT4 MONEY.FR)
)
```

Note that, in this case, it is not necessary to specify the .FR extension of MONEY.FR.

Now do it on your system.

Compare the actual results with yours and with our example.

If there are any errors, it is necessary to reinvoke Speed, make the appropriate corrections, and repeat this compilation. Keep doing it until you get it right.

2. Write the commands to show which files associated with MONEY currently exist on the system. Show the anticipated results:

_____

_____

_____

_____

_____

_____

****

```
) FI/AS/S  MONEY+
DIRECTORY @DPX1:

  MONEY.FR          TXT   26-JUN-79   9:54:45        222
  MONEY.SR           66   26-JUN-79  11:27:47        715
)
```

The FILESTATUS command displays the required information. The .FR version is the speed output. The .SR version was produced by the compiler. Note the use of the "+" template on the MONEY filename.

Do it on your system. Failure to get MONEY.SR indicates that the compilation failed. Check your MONEY.FR against our copy and make sure that is a clean copy.

3. Write the command for displaying the contents of MONEY.SR. *Briefly* describe the anticipated results:

_____

_____

_____

_____

_____

_____

****

```
) TYPE MONEY.SR )

; DGC FORTRAN IV REV 05.20IS


; C       *** TITLE INCOME ***

;         ACCEPT "ENTER YOUR WEEKLY SALARY", SALARY
          .TITL    .MAIN
          .ENT     .MAIN
          .NREL    1
          .TXTM    1
          .EXTU
          .EXTN    .I
          .CSIZ    2
          FS.
.MAIN:
          JMP      @.+1
          L1.
L1.:

          JSR      @.FREA
          .C1
          0
          6
          .TXT     "ENTER YOUR WEEKLY SALARY"
          0
          2
          V.+0              ;SALARY
          5

;         ACCEPT"ENTER YOUR AGE", AGE
          JSR      @.FREA
          .C1
          0
          6
          .TXT     "ENTER YOUR AGE"
          0
          2
          V.+2              ;AGE
          5

;         YERLY = SALARY * 52
          FXFL1
          .C2
          FFLD1
          V.+0              ;SALARY
          FML1
          FFST1
          V.+4              ;YERLY
```

(Continued on Next Page)

```
;       TOTAL = YERLY * (65 - AGE)
        FXFL1
        .C3
        FFLD1
        U.+2            ;AGE
        FSB1
        FFLD1
        U.+4            ;YERLY
        FML1
        FFSI1
        U.+6            ;TOTAL

;       TYPE "TOTAL INCOME FROM" AGE, "TO RETIREMENT AT 65 IS $ ",TOTAL
        JSR     @.FWRI
        .C4
        0
        6
        .TXT    "TOTAL INCOME FROM"

;       TYPE "TOTAL INCOME FROM" AGE, "TO RETIREMENT AT 65 IS $ ",TOTAL
; *** 060 *** CHR 30
        JMP     @.+1
        L2.
L3.:    .TXT    "TO RETIREMENT AT 65 IS $ "
L2.:

;       STOP
        JSR     @.STOP
        .TXT    ""

;       END
        JSR     @.FRET
.C4:    000012
.C3:    000101
.C2:    000064
.C1:    000013

        FS.=10
        SFS.=0
        T.=-167
        U.=200+T.
        TS.=T.+7
        FTS.=T.+0
        US.=U.+7
        FUS.=U.+0
        .END
```

Do it on your system.

If you have a hard-copy Dasher terminal as the system console, use the type command without the /L switch. If you have a line printer, then use the type command with the /L switch.

Save the printout for later reference.

DELETE MONEY.SR to avoid confusion with later exercises. Use the /V/C combination.

4. Compile money and produce a listing in the file titled "MONEY.LS". Show the commands and anticipated results in the space below:

_____

_____

_____

_____

_____

_____

****

```
)
) XEQ FORT4/L=MONEY.LS MONEY)
)
```

/L = filename makes MONEY.LS the listing file on disc.

Note that our example assumes that you have cleaned up any errors.

Now try it on your system.

The listing file is now on your disc.

5. Determine which files associated with money now exist on your system. Show the commands and anticipated responses. Note the length of the files.

_____

_____

_____

_____

_____

_____

****

```
) FI/AS/S MONEY+ )
DIRECTORY @DPX1:

  MONEY.FR          TXT    26-JUN-79    9:54:45        222
  MONEY.LS           66    26-JUN-79   11:31:54        268
  MONEY.SR           66    26-JUN-79   11:31:54        992
)
```

MONEY.FR is the title of your speed output. MONEY.SR is the compiler's assembly language output. MONEY.LS is the listing file specified by the /L switch.

Do it on your system.

6. Write the command for displaying the contents of MONEY.LS. *Briefly describe* the anticipated results:

_____

_____

_____

_____

_____

_____

****

```
) TYPE MONEY.LS )
; DGC FORTRAN IV REV 05.20IS

;C      *** TITLE INCOME ***
;       ACCEPT "ENTER YOUR WEEKLY SALARY", SALARY
;       ACCEPT"ENTER YOUR AGE", AGE
;       YERLY = SALARY * 52
;       TOTAL = YERLY * (65 - AGE)
;       TYPE "TOTAL INCOME FROM", AGE, "TO RETIREMENT AT 65 IS $ ",TOTAL
;       STOP
;       END

)
```

Do it on your system.

MONEY.LS is a listing of the Fortran source language statements.

Again, we used a hard-copy Dasher console and, therefore, only required the type command. To direct the output to a line printer, use the TYPE/L command.

Delete MONEY.LS and MONEY.SR to avoid conflicts with future activities:

```
) DELETE/V/C MONEY.LS MONEY.SR)
MONEY.LS? YES)
Deleted MONEY.LS
MONEY.SR? YES)
Deleted MONEY.SR
)
```

7. Compile money again. This time name the compiler's assembly language output "DUMMY". Show all commands and anticipated results:

_____

_____

_____

_____

_____

_____

_____

****

```
) NEQ FORT4/S=DUMMY  MONEY)

)
```

The "/S DUMMY" switch directs the compiler to make DUMMY.SR the output file.

Do it on your system.

8. Write the commands to determine the output of the previous question (#7). Show the anticipated results. Make note of the length of the files.

_____

_____

_____

_____

****

```
) FI/AS MONEY+ DUMMY+ )
DIRECTORY @DPX1:

  MONEY.FR          TXT    26-JUN-79   9:54:45        222
  DUMMY.SR           66    26-JUN-79  11:35:08        992
)
```

MONEY.FR is the Fortran source file. DUMMY.SR is the compiler's assembly language output.

Do it on your system.

Compare the length of DUMMY.SR to the previous MONEY.SR. Similar? (They should be!!!)

Type the contents of DUMMY.SR. Compare it with the contents of MONEY.SR. Similar? (They should be.)

Now delete DUMMY.SR.

9. Compile money. Make MONEY.ER the error file. Show all commands and anticipated results:

_____

_____

_____

_____

****

```
) XEQ FORT4/E=MONEY.ER MONEY
)
```

MONEY.ER is assigned as the error file by the /E=MONEY.ER switch.

Do it on your system.

10. Write the commands to show all files associated with money that now exist on the system. Show their lengths. Write the anticipated results:

_____

_____

_____

_____

_____

_____

****

```
) FILES MONEY+)
DIRECTORY @DPX1:

  MONEY.FR          TXT    26-JUN-79   9:54:45     222
  MONEY.ER           66    26-JUN-79   11:38:08      0
  MONEY.SR           66    26-JUN-79   11:38:18    715
)
```

Do it on your system.

MONEY.ER is 0 length only if there are no errors associated with the compilation of MONEY.FR.

Display the contents of MONEY.ER with the TYPE command:

```
) TYPE MONEY.ER)
)
```

Anything show up? (It shouldn't; MONEY.ER should be an empty file).

Now delete MONEY.ER and MONEY.SR.

11. Compile money for the last time in this lab exercise. Make MONEY.LS the listing file. Make the Fortran variables equivalenced to identifiers recognizable by the macroassembler. Show all commands and anticipated results:

_____

_____

_____

****

```
) XEQ FORT4/F/L=MONEY.LS  MONEY.FR
)
```

The /L=MONEY.LS switch directs the listing to the MONEY.LS disc file.

The /F switch allows equivalencing.

Do it on your system.

12. Write the command to show all files associated with money that now exist on your system. Show the anticipated results. Include estimated file lengths.

_____

_____

_____

_____

****

```
) FI/AS/S MONEY+)
DIRECTORY @DPX1:

  MONEY.FR          TXT    26-JUN-79   9:54:45        222
  MONEY.LS           66    26-JUN-79  11:42:51        268
  MONEY.SR           66    26-JUN-79  11:42:53       1050
)
```

Do it on your system.

Compare the size of MONEY.SR with the previous MONEY.SR. Is it larger? (It should be: several lines were added with the /F switch).

13. Write the commands to display the contents of MONEY.SR. Briefly describe the anticipated results:

_____

_____

_____

_____

****

```
) TYPE MONEY.SR )

; DGC FORTRAN IV REV 05.20IS


; C      *** TITLE INCOME ***

;        ACCEPT "ENTER YOUR WEEKLY SALARY", SALARY
         .TITL   .MAIN
         .ENT    .MAIN
         .NREL   1
         .TXTM   1
         EXTU
         .EXTN   .I
         .CSIZ   2
         FS.
.MAIN:
         JMP     @.+1
         L1.
L1.:

         JSR     @.FREA
         .C1
         0
         6.
         .TXT    "ENTER YOUR WEEKLY SALARY"
         0
         2
         V.+0            ;SALARY
         5

         ACCEPT"ENTER YOUR AGE", AGE
         JSR     @.FREA
         .C1
         0
         6
         IXI     "ENTER YOUR AGE"
         0
         2
         V.+2            ;AGE
         5

         YERLY = SALARY * 52
         FXFL1
         .C2
```

(CONTINUED)

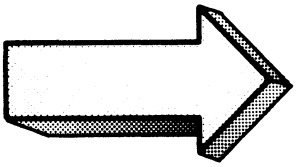
```
            FFLD1
            U.+0            ;SALARY
            FML1
            FFST1
            U.+4            ;YERLY

            TOTAL = YERLY * (65 - AGE)
            FXFL1
            .C3
            FFLD1
            U.+2            ;AGE
            FSB1
            FFLD1
            U.+4            ;YERLY
            FML1
            FFST1
            U.+6            ;TOTAL

            TYPE "TOTAL INCOME FROM",AGE,"TO RETIREMENT AT 65 IS $ ",TOTAL
            JSR     @.FWRI
            .C4
            0
            6
            .TXT    "TOTAL INCOME FROM"
            0
            2
            U.+2            ;AGE
            6
            .TXT    "TO RETIREMENT AT 65 IS $ "
            0
            2
            U.+6            ;TOTAL
            5

            STOP
            JSR     @.STOP
            .TXT    ""

            END
            JSR     @.FRET
.C4:        000012
.C3:        000101
.C2:        000064
.C1:        000013

            FS.=10
            SFS.=0
            T.=-167
            U.=200+T.
            TS.=T.+7
            FTS.=T.+0
            US.=U.+7
            FUS.=U.+0
TOTAL=      U.+6
YERLY=      U.+4
AGE=        U.+2
SALARY=     U.+0
 END

) T\T
```

Do it on your system.

MONEY.SR is the assembly language translation of the Fortran source statements. It consists of both the Fortran statements and the assembly language code that they generate. This version of MONEY.SR has the Fortran variables equivalenced and listed at the tail-end of the listing.

This completes the MP/FORTRAN COMPILER Lab Exercise. Save a copy of MONEY.FR for future references within this course.

Shut down your computer system and continue with the next segment in this Module.



ASSEMBLING FORTRAN .SR FILES

Abstract

This segment discusses the procedures available for translating MP/FORTRAN4 files from the compiled, assembly language state to their object file state.

Objectives

Upon completion of this segment, you will be able to:

1. Given a CLI Macroassembler command line applied in a Fortran context, identify and state the purpose of the switches and arguments.

2. Given a CLI Macroassembler command line applied in a Fortran context, describe the results.

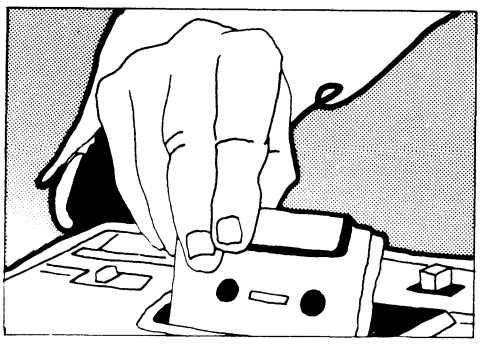
3. Write appropriate CLI Macroassembler command lines to assemble a given Fortran file under stated conditions.

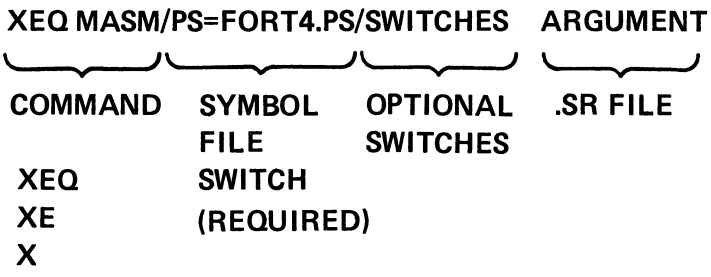
4. Given a Macroassembler error situation in a Fortran context:
   A) identify possible causes;
   B) reference solutions;
   C) write commands to correct the error.

Directions

1. Completion of the MACROASSEMBLER segments of this module are extremely worthwhile at this point.

2. Turn to figure 4-75 on the next page of the Student Guide and listen to the tape for this segment.





CLI FORTRAN/ASSEMBLY COMMAND LINE SYNTAX

Figure 4-75

SYMBOL	ACTION
/B = filename	Gives the name *filename* to the object file. Ordinarily the object file has the name of the first source file in the assembly command line, less the extension .SR (if any) and with a new extension .OB.
/E	Suppresses the error listing if there is a listing file. If there is no listing file, then errors will be displayed on the console despite the /E. See Appendix F for a description of error flags.
/F	Generates or suppresses form feeds as required to produce an even number of assembly pages. This keeps the first page of successive listings on the outsides of paper folds, making refolding unnecessary. By default, the macroassembler generates a form feed at the end of a listing, whether the number of pages is odd or even.
/K	Keeps the macroassembler's temporary symbol file at the end of assembly. Since virtually no programs require the use of this file, the macroassembler deletes it by default.
/L	Produces a listing file on the line printer. Listings always includes a cross reference of symbols in the program showing the page and line number where each symbol is used. If you use the /L switch, program MASMXR.PR must be present in the same directory as the macroassembler itself. MASM.PR itself is found, but, if it is missing, then an error message will be displayed at your console.
/L-*list-file*	Produces a listing file, but instead of sending it to the line printer, sends it to the file designated by *listfile.* If there is already a listing in this file, then the new listing allows it. *Listfile* can be any filename or pathname permitted by the operating system.
/M	Flags any redefinition of semipermanent symbols as multiple definition errors (M).
/N	Produces no object file.
/O	Overrides all listing control pseudo ops: .NOCON, .NOLOC, and .NOMAC. Also overrides listing suppression.

(Continued)

MACROASSEMBLER COMMAND LINE SWITCHES

Figure 4-76

/P	Adds semipermanent symbols to the cross-reference listing. By default they are not included.
/PS-*filename*	Uses *filename* instead of MASM.PS to build symbol table file.
/R	Produces a binary file even if there is an assembly error. By default, if there is an assembly error, the macroassembler does not produce a binary file.
/S	Skips the second assembly pass (produces no.OB file) and saves the macroassembler's symbol table, renaming it MASM.PS. (See below, "Macroassembler Symbol Table Files.")
/U	Includes user symbols in the object file. When the /U switch is also applied to the binder command line, then the debugger is able to find user symbols. This makes debugging easier.
/Z	Lists the DGC proprietory license heading at the top of each assembly and cross-reference page. By default this heading is not listed. Note that this switch is useful to DGC personnel only.

CLI MACROASSEMBLER COMMAND LINE SWITCHES

Figure 4-76

```
) FI/AS/S SUM+↓
DIRECTORY @DPX1:

  SUM.FR              TXT     1-JAN-00   0:09:57          157
  SUM.LS               66    27-JUL-79  10:49:54          203
  SUM.SR               66    27-JUL-79  10:49:55          983
)
```

THE SUM FILES AFTER COMPILATION

Figure 4-77

RESULT

1

```
) XEQ MASM/PS=FORT4.PS SUM.SR
                        .TITL    .MAIN

)
```

- SUM.OB on disc
- errors on console

2

```
) XEQ MASM/PS=FORT4.PS SUM↓
                        .TITL    .MAIN

)
```

- SUM.OB on disc
- errors on console

3

```
) XEQ MASM/PS=FORT4.PS/L=SUMMASM.LS SUM↓

)
```

- SUM.OB, SUMMASM.LS } on disc
- errors on console

/L = filename . . . direct listing to disc filename.

Figure 4-78

RESULT

1

```
) XEQ MASM/PS=FORT4.PS/L SUM↓
)
```

SUM.OB on disc

Errors on Console

/L . . . send listing to LPT

2

```
) XEQ MASM/PS=FORT4.PS/P/L=SUMMASM.LS SUM↓
)
```

SUMMASM.LS } on disc
SUM.OB }

Errors on Console

/P . . . include semi-permanent symbols in cross-reference listing.

3

```
) XEQ MASM/PS=FORT4.PS/K SUM↓
                          .TITL     .MAIN
)
```

SUM.OB } on disc
?MASM.ST.TMP }

Errors on Console

/K . . . save temporary symbol file.

Figure 4-79

RESULT

1

```
) XEQ MASM/PS=FORT4.PS/B=SUMOBJ SUM↓
                          .TITL     .MAIN
)
```

SUMOBJ.OB on disc

Errors on Console

/B = filename . . . make filename the object file.

2

```
) XEQ MASM/PS=FORT4.PS/U SUM
                          .TITL     .MAIN
)
```

SUM.OB on disc

Errors on Console

/U . . . add user-defined symbols to object file.

Figure 4-80

TOPICS

- ASSEMBLING FORTRAN .SR FILES
- CLI MASM COMMAND LINE
- REQUIRED SWITCH
- OPTIONAL SWITCHES
- OBJECT FILE OUTPUT

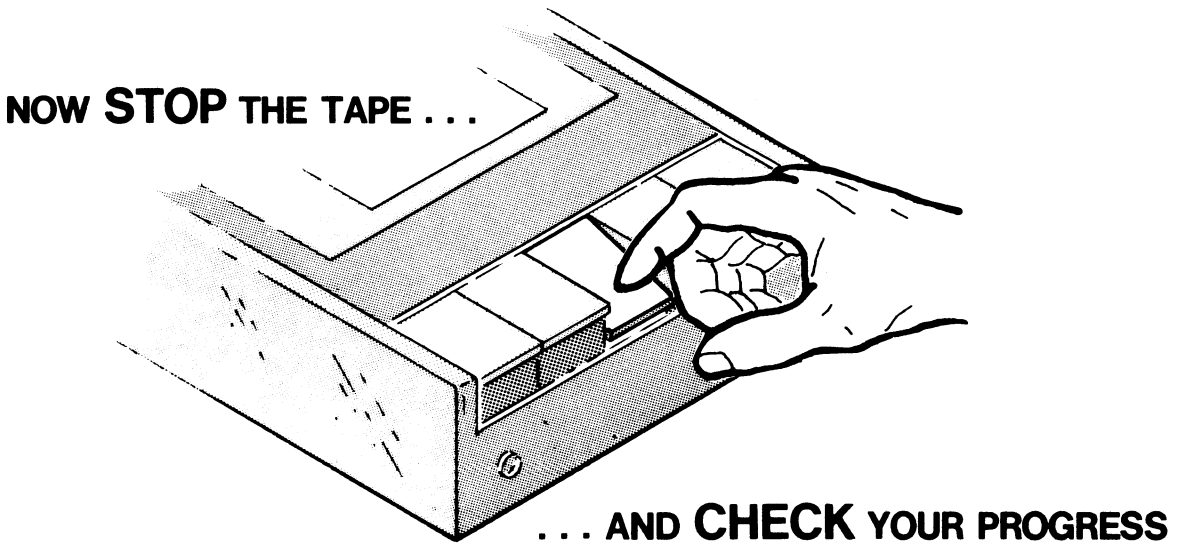


Figure 4-81

ASSEMBLING FORTRAN .SR FILES

QUIZ

Given the following macroassembler command line, identify the switches and arguments and state the purpose of the switches and arguments.

) XEQ MASM/PS=FORT4.PS/L=PROG.LS/K/B=PROG.OB PROG

1. Switches_____

2. Arguments_____

3. Purpose of the switches_____

4. Purpose of the argument: _____

Given the following CLI Macroassembler command lines, briefly describe the anticipated result of a successful assembly:

) XEQ MASM/PS=FORT4.PS PROG.SR

5._____

) XEQ MASM/PS=FORT4.PS PROG

6._____

) XEQ MASM/PS=FORT4.PS/L=PROUT.LS PROG

7. _____

_____

) XEQ MASM/PS=FORT4.PS/B=SPLAT PROG

8. _____

_____

) XEQ MASM/PS=FORT4.PS/P/L PROG

9. _____

_____

) XEQ MASM/PS=FORT4.PS/K PROG

10. _____

_____

) XEQ MASM/PS=FORT4.PS/U PROG

11. _____

_____

Check your answers on

the following page

ASSEMBLING FORTRAN.SR FILES

QUIZ ANSWERS

The switches, arguments, and purposes are as follows:

) XEQ MASM/PS=FORT4.PS/L=PROG.LS/K/B=PROG.OB PROG

1. Switches: /PS=FORT4.PS
   /L=PROG.LS
   /K
   /B=PROG.OB

2. Argument: PROG

3. Purpose of the switches: /PS=FORT4.PS signals the macroassembler to use FORT4.PS instead of MASM.PS to build a symbol table file. Failure to include this switch causes an assembly error and results in no object file. /L=PROG.LS directs the assembly listing to the disc file named PROG.LS. The listing contains the source statements, assembly translations and octal representations. (A detailed description of the assembly listing is in the macroassembler segment of this module.) /K retains the macroassembler temporary symbol table file at the end of the assembly process. This file is titled ?MASM.ST.TMP. /B=PROG.OB directs the assembler to name the assembled file PROG.OB.OB. The double .OB extension is forced by the specified name. Default name would be just PROG.OB.

4. The purpose of the argument: PROG identifies the file to be assembled. The macroassembler first searches for PROG.SR. If the search fails, the macroassembler then searches for PROG without the extension.

The anticipated results of the CLI command lines are as follows:

) XEQ MASM/PS=FORT4.PS PROG.SR

5. The macroassembler will search only for PROG.SR. If the search fails, the macroassembler will issue an error message, "FILE NOT FOUND". PROG.OB will be produced as the assembled file. No listing is produced. Errors are displayed on the console. The .TITL message is displayed on the console with a successful assembly.

) XEQ MASM/PS=FORT4.PS PROG

6. The macroassembler will search for PROG.SR and then PROG. PROG.OB is produced as the assembled file and stored on disc. Errors are displayed on the console. No listing is produced. The .TITL message is displayed on the console.

) XEQ MASM/PS=FORT4.PS/L=PROUT.LS PROG

7. PROG.SR (or PROG, if PROG.SR does not exist) is assembled, producing PROG.OB as the assembled file. The listing is stored under PROUT.LS on the disc. Errors are directed to the console. Note that the MASM listing is different from the compiler listing. MASM's listing includes the octal translations.

) XEQ MASM/PS=FORT4.PS/B=SPLAT PROG

8. PROG.SR (or PROG, if PROG.SR does not exist) is assembled. The assembled file is stored on disc under the name SPLAT.OB. Errors are directed to the console. No listing is produced.

) XEQ MASM/PS=FORT4.PS/P/L PROG

9. PROG.SR (or PROG) is assembled. The assembled file is PROG.OB. The listing is directed to the line printer and contains semi-permanent (instruction mnemonic) symbols. Errors are sent to the console.

) XEQ MASM/PS=FORT4.PS/K PROG

10. PROG.SR (or PROG, if PROG.SR does not exist) is assembled. The assembled file is titled PROG.OB and is stored on disc. Errors go to the console. No listing. The ?MASM.ST.TMP temporary symbol table file is saved on disc. This is a rather large file and is, therefore, not always preserved.

) XEQ MASM/PS=FORT4.PS/U PROG

11. PROG.SR (or PROG, if PROG.SR does not exist) is assembled. The assembled file is titled PROG.OB and is stored on disc. The object file, PROG.OB, includes the user symbols. Errors go to the console. No listing.

A SCORE OF 9 CORRECT ANSWERS OUT OF THE 11 QUESTIONS INDICATES MASTER LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.

ASSEMBLING FORTRAN .SR FILES

LAB EXERCISE

Directions

1. To complete this lab exercise, you must have the following files on your system:

   FORT4.PR . . . . . .The Fortran Compiler
   FORT4.PS. . . . . .Assembler symbols for Fortran
   MASM.PR. . . . . .The macroassembler
   MASMXR.PR . . . . . .The cross-reference for MASM
   MONEY.FR . . . . . .Fortran file from previous exercise.

   Use the filestatus command to verify their presence. If you do not have MONEY.FR then SPEED it in. Make sure it is an accurate copy. As usual, the comments are not necessary, but they make the program easier to read.

```
) TYPE MONEY.FR )
C       *** TITLE INCOME ***
        ACCEPT "ENTER YOUR WEEKLY SALARY", SALARY
        ACCEPT"ENTER YOUR AGE", AGE
        YERLY = SALARY * 52
        TOTAL = YERLY * (65 - AGE)
        TYPE "TOTAL INCOME FROM" ,AGE, "TO RETIREMENT AT 65 IS $ ",TOTAL
        STOP
        END

)
```

2. Delete the current MONEY.SR and MONEY.LS, if they exist. We want to start with a clean slate.

3. Compile money and make MONEY.LS the listing file. This was done in the Fortran compilation lab and is repeated here for your convenience:

```
) XEQ FORT4/L=MONEY.LS MONEY )

)
```

4. Check the success of the operation with the filestatus command:

```
) FI/AS/S MONEY+ }
DIRECTORY @DPX1:

  MONEY.FR              TXT    26-JUN-79    9:54:45          222
  MONEY.LS               66    27-JUL-79   10:40:14          268
  MONEY.SR               66    27-JUL-79   10:40:15          992
)
```

5. TYPE copies of MONEY.LS and MONEY.SR and save them. Note that our development was done on a dual diskette system. SYSDISK was in drive 0 (@DPX0) and FORTDISK was in Drive 1 (@DPX1). Also, our configuration included only a hard-copy Dasher terminal and no line printer.

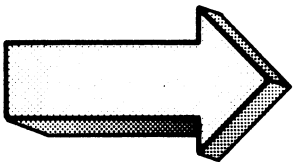
   Copies of MONEY.SR and MONEY.LS are shown below. Compare them with your printouts. Be sure that they are not significantly different. After checking them over, begin the lab exercise.

```
) TYPE MONEY.LS }
; DGC FORTRAN IV REV 05.20IS

; C       *** TITLE INCOME ***
;         ACCEPT "ENTER YOUR WEEKLY SALARY", SALARY
;         ACCEPT"ENTER YOUR AGE", AGE
;         YERLY = SALARY * 52
;         TOTAL = YERLY * (65 - AGE)
;         TYPE "TOTAL INCOME FROM" ,AGE, "TO RETIREMENT AT 65 IS $ ",TOTAL
;         STOP
;         END

)
```

```
) TYPE MONEY.SR )

; DGC FORTRAN IV REV 05.20IS


; C      *** TITLE INCOME ***

;       ACCEPT "ENTER YOUR WEEKLY SALARY", SALARY
        .TITL   .MAIN
        .ENT    .MAIN
        .NREL   1
        .TXTM   1
        .EXTU
        .EXTN   .I
        .CSIZ   2
        FS.
.MAIN:
        JMP     @.+1
        L1.
L1.:

        JSR     @.FREA
        .C1
        0
        6
        .TXT    "ENTER YOUR WEEKLY SALARY"
        0
        2
        V.+0            ;SALARY
        5

;       ACCEPT"ENTER YOUR AGE", AGE
        JSR     @.FREA
        .C1
        0
        6
        .TXT    "ENTER YOUR AGE"
        0
        2
        V.+2            ;AGE
        5

;       YERLY = SALARY * 52
        FXFL1
        .C2
        FFLD1
        V.+0            ;SALARY
        FML1
        FFST1
        V.+4            ;YERLY
```

CONTINUED

```
;       TOTAL = YERLY * (65 - AGE)
        FXFL1
        .C3
        FFLD1
        V.+2            ;AGE
        FSB1
        FFLD1
        V.+4            ;YERLY
        FML1
        FFST1
        V.+6            ;TOTAL

;       TYPE "TOTAL INCOME FROM" ,AGE, "TO RETIREMENT AT 65 IS $ ",TOTAL
        JSR     @.FWRI
        .C4
        0
        6
        .TXT    "TOTAL INCOME FROM"
        0
        2
        V.+2            ;AGE
        6
        .TXT    "TO RETIREMENT AT 65 IS $ "
        0
        2
        V.+6            ;TOTAL
        5

;       STOP
        JSR     @.STOP
         TXT    ""

;       END
        JSR     @.FRET
.C4:    000012
.C3:    000101
.C2:    000064
.C1:    000013

        FS.=10
        SFS.=0
        T.=-167
        V.=200+T.
        TS.=T.+7
        FTS.=T.+0
        VS.=V.+7
        FVS.=V.+0
        .END
```



1. Write the command for assembling the compiled Fortran file MONEY.SR. Show all anticipated results in the space provided:

****

```
) XEQ MASM/PS=FORT4.PS MONEY↓
                        .TITL    .MAIN

)
```

The /PS=FORT4.PS keyboard switch is required.

Note that either MONEY or MONEY.SR is sufficient here. The successful assembly is signalled by the .TITL message and CLI prompt.

If any errors are displayed on your screen, then it is necessary to return to SPEED and edit your MONEY.FR, recompile, and re-assemble. When ready, continue with the next question.

2. Write the commands to determine which files associated with MONEY now exist on your system. Show all anticipated responses:

****

```
) FI/LEN/TYP MONEY+ ↓
DIRECTORY @DPX1:

  MONEY.FR          TXT          222
  MONEY.LS           66          268
  MONEY.SR           66          992
  MONEY.OB          OBF          466
)
```

The /LEN switch displays the length and the /TYP switch displays the file type.

MONEY.SR was input to the macroassembler and MONEY.OB was the assembled result. Note the type "OBF" on MONEY.OB. This indicates a binary, object file.

Do it on your system.

Delete MONEY.OB so that it does not conflict with later questions. (Save the others.)

```
) DELETE/V/C MONEY.OB ↓
MONEY.OB? YES ↓
Deleted MONEY.OB
)
```

3. Assemble MONEY. Make MONEYMASM.LS the list file. Show all commands and anticipated responses:

_____

_____

_____

_____

_____

_____

****

```
) XEQ MASM/PS=FORT4.PS/L=MONEYMASM.LS MONEY )
)
```

The /L=MONEYMASM.LS keyword switch directs the listing to the disc file MONEYMASM.LS.

MONEY or MONEY.SR would suffice.

Do it on your system.

4. Which MONEY files now exist on the system? Write the commands and anticipated responses *before* checking the answer and *before* trying it on your system:

****

```
) FI/LEN/TYP MONEY+ )
DIRECTORY @DPX1:

  MONEY.FR          TXT          222
  MONEY.LS           66          268
  MONEY.SR           66          992
  MONEYMASM.LS      TXT         4478
  MONEY.OB          OBF          466
)
```

Do it on your system. Note the type "TXT" on MONEYMASM.LS.

If you do not have MONEY.OB and MONEYMASM.LS, then the assembly failed. Failure necessitates reviewing the source text, re-invoking SPEED, correcting the errors, re-compiling, and re-assembling.

5. MONEYMASM.LS makes for interesting reading. Write the command for displaying the contents of MONEYMASM.LS. *Briefly describe* the anticipated results (what does it contain):

_____

_____

_____

_____

_____

_____

_____

****

```
) TYPE MONEYMASM.LS↓
 0001  .MAIN  MP/OS  ASSEMBLER REV  1.00          07:27:79 12/52/03
01
02                ; DGC FORTRAN IV REV 05.20IS
03
04
05
06                ; C     *** TITLE INCOME ***
07
08                ;       ACCEPT "ENTER YOUR WEEKLY SALARY", SALARY
                          .TITL   .MAIN
10                        .ENT    .MAIN
11        000001          .NREL   1
12        000001          .TXTM   1
13                        .EXTU
14                        .EXTN   .I
15        000002          .CSIZ   2
16 00000!000010           FS.
17                .MAIN:
18 00001!002401           JMP     @.+1
19 00002!000003!          L1.
20                L1.:
21
22 00003!006000$          JSR     @.FREA
23 00004!000141!          .C1
24 00005!000000           0
25 00006!000006           6
26 00007!042516           .TXT    "ENTER YOUR WEEKLY SALARY"
27        052105
28        051040
29        054517
30        052522
31        020127
32        042505
33        045514
34        054440
35        051501
36        046101
37        051131
38        000000
39 00024!000000           0
40 00025!000002           2
41 00026!000011           V.+0            ;SALARY
42 00027!000005           5
```

CONTINUED

```
43
44                      ;       ACCEPT"ENTER YOUR AGE", AGE
45 00030!006000$                JSR     @.FREA
46 00031!000141!                .C1
47 00032!000000                 0
48 00033!000006                 6
49 00034!042516                 .TXT    "ENTER YOUR AGE"
50        052105
51        051040
52        054517
53        052522
54        020101
55        043505
56        000000
57 00044!000000                 0
58 00045!000002                 2
59 00046!000013                 V.+2            ;AGE
60 00047!000005                 5
 0002 .MAIN
01
02                      ;       YERLY = SALARY * 52
03 00050!000000$                FXFL1
04 00051!000140!                .C2
05 00052!000000$                FFLD1
06 00053!000011                 V.+0            ;SALARY
07 00054!000000$                FML1
08 00055!000000$                FFST1
09 00056!000015                 V.+4            ;YERLY
10
11                      ;       TOTAL = YERLY * (65 - AGE)
12 00057!000000$                FXFL1
13 00060!000137!                .C3
14 00061!000000$                FFLD1
15 00062!000013                 V.+2            ;AGE
16 00063!000000$                FSB1
17 00064!000000$                FFLD1
18 00065!000015                 V.+4            ;YERLY
19 00066!000000$                FML1
20 00067!000000$                FFST1
21 00070!000017                 V.+6            ;TOTAL
22
23                      ;       TYPE "TOTAL INCOME FROM" ,AGE, "TO RETIREMENT AT 65 IS
$ ",TOTAL
24 00071!006000$                JSR     @.FWRI
25 00072!000136!                .C4
26 00073!000000                 0
27 00074!000006                 6
28 00075!052117                 .TXT    "TOTAL INCOME FROM"
29        052101
30        046040
31        044516
32        041517
33        046505
34        020106
35        051117
36        046400
37 00106!000000                 0
38 00107!000002                 2
39 00110!000013                 V.+2            ;AGE
40 00111!000006                 6
41 00112!052117                 .TXT    "TO RETIREMENT AT 65 IS $ "
42        020122
43        042524
44        044522
45        042515
46        042516
```

CONTINUED

```
47        052040
48        040524
49        020066
50        032440
51        044523
52        020044
53        020000
54 00127!000000          0
55 00130!000002          2
56 00131!000017          V.+6              ;TOTAL
57 00132!000005          5
58
59                ;      STOP
60 00133!006000$         JSR       @.STOP
 0003 .MAIN
01 00134!000000          .TXT      ""
02
03                ;      END
04 00135!006000$         JSR       @.FRET
05 00136!000012 .C4:     000012
06 00137!000101 .C3:     000101
07 00140!000064 .C2:     000064
08 00141!000013 .C1:     000013
09
10        000010         FS.=10
11        000000         SFS.=0
12        177611         T.=-167
13        000011         V.=200+T.
14        177620         TS.=T.+7
15        177611         FTS.=T.+0
16        000020         VS.=V.+7
17        000011         FVS.=V.+0
18                       .END
19

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS  0004 .MAIN

FFLD1  000011$ XD    2/05     2/14     2/17
FFST1  000001$ XD    2/08     2/20
FML1   000002$ XD    2/07     2/19
FSB1   000010$ XD    2/16
FS.    000010        1/16     3/10#
FTS.   177611        3/15#
FVS.   000011        3/17#
FXFL1  000005$ XD    2/03     2/12
L1.    000003!       1/19     1/20#
SFS.   000000        3/11#
TS.    177620        3/14#
T.     177611        3/12#    3/13     3/14     3/15
VS.    000020        3/16#
V.     000011        1/41     1/59     2/06     2/09     2/15     2/18     2/21
                     2/39     2/56     3/13#    3/16     3/17
.C1    000141!       1/23     1/46     3/08#
.C2    000140!       2/04     3/07#
.C3    000137!       2/13     3/06#
.C4    000136!       2/25     3/05#
.FREA  000007$ XD    1/22     1/45
.FRET  000006$ XD    3/04
.FWRI  000004$ XD    2/24
.I     000012  XN    1/14
.MAIN  000001! EN    1/10     1/17#
.STOP  000003$ XD    2/60

)
```

Do it on your system.

MONEYMASM.LS is the assembler's listing file. It contains the original Fortran source statements, the assembly translation, the cross-reference listing, and the octal translation. Assembly listings are discussed in depth in the macroassembler concepts segment of this module.

Delete MONEYMASM.LS and MONEY.OB from the system so that they do not conflict with future questions:

```
) DELETE/V/C MONEYMASM.LS MONEY.OB ↓
MONEYMASM.LS? YES↓
Deleted MONEYMASM.LS
MONEY.OB? YES↓
Deleted MONEY.OB
)
```

6. Assemble MONEY. Make MONEYOBJ the object file. Show all commands and anticipated responses:

_____

_____

_____

_____

_____

****

```
) XEQ MASM/PS=FORT4.PS/B=MONEYOBJ MONEY↓
                          .TITL    .MAIN

)
```

Do it on your system.

The /B switch assigns the object file to the specified file.

7. Determine which files now exist on your system. Show the command and response:

****

```
) FI/AS/S MONEY+ ↓
DIRECTORY @DPX1:

  MONEY.FR            TXT    26-JUN-79   9:54:45          222
  MONEY.LS             66    27-JUL-79  10:40:14          268
  MONEY.SR             66    27-JUL-79  10:40:15          992
  MONEYOBJ.OB         OBF    27-JUL-79  13:01:40          466
)
```

Do it on your system. Note the .OB extension on MONEYOBJ. Since this is a binary file, you cannot use the TYPE command to examine it.

Get rid of MONEYOBJ.OB so that it does not conflict with the next questions:

```
) DELETE/V/C MONEYOBJ.OB ↓
MONEYOBJ.OB? YES ↓
Deleted MONEYOBJ.OB
)
```

8. Assemble MONEY. This time, add semi-permanent sybmols to the cross-reference listing. Make MONEYMASM.LS the list file:

_____

_____

_____

_____

_____

_____

****

```
) XEQ MASM/PS=FORT4.PS/P/L=MONEYMASM.LS  MONEY )

)
```

The /P switch directs the assembler to include the semi-permanent symbols (such as instruction mnemonics) to the cross-reference listing. This can be an aid in the debugging process.

Do it on your system. Check your spelling.

9. Which MONEY files now exist on your system. Show the command and anticipated responses:

****

```
) FI/AS/S MONEY+↓
DIRECTORY @DPX1:

  MONEY.FR              TXT    26-JUN-79   9:54:45        222
  MONEY.LS               66    27-JUL-79  10:40:14        268
  MONEY.OB              OBF    27-JUL-79  13:05:34        466
  MONEY.SR               66    27-JUL-79  10:40:15        992
  MONEYMASM.            TXT    27-JUL-79  13:06:05       4574
)
```

Do it on your system.

Note the time appended to the assembler's listing file and the object file. Are they the same? (They shouldn't be.)

10. What would MONEYMASM.LS look like? Is it the same as the previous copy of MONEYMASM.LS? Write the command and *briefly describe* the response:

_____

_____

_____

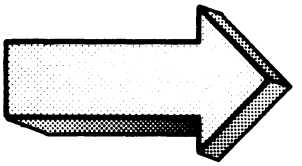
_____

_____

_____

****

```
) TYPE MONEYMASM.LS)
  .
  .
  .
FFLD1  000011$ XD     2/05     2/14     2/17
FFST1  000001$ XD     2/08     2/20
FML1   000002$ XD     2/07     2/19
FSB1   000010$ XD     2/16
FS.    000010         1/16     3/10#
FTS.   177611         3/15#
FUS.   000011         3/17#
FXFL1  000005$ XD     2/03     2/12
JMP    000000         1/18
JSR    004000         1/22     1/45     2/24     2/60     3/04
L1.    000003!        1/19     1/20#
SFS.   000000         3/11#
TS.    177620         3/14#
T.     177611         3/12#    3/13     3/14     3/15
US.    000020         3/16#
U.     000011         1/41     1/59     2/06     2/09     2/15     2/18     2/21
                      2/39     2/56     3/13#    3/16     3/17
.C1    000141!        1/23     1/46     3/08#
.C2    000140!        2/04     3/07#
.C3    000137!        2/13     3/06#
.C4    000136!        2/25     3/05#
.FREA  000007$ XD     1/22     1/45
.FRET  000006$ XD     3/04
.FWRI  000004$ XD     2/24
.I     000012  XN     1/14
.MAIN  000001! EN     1/10     1/17#
.STOP  000003$ XD     2/60

)
```



Note the required spelling of the MONEYMASM listing file.

Do it on your system.

Our example shows only the cross-reference listing. Note the addition of the jump instruction mnemonics (JMP and JSR). Oddly enough, our assembly translation only uses the two jump mnemonics. All other assembly statements involve system calls to assembler subroutines.

Delete MONEYMASM.LS and MONEY.OB from your system.

```
) DELETE/V/C MONEY.OB MONEYMASM.+
MONEY.OB? YES↓
Deleted MONEY.OB
MONEYMASM.? YESY↓
Deleted MONEYMASM.
)
```

11. Assemble MONEY again. This time make CASHLIST the list file and add user symbols to the object file. Show the command and anticipated response:

_____

_____

_____

_____

_____

_____

****

```
) XEQ MASM/PS=FORT4.PS/U/L=CASHLIST MONEY
)
```

The /U switch directs the macroassembler to include the user-defined symbols with the assembled object file.

Do it on your system.

12. Which MONEY and CASH files now exist on your system? Get their sizes (especially the object file). Show the command and estimated response:

****

```
) FI/LEN MONEY+ CASH+ ↓
DIRECTORY @DPX1:

  MONEY.FR                    222
  MONEY.LS                    268
  MONEY.SR                    992
  MONEY.OB                    642
  CASHLIST                   4478
)
```

Do it on your system.

Compare the size of this MONEY.OB with previous MONEY.OBs. Bigger? (It should be, you added the user symbols to the object file.) Also compare the listings.

Get rid of CASHLIST and MONEY.OB:

```
) DEL/V/C CASHLIST MONEY.OB ↓
CASHLIST? YES↓
Deleted CASHLIST
MONEY.OB? YES↓
Deleted MONEY.OB
)
```

13. Assemble MONEY for the final time in this lab exercise. This time,keep the temporary symbol table file at the end of the assembly. Show the command and anticipated responses:

****

```
) XEQ MASM/PS=FORT4.PS/K MONEY↵
                         .TITL    .MAIN

)
```

The /K switch directs the macroassembler to retain the symbol table file upon completion of the assembly. It is stored under ?MASM.ST.TMP.

Do it on your system.

14. Which MONEY files exist on the system now? Show the command and responses. Also, dig up the temporary symbol table file:

****

```
) FI/TYP/LEN MONEY+  ?+ ↓
DIRECTORY @DPX1:

  MONEY.FR          TXT          222
  MONEY.LS           66          268
  MONEY.SR           66          992
  MONEY.OB          OBF          466
  ?MASM.ST.TMP      PST        15360
)
```

Note the use of the + template in the filename argument.

Do it on your system. Note the size and type of the ?MASM.ST.TMP file.

Now clean up your system. Delete the ?MASM.ST.TEMP file. Keep the others for use in future exercises.

```
) DELETE/V/C +?+
?MASM.ST.TMP? YES
Deleted ?MASM.ST.TMP
)
```

This concludes the Lab Exercise "Assembling Fortran .SR files. Shut down your system. Keep a copy of the MONEY source, compiled, and object files (MONEY.FR, MONEY.SR, MONEY.OB).

The next step for the Fortran programmer is the BINDING segment of Module Four. Skip over the Pascal segment and continue to the BINDER.

PASCAL COMPILATION

Abstract

This unit instructs in the concepts and procedures involved in compiling Pascal source files into relocatable binary object files.
NOTE: This segment is of primary interest to the Pascal programmer. Experts in other languages should skip this segment.

Objectives

Upon completion of this unit, you will be able to:

1. Given a CLI Pascal compiler command line, identify and state the purpose of the switches and arguments.

2. List the reference material available on Pascal compiler commands.

3. Given a CLI Pascal compiler command line, describe the results.

4. Write an appropriate CLI Pascal compiler command line to compile a given Pascal program. State the output names with extensions.

5. Given a Pascal compiler error situation:
   A) identify possible causes;
   B) reference solutions;
   C) write commands to correct the error.

6. State the purpose of compiler directives.

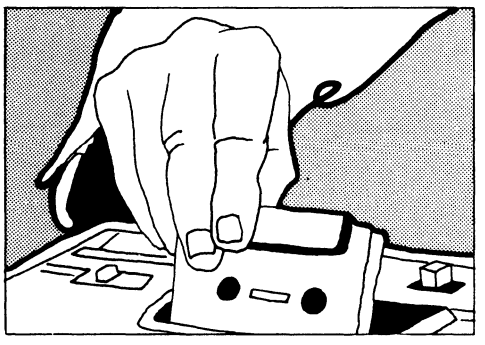
7. Given a compiler directive statement, describe the result.

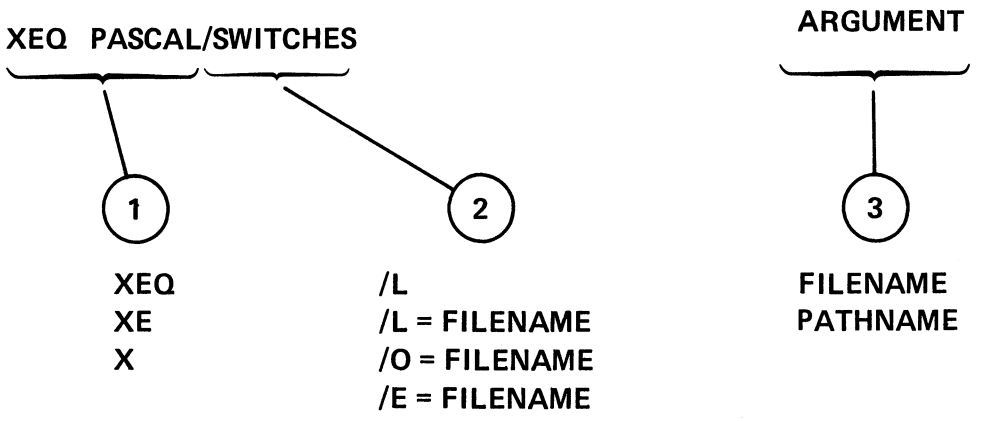
8. Code the compiler directive for producing a stated compiler response.

Directions

1. Turn to figure 4-106 on the next page of the Student Guide.

2. Listen to the tape for this unit.





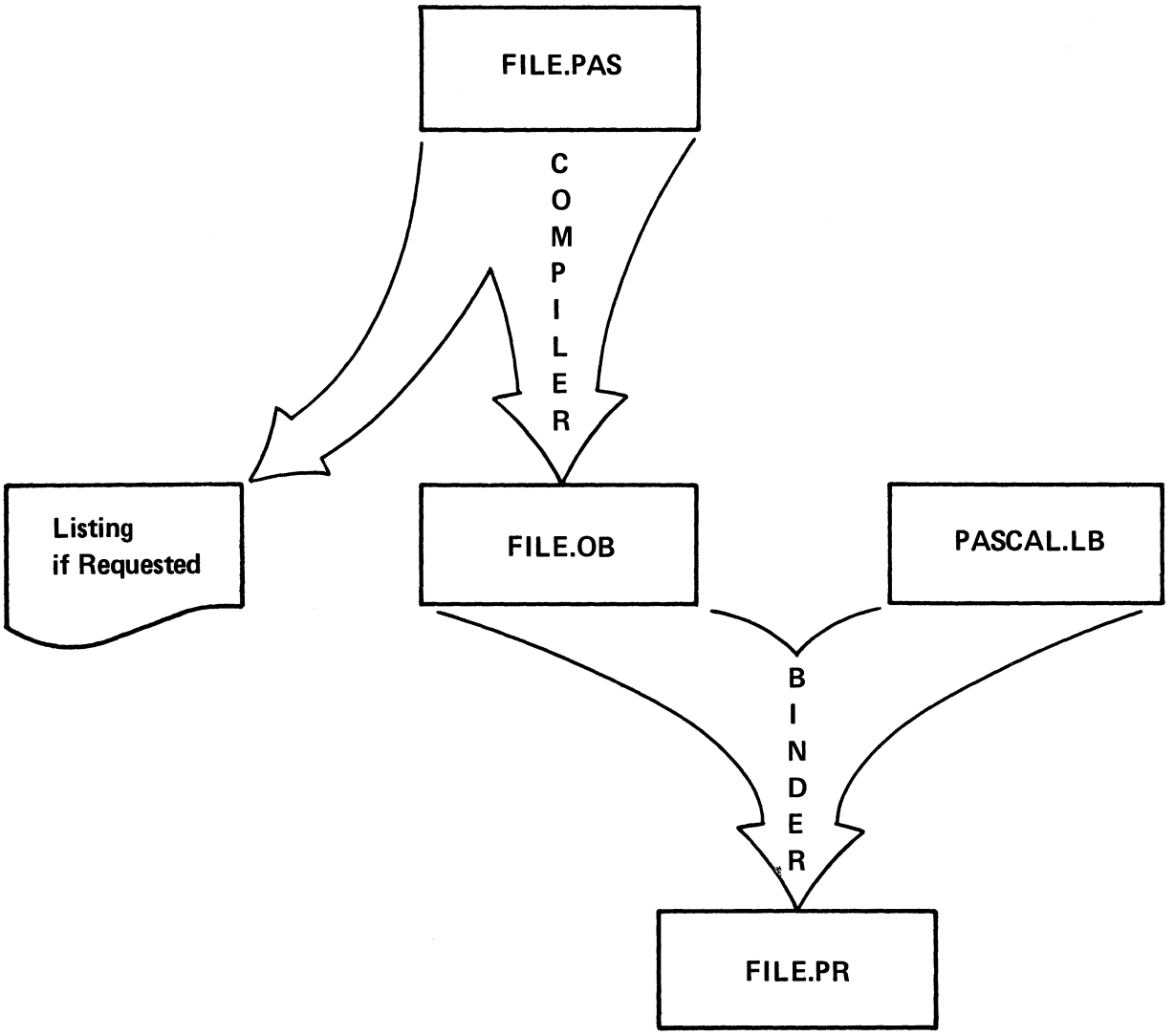
CLI PASCAL COMPILER COMMAND LINE SYNTAX

Figure 4-106

SWITCH	FUNCTION
/L	Direct compiler listing to the line printer.
/L = filename	Direct compiler listing to the disc file "filename"
/O = filename	Give the object file the specified filename.
/E = filename	Write error messages to the specified filename and save it on the disc device.

PASCAL COMPILER COMMAND SWITCHES

Figure 4-107



PASCAL PROGRAM DEVELOPMENT

Figure 4-108

RESULT

1

```
) XEQ PASCAL TEST_ONE↵
No Compilation Errors
)
```

TEST_ONE
TEST.ONE.OB
} On Disc

Error Count on Console

2

```
) X PASCAL TEST_ONE.PAS
No Compilation Errors
)
```

TEST_ONE.PAS
TEST_ONE.OB
} On Disc

Error Count on Console

3

```
) XEQ PASCAL/L TEST_ONE↵
```

TEST_ONE
TEST_ONE.OB
} On Disc

Listing on LPT

Error Count on Console

/L . . . Direct listing to line printer.

Figure 4-109

RESULT

1

```
) XEQ PASCAL/L=TEST_ONE.LS TEST_ONE )
```

TEST_ONE
TEST_ONE.LS
TEST_ONE.OB
} On Disc

Error Count on Console

/L = filename . . . send listing to disc under filename.

2

```
) XEQ PASCAL/O=TESTOBJ TEST_ONE )

No Compilation Errors
)
```

TEST_ONE
TESTOBJ.OB
} On Disc

Error Count on Console

/O = filename . . . make filename the object file.

3

```
) XEQ PASCAL/O=TESTOBJ.OB TEST_ONE )

No Compilation Errors

)
```

TESTONE
TESTOBJ.OB
} On Disc

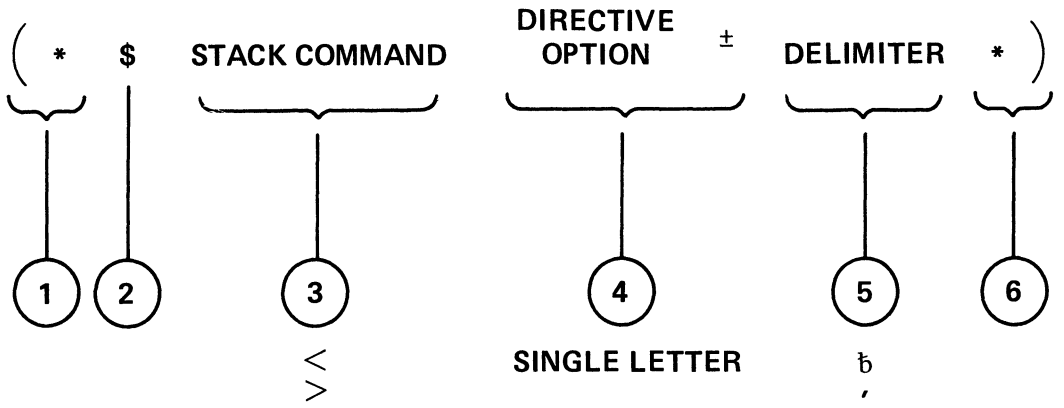
Error Count on Console

Figure 4-110

```
MP/PASCAL              REV 1.00              01-JAN-00       00:01:59

  1.   PROGRAM SMITH_ONE;
  2.   INCLUDE IO CALLS.PAS;
  3.
  4.   { ********************** }
  5.   { PRIMITIVE I/O ROUTINES }
  6.   { ********************** }
  7.
  8.   CONST    MAX_LINE_LTH = 136;
  9.            MAX_PATH_LTH = 128;
 10.
 11.            INCH = 0R8:      {STANDARD INPUT CHANNEL}
 12.            OUCH = 1R8:      {STANDARD OUTPUT CHANNEL}
 13.
 14.            { OPEN OPTIONS}
 15.            EX = 40000R8:    {EXCLUSIVE ACCESS}
 16.            NZ = 10000R8:    {DON'T ZERO BLOCKS ON I/O}
 17.            CR =  4000R8:    {FILE CREATION}
 18.            DE =  2000R8:    {FILE DELETION}
 19.            UC =  1000R8:    {UNCONDITIONAL CREATION}
 20.            AP =   400R8:    {APPEND}
 21.
 22.   TYPE     CHANNEL = 0..15;
 23.            PATHNAME = STRING MAX_PATH_LTH;
 24.            LINE_BUFFER = STRING MAX_LINE_LTH;
 25.            IO_BUFFER = STRING 32767;
 26.            FILE_POSITION = RECORD
 27.                              HIGH: INTEGER;
 28.                              LOW:INTEGER
 29.                            END;
 30.
 31.   EXTERNAL ASSEMBLY PROCEDURE OPENFILE(VAR CHAN: CHANNEL;
 32.                                        FILE: PATHNAME;
 33.                                        OPTIONS: INTEGER;
 34.                                        FILE_TYPE: INTEGER;
 35.                                        ELEM_SIZE: INTEGER;
 36.                                    VAR STATUS: INTEGER);
 37.
 38.   EXTERNAL ASSEMBLY PROCEDURE CLOSEFILE(CHAN: CHANNEL; VAR STATUS:INTEGER);
 39.   EXTERNAL ASSEMBLY PROCEDURE CLDELFILE(CHAN: CHANNEL; VAR STATUS:INTEGER);
 40.
 41.   EXTERNAL ASSEMBLY PROCEDURE LINEREAD(CHAN: CHANNEL;
 42.                                VAR BUFFER: LINE_BUFFER; VAR STATUS: INTEGER);
 43.
 44.   EXTERNAL ASSEMBLY PROCEDURE LINEWRITE(CHAN: CHANNEL; BUFFER: LINE_BUFFER;
 45.                                    VAR STATUS: INTEGER);
 46.
 47.   EXTERNAL ASSEMBLY PROCEDURE CHARREAD(CHAN: CHANNEL;
 48.                                LTH: INTEGER; VAR BUFFER: IO_BUFFER;
 49.                                VAR STATUS: INTEGER);
 50.
 51.   EXTERNAL ASSEMBLY PROCEDURE CHARWRITE(CHAN: CHANNEL; BUFFER: IO_BUFFER;
 52.                                    VAR STATUS: INTEGER);
 53.
 54.   EXTERNAL ASSEMBLY PROCEDURE BYTEREAD(CHAN: CHANNEL;
 55.                                    BUF_ADDRESS: INTEGER; VAR LTH: INTEGER;
 56.                                    VAR STATUS: INTEGER);
 57.
 58.   EXTERNAL ASSEMBLY PROCEDURE BYTEWRITE(CHAN: CHANNEL;
 59.                                    BUF_ADDRESS: INTEGER; LTH: INTEGER;
 60.                                    VAR STATUS: INTEGER);
 61.
 62.
 63.   EXTERNAL ASSEMBLY PROCEDURE GPOSFILE(CHAN: CHANNEL;
 64.                    VAR POSITION: RECAST FILE_POSITION; VAR STATUS:  INTEGER);
 65.
 66.   EXTERNAL ASSEMBLY PROCEDURE SPOSFILE(CHAN: CHANNEL;
 67.                    POSITION: RECAST FILE_POSITION; VAR STATUS: INTEGER);
 68.
 69.   VAR ST:INTEGER;
 70.   BEGIN
 71.   LINEWRITE(OUCH,'THIS IS AN EXAMPLE OF MP/PASCAL DEVELOPMENT <12>'.ST);
 72.   IF ST <> 0 THEN LINEWRITE(OUCH, 'ERROR ON MESSAGE'.ST)
 73.   END.
```

COMPILER LISTING

Figure 4-111



COMPILER DIRECTIVES COMMENT

Figure 4-112

OPTION	FUNCTION
C	Generate code to check all case instructions.
I	List all INCLUDE files.
N	Produce code to update statement line numbers.
O	Generate code to check for overflow on all integer arithmetic operations.
P	Generate code to initialize to zero all pointers and structures containing pointers. Check all pointer references for a value of nil.
R	Generate code to check all subrange assignments.
S	Generate code to check all array subscripts.
V	Generate code to check all variant references.

COMPILER DIRECTIVES.

Figure 4-113

1	(* $ C + *)	CHECK CASE EXPRESSIONS
2	(* $ O – *)	SKIP OVERFLOW CHECKING
3	(* $ I +, R – *)	LIST INCLUDE FILES SKIP UPDATING LINE NUMBERS
4	(*$ < R + *)	REINSTATE LINE NUMBER UPDATING

EXAMPLES OF COMPILER DIRECTIVES COMMENTS

Figure 4-114

```
MP/PASCAL              REV 1.00              03-JUL-79       15:03:26

   1.   PROGRAM SMITH ONE:
   2.   (*$I-*)
   3.   INCLUDE IO_CALLS.PAS:
  70.   VAR ST:INTEGER:
  71.   BEGIN
  72.   LINEWRITE(OUCH,'THIS IS AN EXAMPLE OF MP/PASCAL DEVELOPMENT <12>',ST):
  73.   IF ST <> 0 THEN LINEWRITE(OUCH, 'ERROR ON MESSAGE',ST)
  74.   END.
```

Figure 4-115

TOPICS

- MP/PASCAL COMPILATION
- MP/OS UTILITIES REFERENCE MANUAL
- MP/PASCAL PROGRAMMERS REFERENCE MANUAL
- CLI COMMAND LINE
- OPTIONAL SWITCHES
- COMPILER DIRECTIVES

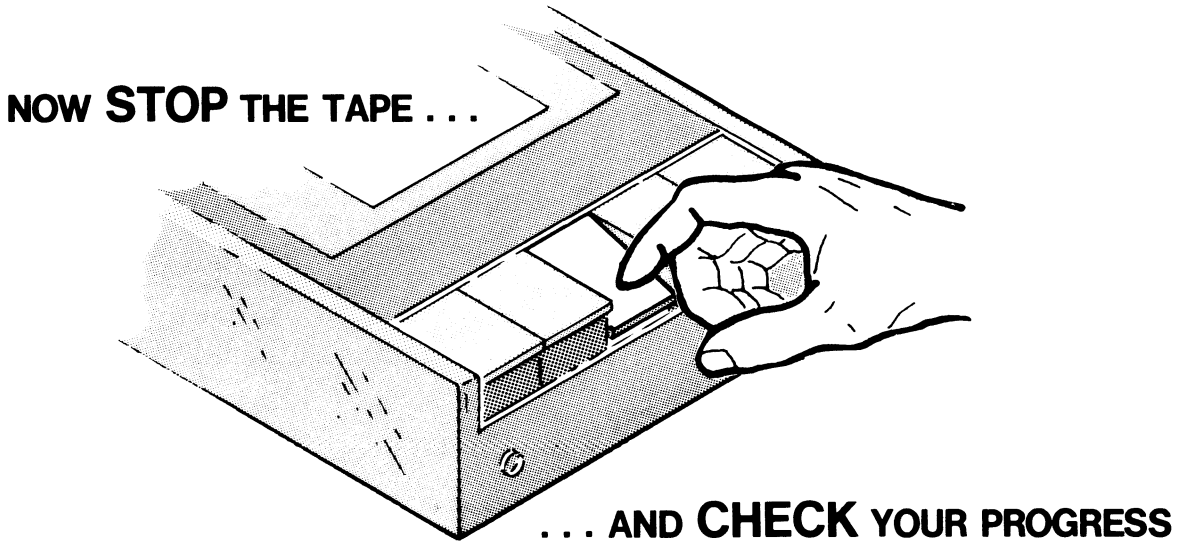


Figure 4-116

PASCAL COMPILATION

QUIZ

Write the answers in the space provided.

Given the following CLI Pascal compiler command line, identify the switches and arguments and state the purpose of the switches and arguments:

) XEQ PASCAL/L=QUIZ.LS/0=QUIZOBJ QUIZ.PAS

1. Switches: _____

2. Arguments: _____

3. Purpose of the switches: _____

4. Purpose of the argument: _____

List three sources of information on MP/Pascal compiler commands:

5. _____

6. _____

7. _____

Given the following CLI MP/Pascal compiler command lines, briefly describe the anticipated result:

```
) XEQ PASCAL HOBBIT
```

8. _____

_____

_____

_____

```
) XEQ PASCAL HOBB.PAS
```

9. _____

_____

_____

```
) XEQ PASCAL/L=HOBB.LS HOBB.PAS
```

10. _____

_____

_____

_____

```
) XEQ PASCAL/0=HOBBOBJ HOBBIT
```

11. _____

_____

_____

_____

What is the purpose of compiler directives?

12. _____

_____

_____

Given the following compiler directive comments coded into separate Pascal programs, describe the anticipated result:

```
(*$I- *)
```

13. _____

_____

_____

```
(*$C- *)
```

14. _____

_____

```
(*$0-,R-,C- *)
```

15. _____

_____

_____

Check your answers on the following page . . .

PASCAL COMPILATION

QUIZ ANSWERS

The switches, arguments, and purposes of the following command line are:

) XEQ PASCAL/L=QUIZ.LS/0=QUIZOBJ QUIZ.PAS

1. Switches: /L=QUIZ.LS
   /0=QUIZOBJ

2. Argument: QUIZ.PAS

3. Purpose of the switches: /L=QUIZ.LS instructs the compiler to store the listing on disc under the name "QUIZ.LS". This can be displayed by the CLI TYPE command. /0=QUIZOBJ directs the compiler to name the binary object module "QUIZOBJ.OB".

4. Purpose of the argument: QUIZ.PAS identifies the Pascal source file to be compiled. The object file is input to the binder to produce an executable program file.

Three sources of information of MP/PASCAL command lines include:

5. MP/OS Utilities Reference Manual (093-40002)

6. The CLI HELP command, if your system is so equipped.

7. This self/study manual

Others: The MP/PASCAL Self/Study course.
MP/PASCAL Programmer's Reference Manual (093-400003).

The anticipated result of the given compiler commands are as follows:

) XEQ PASCAL HOBBIT

8. The compiler is invoked and searches for HOBBIT.PAS. If not found the compiler searches for HOBBIT. The error count is displayed on the console. No listing is produced. A successful compilation is signalled by the message "NO COMPILATION ERRORS" on the console. The object file is titled HOBBIT.OB.

) XEQ PASCAL HOBB PAS

9. This is almost identical to question #8. The only difference is that the compiler searches only for HOBB.PAS. If not found, the error message "FILE NOT FOUND" is displayed on the console. No listing is produced. The object file is named TEST.OB and is stored on disc.

) XEQ PASCAL/L=HOBB.LS HOBB.PAS

10. HOBB.PAS is compiled. HOBB.OB is the object file in a successful compile. The error count is displayed on the console. The listing is stored on disc under the name HOBB.LS.

) XEQ PASCAL/0-HOBBOBJ HOBBIT

11. HOBBIT.PAS (or HOBBIT, if HOBBIT.PAS does not exist) is compiled. The compiled object file is stored on disc under the name "HOBBOBJ.OB". No listing is produced. The error count is directed to the console.

12. The purpose of compiler directives is to request the generation (or non-generation) of code for various Pascal routines.

The anticipated result of the following Pascal program comments is as follows:

(*$I-*)

13. Do not list the "INCLUDE" files in the compiler listing.

(*$C- *)

14. Do not generate the code for checking case statements.

(*$0-, R-, C- *)

15. Skip the code for checking overflow, updating statement numbers, and checking case statements.

A SCORE OF 12 CORRECT ANSWERS OUT OF THE 15 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.

PASCAL COMPILATION

LAB EXERCISE

Directions

This lab exercise is similar to the preceding exercises: it may be completed with or without a functioning system. Learning is enhanced by using a computer. Complete each step of the exercise for maximum benefit.

1. You will need the following files to complete this lab:

PASCAL.PR . . . . . .The Pascal compiler
PASCAL.OL . . . . . .Compiler overlay file.
IOCALLS.PAS . . . . . .Include files for external procedures.

Use the CLI FILESTATUS command to determine the presence of these files. Note that IOCALLS.PAS is a text file which may be printed out for examination.

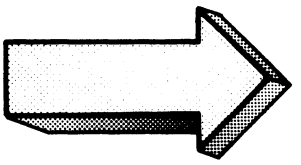
2. SPEED in the program "SMITH_TWO" exactly as shown below. Name it "TWO".

```
PROGRAM SMITH_TWO;
INCLUDE IO_CALLS.PAS;
VAR J,ST:INTEGER;
BEGIN
FOR J:= 1 TO 20 DO LINEWRITE(OUCH,'          * <12>',ST);
IF ST <> Ø THEN LINEWRITE(OUCH, 'ERROR ON WRITE',ST);
LINEWRITE (OUCH,'        MP/OS HAS LANDED <12>',ST);
IF ST <> Ø THEN LINEWRITE (OUCH, 'ERROR ON LANDING',ST)
END.
```

3. Make a backup copy of "TWO". Use the CLI COPY command (or MOVE utility if you are so equipped). Do not give the backup a name with "TWO" in it, to avoid confusion with the lab sequence.

You are now ready for the lab exercise. Remember:

1. Cover the answers;
2. Read the question;
3. Write in your answer;
4. Check the answer;
5. Perform the operation on your system.



1. Compile TWO. Show the commands and anticipated results in the space below. Do not look at the answer until you have written in your response. Do not enter any commands on your system until you have checked your response for accuracy.

_____

_____

_____

_____

_____

****

```
) NEW PASCAL TWO )

No Compilation Errors

)
```

The system searches first for TWO.PAS and then for TWO.

Did your response match the answer? If not, review the source file for possible errors, SPEED in the corrections, and continue.

Now do it on your system.

The message "NO COMPILATION ERRORS" indicates an error-free compilation. If any errors are displayed, it is necessary to return to SPEED, edit the source file, and repeat this first step.

When you are sure you are error-free, continue with question #2.

2. Which files associated with TWO now exist on your system. Write the command and anticipated response:

_____

_____

_____

_____

_____

****

```
DIRECTORY @DPD0:PASCAL

  TWO                TXT    3-JUL-79   15:09:42        293
  TWO.OB             UDF    3-JUL-79   15:20:35       1140
```

Note that our directory is @DPD0:PASCAL, a subdirectory on a 10 megabyte disc. Your directory may be different so adjust your response accordingly.

Now do it on your system.

If you do not have TWO.OB then compilation failed. Reinvoke SPEED, edit your Pascal program, and try again.

Delete TWO.OB so that it does not conflict with succeeding questions.

```
) DELETE/V/C  TWO.OB ↓
TWO.OB?YES ↓
DELETED TWO.OB
)
```

3. Compile TWO. This time make TWO.LS the listing file. Show all commands and anticipated responses in the space provided:

_____

_____

_____

_____

****

```
) XEQ PASCAL/L=TWO.LS    TWO
No Compilation Errors
)
```

Note that either TWO.PAS or TWO would be sufficient in this case.

Now do it on your system.

Be careful of the spelling and spacing.

The "/L=TWO.LS" switch creates a disc file named TWO.LS and writes the listing into it.

4. Which files associated with Two now exist on your system? Show the command and anticipated responses in the space below:

> REMEMBER: TO GET MAXIMUM BENEFIT FROM THIS EXERCISE, WRITE IN YOUR ANSWERS *BEFORE* EXECUTING THEM ON YOUR SYSTEM.

_____

_____

_____

_____

_____

_____

****

```
DIRECTORY @DPDO:PASCAL

  TWO                 TXT    3-JUL-79   15:09:42        293
  TWO.LS              UDF    3-JUL-79   15:18:45       3174
  TWO.OB              UDF    3-JUL-79   15:18:44       1140
```

The FILESTATUS command displays the information.

Now do it on your system.

TWO.LS, the requested listing, makes for interesting reading. Print out a copy of this file (TYPE/L). You will note that it is considerably longer than the few source lines you entered.

(LISTING IS ON THE NEXT PAGE.)

```
PROGRAM SMITH_TWO;
INCLUDE IO_CALLS.PAS;

{ ********************* }
{ PRIMITIVE I/O ROUTINES }
{ ********************* }

CONST   MAX_LINE_LTH = 136;
        MAX_PATH_LTH = 128;

        INCH = 0B8:     {STANDARD INPUT CHANNEL}
        OUCH = 1B8:     {STANDARD OUTPUT CHANNEL}

        { OPEN OPTIONS}
        EX = 40000B8:   {EXCLUSIVE ACCESS}
        NZ = 10000B8:   {DON'T ZERO BLOCKS ON I/O}
        CR =  4000B8:   {FILE CREATION}
        DE =  2000B8:   {FILE DELETION}
        UC =  1000B8:   {UNCONDITIONAL CREATION}
        AP =   400B8;   {APPEND}

TYPE    CHANNEL = 0..15;
        PATHNAME = STRING MAX_PATH_LTH;
        LINE_BUFFER = STRING MAX_LINE_LTH;
        IO_BUFFER = STRING 32767;
        FILE_POSITION = RECORD
                        HIGH: INTEGER;
                        LOW:INTEGER
                      END;

EXTERNAL ASSEMBLY PROCEDURE OPENFILE(VAR CHAN: CHANNEL;
                                     FILE: PATHNAME;
                                     OPTIONS: INTEGER;
                                     FILE_TYPE: INTEGER;
                                     ELEM_SIZE: INTEGER;
                                   VAR STATUS: INTEGER);

EXTERNAL ASSEMBLY PROCEDURE CLOSEFILE(CHAN: CHANNEL; VAR STATUS:INTEGER);
EXTERNAL ASSEMBLY PROCEDURE CLDELFILE(CHAN: CHANNEL; VAR STATUS:INTEGER);

EXTERNAL ASSEMBLY PROCEDURE LINEREAD(CHAN: CHANNEL;
                           VAR BUFFER: LINE_BUFFER; VAR STATUS: INTEGER);

EXTERNAL ASSEMBLY PROCEDURE LINEWRITE(CHAN: CHANNEL; BUFFER: LINE_BUFFER;
                                      VAR STATUS: INTEGER);

EXTERNAL ASSEMBLY PROCEDURE CHARREAD(CHAN: CHANNEL;
                          LTH: INTEGER; VAR BUFFER: IO_BUFFER;
                          VAR STATUS: INTEGER);

EXTERNAL ASSEMBLY PROCEDURE CHARWRITE(CHAN: CHANNEL; BUFFER: IO_BUFFER;
                                      VAR STATUS: INTEGER);

EXTERNAL ASSEMBLY PROCEDURE BYTEREAD(CHAN: CHANNEL;
                                   BUF_ADDRESS: INTEGER; VAR LTH: INTEGER;
                                   VAR STATUS: INTEGER);

EXTERNAL ASSEMBLY PROCEDURE BYTEWRITE(CHAN: CHANNEL;
                                   BUF_ADDRESS: INTEGER; LTH: INTEGER;
                                   VAR STATUS: INTEGER);


EXTERNAL ASSEMBLY PROCEDURE GPOSFILE(CHAN: CHANNEL;
                VAR POSITION: RECAST FILE_POSITION; VAR STATUS:  INTEGER);

EXTERNAL ASSEMBLY PROCEDURE SPOSFILE(CHAN: CHANNEL;
                  POSITION: RECAST FILE_POSITION; VAR STATUS: INTEGER);

VAR J,ST:INTEGER;
BEGIN
FOR J:= 1 TO 20 DO LINEWRITE(OUCH,'          * <12>',ST);
IF ST <> 0 THEN LINEWRITE(OUCH, 'ERROR ON WRITE',ST);
LINEWRITE (OUCH,'        MICRON HAS LANDED <12>',ST);
IF ST <> 0 THEN LINEWRITE (OUCH, 'ERROR ON LANDING',ST)
END.
```

Note the addition of the IO_CALLS.PAS information.

DELETE TWO.LS and TWO.OB to avoid confusion with later exercises. Keep the printout of TWO.LS for later questions:

```
) DEL/V TWO.OB TWOL.S↓
Deleted TWO.OB
Deleted TWO.LS
)
```

5. Compile TWO. Make TWOOBJ the object file. Show all commands and anticipated responses:

_____

_____

_____

_____

_____

****

```
) XEQ PASCAL/O=TWOOBJ  TWO↓
No Compilation Errors
)
```

Now do it on your system.

The "/O=TWOOBJ" keyword switch assigns the disc file "TWOOBJ.OB" as the object file pending a successful compilation.

6. Which files associated with TWO now exist on your system? Show all commands and anticipated responses?

_____

_____

_____

_____

_____

****

```
) FI TWO+↓
DIRECTORY @DPD0:PASCAL

        TWO
        TWOOBJ.OB
)
```

Now do it on your system.

The default name for the object file is created by stripping the .PAS extension, if any, from the source file. You overrode the default by appending the /0 switch to the Pascal command.

Compare the length of TWO.OB from previous compilations with TWOOBJ.OB from #5. Are they the same? (They should be!).

Now delete TWOOBJ.OB

```
) DELETE/V/C TWOOBJ.OB ↓
TWOOBJ? YES↓
DELETED TWOOBJ
)
```

7. Your listing file was stretched out by the listing of the "INCLUDE" file IO_CALLS.PAS. This time we want to create a listing that does not write out each line of IO_CALLS.PAS. Write the Pascal program comment that suppresses the listing of INCLUDE files. (Just write the comment, not the whole program.)

_____

_____

****

```
PROGRAM SMITH_TWO;
 (*EXAMPLE OF COMPILER DIRECTIVE OPTION 'I' *)
(*$I-*)
INCLUDE IO_CALLS.PAS;
VAR J,ST:INTEGER;
BEGIN
FOR J:= 1 TO 20 DO LINEWRITE(OUCH,'          * <12>',ST);
IF ST <> 0 THEN LINEWRITE(OUCH, 'ERROR ON WRITE',ST);
LINEWRITE (OUCH,'          MP/OS HAS LANDED <12>',ST);
IF ST <> 0 THEN LINEWRITE (OUCH, 'ERROR ON LANDING',ST)
END.
```

Notice that our comment was inserted via SPEED.

SPEED the comment into your program. Rename your output so that TWO is the new, edited version. The SPEED FB or FU commands are useful for this.

8. Now compile TWO. Produce a listing on the line printer. *Briefly* describe the contents of the listing file as they should now appear:

_____

_____

_____

_____

_____

****

```
MP/PASCAL              REV 00.00              03-JUL-79        15:32:42

   1.   PROGRAM SMITH_TWO;
   2.    (*EXAMPLE OF COMPILER DIRECTIVE OPTION 'I' *)
   3.   (*$I-*)
   4.   INCLUDE IO_CALLS.PAS;
  71.   VAR J,ST:INTEGER;
  72.   BEGIN
  73.   FOR J:= 1 TO 20 DO LINEWRITE(OUCH,'          * <12>',ST);
  74.   IF ST <> 0 THEN LINEWRITE(OUCH, 'ERROR ON WRITE',ST);
  75.   LINEWRITE (OUCH,'          MP/OS HAS LANDED <12>',ST);
  76.   IF ST <> 0 THEN LINEWRITE (OUCH, 'ERROR ON LANDING',ST)
  77.   END.
```

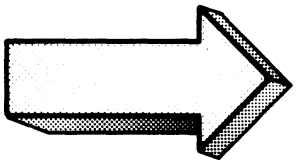
Do it on your system.

The shorter version is forced by the 'I-' compiler directive.

Note that if you did not have any INCLUDE statements in your program, the code would not have been generated.

This concludes the Pascal Compilation Lab Exercise. You may want to try the other compiler directives to see their results. The TWO program is a good experimental example.

When you are ready, shut down the system. Keep a copy of TWO.PAS and TWO.OB for future reference in this course. Continue with the next segment of this module.



BINDING

Abstract

This unit instructs in the concepts and procedures involved in binding MP/Fortran, MP/Assembly, and MP/Pascal relocatable binary object modules into executable program files.

Note: This segment should be completed by programmers of all three languages.

Objectives

Upon completion of this unit, you will be able to:

1. Given a CLI BINDER command line, identify and state the purpose of the switches and arguments.

2. List the reference material available on BINDER commands.

3. Given a CLI BINDER command line, describe the results.

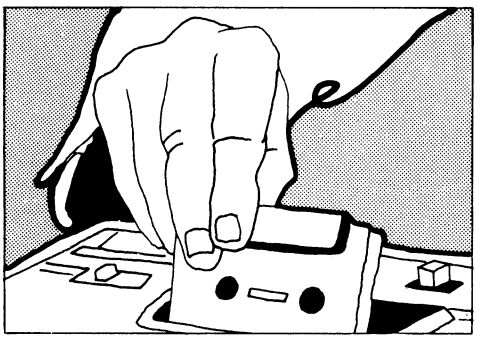
4. Write an appropriate CLI BINDER command line to bind a given object module into a program file.

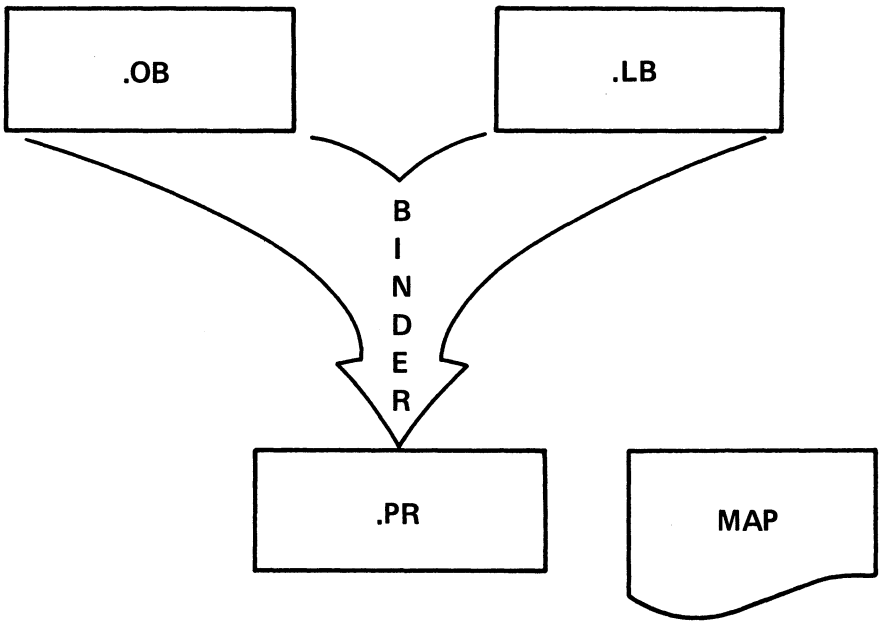
5. Given a BINDER error situation:
   - A) Identify possible causes;
   - B) Reference solutions;
   - C) Write commands to correct the error.

Directions

1. Turn to figure 4-128 on the next page of the Student Guide.

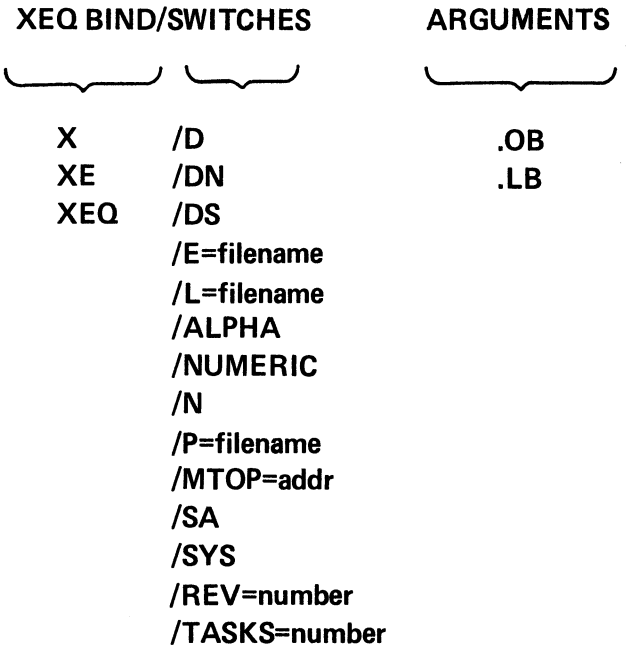
2. Listen to the tape for this segment.





FORTRAN OR ASSEMBLY OR PASCAL
SIMPLIFIED BINDING PROCESS

Figure 4-128



CLI BINDER COMMAND LINE

Figure 4-129

SWITCH	FUNCTION
/D	Include the Debugger, and place the symbol table in the program immediately above the impure area.
/DN	Include the Debugger, but not the symbol table.
/DS	Include the Debugger and the symbol table, but put the symbol table at the top of memory (or just below the pure area if the /SA switch is specified).
/E=*name*	Put error messages into file *name*. If you do not use this switch, error messages will go to ?OUCH, the standard output channel.
/L=*name*	Put the load map in file *name*. If you do not use this switch, no map will be generated.
/ALPHA	(Used with /L) Sort the list of symbols into alphabetical order.
/NUMERIC	(Used with /L) Sort the list of symbols by numeric value.
/N	Do not search the standard library file, MSL.LB.
/P=*name*	Put the program in file *name*.PR.
/MTOP=*addr*	Generate program file for a system where the highest available memory address is *addr*. If you do not use this switch, the program file will be generated with the assumption that the highest available address is that of the current system. *Addr* must be octal.
/SA	Generate a stand-alone program: load the impure area at location 400_8 and the pure area at the top of memory. Also, do not put the usual MP/OS header data in the program file; do not search MSL.LB (same as /N); and assume that the highest memory address (/MTOP) is 077777_8.
/SYS	Generate an MP/OS system file. (For more information on system generation, see *MP/OS Assembly Language Programmer's Reference*).
/REV=*value*	Set the program's revision number to the value specified. The number may contain a dot (.) to separate the major and minor revision numbers.
/TASKS= *number*	Specify the maximum number of tasks that the program may have active at one time.

CLI BINDER COMMAND SWITCHES

Figure 4-130

RESULT

1

```
) XEQ BIND ASMPROG MSL.LB )
```

ASMPROG.PR ⟶ On Disc

Errors on Console

2

```
) XEQ BIND FORTPROG FORT4.LB )
```

FORTPROG.PR ⟶ On Disc

Errors on Console

3

```
) XEQ BIND PASCALPROG PASCAL.LB )
```

PASCALPROG.PR ⟶ On Disc

Errors on Console

LIBRARY FILES IN THE BIND LINE

Figure 4-131

RESULT

1

```
) XEQ BIND/E=PROG.ER PROG LIBE.LB ↓
```

PROG.PR
PROG.ER
} On Disc

Errors on Console

/E = file . . . make file the error file.

2

```
) XEQ BIND/L=PROGMAP PROG LIBE.LB ↓
```

PROG.PR
PROGMAP
} On Disc

Errors on Console

/L = filename . . . make filename the listing file on disc.

3

```
) XEQ BIND/ALPHA/L=PROGMAP PROG LIBE.LB ↓
```

PROG.PR
PROGMAP
} On Disc

Errors on Disc

/ALPHA . . . list symbols in alphabetical order in load map.

Figure 4-132

RESULT

1

```
) XEQ BIND/NUMERIC/L=PROGMAP PROG LIBE.LB ↓
```

PROG.PR
PROGMAP
} On Disc

Errors on Console

/NUMERIC . . . list symbols by numeric (address)

2

```
) XEQ BIND/P=PSEUDO PROG LIBE.LB ↓
```

PSEUDO.PR On Disc

Errors on Console

/P=filename . . . name the executable program file, filename.

3

```
) XEQ BIND/REV=1.0 PROG LIBE.LB ↓
```

PROG.PR On Disc

Errors on Console

/REV=number . . . assign a revision number to the program file.

Figure 4-133

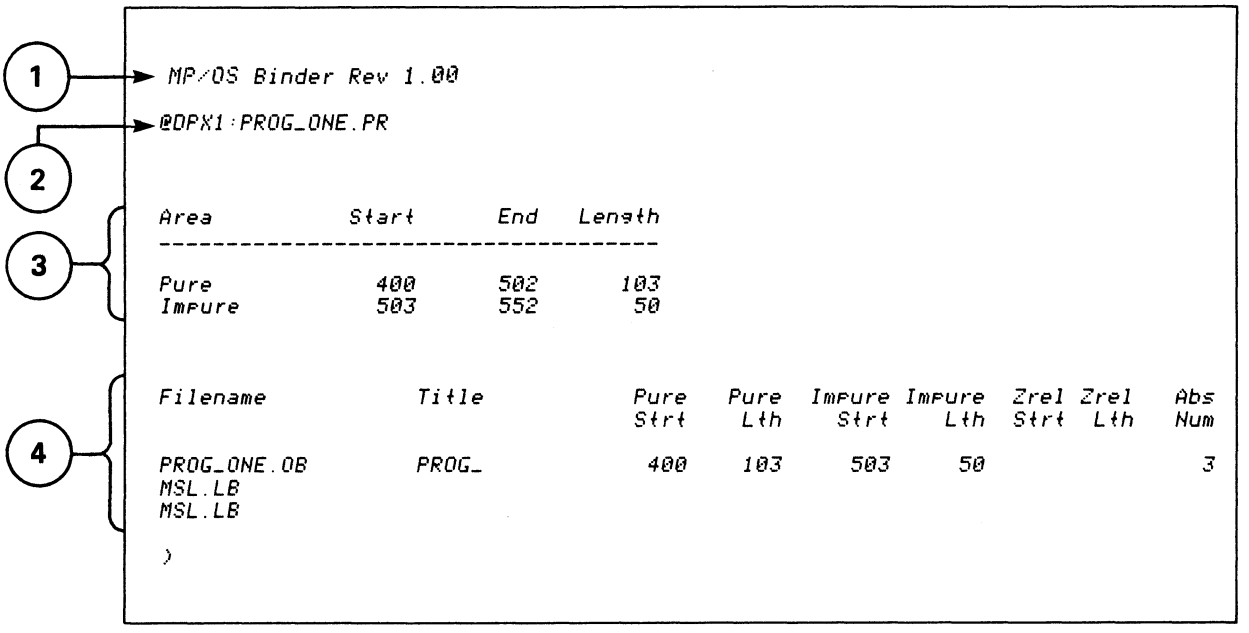
	RESULT	
`) XEQ BIND/D PROG LIBE.LB )`	PROG.PR	On Disc
	Errors on Console	

/D . . . include Debugger and symbol table.

	RESULT	
`) XEQ BIND/DN PROG LIBE.LB )`	PROG.PR	On Disc
	Errors on Console	

/DN . . . include Debugger, exclude symbol table.

Figure 4-134



SAMPLE BINDER LOAD MAP

Figure 4-135

TOPICS

- BINDING OBJECT FILES INTO PROGRAM FILES
- BINDER COMMAND LINES
- FUNCTION SWITCHES
- ERROR COUNT
- LOAD MAP

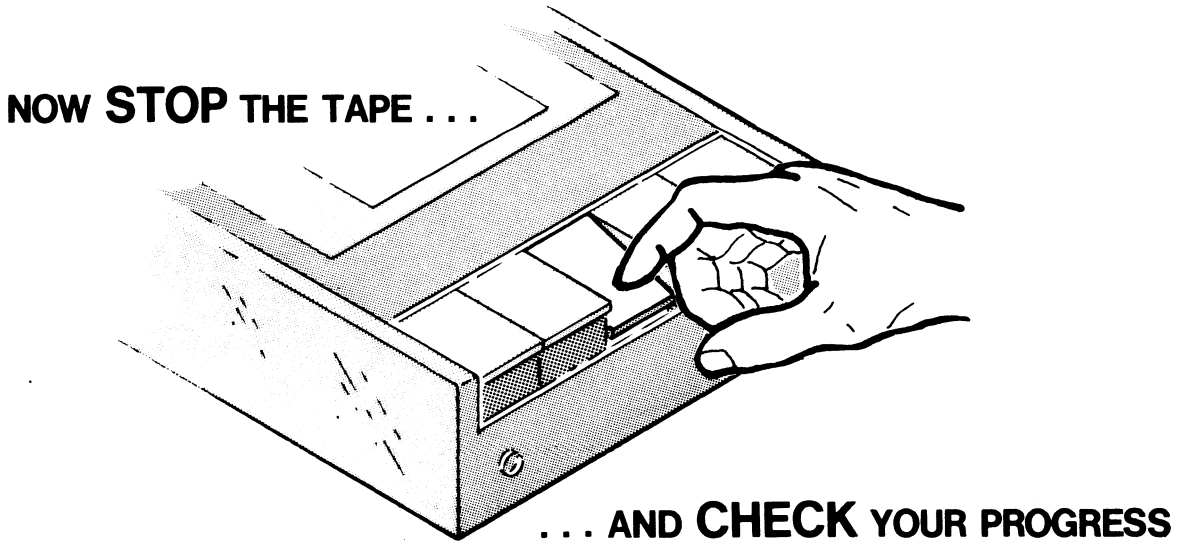


Figure 4-136

BINDER QUIZ

The questions in this quiz apply to MP/Pascal, MP/Fortran, and assembly languages.

Write the answers in the space provided.

Given the following CLI BINDER command line, identify the switches and arguments and state the purpose of the switches and arguments:

) XEQ BIND/ALPHA/L=TEST.MAP/E=TEST.ER TEST XXX.LB

(Note: XXX.LB refers to the library file for your language.)

1. Switches: _____

2. Arguments: _____

3. Purpose of the switches: _____

4. Purpose of the Arguments: _____

List at least three sources of information on BINDER commands:

5. _____

6. _____

7. _____

Given the following CLI BINDER command lines, briefly describe the anticipated result: (Note that the library file has been blank-checked as XXX.LB. Just imagine that it is the library file for your language.)

```
) XEQ BIND TEST XXX.LB
```

8. _____

_____

_____

```
) XEQ BIND TEST.OB XXX.LB
```

9. _____

_____

_____

```
) XEQ BIND/L=TEST.LS TEST XXX.LB
```

10. _____

_____

_____

```
) XEQ BIND/REV=1.5/P=TWO TEST XXX.LB
```

11. _____

_____

_____

CHECK YOUR ANSWERS ON THE FOLLOWING PAGES.

BINDER QUIZ

ANSWERS

The switches, arguments, and purposes of the binder line are as follows:

) XEQ BIND/ALPHA/L=TEST.MPA/E=TEST.ER TEST XXX.LB

1. Switches: /ALPHA
   /L=TEST.MAP
   /E=TEST.ER

2. Arguments: TEST
   XXX.LB (The pseudo library)

3. Purpose of the switches: /ALPHA sorts the map symbols alphabetically and forces their inclusion in the printout.
   /L=TEST.MAP directs the load map to be stored on disc under the name TEST.MAP.
   /E=TEST.ER directs error messages to be stored under TEST.ER on disc.
   TEST.ER is created whether or not there are any errors.

4. Purpose of the arguments: TEST is the name of the object file to be processed by the BINDER. XXX.LB is the name of the library file to be searched for the appropriate routines. (Substitute PASCAL.LB or MSL.LB or FORT4.LB for XXX.LB).

Sources of information on BINDER commands include the following:

5. MP/OS Utilities Reference Manual 093-40002

6. CLI HELP command (if your system is so equipped).

7. This Self-Study course.
   Also: The MP/Pascal Self-Study Course.
   The Programmer's Reference Manual for each language.

Given the following CLI BINDER command lines, briefly describe the anticipated result:

```
) XEQ  BIND  TEST  XXX.LB
```

8. The BINDER searches for TEST.OB and then TEST. A successful seek and bind produces TEST.PR as the program file. The binder will search XXX.LB library for the routines referenced in TEST. No listing (load map). Errors and warning messages default to the console.

```
) XEQ  BIND  TEST.OB  XXX.LB
```

9. Identical result as in #8. TEST.PR is the program file. No load map is produced. Errors default to the console. The Binder searches only for TEST.OB.

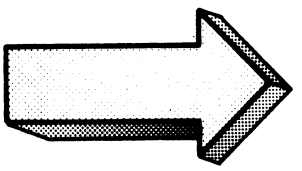
```
) XEQ  BIND/L=TEST.LS  TEST  XXX.LB
```

10. Again, similar to #8. The load map is stored on disc under TEST.LS. TEST.PR is the program file. Messages default to the console. (Errors are also stored in the load map).

```
) XEQ  BIND/REV=1.5/P=TWO  TEST  XXX.LB
```

11. TEST is bound with the requested routines from XXX.LB to produce the program file TWO.PR. TWO's revision number is 1.05. No map. Errors to the console.

A SCORE OF 9 CORRECT ANSWERS OUT OF THE 11 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.



BINDER

LAB EXERCISE

Once again, this lab is designed for your active participation and it can be completed whether or not you have access to a functioning system.

1. First you will need a series of files for completing the lab (skip this requirement if you are going to "paper and pencil" the lab and not perform the exercises on a system):

   BINDER.PR . . . . . .Binder program
   BINDER.OL . . . . . .Binder overlays
   MSL.LB. . . . . .The assembly language library
   (required by all three languages)

2. Use the CLI FILESTATUS command to be sure that you have these files. A note about MSL.LB: even though the Fortran and Pascal bind lines do not specify MSL.LB, it is still searched and used by the binder in these cases. Look for it in the load map.

3. It is also worthwhile to set your SEARCHLIST to be sure that these files are accessible to your directory.

At this point the lab branches to accommodate the different languages. This is done for the sake of accuracy and to avoid confusion of those minor details that usually become major obstacles down the road. Turn to the section of this lab exercise appropriate for your language.

IF YOU ARE PROGRAMMING IN ASSEMBLY LANGUAGE, THIS IS YOUR SECTION.

The assembly language exercises were executed on a system configured with dual diskettes and a hard-copy Dasher terminal. Make the necessary adjustments for your system.

1. SPEED in PROG_ONE (if it is not left over from previous lab exercises.) A copy of PROG_ONE is shown on the following page.

```
) TYPE PROG_ONE)
        .TITL   PROG_ONE;TITLE IDENTIFIER
        .ENT    START   ;ENTRY POINT
        .NREL   1       ;PURE CODE

        ;MAIN ROUTINE

START:  LDA     0,PATH  ;BYTEPOINTER TO PATHNAME
        ?OPEN           ;OPEN CHANNEL TO @TTO
        JMP     ERTN    ;ERROR ON OPEN...CLI
        STA     0,CHAN  ;SAVE CHANNEL #
LOOP:   LDA     0,CHAN  ;IN CASE IT WANDERED
        LDA     1,STPTR ;BYTE POINTER TO STAR LINE
        LDA     2,36    ;36 OCTAL BYTE MSG

AUTHR:  ?WRITE  DS      ;DATA SENSITIVE OPTION
        JMP     ERTN    ;ERROR ON WRITE ..TO CLI
        DSZ     TWEN    ;LOOP 20 TIMES
        JMP     AUTHR   ;WRITE AGAIN, UNTIL 0

MESSG:  LDA     1,MPTR
        LDA     2,55    ;MSG IS 55 OCTAL BYTES
        ?WRITE  DS      ;WRITE 'MP/OS...' MESSAGE
        JMP     ERTN    ;ERROR ON WRITE
CLEAN:  LDA     0,CHAN  ;PREP FOR CLOSE
        ?CLOSE          ;CLOSE CHANNEL TO @TTO
        JMP     ERTN    ;ERROR ON ?CLOSE
        SUB     0,0     ; CLEAR ACS FOR CLEAN RETURN
        SUB     1,1     ;... AC1 HAS ERROR LENGTH
        SUB     2,2     ;... AC2 HAS POINTER TO ERROR

ERTN:   ?RETURN         ;TO CLI, ERROR RETURNS KEEP
                        ;   CODES IN ACS

        ;DATA AREAS

CHAN:   0               ;CHANNEL # SAVE AREA
PATH:   .+1#2
        .TXT    /@TTO/  ;OUTPUT TO CONSOLE
STPTR:  .+1#2
        .TXT    /                       #<12>/
TWEN:   24              ;24 OCTAL = 20 DEC
MPTR:   .+1#2
        .TXT    /                   THE  MP/OS HAS LANDED<12>/
        ;THE  MP/OS STACK

        .NREL   0       ;IMPURE CODE
STK:    .BLK    50      ;RESERVE 50 WORDS
        .LOC    40      ;LOC 40 HAS ...
        STK             ;... THE STACK POINTER
        STK             ;LOCATION 41
        STK+45          ;LOC 42 HAS STACK LIMIT

        .END    START   ;END OF ASM INPUT

)
```

2. Assemble PROG_ONE and get a listing. This was done in previous lab exercises but the command line is given here for convenience:

```
) XEQ MASM/L=PROG.LS PROG_ONE)

)
```

3. Get a copy of PROG.LS and check it for stray errors.

```
) TYPE PROG.LS
 0001 PROG_ .MP/OS  ASSEMBLER REV 00.05          07:27:79 21/37/13
                          .TITL   PROG_ONE;TITLE IDENTIFIER
02                        .ENT    START   ;ENTRY POINT
03        000001          .NREL   1       ;PURE CODE
04
05                        ;MAIN ROUTINE
06
07 00000!020434 START:    LDA     0,PATH  ;BYTEPOINTER TO PATHNAME
08                        ?OPEN           ;OPEN CHANNEL TO @TTO
09 00003!000426           JMP     ERTN    ;ERROR ON OPEN...CLI
10 00004!040427           STA     0,CHAN  ;SAVE CHANNEL #
11 00005!020426 LOOP:     LDA     0,CHAN  ;IN CASE IT WANDERED
12 00006!024432           LDA     1,STPTR ;BYTE POINTER TO STAR LINE
13 00007!030036           LDA     2,36    ;36 OCTAL BYTE MSG
14
15                AUTHR:  ?WRITE  DS      ;DATA SENSITIVE OPTION
16 00012!000417           JMP     ERTN    ;ERROR ON WRITE ..TO CLI
17 00013!014442           DSZ     TWEN    ;LOOP 20 TIMES
18 00014!000774           JMP     AUTHR   ;WRITE AGAIN, UNTIL 0
19
20 00015!024441 MESSG:    LDA     1,MPTR
21 00016!030055           LDA     2,55    ;MSG IS 55 OCTAL BYTES
22                        ?WRITE  DS      ;WRITE 'MP/OS!...' MESSAGE
23 00021!000410           JMP     ERTN    ;ERROR ON WRITE
24 00022!020411 CLEAN:    LDA     0,CHAN  ;PREP FOR CLOSE
25                        ?CLOSE          ;CLOSE CHANNEL TO @TTO
26 00025!000404           JMP     ERTN    ;ERROR ON ?CLOSE
27 00026!102400           SUB     0,0     ; CLEAR ACS FOR CLEAN RETURN
28 00027!126400           SUB     1,1     ;... AC1 HAS ERROR LENGTH
29 00030!152400           SUB     2,2     ;... AC2 HAS POINTER TO ERROR
30
31                ERTN:   ?RETURN         ;TO CLI, ERROR RETURNS KEEP
32                                        ;   CODES IN ACS
33
34                        ;DATA AREAS
35
36 00033!000000 CHAN:     0               ;CHANNEL # SAVE AREA
37 00034!000072&PATH:     .+1*2
38 00035!040124           .TXT    /@TTO/  ;OUTPUT TO CONSOLE
39        052117
40        000000
41 00040!000102&STPTR:    .+1*2
42 00041!020040           .TXT    /                       *<12>/
43        020040
44        020040
45        020040
```

(Continued)

```
46        020040
47        020040
48        020040
49        020040
50        020040
51        020040
52        020052
53        005000
54 00055!000024 TWEN:    24                ;24 OCTAL = 20 DEC
55 00056!000136&MPTR:    .+1#2
56 00057!020040          .TXT    /                  THE  MP/OS  HAS LANDED<12>/
57        020040
58        020040
59        020040
60        020040
 0002 PROG_
01        020040
02        020040
03        020040
04        052110
05        042440
06        046511
07        041522
08        047516
09        020110
10        040523
11        020114
12        040516
13        042105
14        042012
15        000000
16                         ;THE  MP/OS  STACK
17
18        000000           .NREL   0         ;IMPURE CODE
19 00000'000050 STK:       .BLK    50        ;RESERVE 50 WORDS
20        000040           .LOC    40        ;LOC 40 HAS ...
21 00040 000000'           STK               ;... THE STACK POINTER
22 00041 000000'           STK               ;LOCATION 41
23 00042 000045'           STK+45            ;LOC 42 HAS STACK LIMIT
24
25                         .END    START     ;END OF ASM INPUT
26

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS  0003 PROG_

AUTHR 000010!        1/15#   1/18
CHAN  000033!        1/10    1/11    1/24    1/36#
CLEAN 000022!        1/24#
ERTN  000031!        1/09    1/16    1/23    1/26    1/31#
LOOP  000005!        1/11#
MESSG 000015!        1/20#
MPTR  000056!        1/20    1/55#
PATH  000034!        1/07    1/37#
START 000000! EN     1/02    1/07#   2/25
STK   000000'        2/19#   2/21    2/22    2/23
STPTR 000040!        1/12    1/41#
TWEN  000055!        1/17    1/54#
?CLOS 002203! MC     1/25
?I    000013         1/09#   1/16#   1/23#   1/26#   1/32#
?J    000000         1/09#   1/16#   1/23#   1/26#   1/32#
?K    000002         1/16#   1/23#
?OPEN 001743! MC     1/08
?RETU 000203! MC     1/31
?SYSE 000001$ XD     1/09    1/16    1/23    1/26    1/32
?WRIT 002143! MC     1/15    1/22

)
```

You are now ready for the BIND lab exercise. Remember:

1. cover the answer;
2. read the question;
3. write the answer
4. check the answer
5. perform the operation on your system.

1. Bind PROG_ONE into an executable program file. Write your commands and the anticipated responses in the space below *before* checking the answers and before entering any commands on your system:

_____

_____

_____

_____

_____

****

```
) KEQ BIND PROG_ONE MSL.LB)

)
```

Now do it on your system.

If you fail to include MSL.LB you will get a series of binder errors and unresolved external references. This is fun if you have a Dasher CRT console, but tedious with a Dasher printer terminal. You also fail to get an executable program file.

2. Which files associated with PROG_ONE now exist on your system? Show the command and anticipated responses:

****

```
) FILES PROG+)
DIRECTORY @DPX1:

  PROG_ONE              TXT    18-JUN-79  11:48:02        1293
  PROG_ONE.BU           TXT    18-JUN-79  11:49:51        1293
  PROG.LS               TXT    27-JUL-79  21:38:43        3807
  PROG_ONE.PR           PRG    27-JUL-79  21:43:44        1024
  PROG_ONE.OB           OBF    27-JUL-79  21:38:05         312
)
```

Note that we created a backup copy of PROG_ONE titled PROG_ONE.BU. The executable program is named PROG_ONE.PR.

Now do it on your system.

If you fail to get a .PR file then the bind did not succeed. Go back, try to find the error, and try again.

3. To execute PROG_ONE type the following command and watch the result:

```
 ) XEQ PROG_ONE)
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                    THE  MP/OS HAS LANDED
```

Aren't you impressed? Don't forget the XEQ.

Do it on your system and then delete PROG_ONE.PR to avoid confusion with later exercises.

```
) DELETE/V/C PROG_ONE.PR)
PROG_ONE.PR? YES)
Deleted PROG_ONE.PR
)
```

4. Bind PROG_ONE and make PROGERRS the error file. Write your command and anticipated responses below:

_____

_____

_____

_____

****

```
) XEQ BIND/E=PROGERRS PROG_ONE MSL.LB)
)
```

The /E=PROGERRS keyword switch creates PROGERRS as the error file on disc.

Now do it on your system.

The /E=PROGERRS sets up an error file on disc whether or not there are any errors.

5. Which files associated with PROG_ONE now exist on your system. Show the command and anticipated responses. (Get their lengths.)

_____

_____

_____

_____

_____

_____

_____

_____

****

```
) FI/LEN/TYP PROG+)
DIRECTORY @DPX1:

  PROG_ONE            TXT          1293
  PROG_ONE.BU         TXT          1293
  PROG.LS             TXT          3807
  PROGERRS            TXT             0
  PROG_ONE.PR         PRG          1024
  PROG_ONE.OB         OBF           312
)
```

The /LEN switch displays the file's byte length.
The /TYP switch displays the type.

Do it on your system.

Note that since there are no errors, PROGERRS has a length of 0. The executable program file is PROG_ONE.PR.

Type PROGERRS (you should get nothing) and then delete it:

```
) TYPE PROGERRS)

) DEL/V PROGERRS)
Deleted PROGERRS
)
```

6. Now for a load map. Bind PROG_ONE and make PROGMAP the binder load map. Show the command and anticipated responses:

_____

_____

_____

_____

_____

_____

****

```
) XEQ BIND/L=PROGMAP PROG_ONE MSL.LB↓

)
```

The /L=PROGMAP switch assigns PROGMAP as the disc file (or diskette file) with the load map.

Do it on your system.

Check the lengths of the PROG files (especially the map):

```
) FI/LEN PROG+↓
DIRECTORY @DPX1:

  PROG_ONE                    1293
  PROG_ONE.BU                 1293
  PROG.LS                     3807
  PROGMAP                      611
  PROG_ONE.PR                 1024
  PROG_ONE.OB                  312
)
```

7. Get a hard-copy printout of PROGMAP. *Briefly describe* what you expect it to contain:

_____

_____

_____

_____

_____

****

```
) TYPE PROGMAP↓

MP/OS Binder Rev 0.04

@DPX1:PROG_ONE.PR


Area          Start      End   Length
-------------------------------------

Pure            400      502      103
Impure          503      552       50


Filename          Title          Pure   Pure  Impure Impure  Zrel Zrel   Abs
                                 Strt    Lth    Strt    Lth  Strt  Lth   Num

PROG_ONE.OB       PROG_           400    103     503     50                 3
MSL.LB
MSL.LB

)
```

Do it on your system.

Note that your addresses may differ a bit.

Save the printout. Delete PROGMAP and PROG_ONE.PR:

```
) DELETE/V/C PROGMAP PROG_ONE.PR↓
PROGMAP? YES↓
Deleted PROGMAP
PROG_ONE.PR? YES↓
Deleted PROG_ONE.PR
)
```

8. Now let's try a variation of the load map. Bind PROG_ONE and get a map (PROGMAP) with the symbols included in alphabetical order. Show the command and anticipated result in the space below:

_____

_____

_____

_____

****

```
) XEQ BIND/ALPHA/L=PROGMAP PROG_ONE MSL.LB↓

)
```

The /ALPHA switch, in conjunction with the /L=PROGMAP switch, will produce an inclusive, alphabetized load map.

Do it on your system.

Check the length of your files this time:

```
) FI/S/LEN/TYP PROG+↓
DIRECTORY @DPX1:

  PROG.LS              TXT            3807
  PROGMAP              TXT             852
  PROG_ONE             TXT            1293
  PROG_ONE.BU          TXT            1293
  PROG_ONE.OB          OBF             312
  PROG_ONE.PR          PRG            1024
)
```

Compare the length of PROGMAP with previous PROGMAPS. Longer? (It should be.)

9. Get a copy of this version of PROGMAP (from #8). *Briefly describe* the expected difference between this PROGMAP (/ALPHA) and the previous PROGMAP (/ALPHA):

_____

_____

_____

****

```
) TYPE PROGMAP)

 MP/OS  Binder Rev 0.04

@DPX1:PROG_ONE.PR


Area            Start       End    Length
-------------------------------------

Pure              400       502       103
Impure            503       552        50


Filename             Title             Pure    Pure  Impure Impure  Zrel Zrel   Abs
                                       Strt     Lth    Strt    Lth  Strt  Lth   Num

PROG_ONE.OB          PROG_              400     103     503     50                 3
MSL.LB
MSL.LB
?ERTN        000011          ?NRTN        000010          ?SYSE        000017
?ICAL        000012          ?STBL        000045          START        000400
?LDBL        000044          ?SVLO        000043

)
```

Do it on your system.

Note that the alphabetized listing goes down column one, then to column two, and so on.

If you would like, you can try the /NUMERIC switch in place of /ALPHA:

) XEQ BIND/NUMERIC/L=PROGMAP PROG_ONE MSL.LB

The only difference is that the symbols are sorted by address. DELETE PROGMAP and *save* PROG_ONE.PR

```
) DELETE PROGMAP )
)
```

10. This time bind PROG_ONE and make MESSAGE the executable program file. Show the command and expected response:

_____

_____

_____

_____

_____

****

```
) XEQ BIND/P=MESSAGE PROG_ONE MSL.LB )
)
```

The /P=MESSAGE switch sets up MESSAGE.PR as the executable program file.

Do it on your system.

Note that if MESSAGE already exists, the system deletes it and creates a new one with its contents the executable program file from this bind.

11. Which files associated with PROG and MESSAGE now exist on your system? Are the MESSAGE and PROG_ONE program files the same? Show all entries:

****

```
) FI/AS/S PROG+ MESSAGE+)
DIRECTORY @DPX1:

  MESSAGE.PR          PRG   27-JUL-79  22:07:16        1024
  PROG.LS             TXT   27-JUL-79  21:38:43        3807
  PROG_ONE            TXT   18-JUN-79  11:48:02        1293
  PROG_ONE.BU         TXT   18-JUN-79  11:49:51        1293
  PROG_ONE.OB         OBF   27-JUL-79  21:38:05         312
  PROG_ONE.PR         PRG   27-JUL-79  21:59:45        1024
)
```

Do it on your system.

You should now have MESSAGE.PR added to your disc device in the working directory.

12. Execute MESSAGE. Is it the same as PROG_ONE.PR?

****

```
) XEQ MESSAGE )
                    *
                    *
                    *
                    *
                    *
                    *
                    *
                    *
                    *
                    *
                    *
                    *
                    *
                    *
                    *
                    *
                    *
                    *
                    *
                    *
                    *
                THE  MP/OS  HAS LANDED
)
```

Do it!

The results should be identical. The only functional difference is the filename.

13. Last one. This time bind PROG_ONE and give it a revision number of 3.2. Show your entry before executing:

_____

_____

_____

_____

_____

****

```
) XEQ BIND/REV=3.2 PROG_ONE MSL.LB⟩

)
```

Do it.

Use the CLI REV command to determine its success:

```
) REVISION PROG_ONE.PR⟩
03.02
) REV PROG_ONE.PR⟩
03.02
) REV PROG_ONE⟩
Error: Illegal file type
REV,PROG_ONE
) REV PROG_ONE.PR⟩
03.02
)
```

Don't forget the .PR extension!! Also, REVISION may be shortened to REV (as shown above).

Note that we did not delete PROG_ONE.PR before binding another PROG_ONE.PR with this exercise. In this case, the system deleted the old PROG_ONE.PR and created a new one.

You may wish to experiment with the other command switches. If so, this is the time to do it.

Some points to research are:

How does the /D switch affect the load map? (Check the impure area's allocation and note the inclusion of the symbol table. We use this switch in Module Five.

How do the /D and /DN switches affect the program length? Is the program file longer?

What affect does the /SA switch have? What is the executable program file like? The load map?

For the Assembly Language programmer, this concludes the Binder Lab Exercise and also concludes Module Four. At this point you should be able to develop a given source language program for execution under MP/OS.

ON TO MODULE FIVE

IF YOU ARE PROGRAMMING IN FORTRAN4 THIS IS YOUR SECTION.

This section is for the Fortran programmer. Experts in other languages may skip this section.

The FORTRAN4 binder exercises were executed on a system configured with dual diskettes and a hard-copy Dasher terminal.

1. SPEED in MONEY (if it is not left over from previous lab exercises).

```
) TYPE MONEY.FR )
C       *** TITLE INCOME ***
        ACCEPT "ENTER YOUR WEEKLY SALARY", SALARY
        ACCEPT"ENTER YOUR AGE", AGE
        YERLY = SALARY * 52
        TOTAL = YERLY * (65 - AGE)
        TYPE "TOTAL INCOME FROM" ,AGE, "TO RETIREMENT AT 65 IS $ ",TOTAL
        STOP
        END

)
```

Also make sure that you have FORT4.LB, the Fortran Library.

2. Compile MONEY and get a listing. This was done in previous lab exercises but the command lines are given here for convenience:

```
) XEQ FORT4/L=MONEY.LS MONEY )

)
```

3. Get a copy of MONEY.LS and check it for stray errors.

```
) TYPE MONEY.LS↓
; DGC FORTRAN IV REV 05.20IS

;C      *** TITLE INCOME ***
;       ACCEPT "ENTER YOUR WEEKLY SALARY", SALARY
;       ACCEPT"ENTER YOUR AGE", AGE
;       YERLY = SALARY * 52
;       TOTAL = YERLY * (65 - AGE)
;       TYPE "TOTAL INCOME FROM" ,AGE, "TO RETIREMENT AT 65 IS $ ",TOTAL
;       STOP
;       END

)
```

4. Get a copy of MONEY.SR, the Fortran compiler's translation of MONEY.FR, and check it for errors.

```
) TYPE MONEY.SR↓

; DGC FORTRAN IV REV 05.20IS


;C      *** TITLE INCOME ***

;       ACCEPT "ENTER YOUR WEEKLY SALARY", SALARY
        .TITL   .MAIN
        .ENT    .MAIN
        .NREL   1
        .TXTM   1
        .EXTU
        .EXTN   .I
        .CSIZ   2
        FS.
.MAIN:
        JMP     @.+1
        L1.
L1.:

        JSR     @.FREA
        .C1
        0
        6
        .TXT    "ENTER YOUR WEEKLY SALARY"
        0
        2
        U.+0            ;SALARY
        5
```

(Continued)

```
;       ACCEPT"ENTER YOUR AGE", AGE
        JSR     @.FREA
        .C1
        0
        6
        .TXT    "ENTER YOUR AGE"
        0
        2
        V.+2            ;AGE
        5

;       YERLY = SALARY * 52
        FXFL1
        .C2
        FFLD1
        V.+0            ;SALARY
        FML1
        FFST1
        V.+4            ;YERLY

;       TOTAL = YERLY * (65 - AGE)
        FXFL1
        .C3
        FFLD1
        V.+2            ;AGE
        FSB1
        FFLD1
        V.+4            ;YERLY
        FML1
        FFST1
        V.+6            ;TOTAL

;       TYPE "TOTAL INCOME FROM" ,AGE, "TO RETIREMENT AT 65 IS $ ",TOTAL
        JSR     @.FWRI
        .C4
        0
        6
        .TXT    "TOTAL INCOME FROM"
        0
        2
        V.+2            ;AGE
        6
        .TXT    "TO RETIREMENT AT 65 IS $ "
        0
        2
        V.+6            ;TOTAL
        5

;       STOP
        JSR     @.STOP
        .TXT    ""

;       END
        JSR     @.FRET
.C4:    000012
.C3:    000101
.C2:    000064
.C1:    000013

        FS.=10
        SFS.=0
        T.=-167
        V.=200+T.
        TS.=T.+7
        FTS.=T.+0
        VS.=V.+7
        FVS.=V.+0
        .END

)
```

5. Assemble MONEY.SR, get an assembly listing, and check it for errors.

```
) XEQ MASM/L=MONMASM.LS/PS=FORT4.PS MONEY)

)
```

```
) TYPE MONMASM.LS
 0001 .MAIN  MP/OS ASSEMBLER REV 00.05          07:28:79 13/32/55
01
02                ; DGC FORTRAN IV REV 05.20IS
03
04
05
06                ; C     *** TITLE INCOME ***
07
08                ;       ACCEPT "ENTER YOUR WEEKLY SALARY", SALARY
                          .TITL   .MAIN
10                        .ENT    .MAIN
11        000001          .NREL   1
12        000001          .TXTM   1
13                        .EXTU
14                        .EXTN   .I
15        000002          .CSIZ   2
16 00000!000010           FS.
17                 .MAIN:
18 00001!002401           JMP     @.+1
19 00002!000003!          L1.
20                 L1.:
21
22 00003!006000$          JSR     @.FREA
23 00004!000141!          .C1
24 00005!000000           0
25 00006!000006           6
26 00007!042516           .TXT    "ENTER YOUR WEEKLY SALARY"
27        052105
28        051040
29        054517
30        052522
31        020127
32        042505
33        045514
34        054440
35        051501
36        046101
37        051131
38        000000
39 00024!000000           0
40 00025!000002           2
41 00026!000011           V.+0            ;SALARY
42 00027!000005           5
43
44                ;       ACCEPT"ENTER YOUR AGE", AGE
45 00030!006000$          JSR     @.FREA
46 00031!000141!          .C1
47 00032!000000           0
```

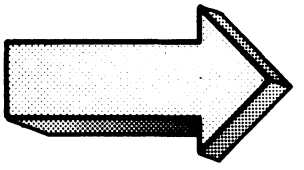
(Continued)

```
48 00033!000006          6
49 00034!042516          .TXT    "ENTER YOUR AGE"
50       052105
51       051040
52       054517
53       052522
54       020101
55       043505
56       000000
57 00044!000000          0
58 00045!000002          2
59 00046!000013          V.+2            ;AGE
60 00047!000005          5
 0002 .MAIN
01
02                ;      YERLY = SALARY * 52
03 00050!000000$         FXFL1
04 00051!000140!         .C2
05 00052!000000$         FFLD1
06 00053!000011          V.+0            ;SALARY
07 00054!000000$         FML1
08 00055!000000$         FFST1
09 00056!000015          V.+4            ;YERLY
10
11                ;      TOTAL = YERLY * (65 - AGE)
12 00057!000000$         FXFL1
13 00060!000137!         .C3
14 00061!000000$         FFLD1
15 00062!000013          V.+2            ;AGE
16 00063!000000$         FSB1
17 00064!000000$         FFLD1
18 00065!000015          V.+4            ;YERLY
19 00066!000000$         FML1
20 00067!000000$         FFST1
21 00070!000017          V.+6            ;TOTAL
22
23                ;      TYPE "TOTAL INCOME FROM" ,AGE, "TO RETIREMENT AT 65 IS
$ ",TOTAL
24 00071!006000$         JSR     @.FWRI
25 00072!000136!         .C4
26 00073!000000          0
27 00074!000006          6
28 00075!052117          .TXT    "TOTAL INCOME FROM"
29       052101
30       046040
31       044516
32       041517
33       046505
34       020106
35       051117
36       046400
37 00106!000000          0
38 00107!000002          2
39 00110!000013          V.+2            ;AGE
40 00111!000006          6
41 00112!052117          .TXT    "TO RETIREMENT AT 65 IS $ "
42       020122
43       042524
44       044522
45       042515
46       042516
47       052040
48       040524
49       020066
50       032440
51       044523
52       020044
53       020000
54 00127!000000          0
55 00130!000002          2
56 00131!000017          V.+6            ;TOTAL
57 00132!000005          5
```

(Continued)

```
58
59                      ;       STOP
60 00133!006000$                JSR     @.STOP
 0003 .MAIN
01 00134!000000                 .TXT    ""
02
03                      ;       END
04 00135!006000$                JSR     @.FRET
05 00136!000012 .C4:            000012
06 00137!000101 .C3:            000101
07 00140!000064 .C2:            000064
08 00141!000013 .C1:            000013
09
10        000010                FS.=10
11        000000                SFS.=0
12        177611                T.=-167
13        000011                U.=200+T.
14        177620                TS.=T.+7
15        177611                FTS.=T.+0
16        000020                US.=U.+7
17        000011                FUS.=U.+0
18                              .END
19

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS  0004 .MAIN

FFLD1 000011$ XD    2/05    2/14    2/17
FFST1 000001$ XD    2/08    2/20
FML1  000002$ XD    2/07    2/19
FSB1  000010$ XD    2/16
FS.   000010        1/16    3/10#
FTS.  177611        3/15#
FUS.  000011        3/17#
FXFL1 000005$ XD    2/03    2/12
L1.   000003!       1/19    1/20#
SFS.  000000        3/11#
TS.   177620        3/14#
T.    177611        3/12#   3/13    3/14    3/15
US.   000020        3/16#
U.    000011        1/41    1/59    2/06    2/09    2/15    2/18    2/21
                    2/39    2/56    3/13#   3/16    3/17
.C1   000141!       1/23    1/46    3/08#
.C2   000140!       2/04    3/07#
.C3   000137!       2/13    3/06#
.C4   000136!       2/25    3/05#
.FREA 000007$ XD    1/22    1/45
.FRET 000006$ XD    3/04
.FWRI 000004$ XD    2/24
.I    000012  XN    1/14
.MAIN 000001! EN    1/10    1/17#
.STOP 000003$ XD    2/60

)
```

Once you are satisfied with the preparation of MONEY.OB, then start the Lab Exercise.



1. Bind MONEY into an executable program file. Write your commands and the anticipated responses in the space below *before* checking the answers and before entering any commands on your system:

_____

_____

_____

_____

****

```
) XEQ BIND MONEY FORT4.LB\

)
```

If you fail to include FORT4.LB you will get a series of binder errors and unresolved external references. This is fun if you have a Dasher CRT, but tedious with a Dasher terminal printer. You can try it, but you will not get an executable program file.

Now do it on your system.

If you get any errors, you may have to reinvoke SPEED, edit the file, recompile, reassemble, and rebind.

2. Which files associated with MONEY now exist on your system? (Assuming a successful bind). Show the command and anticipated responses:

_____

_____

_____

_____

_____

_____

_____

****

```
) FI/AS/S MONEY+)
DIRECTORY @DPX1:

  MONEY.FR          TXT    26-JUN-79   9:54:45      222
  MONEY.LS           66    28-JUL-79  13:27:31      268
  MONEY.OB          OBF    28-JUL-79  13:33:31      466
  MONEY.PR          PRG    28-JUL-79  13:42:48     9728
  MONEY.SR           66    28-JUL-79  13:27:32      992
)
```

Now do it on your system.

MONEY.FR is the original source file (from SPEED). MONEY.LS is the compiler listing. MONEY.OB is the object file (from MASM). MONEY.PR is the executable program file (from the binder). MONEY.SR is the source file output by the compiler. MONMASM.LS is the macro-assembler listing.

If you fail to get a .PR file then the bind did not succeed. Go back and try to find the error, then try again before continuing.

3. To execute MONEY, type the following command, answer the questions, and watch the result: (Press <NEW-LINE> after each response).

```
) XEQ MONEY⤶
ENTER YOUR WEEKLY SALARY100⤶
ENTER YOUR AGE21⤶
TOTAL INCOME FROM     0.210000E  2TO RETIREMENT AT 65 IS $
     0.228800E  6
Stop

)
```

You must type the XEQ (or its abbreviation). You do not need the .PR extension on the program file.

Do it on your system.

In our example, a twenty-one year old earning $100 per week will accumulate $228,000 by the time of retirement at age 65. Impressive, isn't it?

Delete MONEY.PR to avoid confusion with later exercises:

```
) DEL/U/C MONEY.PR⤶
MONEY.PR? YES⤶
Deleted MONEY.PR
)
```

4. Bind MONEY and make MONEYERR the error file. Write your command and anticipated responses below:

_____

_____

_____

_____

_____

****

```
) XEQ BIND/E=MONEYERR MONEY FORT4.LB)

)
```

Now do it on your system.

The /E=MONEYERR switch sets up an error file on disc. The file is created whether or not there are any errors.

Now do it on your system.

5. Which files associated with MONEY now exist on your system? Show the command and anticipated responses: (Get their lengths.)

_____

_____

_____

_____

_____

_____

****

```
) FI/LEN/TYP MONEY+ )
DIRECTORY @DPX1:

  MONEY.FR          TXT          222
  MONEY.LS           66          268
  MONEY.SR           66          992
  MONEYERR          TXT            0
  MONEY.PR          PRG         9728
  MONEY.OB          OBF          466
)
```

The /LEN switch shows the byte length. The /TYP switch shows the file types

Do it on your system.

Note that since there are no errors, MONEYERR has a length of 0. The executable program file is MONEY.PR.

TYPE MONEYERR (you should get nothing) and then delete it:

```
) TYPE MONEYERR)

) DEL/V/C MONEYERR)
MONEYERR? YES)
Deleted MONEYERR
)
```

6. Now for a load map. Bind MONEY and make MONEYMAP the binder load map. Show the command and anticipated responses:

_____

_____

_____

_____

****

```
) XEQ BIND/L=MONEYMAP MONEY FORT4.LB↓

)
```

The /L=MONEYMAP switch assigns MONEYMAP as the disc file (or diskette file) with the load map.

Do it on your system.

Check the lengths of the money files (especially the map):

```
) FI/S/LEN MONEY+↓
DIRECTORY @DPX1:

  MONEY.FR                    222
  MONEY.LS                    268
  MONEY.OB                    466
  MONEY.PR                   9728
  MONEY.SR                    992
  MONEYMAP                    648
)
```

Write down the file lengths for comparison with later exercises.

7. Get a hard-copy printout of MONEYMAP. *Briefly* describe what you expect it to contain:

_____

_____

_____

****

```
) TYPE MONEYMAP↓

MP/OS Binder Rev 0.04
@DPX1:MONEY.PR


Area          Start      End   Length
-------------------------------------

Page zero        50      164      115
Pure            400    10262     7663
Impure        10263    11206      724


Filename          Title          Pure   Pure  Impure Impure  Zrel Zrel   Abs
                                 Strt    Lth    Strt    Lth  Strt  Lth   Num

MONEY.OB          .MAIN           400    142
FORT4.LB                          542   7220   10265    722    50  115     3
MSL.LB                           7762    301

)
```

Do it on your system.

Note that your addresses may differ a bit.

Save the printout. DELETE MONEYMAP and MONEY.PR to avoid conflicts with later questions.

```
) DELETE/V/C MONEYMAP MONEY.PR↓
MONEYMAP? YES↓
Deleted MONEYMAP
MONEY.PR? YES↓
Deleted MONEY.PR
)
```

8. Now let's try a variation of the load map. Bind MONEY and get a map (MONEYMAP) with the symbols included in alphabetic order. Show the command and anticipated result in the space below:

_____

_____

_____

_____

****

```
) XEQ BIND/ALPHA/L=MONEYMAP MONEY FORT4.LB↓

)
```

The /ALPHA switch, in conjunction with the /L=MONEYMAP switch, will produce an inclusive, alphabetized load map.

Now do it on your system.

Check the length of your files this time. Compare the lengths with previous binds:

```
) FI/AS/S MONEY+↓
DIRECTORY @DPX1:

  MONEY.FR          TXT    26-JUN-79   9:54:45      222
  MONEY.LS           66    28-JUL-79  13:27:31      268
  MONEY.OB          OBF    28-JUL-79  13:33:31      466
  MONEY.PR          PRG    28-JUL-79  14:07:24     9728
  MONEY.SR           66    28-JUL-79  13:27:32      992
  MONEYMAP          TXT    28-JUL-79  14:07:25     6170
)
```

9. Get a copy of this version of MONEYMAP (from #8). *Briefly* describe the expected difference between this (/ALPHA) and the previous MONEYMAP:

_____

_____

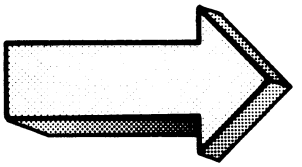
_____

****

```
) TYPE MONEYMAP)

MP/OS  Binder Rev 0.04

@DPX1:MONEY.PR


Area          Start      End   Length
-------------------------------------

Page zero        50      164      115
Pure            400    10262     7663
Impure        10263    11206      724


Filename          Title          Pure   Pure  Impure Impure  Zrel Zrel   Abs
                                 Strt    Lth    Strt    Lth  Strt  Lth   Num

MONEY.OB          .MAIN           400    142
FORT4.LB                          542   7220   10265    722    50  115     3
MSL.LB                           7762    301
.?DLY      010245          ALLOC       005031          FXL         006412
.?ERM      010021          APPEN       005477          I.OCA       010607
.?XQT      007762          ARYSZ       005605          IOPR.       005563
.ALLO      000101          BRD         000755          ISA.E       002140
.ARYS      000112          BWR         000761          ISA.N       002137
.BASC      007534          D?IV        073101          ISAE.       007034
.BDAS      007453          D?IVS       077001          ISAN.       007036
.BRD       000060          DB.E        000115          L?B00       060401
.BWR       000061          DFT.0       005447          L?B01       064401
.FARG      000161          DVD         006164          L?B02       070401
.FARL      000160          EXIT        000733          L?B03       074401
.FCAL      000141          FA          006142          L?B10       060501
.FLSP      007750          FAD1        006117          L?B11       064501
.FLSZ      177777          FARG.       007705          L?B12       070501
.FREA      000056          FARL.       007677          L?B13       074501
.FRET      000144          FB          006136          L?B20       060601
.FRG0      000156          FCALL       006141          L?B21       064601
.FRG1      000157          FCEQ1       006133          L?B22       070601
.FSAV      000143          FCGE1       006131          L?B23       074601
.FSBR      000114          FCGT1       006132          L?B30       060701
.FSUB      000113          FCLE1       006127          L?B31       064701
.FWRI      000057          FCLT1       006130          L?B32       070701
.I         000635          FD          006647          L?B33       074701
.IOPR      000111          FDV1        006122          LDB.        007567
.ISAE      000140          FEQ         006605          M?FF0       060201
```

(CONTINUED)

```
.ISAN   000137   FERT0   000734   M?FF1   064201
.LDBT   000152   FERT1   000734   M?FF2   070201
.MAD    000154   FERTN   000733   M?FF3   074201
.MADO   000155   FFLD1   006115   M?FS0   061201
.MAIN   000401   FFST1   006116   M?FS1   065201
.MEMA   000737   FGE     006601   M?FS2   071201
.MVBC   000134   FGT     006603   M?FS3   075201
.MVBT   000135   FHMA    177777   M?TF0   060001
.NDSP   000052   FL      006013   M?TF1   064001
.OFLO   007162   FL.AT   000123   M?TF2   070001
.OVFL   000053   FLE     006575   M?TF3   074001
.RDFC   000104   FLFX1   006124   M?TS0   061001
.RDFL   000103   FLSP    000674   M?TS1   065001
.READ   000106   FLT     006577   M?TS2   071001
.REDS   000110   FLX     006447   M?TS3   075001
.RTE0   000147   FM      006256   M?UL    073301
.RTER   000146   FML1    006121   M?ULS   077201
.RTES   000150   FNEG1   006126   MAD     007621
.SPSZ   000310   FNG     006540   MADO    007622
.STBT   000153   FQRET   002145   MPY     006163
.STOP   000136   FRCAL   006142   MPY0    006162
.SV0    000055   FREAD   000766   MVBC    006757
.THRE   000102   FRET    002144   MVBT    006753
.WRCH   000151   FRG0    007650   NSP     000051
.WRIT   000105   FRG1    007644   OPEN    005457
.WRTS   000107   FRTSK   177777   P?OP0   061601
?ERTN   000011   FS      006054   P?OP1   065601
?ICAL   000012   FSAV    002143   P?OP2   071601
?LDBL   000044   FSB1    006120   P?OP3   075601
?NMAX   011207   FSBR    005770   P?SH0   061401
?NRTN   000010   FSG     006517   P?SH1   065401
?STBL   000045   FSGN1   006125   P?SH2   071401
?SVLO   000043   FSUBA   005621   P?SH3   075401
?SYSE   000017   FWRIT   000772   P?SO0   061501
AFSE    000054   FXFL1   006123   P?SO1   065501
P?SO2   071501   RTE0    007232   STB.    007602
P?SO3   075501   RTER    007216   STOP    007012
QRSTR   007160   RTESP   007220   THREA   005057
R?ET    062601   S?AV    062401   WRCH    007433
RDFCH   005101   S?AVE   162501   WRITL   005213
RDFLD   005075   SAV0    007047   WRITS   005435
READL   005205   SAV2    007121
REDS    005427   SAV3    007131
RSTR.   007147   SN.L    000115

)
```



Do it on your system.

Note that the alphabetized listing goes down column one, then to column two, and so on.

If you would like, you can try the /NUMERIC switch in place of /ALPHA. The command line is shown below:

```
) XEQ BIND/L=MONEYMAP/NUMERIC MONEY FORT4.LB)
)
```

The only difference between the /ALPHA and /NUMERIC maps is that the symbols are sorted by address. DELETE MONEYMAP and *save* MONEY.PR.

10. This time bind MONEY, make CASH the executable program file, and CASHMAP the listing. Show the command and expected response:

_____

_____

_____

****

```
) XEQ BIND/L=CASHMAP/P=CASH MONEY FORT4.LB )

)
```

The /P=CASH switch sets up CASH.PR as the executable program file. The /L=CASHMAP directs the load map to the =CASHMAP file on disc.

Do it on your system.

11. Which files associated with MONEY and CASH now exist on your system? Are the CASH and MONEY program files the same? Show all entries:

_____

_____

_____

_____

_____

_____

_____

****

```
) FI/LEN/TYP/S MONEY+ CASH+⌇
DIRECTORY @DPX1:

  CASH.PR          PRG        9728
  CASHMAP          TXT         647
  MONEY.FR         TXT         222
  MONEY.LS          66         268
  MONEY.OB         OBF         466
  MONEY.PR         PRG        9728
  MONEY.SR          66         992
  MONEYMAP         TXT        6170
)
```

Do it on your system.

Execute CASH. Is it the same as MONEY? You can use any one of the following four to execute the program file:

```
) X CASH⌇

) XE CASH⌇

) XEQ CASH⌇

) EXECUTE CASH⌇
```

12. Last one. This time bind MONEY and give it a revision number of 9.8. Show your entry before executing:

_____

_____

_____

_____

****

```
) XEQ BIND/REV=9.8 MONEY FORT4.LB↓

)
```

The /REV=number switch assigns a revision number to the program file.

Do it.

Use the CLI REV command to determine its success:

```
) REVISION MONEY.PR↓
09.08
) REV MONEY.PR↓
09.08
) REV MONEY↓
Error: File does not exist
REV,MONEY
) REV MONEY.PR↓
09.08
)
```

As shown above:

Don't forget the .PR extension!!

The REVISION command may be abbreviated.

You have covered the binder and the most frequently used switches. You may wish to experiment with the remaining switches, especially the debugger switches (/D, /DN). Do your experimenting now with the MONEY and CASH programs. Minimal harm is done if you lose these programs.

This concludes the Binder Lab exercise and Module Four for the Fortran Programmer. You should now be able to develop a given source language program for execution under MP/OS. Skip the next section and proceed to Module Five.

IF YOU ARE PROGRAMMING IN PASCAL THIS IS YOUR SECTION.

The Pascal binder exercises were executed on a system configured with a 10 meg disc, Dasher CRT, and Dasher LP2 printer.

Note: This exercise is for the Pascal programmer. Experts in other languages may skip this section.

Directions

1. SPEED in "TWO" (if it is not left over from previous lab exercises.)

```
PROGRAM SMITH_TWO;
INCLUDE IO_CALLS.PAS;
VAR J,ST:INTEGER;
BEGIN
FOR J:= 1 TO 20 DO LINEWRITE(OUCH,'          * <12>',ST);
IF ST <> 0 THEN LINEWRITE(OUCH, 'ERROR ON WRITE',ST);
LINEWRITE (OUCH,'        MICRON HAS LANDED <12>',ST);
IF ST <> 0 THEN LINEWRITE (OUCH, 'ERROR ON LANDING',ST)
END.
```

Also make sure that you have PASCAL.LB and IO_CALLS.PAS on your system.

2. Compile TWO and get a listing (TWO.LS). This was done in the Pascal compilation lab and is repeated here for your convenience:

```
) XEQ PASCAL/L=TWO.LS  TWO )

No Compilation Errors
)
```

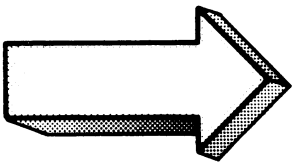
3. Get a copy of TWO.LS and check it for stray errors:

```
MP/PASCAL              REV 00.00              10-JUL-79       12:29:10

    1.   PROGRAM SMITH_TWO;
    2.   INCLUDE IO_CALLS.PAS;
    3.
    4.   { ********************* }
    5.   { PRIMITIVE I/O ROUTINES }
    6.   { ********************* }
    7.
    8.   CONST   MAX_LINE_LTH = 136;
    9.           MAX_PATH_LTH = 128;
   10.
   11.           INCH = 0R8:      {STANDARD INPUT CHANNEL}
   12.           OUCH = 1R8:      {STANDARD OUTPUT CHANNEL}
   13.
   14.           { OPEN OPTIONS}
   15.           EX = 40000R8:    {EXCLUSIVE ACCESS}
   16.           NZ = 10000R8;    {DON'T ZERO BLOCKS ON I/O}
   17.           CR =  4000R8;    {FILE CREATION}
   18.           DE =  2000R8;    {FILE DELETION}
   19.           UC =  1000R8:    {UNCONDITIONAL CREATION}
   20.           AP =   400R8:    {APPEND}
   21.
   22.   TYPE    CHANNEL = 0..15:
   23.           PATHNAME = STRING MAX_PATH_LTH;
   24.           LINE_BUFFER = STRING MAX_LINE_LTH;
   25.           IO_BUFFER = STRING 32767;
   26.           FILE_POSITION = RECORD
   27.                           HIGH: INTEGER:
   28.                           LOW:INTEGER
   29.                         END;
   30.
   31.   EXTERNAL ASSEMBLY PROCEDURE OPENFILE(VAR CHAN: CHANNEL:
   32.                                        FILE: PATHNAME;
   33.                                        OPTIONS: INTEGER:
   34.                                        FILE_TYPE: INTEGER;
   35.                                        ELEM_SIZE: INTEGER;
   36.                                      VAR STATUS: INTEGER);
   37.
   38.   EXTERNAL ASSEMBLY PROCEDURE CLOSEFILE(CHAN: CHANNEL: VAR STATUS:INTEGER);
   39.   EXTERNAL ASSEMBLY PROCEDURE CLDELFILE(CHAN: CHANNEL; VAR STATUS:INTEGER);
   40.
   41.   EXTERNAL ASSEMBLY PROCEDURE LINEREAD(CHAN: CHANNEL:
   42.                                VAR BUFFER: LINE_BUFFER; VAR STATUS: INTEGER);
   43.
   44.   EXTERNAL ASSEMBLY PROCEDURE LINEWRITE(CHAN: CHANNEL; BUFFER: LINE_BUFFER;
   45.                                     VAR STATUS: INTEGER);
   46.
   47.   EXTERNAL ASSEMBLY PROCEDURE CHARREAD(CHAN: CHANNEL:
   48.                                LTH: INTEGER; VAR BUFFER: IO_BUFFER;
   49.                                VAR STATUS: INTEGER);
   50.
   51.   EXTERNAL ASSEMBLY PROCEDURE CHARWRITE(CHAN: CHANNEL; BUFFER: IO_BUFFER;
   52.                                     VAR STATUS: INTEGER);
   53.
   54.   EXTERNAL ASSEMBLY PROCEDURE BYTEREAD(CHAN: CHANNEL:
```

(Continued)

```
55.                                 BUF_ADDRESS: INTEGER; VAR LTH: INTEGER;
56.                                 VAR STATUS: INTEGER);
57.
58.   EXTERNAL ASSEMBLY PROCEDURE BYTEWRITE(CHAN: CHANNEL;
59.                                 BUF_ADDRESS: INTEGER; LTH: INTEGER;
60.                                 VAR STATUS: INTEGER);
61.
62.
63.   EXTERNAL ASSEMBLY PROCEDURE GPOSFILE(CHAN: CHANNEL;
64.                   VAR POSITION: RECAST FILE_POSITION; VAR STATUS:  INTEGER);
65.
66.   EXTERNAL ASSEMBLY PROCEDURE SPOSFILE(CHAN: CHANNEL:
67.                   POSITION: RECAST FILE_POSITION; VAR STATUS: INTEGER);
68.
69.   VAR J,ST:INTEGER;
70.   BEGIN
71.   FOR J:= 1 TO 20 DO LINEWRITE(OUCH,'          * <12>',ST);
72.   IF ST <> 0 THEN LINEWRITE(OUCH, 'ERROR ON WRITE',ST);
73.   LINEWRITE (OUCH,'         MICRON HAS LANDED <12>',ST);
74.   IF ST <> 0 THEN LINEWRITE (OUCH, 'ERROR ON LANDING',ST)
75.   END.
```

Once you are certain that you have the required programs and that TWO is error-free, then you are ready for the Lab Exercise.



1. Bind TWO into an executable program file. Write your commands and the anticipated responses in the space below *before* checking the answers and before entering any commands on your system:

_____

_____

_____

_____

****

```
) XEQ BIND TWO PASCAL.LB)
```

Nothing fancy is required here.

Now do it on your system.

If you fail to include PASCAL.LB, you will get a series of binder errors, unresolved external references, and warnings. You will also fail to get an executable program file.

2. Which files associated with TWO now exist on your system? Show the command and anticipated responses: (assuming a successful bind from question #1).

_____

_____

_____

_____

_____

_____

_____

****

```
FI/AS/S TWO+ )
DIRECTORY @DPD0:PASCAL

  TWO                TXT    1-JAN-00    0:00:41        293
  TWO.LS             UDF   10-JUL-79   12:29:32       3174
  TWO.OB             UDF   10-JUL-79   12:29:32       1140
  TWO.PR             PRG   10-JUL-79   12:34:10       3072
```

The CLI FILESTATUS command displays the requested information. Note the use of the "+" template.

TWO is the original Pascal source file. TWO.LS is the compiler listing. TWO.OB is the object file (from the compiler). TWO.PR is the executable program file. The directory @DPD0:PASCAL is a subdirectory on a 10 MG disc. Make the appropriate adjustments for your system configuration.

Now do it on your system.

If you fail to get a .PR file then the bind did not succeed. Go back and try to find the error. Then try again.

3. To execute TWO, type the following command and watch the result:

```
) XEQ TWO↓
          *
          *
          *
          *
          *
          *
          *
          *
          *
          *
          *
          *
          *
          *
          *
          *
          *
          *
          *
          *
          *
     THE  MP/OS HAS LANDED

)
```

Impressive, isn't it? Make sure you include the CLI XEQ command.

Do it on your system.

Delete TWO.PR to avoid confusion with later exercises.

```
) DEL/V/C TWO.PR↓
TWO.PR? YES↓
Deleted TWO.PR
)
```

4. Bind TWO and make TWOERR the error file. Write your command and anticipated responses below:

_____

_____

_____

_____

_____

_____

****

```
) XEQ BIND/E=TWOERR  TWO  PASCAL.LB )
)
```

The /E=TWOERR switch sets up an error file on disc. TWOERR is created whether or not there are any errors. The file is cumulative. Each additional bind adds to the TWOERR file.

Now do it on your system.

5. Which files associated with TWO now exist on your system. Show the command and anticipated responses. (Get their lengths)

_____

_____

_____

_____

_____

_____

_____

****

```
)FI/AS TWO+)
 DIRECTORY @DPD0:PASCAL

   TWO                TXT    1-JAN-00    0:00:41        293
   TWO.LS             UDF   10-JUL-79   12:29:32       3174
   TWO.OB             UDF   10-JUL-79   12:29:32       1140
   TWO.PR             PRG   10-JUL-79   12:39:30       3072
   TWOERR             TXT   10-JUL-79   12:39:02          0
```

Note that since there are no errors, TWOERR has a length of 0. The executable program file is TWO.PR.

Try it on your system.

TYPE TWOERR (you should get nothing) and then delete it.

```
) TYPE TWOERR )

) DELETE/V TWOERR )
DELETED TWOERR
)
```

6. Now for a load map. Bind TWO and make TWOMAP the binder load map. Show the command and anticipated responses:

_____

_____

_____

_____

_____

****

```
XEQ BIND/L=TWOMAP TWO PASCAL.LB)
```

The /L=TWOMAP switch assigns TWOMAP as the disc file with the load map.

Now do it on your system.

Check the lengths of the "TWO" files (especially the map):

```
FI/LEN TWO+)
DIRECTORY @DPDO:PASCAL

  TWO.OB                    1140
  TWO.LS                    3174
  TWOMAP                    1772
  TWO.PR                    3072
  TWO                        293
```

Make note of the sizes (circle the printout or write down the CRT information) for comparison with later questions.

7. Get a hard-copy printout of TWOMAP. *Briefly* describe what you expect it to contain:

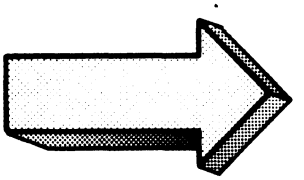
_____

_____

_____

****

```
MP/OS BINDER REV 0.0

@DPD0:PASCAL:TWO.PR


AREA            START       END    LENGTH
--------------------------------------------

PAGE ZERO          50       102        33
PURE              400      2016      1417
IMPURE           2017      2461       443


FILENAME            TITLE            PURE    PURE  IMPURE IMPURE  ZREL ZREL   ABS
                                     STRT     LTH    STRT    LTH  STRT  LTH   NUM

TWO.OB              SMITH_TWO         400     223    2017      2
PASCAL.LB           SYSIO             623     202
                    ?PASC            1025     206    2021    441    50   33     3
                    V?GAD            1233       7
                    V?PGB            1242       7
                    V?RGE            1251      21
                    W?COM            1272      55
                    V?ENT            1347      71
                    V?CSY            1440      15
                    V?POP            1455       5
                    V?FUP            1462      30
                    V?FAL            1512       7
                    V?GIT            1521      21
                    V?JMP            1542      31
MSL.LB              ERMSG            1573     224
                    UNINS
                    SYSEN
```



Do it on your system.

Note that your addresses may differ a bit.

Save the printout. Delete TWOMAP and TWO.PR.

```
) DEL/V/C TWOMAP TWO.PR↓
TWOMAP? YES↓
Deleted TWOMAP
TWO.PR?  YES↓
Deleted TWO.PR
)
```

8. Now let's try a variation of the load map. Bind TWO and get a map (TWOMAP) with the symbols included in alphabetical order. Show the command and anticipated result in the space below:

_____

_____

_____

_____

****

```
) XEQ BIND/ALPHA/L=TWOMAP TWO PASCAL.LB )
```

The /ALPHA switch, in conjunction with the /L=TWOMAP switch, will produce an inclusive, alphabetized load map. You must include the listing switch.

Do it on your system.

Check the length of your files this time (is the map longer?):

```
) FI/LEN TWO+ )
  DIRECTORY @DPDO:PASCAL

     TWO.OB                    1140
     TWO.LS                    3174
     TWOMAP                    4253
     TWO.PR                    3072
     TWO                        293
```

In Question #6, TWOMAP had a length of 1772. What caused the different sizes?

9. Get a copy of this version of TWOMAP (from question #8). Briefly describe the expected difference between this (/ALPHA) version and the previous TWOMAP:

_____

_____

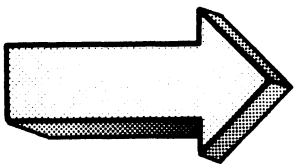
_____

****

```
MP/OS BINDER REV 0.0

@DPDO:PASCAL:TWO.PR


AREA          START      END    LENGTH
--------------------------------------------------------

PAGE ZERO        50      102        33
PURE            400     2016      1417
IMPURE         2017     2461       443


FILENAME          TITLE          PURE    PURE  IMPURE IMPURE  ZREL ZREL   ABS
                                 STRT     LTH    STRT    LTH  STRT  LTH   NUM

TWO.OB            SMITH_TWO       400     223    2017      2
PASCAL.LB         SYSIO           623     202
                  ?PASC          1025     206    2021    441    50   33     3
                  V?GAD          1233       7
                  V?PGB          1242       7
                  V?RGE          1251      21
                  W?COM          1272      55
                  V?ENT          1347      71
                  V?CSY          1440      15
                  V?POP          1455       5
                  V?FUP          1462      30
                  V?FAL          1512       7
                  V?GIT          1521      21
                  V?JMP          1542      31
MSL.LB            ERMSG          1573     224
                  UNINS
                  SYSEN
```

(Continued)

.?ERM	001573	CN8	000102	R?ET	062601
.BITS	000055	D?IV	073101	S?AV	062401
.ERIA	000054	GPOSF	000700	SPOSF	000712
.ERIO	000053	LINER	000752	V?BEG	001406
.ERRO	000052	LINEW	000767	V?CAD	001233
.HEAP	000050	M?FF0	060201	V?CAL	001347
.REGB	000051	M?FF1	064201	V?CJP	001542
?CLOC	177777	M?FF2	070201	V?CSY	001440
?ERTN	000011	M?FF3	074201	V?END	001117
?ICAL	000012	M?FS0	061201	V?ENT	001355
?LDBL	000013	M?FS1	065201	V?FAL	001512
?NMAX	002462	M?FS2	071201	V?FIN	001462
?NRTN	000010	M?FS3	075201	V?FJP	001564
?PASC	001025	M?TF0	060001	V?FUP	001474
?STBL	000014	M?TF1	064001	V?GAD	001233
?SVLO	000004	M?TF2	070001	V?GIT	001521
?SYSE	000017	M?TF3	074001	V?JMP	001555
BYTER	000727	M?TS0	061001	V?PGB	001242
BYTEW	000742	M?TS1	065001	V?POP	001455
C15	000100	M?TS2	071001	V?RGE	001251
C16	000071	M?TS3	075001	V?TRU	001513
C2	000074	M?UL	073301	V?VAL	001233
C3	000076	OPENF	000623	V?WEQ	001301
C377	000101	P?OP0	061601	V?WGR	001313
C4	000073	P?OP1	065601	V?WLS	001272
C5	000077	P?OP2	071601	V?WNE	001331
C8	000072	P?OP3	075601	V?WNG	001340
CHARR	001000	P?SH0	061401	V?WNL	001322
CHARW	001014	P?SH1	065401		
CLDEL	000672	P?SH2	071401		
CLOSE	000664	P?SH3	075401		



The load map has been expanded by the addition of each symbol.

Do it on your system.

Note that the alphabetized listing goes down column one, then to column two, and so on.

If you would like, you can try the /NUMERIC switch in place of /ALPHA, as shown below:

```
) XEQ BIND/NUMERIC/L=TWOMAP TWO PASCAL.LB )
```

The only difference between /ALPHA and /NUMERIC is that the symbols are sorted by numerical value (address) with /NUMERIC. Delete TWOMAP and TWO.PR to avoid confusion with later questions.

10. This time bind TWO, makes STARS the executable program file, and STARMAP the listing. Show the command and expected response:

_____

_____

_____

_____

****

```
) XEQ BIND/L=STARMAP/P=STARS TWO PASCAL.LB )
```

The /P=STARS switch sets up STARS.PR as the executable program file. /L=STARMAP makes the STARMAP file the disc file load map.

Do it on your system.

11. Which files associated with TWO and STARS now exist on your system? Are the STARS and TWO program files the same? Show all entries:

_____

_____

_____

_____

_____

_____

****

```
FI/AS/S STAR+ TWO+
DIRECTORY @DPDO:PASCAL

  STARMAP          TXT   10-JUL-79   12:59:37        1774
  STARS.PR         PRG   10-JUL-79   12:59:36        3072
  TWO              TXT    1-JAN-00    0:00:41         293
  TWO.LS           UDF   10-JUL-79   12:29:32        3174
  TWO.OB           UDF   10-JUL-79   12:29:32        1140
  TWO.PR           PRG   10-JUL-79   12:55:52        3072
  TWOMAP           TXT   10-JUL-79   12:55:52        4253
```

Notice that STARS.PR and TWO.PR are the same length. Notice also that STARMAP and TWOMAP are not the same length because they were bound with different switches.

Do it on your system.

Execute STARS. Is it the same? You can use any one of the following four commands to get STARS flying:

```
) X STARS↓

) XE STARS↓

) XEQ STARS↓

) EXECUTE STARS
```

After you get STARS to run, DELETE TWO.PR before going on to the next question.

12. Last one. This time bind TWO and give it a revision number of 7.1. Show your entry before execution:

_____

_____

_____

****

```
) XEQ BIND/REV=7.1 TWO PASCAL.LB )
```

The /REV=number siwtch assigns a revision number to the executable program file.

Do it.

Use the CLI REV command to determine the success of the operation:

```
) REV TWO.PR )
  07.01
)
```

Don't forget the .PR extension on the REV command.

This concludes the Pascal binding lab exercise. You may wish to experiment with the other switches. Especially recommended for research are the debugger switches (/D and /DN). Try all of them. Now is the time for experimentation. The loss of STARS and TWO would be minimal.

This concludes Module Four for the Pascal programmer. At this point you should be able to edit, compile, bind, and execute a given Pascal source program.

ON TO MODULE FIVE

MODULE FIVE
SYMBOLIC DEBUGGER

SYMBOLIC DEBUGGER

Abstract

This module instructs in the fundamental concepts and procedures involved in the Symbolic Debugger. Topics include operating requirements and procedures, memory searching and monitoring, breakpoints, accumulator monitoring, display formats, and program execution.

Note: The Debugger operates on the assembly language level. This Module is designed to allow any language expert to learn Debugging. Assembly language knowledge is an asset.

Objectives

Upon completion of this module you will be able to:

1. Use the debugger commands to:
   a) set and display memory;
   b) set and display breakpoints;
   c) search memory;
   d) set and display accumulators and registers;
   e) execute a program from various locations;
   f) invoke and exit the debugger.

2. Write the command(s) for solving a given debugging situation.

3. Determine the result of a given debugger command.

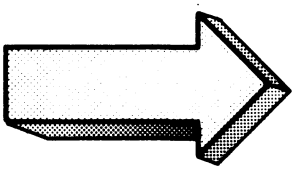
4. Solve a debugger error situation by:
   a) identifying the cause of the error,
   b) referencing the solution,
   c) entering the command(s) to solve the error.

5. Define, in your own words, the following Symbolic Debugger terms:
   a) breakpoint
   b) symbol
   c) break proceed counter
   d) conditional breakpoint

e) word register
f) increment register
g) number register
h) interrupt register
i) search output device register
j) carry register
k) console register
l) location register

Directions

Begin the first segment of Module Five on the next page of the Student Guide.



OPERATING PRINCIPLES

Abstract

This segment of Module Five discusses the features, operating procedures and requirements, console control, and errors involved in the Symbolic Debugger.

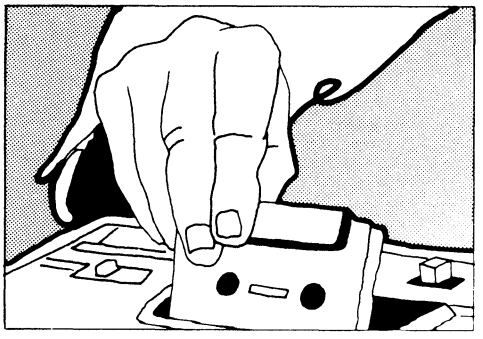
Objectives

Upon completion of this segment, you will be able to:

1. State the procedures for successfully invoking and exiting the Debugger;
2. correct typing errors at the console;
3. generate special command symbols;
4. state the function of two error responses
5. open, modify, and close memory locations.

Directions

Turn to Figure 5-1 on the next page of the Student Guide and listen to the tape for the first segment of Module Five.



- CONTROL PROGRAM EXECUTION
- SET & DELETE BREAKPOINTS
- SET CONDITIONAL BREAKPOINTS
- DISPLAY & MODIFY MEMORY LOCATIONS
- SEARCH MEMORY
- DISPLAY & MODIFY ACCUMULATORS & REGISTERS
- MODIFY DISPLAY FORMATS

FEATURES of the SYMBOLIC DEBUGGER

Figure 5-1

1 XEQ MASM/U ASMPROG ↵

- ADD /V TO THE MACROASSEMBLER COMMAND

2 XEQ BIND/D ASMPROG MSL.LB ↵

- ADD /D TO THE BINDER COMMAND

3 XEQ/D ASMPROG ↵

OR

DEBUG ASMPROG

COMMAND LINES AND THE DEBUGGER

Figure 5-2

```
) DEBUG DELTR )

DELTR
Error on ?EXEC
Error: No debugger present
DEBUG.DELTR
)
```

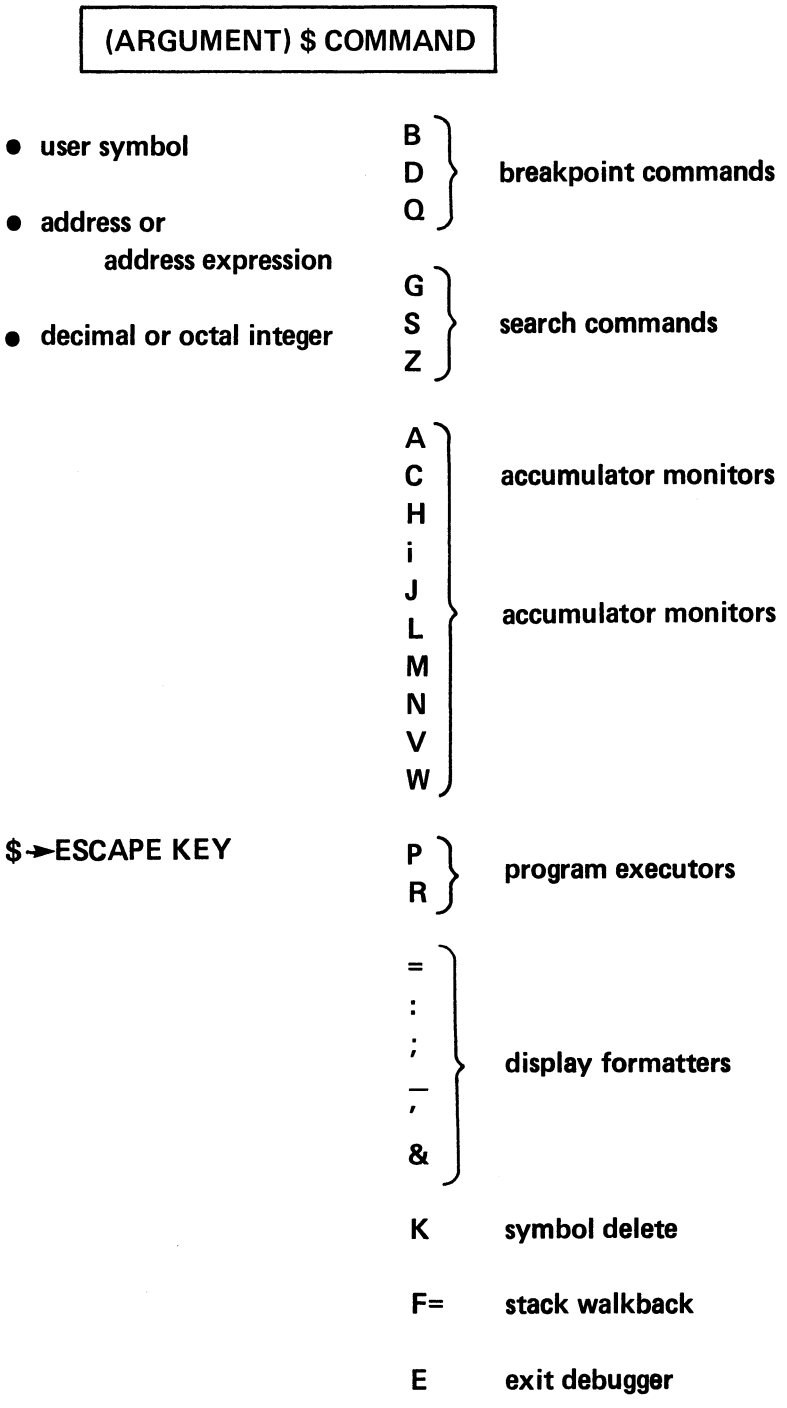
SYSTEM ERROR RESPONSE

Figure 5-3

```
) XEQ/D DELTR
*
```

DEBUGGER PROMPT AT STARTUP

Figure 5-4



DEBUGGER COMMAND FORMAT

Figure 5-5

```
BARY/U
BARRY/U
BARRY!U
BARRY$U
```

UNDEFINED SYMBOL

1. `STA#?`
2. `-7$R?`
3. `ST1234567?`

DEBUGGER ERROR RESPONSES

Figure 5-6

1

```
405!



405/126400
```

address! . . . open address.
address/. . . open address, display contents

2

```
405/126400 ↓
START+6    152400 ↓
START+7    006017 ↓
START+10    000013 ↓
START+11    001024
```

Using <NEW-LINE> to examine subsequent memory locations.

3

```
405/126400 ^
START+4    102400 ^
START+3    000404 ^
START+2    000005
```

Using <SHIFT-6> to examine preceding memory locations.

OPENING AND DISPLAYING MEMORY LOCATIONS

Figure 5-7

1

```
405/126400 <NEW LINE>

477/060277 <NEW LINE>
```

<NEW-LINE> . . . closes a memory location

2

```
405/126400 $B


477/060277 $B
```

CLOSING MEMORY LOCATIONS

Figure 5-8

1

```
700/020406 000000 <NEW LINE>
                  ↖—— insert new contents

700/000000
```

2

```
700/000000 020406 <NEW LINE>   re-insert old contents
700/020406 <NEW LINE>
```

3

```
1000!111111 <NEW LINE>   insert new contents

1000/111111 <NEW LINE>
```

MODIFYING MEMORY LOCATIONS

Figure 5-9

INTEGERS

0 to 177777 octal

0.to 65535. decimal

ASCII CHARACTERS

One- or two-character strings

Double quotes

Packed left to right

SYMBOLS

Legal name

Defined as Entry Point in Program

Recognizable by Debugger

Truncated to 5 characters

+ AND -

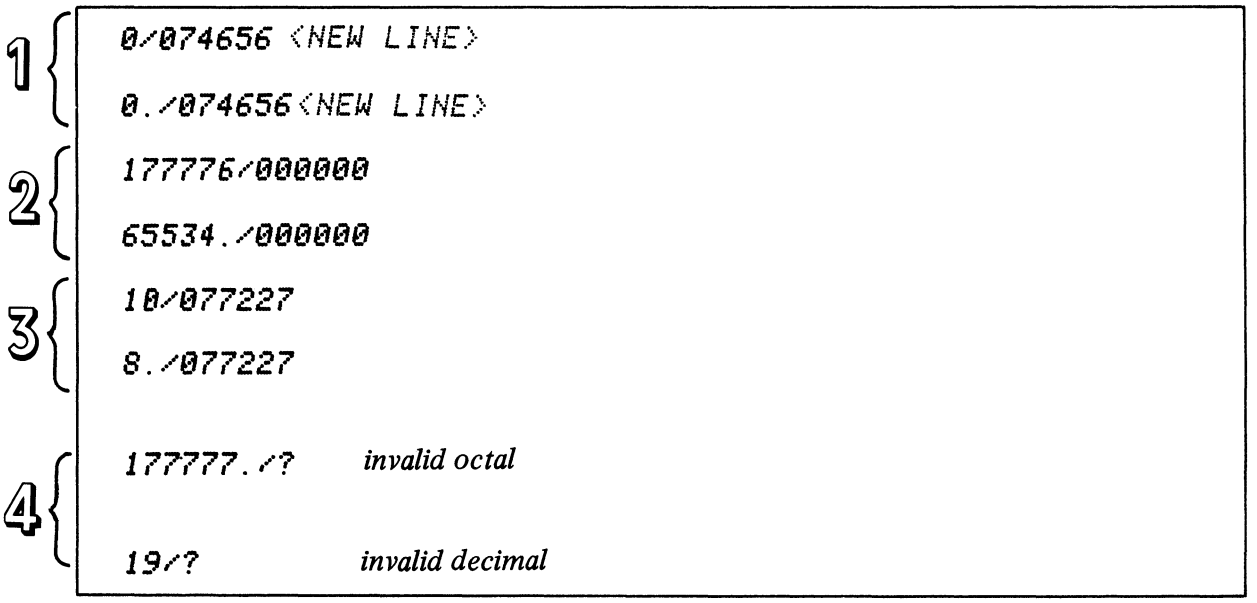
Within expressions

INSTRUCTIONS

Instruction mnemonic

DEBUGGER INPUT

Figure 5-10



OCTAL 0 to 177777

DECIMAL 0. to 65535.

VALID AND INVALID INTEGERS AS DEBUGGER INPUT

Figure 5-11

1

```
START/020411<NEW LINE>

GOBAK/102400<NEW LINE>
```

legal names

2

```
DLERR/U

FLPTR/U
```

Not defined as entry points

3

```
STAR/U

STARTUPAG?
```

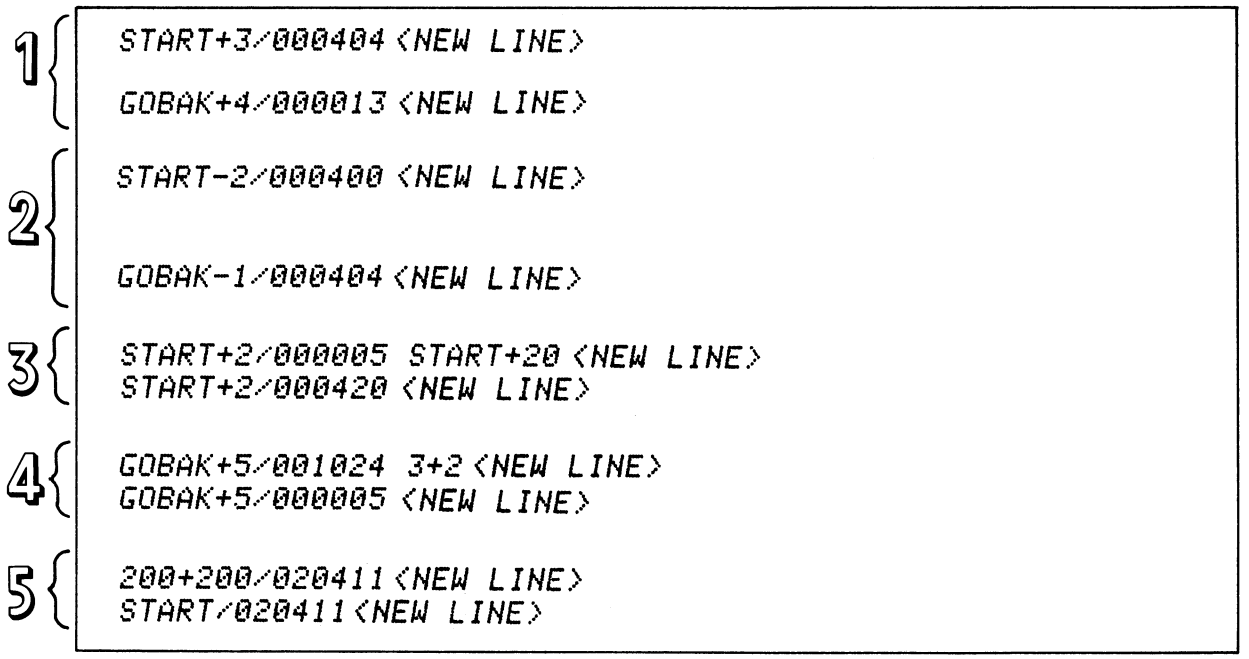
incorrect length

```
.ENT    START   ;ENTRY POINT DEFINED
.ENT    GOBAK   ;ANOTHER ENTRY POINT
```

Legal names are defined as entry points.

SYMBOLS AS DEBUGGER INPUT

Figure 5-12



+ and – OPERATORS AS DEBUGGER INPUT

Figure 5-13

1

```
500/014354 LDA 0,START<NEW LINE>
500/020700 <NEW LINE>
```

2

```
600/176660 SUB 0,0 <NEW LINE>
600/102400 <NEW LINE>
```

3

```
1000/060000 JMP 1 <NEW LINE>
1000/000001 <NEW LINE>
```

INSTRUCTION INPUT

Figure 5.14

TOPICS

- DEBUGGER FEATURES
- PREPARATION
- INPUT FORMATS
- ERROR RESPONSES
- MEMORY DISPLAY
- MEMORY MODIFICATIONS

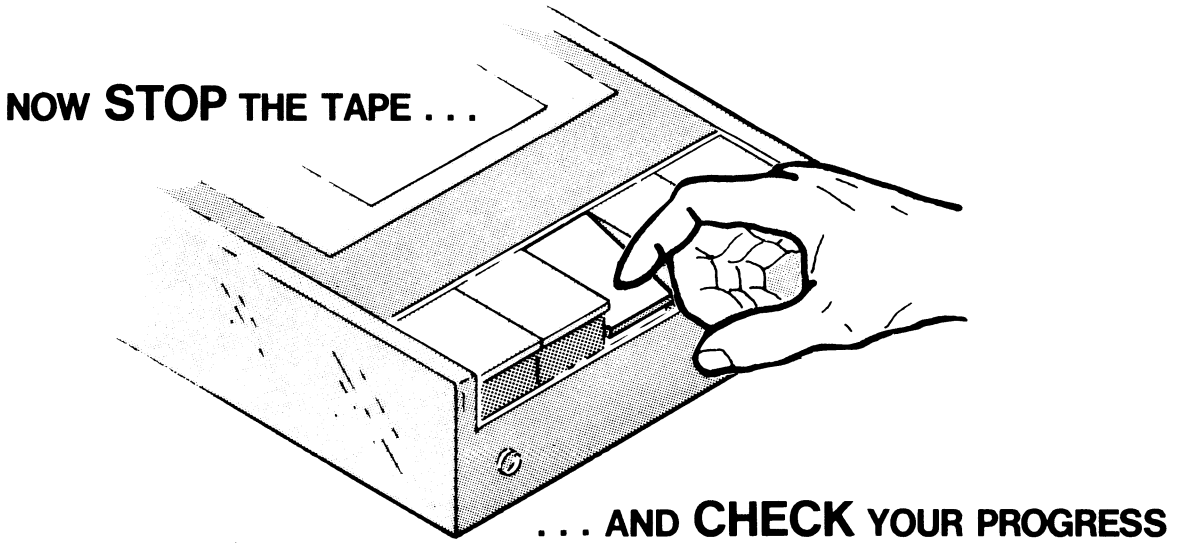


Figure 5-15

OPERATING PRINCIPLES QUIZ

Circle the letter of the correct answer. Note that a question may have more than one correct answer.

1. To make user symbols recognizable to the debugger, your preparation must include:

   A. XEQ BIND/U . . .
   B. XEQ MASM/U . . .
   C. XEQ DEBUG/U
   D. XEQ MASM/D

2. To make user symbols recognizable to the debugger, another step you must take is to:

   A. define the symbols as entry points.
   B. define the symbols in the system stack.
   C. define the symbols as offsets of the starting address.
   D. give the symbols valid names.

3. To be able to invoke the debugger, you must first use the command:

   A. XEQ BIND/U . . .
   B. XEQ MASM/U . . .
   C. XEQ BIND/D . . .
   D. XEQ MASM/D . . .

4. The command for invoking the debugger is:

   A. X/D programfile
   B. DEBUG programfile
   C. XEQ/D programfile
   D. DEBUG/D programfile

5. Debugger typing corrections may be made by using:

   A. RUBOUT or DELETE
   B. ESCAPE
   C. ERASE
   D. BACKSLASH

6. The $ in debugger commands is generated by pressing:

   A. SHIFT-4
   B. SHIFT-6
   C. ESCAPE
   D. SLASH (/)

7. The error message "U" indicates:

   A. an undefined symbol.
   B. an error other than an undefined symbol.
   C. an unintelligible entry.
   D. an incorrect address.

8. The error message "?" indicates:

   A. an undefined symbol.
   B. an error other than an undefined symbol.
   C. an illegal address.
   D. a Debugger prompt.

9. To open location 1200 you type:

   A. 1200!
   B. 1200/
   C. 1200 DISPLAY
   D. 1200 <NEW-LINE>

10. You have opened location 1200. To open location 1177 you type:

    A. 1177!
    B. 1177/
    C. SHIFT-6
    D. <RETURN> or <CR>

11. You have opened location 1200. To open location 1201, you type:

    A. 1201!
    B. 1201/
    C. SHIFT-6
    D. <RETURN> or <CR>

12. You have opened location 1201. To close it, you type:

    A. 1201!
    B. 1201/
    C. SHIFT-6
    D. <NEW-LINE> or <LINE FEED>

13. Location 1600 has all zeroes in it. Change it to all ones and close it.

    A. 1600! 111111 <NEW-LINE>
    B. 1600/000000 111111 <NEW-LINE>
    C. 1600/000000 111111<CR>
    D. 1600/000000 1<CR>

14. Valid integer input to the Debugger includes:

    A. 0
    B. 0.
    C. 177777
    D. 177777.

15. Valid integer input to the Debugger includes:

    A. 800
    B. 800.
    C. 65537
    D. 65537.

16. Valid modifications of location 750 include (750 contains 063711).

    A. 750/LDA 0, 1
    B. 750/063711 LDA 0, 1
    C. 750/063711 020001
    D. 750/063711/LDA 0, 1

17. The valid symbol GCHAR is at location 700. GCHAR may be referenced by:

    A. 700.
    B. 700 +0
    C. 500 + 200
    D. 1000 - 100

NOW CHECK YOUR ANSWERS
ON THE FOLLOWING PAGES.

OPERATING PRINCIPLES QUIZ
ANSWERS

1. To make user symbols recognizable to the debugger, your preparation must include:

   A. XEQ BIND/U . . .
   (B.) XEQ MASM/U . . . This includes the user symbols with the object file.
   C. XEQ DEBUG/U
   D. XEQ MASM/D.

2. To make user symbols recognizable to the debugger, another step you must take is to:

   (A.) define the symbols as entry points.
   Yes, use the .ENT statement.
   B. define the symbols in the system stack.
   No.
   C. define the symbols as offsets of the starting address.
   No, this is automatically by the Debugger.
   (D.) give the symbols valid names.
   Yes, a basic requirement.

3. To be able to invoke the debugger, you must first use the command:

   A. XEQ BIND/U . . .
   B. XEQ MASM/U . . .
   No, this only includes user symbols with the object file. You can debug without this.
   (C.) XEQ BIND/D . . .
   This binds the object files, library files, and debugger into a program file.
   D. XEQ MASM/D . . .

4. The command for invoking the debugger is:

   (A.) X/D programfile
   Yes, XEQ is abbreviated
   (B.) DEBUG programfile
   Yes.
   (C.) XEQ/D programfile
   Yes.
   D. DEBUG/D programfile.
   The /D switch produces the error: "unknown switch specified".

5. Debugger typing corrections may be made by using:

(A.) RUBOUT or DELETE
Yes, depending on your terminal.

B. ESCAPE
No, produces a $.

C. ERASE
No.

D. BACKSLASH.
No.

6. The $ in debugger commands is generated by pressing:

A. Shift-4
This produces a "$" but does not function in a command.

B. SHIFT-6
No, generates an up-arrow, which opens and displays the previous location.

(C.) ESCAPE
Yes, echoed as $.

D SLASH. (/)
No, opens and displays an address.

7. The error message "U" indicates:

(A.) an undefined symbol.

B. an error other than an undefined symbol.
No, these receive the "?".

C. an unintelligible entry.
Only if the entry is an undefined symbol.

D. an incorrect address.
Not unless the address is referenced as a symbol.

8. The error message "?" indicates:

A. an undefined symbol.
B. an error other than an undefined symbol.
C. an illegal address.

Yes, this is an error that is other than an undefined symbol.

D. a Debugger prompt.

No, the only visible debugger prompt is the initial *.

9. To open location 1200 you type:

A. 1200!

Yes, nothing is displayed.

B. 1200/

Yes, the contents are displayed.

C. 1200 DISPLAY

No, receives a "?" error.

D. 1200 <NEW-LINE>

No, the new-line (or line-feed) closes the location that was never opened.

10. You have opened location 1200. To open location 1177 you type:

A. 1177!

Yes, nothing is displayed.

B. 1177/

Yes, contents are displayed.

C. Shift-6

Yes, opens and displays contents of previous location ( ).

D. <RETURN> or <CR>

No, opens and displays 1201.

11. You have opened location 1200. To open location 1201, you type:

A. 1201!

Yes, nothing is displayed.

B. 1201/

Yes, opens and displays

C. SHIFT-6

No, opens and displays 1177.

D. <RETURN> or <CR>

Yes, opens and displays the succeeding location.

12. You have opened location 1201. To close it, you type:

A. 1201!
No, the "!" opens it again.

B. 1201/
No, the "/" keeps it open.

(C.) SHIFT-6
Yes, closes 1201, opens and displays 1200.

(D.) <NEW-LINE> or <LINE FEED>
Yes, the choice depends on your console keyboard.

13. Location 1600 has all zeroes in it. Change it to all ones and close it.

(A.) 1600! 111111 <NEW-LINE>
Yes

(B.) 1600/000000 111111 <NEW-LINE>
Yes, again.

(C.) 1600/000000 111111 <CR>
Yes, also opens and displays 1601.

D. 1600/000000 1 <CR>
No, only inserts a single one.

14. Valid integer input to the Debugger includes:

(A.) 0
An octal zero.

(B.) 0.
A decimal zero.

(C.) 177777
The maximum octal integer.

D. 177777.
No, exceeds the 65,535 maximum for decimal integers.

15. Valid integer input to the Debugger includes:

A. 800

No, "8" is an invalid octal digit.

(B.) 800.

Yes, a valid decimal integer.

(C.) 65537

Yes, a valid octal integer.

D. 65537.

No, exceeds the decimal maximum of 65,535.

16. Valid modifications of location 750 include (750 contains 063711).

A. 750/LDA 0, 1

No, the "/" should display 750's contents first.

(B.) 750/063711 LDA 0, 1

Yes, instructions are valid input.

(C.) 750/063711 020001

Yes. This just happens to be a "LDA 0, 1."

D. 750/063711/LDA 0, 1

No, nothing is modified. Note that the second "/" opens and displays location 063711.

17. The valid symbol GCHAR is at location octal 700. GCHAR may be referenced by:

A. 700.

No, decimal 700 is different from octal 700.

(B.) 700 + 0

Yes, a valid expression.

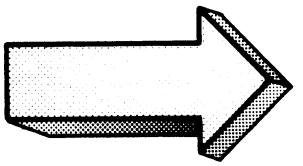
(C.) 500 + 200

Yes, a valid expression.

(D.) 1000 - 100

Yes, octal 1000 minus octal 100 equals octal 700.

A SCORE OF 14 CORRECT ANSWERS OUT OF THE 17 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.



OPERATING PRINCIPLES

LAB EXERCISE

Abstract

This exercise covers debugger preparation, error correction, error messages, and memory opening, display, modification, and closing.

Directions

This lab follows the procedures of previous labs:

1. Cover the answer;
2. Read the question;
3. Write your answer;
4. Check the answer;
5. Perform the operation on your system.

We use the PROG_ONE program in this lab. SPEED it into your system (if it is not already there). Note the changes to the .ENT statement and the text message. Make the appropriate changes in your file. The PROG_ONE text is illustrated on the following page. When you are satisfied with the accuracy of PROG_ONE then begin the lab.

```
   TYPE PROG_ONE)
        .TITL   PROG_ONE;TITLE IDENTIFIER
        .ENT    START   ;ENTRY POINT
        .ENT    LOOP
        .ENT    MESSG
        .ENT    CLEAN
        .ENT    ERTN
        .NREL   1       ;PURE CODE

        ;MAIN ROUTINE

START:  LDA     0,PATH  ;BYTEPOINTER TO PATHNAME
        ?OPEN           ;OPEN CHANNEL TO @TTO
        JMP     ERTN    ;ERROR ON OPEN...CLI
        STA     0,CHAN  ;SAVE CHANNEL #
LOOP:   LDA     0,CHAN  ;IN CASE IT WANDERED
        LDA     1,STPTR ;BYTE POINTER TO STAR LINE
        LDA     2,36    ;36 OCTAL BYTE MSG

AUTHR:  ?WRITE  DS      ;DATA SENSITIVE OPTION
        JMP     ERTN    ;ERROR ON WRITE ..TO CLI
        DSZ     TWEN    ;LOOP 20 TIMES
        JMP     AUTHR   ;WRITE AGAIN, UNTIL 0

MESSG:  LDA     1,MPTR
        LDA     2,55    ;MSG IS 55 OCTAL BYTES
        ?WRITE  DS      ;WRITE 'MP/OS....' MESSAGE
        JMP     ERTN    ;ERROR ON WRITE
CLEAN:  LDA     0,CHAN  ;PREP FOR CLOSE
        ?CLOSE          ;CLOSE CHANNEL TO @TTO
        JMP     ERTN    ;ERROR ON ?CLOSE
        SUB     0,0     ; CLEAR ACS FOR CLEAN RETURN
        SUB     1,1     ;... AC1 HAS ERROR LENGTH
        SUB     2,2     ;... AC2 HAS POINTER TO ERROR

ERTN:   ?RETURN         ;TO CLI, ERROR RETURNS KEEP
                        ;   CODES IN ACS

        ;DATA AREAS

CHAN:   0               ;CHANNEL # SAVE AREA
PATH:   .+1*2
        .TXT    /@TTO/  ;OUTPUT TO CONSOLE
STPTR:  .+1*2
        .TXT    /                       *<12>/
TWEN:   24              ;24 OCTAL = 20 DEC
MPTR:   .+1*2
        .TXT    %                   THE MP/OS! HAS LANDED<12>%
        ;THE MP/OS STACK

        .NREL   0       ;IMPURE CODE
STK:    .BLK    50      ;RESERVE 50 WORDS
        .LOC    40      ;LOC 40 HAS ...
        STK             ;... THE STACK POINTER
        STK             ;LOCATION 41
        STK+45          ;LOC 42 HAS STACK LIMIT

        .END    START   ;END OF ASM INPUT
)
```

1. Assemble PROG_ONE. Include user symbols and make PROG.LS the list file:

_____

_____

_____

****

```
) XEQ MASM/U/L=PROG.LS PROG_ONE )
)
```

The /U directs the Macroassembler to include user symbols (e.g., START, LOOP) in the object file. /L=PROG.LS assigns the PROG.LS disc file as the listing file.

Do it on your system.

If you receive an "assembly error" message, then re-invoke SPEED, correct the source file, and try again.

2. No question here. Get a hard-copy listing of PROG.LS and save it for later.

```
) TYPE PROG.LS }
 0001 PROG_  MP/OS  ASSEMBLER REV 00.05          08:01:79 14/03/02
                          .TITL   PROG_ONE;TITLE IDENTIFIER
02                        .ENT    START   ;ENTRY POINT
03                        .ENT    LOOP
04                        .ENT    MESSG
05                        .ENT    CLEAN
06                        .ENT    ERTN
07        000001          .NREL   1       ;PURE CODE
08
09                        ;MAIN ROUTINE
10
11 00000!020434 START:    LDA     0,PATH  ;BYTEPOINTER TO PATHNAME
12                        ?OPEN           ;OPEN CHANNEL TO @TTO
13 00003!000426           JMP     ERTN    ;ERROR ON OPEN...CLI
14 00004!040427           STA     0,CHAN  ;SAVE CHANNEL #
15 00005!020426 LOOP:     LDA     0,CHAN  ;IN CASE IT WANDERED
16 00006!024432           LDA     1,STPTR ;BYTE POINTER TO STAR LINE
17 00007!030036           LDA     2,36    ;36 OCTAL BYTE MSG
18
19                AUTHR:  ?WRITE  DS      ;DATA SENSITIVE OPTION
20 00012!000417           JMP     ERTN    ;ERROR ON WRITE ..TO CLI
21 00013!014442           DSZ     TWEN    ;LOOP 20 TIMES
22 00014!000774           JMP     AUTHR   ;WRITE AGAIN, UNTIL 0
23
24 00015!024441 MESSG:    LDA     1,MPTR
25 00016!030055           LDA     2,55    ;MSG IS 55 OCTAL BYTES
26                        ?WRITE  DS      ;WRITE 'MP/OS....' MESSAGE
27 00021!000410           JMP     ERTN    ;ERROR ON WRITE
28 00022!020411 CLEAN:    LDA     0,CHAN  ;PREP FOR CLOSE
29                        ?CLOSE          ;CLOSE CHANNEL TO @TTO
30 00025!000404           JMP     ERTN    ;ERROR ON ?CLOSE
31 00026!102400           SUB     0,0     ; CLEAR ACS FOR CLEAN RETURN
32 00027!126400           SUB     1,1     ;... AC1 HAS ERROR LENGTH
33 00030!152400           SUB     2,2     ;... AC2 HAS POINTER TO ERROR
34
35                ERTN:   ?RETURN         ;TO CLI, ERROR RETURNS KEEP
36                                        ;   CODES IN ACS
37
38                        ;DATA AREAS
39
40 00033!000000 CHAN:     0               ;CHANNEL # SAVE AREA
41 00034!000072&PATH:     .+1*2
42 00035!040124           .TXT    /@TTO/  ;OUTPUT TO CONSOLE
43        052117
44        000000
45 00040!000102&STPTR:    .+1*2
46 00041!020040           .TXT    /                       *<12>/
47        020040
49        020040
49        020040
50        020040
51        020040
52        020040
53        020040
54        020040
55        020040
56        020052
```

(CONTINUED)

```
57        005000
58 00055!000024 TWEN:   24              ;24 OCTAL = 20 DEC
59 00056!000136&MPTR:   .+1#2
60 00057!020040         .TXT    %               THE MP/OS! HAS LANDED<12>%
 0002 PROG_
01        020040
02        020040
03        020040
04        020040
05        020040
06        020040
07        020040
08        052110
09        042440
10        046520
11        027517
12        051441
13        020110
14        040523
15        020114
16        040516
17        042105
18        042012
19        000000
20                        ;THE MP/OS STACK
21
22        000000          .NREL   0       ;IMPURE CODE
23 00000'000050 STK:      .BLK    50      ;RESERVE 50 WORDS
24        000040          .LOC    40      ;LOC 40 HAS ...
25 00040 000000'          STK             ;... THE STACK POINTER
26 00041 000000'          STK             ;LOCATION 41
27 00042 000045'          STK+45          ;LOC 42 HAS STACK LIMIT
28
29                        .END    START   ;END OF ASM INPUT
30

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS  0003 PROG_

AUTHR 000010!         1/19#   1/22
CHAN  000033!         1/14    1/15    1/28    1/40#
CLEAN 000022! EN      1/05    1/28#
ERTN  000031! EN      1/06    1/13    1/20    1/27    1/30    1/35#
LOOP  000005! EN      1/03    1/15#
MESSG 000015! EN      1/04    1/24#
MPTR  000056!         1/24    1/59#
PATH  000034!         1/11    1/41#
START 000000! EN      1/02    1/11#   2/29
STK   000000'         2/23#   2/25    2/26    2/27
STPTR 000040!         1/16    1/45#
TWEN  000055!         1/21    1/58#
?CLOS 002203! MC      1/29
?I    000013          1/13#   1/20#   1/27#   1/30#   1/36#
?J    000000          1/13#   1/20#   1/27#   1/30#   1/36#
?K    000002          1/20#   1/27#
?OPEN 001743! MC      1/12
?RETU 000203! MC      1/35
?SYSE 000001$ XD      1/13    1/20    1/27    1/30    1/36
?WRIT 002143! MC      1/19    1/26

)
```

3. Bind PROG_ONE with the Debugger and library files. Make PROGMAP the binder listing (load map):

_____

_____

_____

****

```
) XEQ BIND/D/L=PROGMAP PROG_ONE MSL.LB )

)
```

The /D switch on the BIND command directs the binder to include the Debugger with the program file.

Now do it on your system.

Don't forget MSL.LB on the tail-end of the bind command line!

4. Make PROGTWO.PR a backup copy of PROG_ONE.PR. Then determine which PROG files now exist on your system? Show the commands and anticipated responses.

****

```
) COPY PROGTWO.PR PROG_ONE.PR ↵
)


) FI/AS/S PROG+ ↵
DIRECTORY @DPX1:

  PROG.LS              TXT     1-AUG-79  14:04:41          3948
  PROGMAP              TXT     1-AUG-79  14:11:56           651
  PROGTWO.PR           PRG     1-AUG-79  14:12:59          6144
  PROG_ONE             TXT     1-AUG-79  10:02:43          1338
  PROG_ONE.BU          TXT    18-JUN-79  11:49:51          1293
  PROG_ONE.OB          OBF     1-AUG-79  14:04:01           534
  PROG_ONE.PR          PRG     1-AUG-79  14:11:55          6144
)
```

Now do it on your system.

The backup copy of PROG_ONE may come in handy if you lose your original.

5. No question here. Get a hard-copy printout of the binder's load map (PROGMAP). Keep it for later.

```
) TYPE PROGMAP }

MP/OS  Binder Rev 0.04

@DPK1:PROG_ONE.PR


Area          Start      End   Length
-------------------------------------

Pure            400      502      103
Impure          503     5665     5163
Symbol         5666     5742       55


Filename          Title          Pure   Pure  Impure Impure  Zrel Zrel   Abs
                                 Strt    Lth    Strt    Lth  Strt  Lth   Num

PROG_ONE.OB       PROG_           400    103     503     50                 3
MSL.LB                                        553   5113                23
MSL.LB

)
```

6. Invoke the Debugger for PROG_ONE. Write down whichever command you choose along with the anticipated response.

_____

_____

_____

****

```
) DEBUG PROG_ONE )
*
```

We used the DEBUG command. You could choose the XEQ command with the /D switch appended. (XEQ/D PROG_ONE) and get the same result.

Try it on your system.

7. Open the location referenced by the symbol START, display its contents, and close it. Show the command:

_____

_____

****

```
START/020434 <NEW LINE>
```

Since START is recognizable to the Debugger, we can use it to reference locations. The slash (/) command opens and displays START's contents. Pressing new-line or line-feed closes the location.

Do it on your system.

If you receive a "U" error message:

a) check your typing for possible misspelling,
b) check your listing. START must be defined in a .ENT statement.

Compare the START location in the program listing with your console display. They should show 020434.

8. Use the "+" operator to open, display, and close the location following START. Show the anticipated result.

_____

_____

****

```
START+1/006017 <NEW LINE>
```

"START+1/" opens the location referenced by the expression START+1 and displays its contents.

To get to the location following start, you can use any one of the following

401/ . . . . . .because START is at location 400
400+1/ . . . . . .because this = 401.
START+1/ . . .because this is a valid expression

Note that 006017 is a jump to the system call handler.

Do it on your system.

Make sure you close the location. Use the new-line or line feed (console-dependent).

Note that we usually space down several lines between commands, by pressing line feed. We do this only to produce uncluttered examples.

9. LOOP is a valid symbol name. Open the location referenced by LOOP, display the contents, and close it. Then display the next three locations following LOOP. Show the entries and anticipated responses. (Hint: the program listing helps here.)

_____

_____

_____

_____

****

```
LOOP/020426<CR>
LOOP+1    024432<CR>
LOOP+2    030036<CR>
LOOP+3    006017<CR>
```

"LOOP/" opens the location referenced by LOOP and displays its contents.

Pressing RETURN or CR closes the location, opens the next and displays its contents.

Try it on your system.

Make sure you close LOOP+3 with a new-line.

LOOP is a valid reference because it was defined in an entry statement (.ENT) and was included as a user symbol (/U).

10. The symbol AUTHR is part of your program. Open the location referenced by AUTHR and display its contents. Show the command and anticipated response:

_____

_____

****

```
AUTHR/U
```

Since AUTHR is not defined as an entry point in a .ENT statement, the system responds with the "U" error.

Try it on your system.

11. Now open the location referenced by the symbol MESSG. Display its contents, close it, and take a look at the next six addresses (seven in all). Show the command and anticipated responses:

_____

_____

_____

_____

_____

_____

_____

****

```
MESSG/024441<CR>
MESSG+1    030055<CR>
MESSG+2    006017<CR>
MESSG+3    002004<CR>
MESSG+4    000410<CR>
CLEAN  020411<CR>
CLEAN+1    006017<CR>
```

"MESSG/" opens the location referenced by MESSG and displays the locations contents.

CLEAN is a valid symbol name. The debugger shifts to the different address reference when it reaches another valid symbol.

Do it on your system.

All except the last address are terminated by a CR or RETURN. The last address is closed with a NEW-LINE or LINE FEED.

Compare the contents with your program listing.

12. Open the address referenced by MESSG. Display the contents. Change the contents to 000400. Close the location. Then display it to verify the modification. Show how first:

_____

_____

****

```
MESSG/024441 000400<NEW LINE>
MESSG/000400<NEW LINE>
```

MESSG/ opens the location. The contents are displayed as 02441. You then type 000400 to change the contents. A NEWLINE closes the location. Re-opening the location displays the new contents.

Do it on your system.

13. The MESSG location must be changed back to its original contents of 024441. Open, display, modify, and close the location. Then re-open and display its re-instated contents:

_____

_____

****

```
MESSG/000400 024441<NEW LINE>
MESSG/024441<NEW LINE>
```

The routine is almost identical to that followed in #12. MESSG/ opens the location referenced by MESSG.

Do it on your system.

Make sure you close the location.

14. Use the minus operator to open, display, and close the location four short of LOOP:

****

```
LOOP-4/006017 <NEW LINE>
```

LOOP-4 is also referenced as START+1. The 006017 is the numeric representation of a jump to the system call handler.

Do it on your system.

Close the location.

15. Open, display, and close your program's first two addresses. Use both decimal and octal notation to reference the first address.

_____

_____

_____

_____

****

```
256./020434<CR>
START+1    006017<NEW LINE>


400/020434<CR>
START+1    006017<NEW LINE>
```

Decimal 256 is the equivalent of octal 400, the first address in the program. Note the decimal point indicating decimal notation.

Try it on your system.

Use the CR to move to the next location. Use the NEW-LINE to close the last locations.

16. Open and display location 401. Use the SHIFT-6 to get to location 400. Show it:

_____

_____

****

```
401/006017 ^
START  020434<NEW LINE>
```

401/ opens and displays address 401. It contains 006017. Pressing SHIFT-6 is echoed as an up-arrow. This closes 401, opens 400 and displays its contents. Location 400 is referenced by the symbol START.

Try it.

Close the last location.

17. This will hold us for now. To exit the debugger, use the exit command as shown:

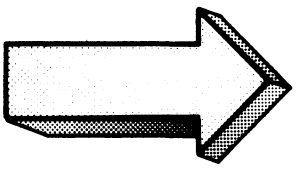
```
$E
)
```

Press ESCAPE followed by "E". The ESCAPE is echoed by the "$".

Do it on your system.

Wait a few seconds for the CLI prompt to return.

This concludes the Debugger Operating Principles Lab Exercise. Shut down your system and continue with the second segment of Module Five.



SEARCH AND DISPLAY

Abstract

This segment of Module Five instructs in the Debugger commands for altering display formats and searching memory.

Objectives

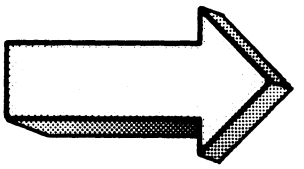
Upon completion of this segment, you will be able to:

1. Write the search commands to display any given part of main memory;

2. Write the command for displaying storage values in the following permanent and/or individual formats:
   a) numeric
   b) symbolic
   c) instruction
   d) byte
   e) ASCII
   f) byte pointer
   g) symbol format

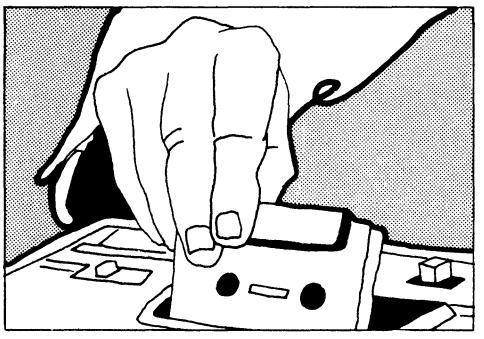
3. Describe the function of the:
   a) word register
   b) mask register
   c) increment register
   d) number register

4. Write the commands for loading the four registers above.



Directions

Turn to Figure 5-34 on the next page of the Student Guide and listen to the tape for the second segment of Module Five.



1

```
START+3/000404 <NEW LINE>
```

2

```
START+3/000404 ;JMP GOBAK+3 <NEW LINE>
```

; . . . display contents in instruction format.

Figure 5-34

1

```
GOBAK+2/152400 ;SUB 2 2 <NEW LINE>
```

2

```
GOBAK+2/152400 ;SUB 2 2 =152400 <NEW LINE>
```

= . . . display contents in numeric format.

Figure 5-35

SYMBOL		DISPLAY FORMAT
=	(Equals)	Numeric format
:	(Colon)	Symbolic format
;	(Semi-colon)	Instruction format
_	(Under-line)	Byte format
'	(Single quote)	ASCII format
&	(Ampersand)	Byte Pointer format
*	(Asterisk)	Symbol format with Bit 0 = 0.
>	(Right arrow)	Display String given by Byte Pointer

SINGLE ITEM DATA DISPLAY FORMATS

Figure 5-36

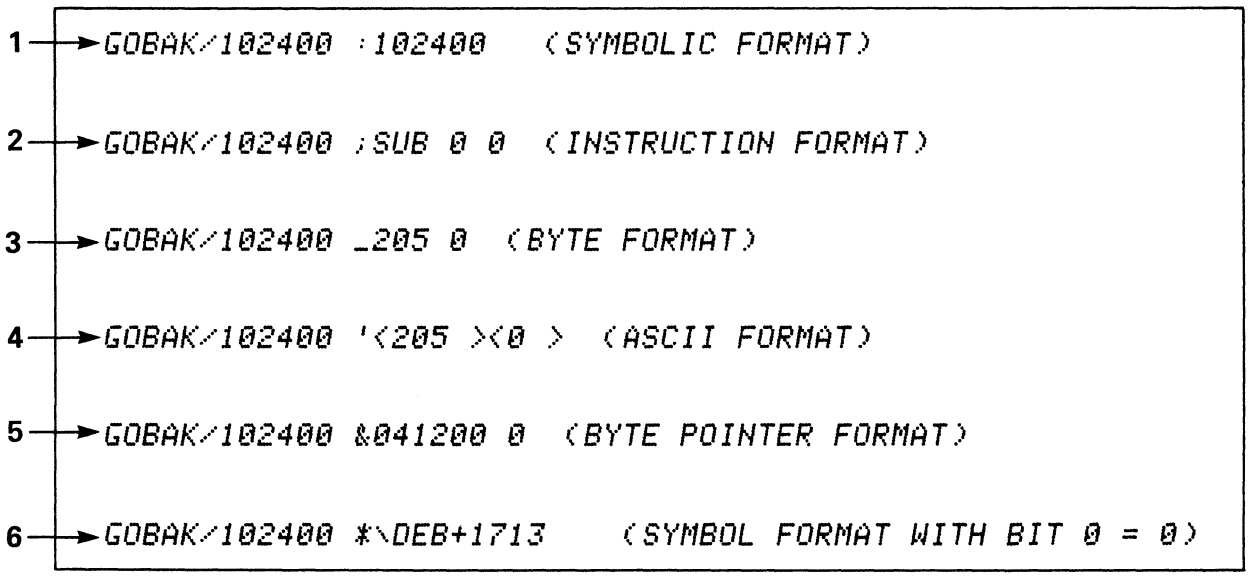
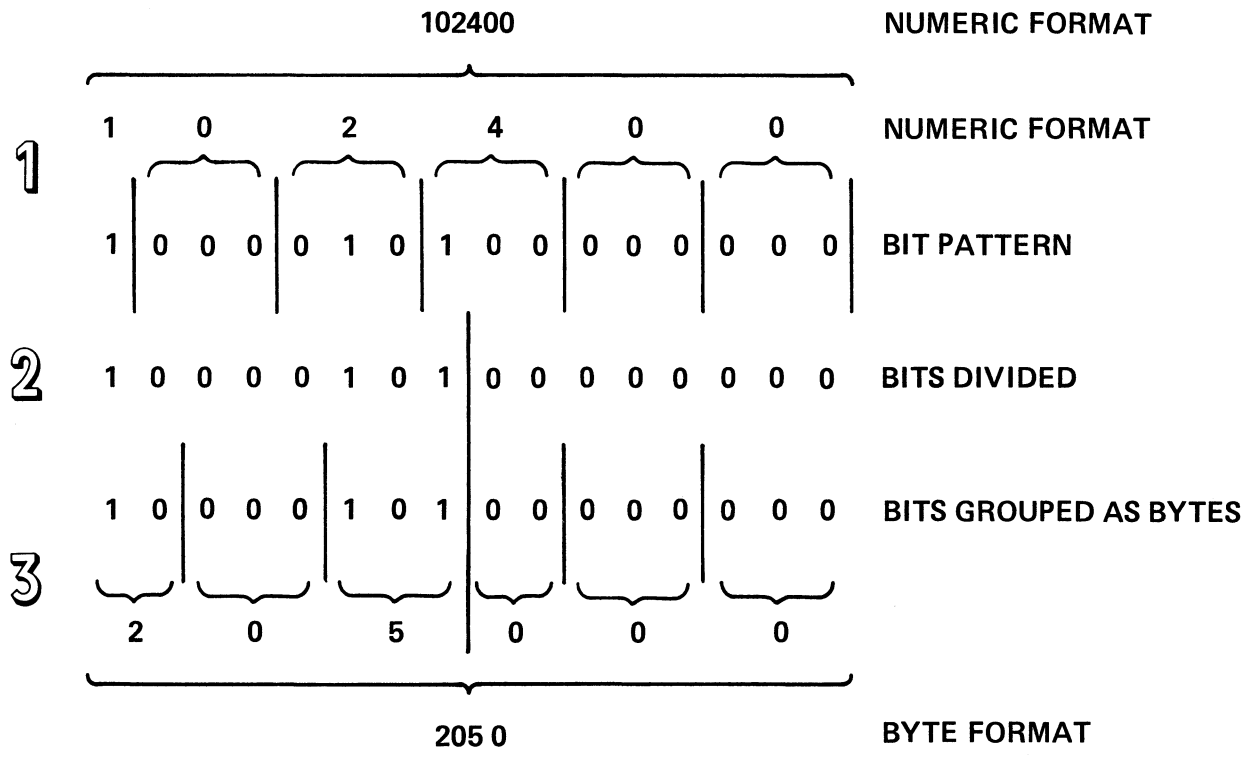

1 GOBAK/102400 :102400 (SYMBOLIC FORMAT)
2 GOBAK/102400 ;SUB 0 0 (INSTRUCTION FORMAT)
3 GOBAK/102400 _205 0 (BYTE FORMAT)
4 GOBAK/102400 '<205 ><0 > (ASCII FORMAT)
5 GOBAK/102400 &041200 0 (BYTE POINTER FORMAT)
6 GOBAK/102400 *\DEB+1713 (SYMBOL FORMAT WITH BIT 0 = 0)

Figure 5-37



NUMERIC FORMAT TO BYTE FORMAT

Figure 5-38

```
GOBAK+6/053530  'WX <CR>
GOBAK+7    054532  'YZ<NEW LINE>
```

ASCII DISPLAY FORMAT

Figure 5-39

1

```
$N 000000<NEW LINE>
```

$N . . . . Open & Display the number register

2

```
$N 000000 177777<NEW LINE>

START/+8457.<CR>
START+1    +3087.<CR>
START+2    +5.<CR>
START+3    +260.<CR>
GOBAK   -31488.<CR>
GOBAK+1    -21248.<CR>
GOBAK+2    -11008.<CR>
GOBAK+3    +3087.<CR>
GOBAK+4    +11.<CR>
GOBAK+5    +532.<CR>
GOBAK+6    +22360.<CR>
GOBAK+7    +22874.<CR>
GOBAK+10    +0.<CR>
GOBAK+11    +0.<NEW LINE>
```

Number Register = 0 ? . . . Display is octal format.
Number Register = 0 ? . . . Display is decimal format.

3

```
$N -1.000000<NEW LINE>
START/020411<CR>
START+1    006017<CR>
START+2    000005<CR>
START+3    000404<CR>
GOBAK   102400<NEW LINE>
```

DECIMAL NUMERIC FORMAT DISPLAY

Figure 5-40

SYMBOL		FORMAT
$=	(ESCAPE, EQUALS)	Numeric Format
$:	(ESCAPE, COLON)	Symbolic Format
$;	(ESCAPE, SEMI-COLON)	Instruction Format
$_	(ESCAPE, UNDER-SCORE)	Byte Format
$'	(ESCAPE, SINGLE QUOTE)	ASCII Format
$&	(ESCAPE, AMPERSAND)	Byte Pointer Format

PERMANENT DATA DISPLAY FORMAT COMMANDS

Figure 5-41

1

```
$;
START/LDA 0 GOBAK+5 <CR>
START+1   JSR @\SYSE <CR>
START+2   CLE <CR>
START+3   JMP GOBAK+3 <CR>
GOBAK  SUB 0 0 <CR>
GOBAK+1   SUB 1 1 <CR>
GOBAK+2   SUB 2 2 <CR>
GOBAK+3   JSR @\SYSE <CR>
GOBAK+4   JMP \ICAL+1 <CR>
GOBAK+5   JMP +24 2 <CR>
GOBAK+6   STA 2 @+130 3 <CR>
GOBAK+7   STA 3 \DEB+60 <CR>
GOBAK+10    0 <CR>
GOBAK+11    0 <NEW LINE>
```

$; . . . Display contents in Instruction Format.

2

```
$:
START/20411 <CR>
START+1   \STEN+154  ? <CR>

START+2   +5 <CR>
START+3   GOBAK <CR>
GOBAK  102400 <CR>
GOBAK+1   126400 <CR>
GOBAK+2   152400 <CR>
GOBAK+3   \STEN+154 <CR>
GOBAK+4   \ICAL+1 <CR>
GOBAK+5   \DEB+337 <CR>
GOBAK+6   53530 <CR>
GOBAK+7   54532 <CR>
GOBAK+10   +0 <CR>
GOBAK+11   +0 <CR>
GOBAK+12   +0 <NEW LINE>
```

$: . . . Display Contents in Symbolic Format.

Figure 5-42

1

```
$=
START/020411<CR>
START+1    006017<CR>
START+2    000005<CR>
START+3    000404<CR>
GOBAK   102400<CR>
GOBAK+1    126400<CR>
GOBAK+2    152400<CR>
GOBAK+3    006017<CR>
GOBAK+4    000013<CR>
GOBAK+5    001024<CR>
GOBAK+6    053530<CR>
GOBAK+7    054532<CR>
GOBAK+10    000000<CR>
GOBAK+11    000000<CR>
GOBAK+12    000000<NEW LINE>
```

$= . . . display contents in numeric format.

2

```
$_
START/41 11<CR>
START+1    14 17<CR>
START+2    0 5<CR>
START+3    1 4 ?<CR>

GOBAK   205 0<CR>
GOBAK+1    255 0<CR>
GOBAK+2    325 0<CR>
GOBAK+3    14 17<CR>
GOBAK+4    0 13<CR>
GOBAK+5    2 24<CR>
GOBAK+6    127 130<CR>
GOBAK+7    131 132<CR>
GOBAK+10    0 0<CR>
GOBAK+11    0 0<CR>
GOBAK+12    0 0<NEW LINE>
```

$– . . . display contents in byte format.

Figure 5-43

1

```
$'
START/!<11 ><CR>
START+1    <14 ><17 ><CR>
START+2    <0 ><5 ><CR>
START+3    <1 ><4 ><CR>
GOBAK   <205 ><0 ><CR>
GOBAK+1    <255 ><0 ><CR>
GOBAK+2    <325 ><0 ><CR>
GOBAK+3    <14 ><17 ><CR>
GOBAK+4    <0 ><13 ><CR>
GOBAK+5    <2 ><24 ><CR>
GOBAK+6    WX<CR>
GOBAK+7    YZ<CR>
GOBAK+10    <0 ><0 ><CR>
GOBAK+11    <0 ><0 ><CR>
GOBAK+12    <0 ><0 ><CR>
GOBAK+13    <0 ><0 ><NEW LINE>
```

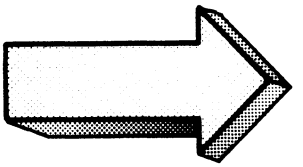
$' . . . display contents in ASCII format.

2

```
$&
START/010204 1<CR>
START+1    003007 1<CR>
START+2    000002 1<CR>
START+3    000202 0<CR>
GOBAK   041200 0<CR>
GOBAK+1    053200 0<CR>
GOBAK+2    065200 0<CR>
GOBAK+3    003007 1<CR>
GOBAK+4    000005 1<CR>
GOBAK+5    000412 0<CR>
GOBAK+6    025654 0<CR>
GOBAK+7    026255 0<CR>
GOBAK+10    000000 0<CR>
GOBAK+11    000000 0<CR>
GOBAK+12    000000 0<NEW LINE>
```

$& . . . display contents in byte pointer format.

Figure 5-44



DISPLAY FORMATS QUIZ

Match the symbols on the left with the display format on the right.

1. _____	=		A.)	Byte format
2. _____	:		B.)	symbolic format
3. _____	;		C.)	byte pointer format
4. _____	_	(under-score)	D.)	numeric format
5. _____	'	(single quote)	E.)	symbol format (bit 0 = 0).
6. _____	*		F.)	instruction format
7. _____	&		G.)	ASCII format

Given the following displays, identify the display format:

8. START/020411 _____

9. START/020411 :20411 _____

10. START/020411 ;LDA 0 GOBAK+5 _____

11. START/020411 _41 11 _____

12. START/020411 '!<11> _____

13. START/020411 *20411 _____

Now check your answers
on the following pages

DISPLAY FORMATS
QUIZ ANSWERS

Match the symbols on the left with the display format on the right.

1. D	=	A.)	Byte format
2. B	:	B.)	Symbolic format
3. F	;	C.)	byte pointer format
4. A	– (under-score)	D.)	numeric format
5. G	' (single quote)	E.)	symbol format (bit 0 = 0).
6. E	*	F.)	instruction format
7. C	&	G.)	ASCII format

Given the following displays, identify the format:

8. START/020411 Numeric format (by default, the display format is numeric).

9. START/020411 :20411 Symbolic format

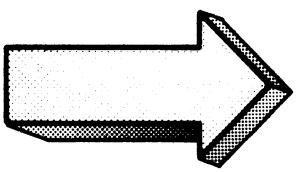
10. START/020411 ;LDA 0 GOBAK+5 Instruction format

11. START/020411 _41 11 Byte format

12. START/020411 '!<11> ASCII format

13. START/020411 *20411 Symbol format (Bit 0 = 0)

A SCORE OF 11 CORRECT ANSWERS OUT OF THE 13 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE AND RETURN TO THE TAPE.



1 SPECIFY VALUE TO FIND
(WORD REGISTER)

2 SPECIFY SEARCH INCREMENT
(INCREMENT REGISTER)

3 DETERMINE MASK VALUE
(MASK REGISTER)

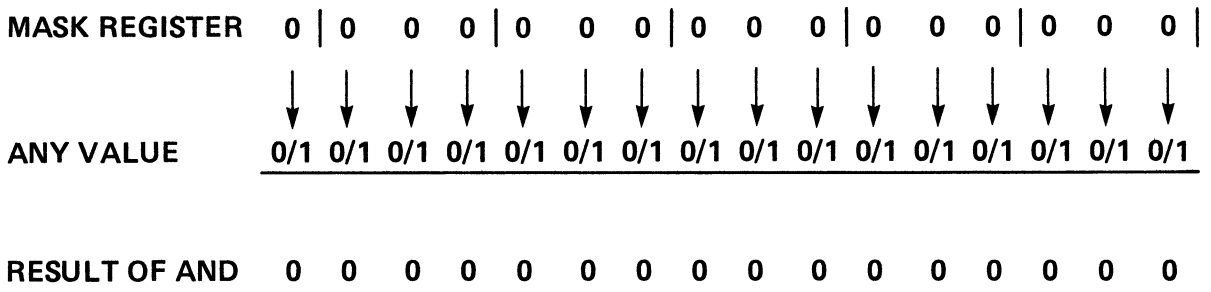
4 SPECIFY RANGE OF SEARCH
(SEARCH COMMAND)

STEPS FOR SEARCHING MEMORY

Figure 5-45

```
$W 000000<NEW LINE>

$M 000000<NEW LINE>

$J 000001<NEW LINE>



$S
+0  076112
+1  066233
+2  061511
+3  066437
+4  000000
+5  000465
+6  050577
+7  000000
\NRTN 077227
\ERTN 077235
\ICAL 075601
\ICAL+1  076144
```

$W . . . open and display the word register
$M . . . open and display the mask register
$J . . . open and display the increment register.



SEARCH ALL OF MEMORY

Figure 5-46

```
$W 000000
$M 000000
$J 000001

$;

$S
+0  DOBS 3  PTR
+1  DOBC 1  DKP
+2  DIBS 0  TTO
+3  DIC 1  ALM+3
+4   0
+5  JMP \STBL+25
+6  STA 2 \STBL+140
```

$W . . . open Word register
$M . . . open Mask register
$J . . . open Increment register

MEMORY SEARCH WITH INSTRUCTION DISPLAY FORMAT

Figure 5-47

COMMAND	DESCRIPTION	EXAMPLE
$S	search all of memory	$S search all of memory
end-address$S	search memory from location 0 to the specified address	400$S search from 0 to 400 (inclusive).
start-address<$S	search memory from the specified address to the memory limit (inclusive)	1000<$S search from 1000 to limit
start-address<end-address$S	search memory from the specified start address to the specified end address, inclusive.	100<200$S search from 100 to 200, inclusive.

SEARCH COMMANDS

Figure 5-48

```
$W 000000
$M 000000
$J 000001

15$S
+0   076112
+1   066233
+2   061511
+3   066437
+4   000000
+5   000465
+6   050577
+7   000000
\NRTN 077227
\ERTN 077235
\ICAL 075601
\ICAL+1   076144
\ICAL+2   076011
\ICAL+3   066400
```

n$S . . . search from 0 to n

```
1100(<$S
\DEB+413   010677
\DEB+414   000772
\DEB+415   024714
\DEB+416   030675
\DEB+417   034656
\DEB+420   142433
\DEB+421   117705
\DEB+422   000422
```

n<$S . . . search from n to limit.

$ (ESCAPE) . . . halt the search.

SEARCH COMMANDS

Figure 5-49

```
$W 000000
$M 000000
$J 000001

700<707$S
\DEB+213  020406
\DEB+214  107037
\DEB+215  125401
\DEB+216  002470
\DEB+217  006436
\DEB+220  000763
\DEB+221  177774
\DEB+222  000000
```

n<x$S . . . search from n to x.

Figure 5-50

WORD REGISTER

```
$W 000000 400<NEW LINE>
$W 000400 <NEW LINE>
```

MASK REGISTER

```
$M 000000 177777 <NEW LINE>
$M 177777 <NEW LINE>
```

```
$S
\STBL+331   000400
\DEB+310   000400
3512   000400
\STST+20   000400
\STEN+27   000400
\STEN+146   000400
\STEN+166   000400
```

$S . . . search from 0 to limit

LOCATION 3512 = 000400 =	0	000	000	100	000	000
MASK REGISTER = 177777 =	1	111	111	111	111	111
AND						
	0	000	000	100	000	000
WORD REGISTER = 000400 =	0	000	000	100	000	000

WORD REGISTER MATCHES LOCATION VALUE

SEARCH FOR A SPECIFIC VALUE

Figure 5-51

```
$W 000400 HALT <NEW LINE>
$W 063077<CR>

$M 177777

5000<6000$S
5017  HALT
5020  HALT
5021  HALT
5022  HALT
5023  HALT
5024  HALT
5025  HALT
5026  HALT
5027  HALT
5030  HALT
5031  HALT
5032  HALT
5033  HALT
5034  HALT
5035  HALT
5036  HALT
5444  HALT
```

SEARCH FOR A SPECIFIC VALUE (HALT)

Figure 5-52

```
$W 063077 MOVL# 0,2 SKP
$W 111111

$M 177777

$S
3174  MOVL# 0 2 SKP
3512  MOVL# 0 2 SKP
```

SEARCH FOR A SPECIFIC VALUE (111111)

Figure 5-53

SEARCH AND DISPLAY QUIZ

Circle the correct answers. A question may have more than one correct answer.

1. The default display format is:

   A. numeric
   B. symbolic
   C. byte
   D. instruction

2. To set the permanent display format to instruction format use:

   A. $=
   B. $;
   C. $:
   D. $'

3. To set the permanent display to numeric format, you use:

   A. $=
   B. $,
   C. $_
   D. $&

4. To set the display of data to symbolic format, use:

   A. $=
   B. $:
   C. $;
   D. $_

5. To set the display to ASCII format, use:

   A. $:
   B. $_
   C. $'
   D. $&

6. To set the display of data to Byte format, use:

   A. $:
   B. $_
   C. $'
   D. $&

7. To set the display of data to byte pointer format, use:

   A. $'
   B. $_
   C. $&
   D. $:

8. Assume the permanent display is numeric. To display an individual data item in instructional format, use:

   A. *
   B. I
   C. ;
   D. :

9. To display #8's data item in ASCII format, use:

   A. &
   B. ;
   C. '
   D. =

10. Assume the word and mask registers contain zeroes. The command to search and display all of memory is:

    A. $A
    B. $S
    C. 0$S
    D. S$

11. The Word and Mask registers contain zeroes. Search and display from location 0 to location 200.
    A. 0 - 200$S
    B. 0,200$S
    C. 200$S
    D. 0<200$S

12. The word and mask registers contain zeroes. Search and display from address 200 to the memory limit.
    A. 200<$S
    B. 200>$S
    C. 200$S
    D. 200>0$S

13. The word and mask registers contain zeroes. Search and display from address 200 to address 200.
    A. 220, 200$S
    B. 200<220$S
    C. 200>220$S
    D. 200 - 220$S

14. To search for one specific value, set the mask register to:

    A. 000000
    B. 111111
    C. 177777
    D. −1.

15. To search only for 000250, set the word register to:

A. 177777
B. 000000
C. 250
D. +168.

Check your answers
on the following pages

SEARCH AND DISPLAY
QUIZ ANSWERS

1. The default display format is:

(A.)	numeric	six bit, unsigned octal integers.
B.	symbolic	
C.	byte	
D.	instruction	

2. To set the permanent display format to instruction format use:

A.	$=	numeric
(B.)	$;	yes, instruction
C.	$:	symbolic
D.	$'	ASCII

3. To set the permanent display to numeric, you use:

(A.)	$=	
B.	$;	instruction
C.	$_	byte
D.	$&	byte pointer

4. To set the display of data to symbolic format, use:

A.	$=	numeric
(B.)	$:	symbolic
C.	$;	instruction
D.	$_	byte

5. To set the display to ASCII format, use:

A.	$:	symbolic
B.	$_	byte
(C.)	$'	ASCII
D.	$&	byte pointer

6. To set the display of data to Byte format, use:

A.	$:	symbolic
(B.)	$_	byte
C.	$'	ASCII
D.	$&	byte pointer

7. To set the display of data to byte pointer format, use:

A.	$'	ASCII
B.	$_	byte
(C.)	$&	byte pointer
D.	$:	symbolic

8. Assume the permanent display is numeric. To display an individual data item in instructional format, use:

A.	*	symbol with bit 0 = 0
B.	I	nothing occurs
(C.)	;	
D.	:	symbol

9. To display #8's data item in ASCII format, use:

A.	&	byte pointer
B.	;	instruction
(C.)	'	ASCII
D.	=	numeric

10. Assume the word and mask registers contain zeroes. The command to search and display all of memory is:

A.	$A	displays accumulators, plus.
(B.)	$S	
C.	0$S	searches and displays only location 0.
D.	S$	undefined symbol error $SU.

11. The Word and Mask registers contain all zeroes. Search and display from location 0 to location 200.

A.	0 - 200$S	No, evaluates (0 - 200) = 177600. Then tries to search from 0 to 177600.
B.	0,200$S	No, error: "0, 200$?"
(C.)	200$S	Yes.
(D.)	0<200$S	Yes.

12. The word and mask registers contain zeroes. Search and display from location 200 to the memory limit.

(A.)	200<$S	Yes.
B.	200>$S	Starts at zero and continues beyond 200.
C.	200$S	No, searches from 0 to 200.
D.	200>0$S	No, searches only location 0.

13. The word and mask registers contain zeroes. Search and display from address 200 to address 200.

A.	200,200$S.	No, produces error message: "200, 220$?"
(B.)	200<220$S.	Yes.
C.	200>220$S.	No, starts at 0 and goes to 220.
D.	200 - 220$S.	No, tries to cover 0 to 177760.

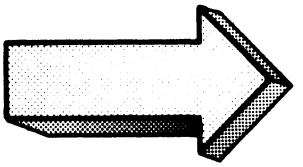
14. To search for one specific value, set the mask register to:

A.	000000	
B.	111111	
(C.)	177777	Yes, the octal value.
(D.)	−1.	Yes, the decimal value.

15. To search only for 000250, set the word register to:

A. 177777
B. 000000
C. 250 — Yes, the octal value.
D. +168 — Yes, the decimal value.

A SCORE OF 12 CORRECT ANSWERS OUT OF THE 15 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.



SEARCH AND DISPLAY
LAB EXERCISE

Abstract

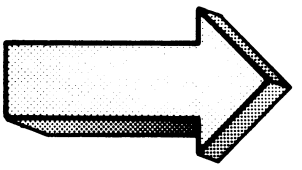
This lab exercise covers permanent display commands, single value display commands, the number register, and the search commands.

Directions

You will need the version of PROG_ONE that includes the symbol table and the Debugger. Preparation of this program was performed in the Operating Principles Lab Exercise. If you do not have a properly prepared PROG_ONE then return to the previous lab exercise and complete the preparatory steps. When ready, continue with this lab.

Once again, follow the lab steps:

1. Cover the answer;
2. Read the question.
3. Write down the answer,
4. Check the answer,
5. Perform it on your system.



1. Write a command for bringing PROG_ONE back in for debugging:

_____

_____

****

```
) DEBUG PROG_ONE )
*
```

Of course, you can use any variation of the XEQ command:

X/D PROG_ONE
XE/D PROG_ONE
XEQ/D PROG_ONE

Do it on your system.

The asterisk prompt indicates you are ready to roll.

2. Write the command for displaying location 400 in numeric format. Show the anticipated response:

_____

_____

****

```
400/020434 <NEW LINE>
```

The command 400/ opens and displays the contents of address 400.

Since numeric format is the default display, you need not enter any "$=" commands.

Do it on your system.

Close 400 with a new-line or line-feed, depending on your console keyboard.

3. Open, display, and close location 414 in numeric and instruction format.

_____

_____

_____

_____

****

```
414/000774 ;JMP LOOP+3<NEW LINE>

414/000774 <NEW LINE>
$;
414/JMP LOOP+3 <NEW LINE>
$=
```

The command 414/ opens location 414 and displays its contents as 000774. The ";" command translates 000774 into the JMP instruction.

You have your choice of doing this on one line or several.

Do it on your system.

Be sure to close the location and return the display to numeric format ($=).

4. CLEAN is a valid symbol for this program, (it was defined in a .ENT statement). Open, display, and close the location four addresses beyond CLEAN. Make the display in numeric and instruction format:

_____

_____

****

```
CLEAN+4/102400 ;SUB 0 0 <NEW LINE>
```

The command CLEAN+4/ opens location CLEAN+4 and displays its contents as 102400. The ";" command translates the contents to the subtract instruction.

CLEAN+4 is a valid expression containing the "+" operator. The display change can be made on one line. The altered format holds for one item only.

Do it on your system.

5. Display the location three words beyond START in numeric, symbolic, and numeric formats:

_____

_____

****

```
START+3/000426 :CLEAN+4   =000426 <NEW LINE>
```

The command START+B/ opens location START+3 and displays its contents as 000426.

The ":" is for symbolic format and "=" is for numeric format.

Try it on your system.

6. Open, display, and close decimal location 313. Make the display in numeric and ASCII formats.

_____

_____

****

```
313./046520 'MP <NEW LINE>
```

The command 313./ opens location 313. and displays its contents as 046520.

Decimal notation is indicated by the decimal point following 313. ASCII format is displayed by the single quote ('). The ASCII translation of 046520 is "MP".

Do it to it.

ASCII format is excellent for text strings defined in .TXT statements.

7. Open, display, and close the word register, mask register, and increment register:

_____

_____

_____

****

```
$W 000000 <NEW LINE>
$M 000000 <NEW LINE>
$J 000001 <NEW LINE>
```

$W opens and displays the word register, which is the value searched for in search sequences. $M opens and displays the mask register. $J opens and displays the search increment register. (Note that $I opens and displays the interrupt register.)

Do it on your system.

The mask and word registers should display their default values of zero. The increment register should contain a one. Insert these values if they do not already have them.

8. Use a search command to search and display addresses 425 to 430. Make the display instruction format. Write your commands and anticipated responses:

_____

_____

_____

_____

_____

_____

****

```
$;
425<430$S
CLEAN+3  JMP ERTN
CLEAN+4  SUB 0 0
CLEAN+5  SUB 1 1
CLEAN+6  SUB 2 2
```

The "$;" commands sets the display to instruction format. The "25<30" sets the search range. "$S" kicks off the search.

Try the sequence on your system. Compare the results with your program listing. They should match.

9. Use byte format in a search and display of the words from ERTN+25 to ERTN+35. Write the commands and briefly describe the anticipated responses:

****

```
$_
ERTN+25<ERTN+35$S
ERTN+25   2 136
ERTN+26   40 40
ERTN+27   40 40
ERTN+30   40 40
ERTN+31   40 40
ERTN+32   40 40
ERTN+33   40 40
ERTN+34   40 40
ERTN+35   40 40
```

Byte format is set with the "$_" command. The search commands accept the address expressions for specifying the search range. ERTN+25 <ERTN+35 is the specified search range. $S initiates the search.

Enter the sequence on your system.

The sequence of 40's constitute the sixteen spaces in the print line.

10. Search and display locations 467 to 501. Use ASCII format. Write the commands and briefly describe the results:

****

```
$'
467<501$S
ERTN+36  TH
ERTN+37  E
ERTN+40  MP
ERTN+41  /O
ERTN+42  S!
ERTN+43   H
ERTN+44  AS
ERTN+45   L
ERTN+46  AN
ERTN+47  DE
ERTN+50  D<12 >
```

ASCII format is set with the $' command. The range is from 467 to 501, (467 < 501). This is a text message in a print-line of the PROG_ONE program.

Perform it on your system.

Note that the addresses are specified in octal notation.

11. Search and display the first ten memory locations in your system in decimal numeric format. Write the *full* commands and briefly describe the results.

****

```
$=
$N 000000 1 <NEW LINE>
$N +1.11$S
+0   +31818.
+1   +27B03.
+2   +25417.
+3   +27935.
+4   +0.
+5   +363.
+6   +20863.
+7   +0.
\NRTN +32407.
\ERTN +32413.
```

$= insures that the display is in numeric format.
$N opens and displays the number register.

The number register determines octal and decimal numeric displays. When the number register contains zero, the display is octal (the default). When equal to any non-zero number, the display is decimal.

The search command says to search from 0 to 11 (first 10).

Do it on your system.

You could also use 0 <10$S for the search.

12. *A new command.* "$Z" directs the Debugger to repeat the previous search command using the same limits.

    Set the display back to octal and repeat the search of locations 0 to 10:

****

```
$N +1.0<NEW LINE>
$N 000000 <NEW LINE>
$Z
+0  076112
+1  066233
+2  061511
+3  066437
+4  000000
+5  000553
+6  050577
+7  000000
\NRTN 077227
\ERTN 077235
```

$N opens and displays the number register.

First re-set the number register to zero for octal display. The "$Z" command repeats the previous search as it was specified.

Try it.

Compare the results. They should be the same.

13. This is a little harder. Search locations 400 to 420 for the value of 006017. Show *all* commands and anticipated responses:

****

```
1  $W 000000 6017 <NEW LINE>
   $W 006017 <NEW LINE>

2  $M 000000 177777 <NEW LINE>
   $M 177777 <NEW LINE>

3  400<420$S
   START+1   006017
   LOOP+3    006017
   MESSG+2   006017
```

First set the word register to 6017, the value to find (1). Then set the mask register to 177777, so that only the specified value will be matched. (2) Close both registers.

400 to 420 specifies the search range. $S starts the search (3).

Do it.

"6017" is a JSR instruction. This instruction is used in three locations within our search range.

14. We get fancy before the last questions. Search every fifth location in memory for the value 1. (Numeric display.) Show all. Briefly describe the expected result:

****

```
1  $J 000001 5.<NEW LINE>
   $J 000005 <NEW LINE>

2  $W 006017 400<NEW LINE>
   $W 000400 <NEW LINE>

   $M 177777 <NEW LINE>

3  $S
   3600   000400
   70635  000400
   70717  000400
```

"$J" opens and displays the increment register. We inserted a decimal five, closed it, and then checked the success of the alteration (1). The word register was then modified to 400 (2). "$S" starts the search of memory (3).

Try it on your system.

Your results may easily be different. Fiddle with the increment to get a match of 400.

15. *Two new commands.* $R restarts the program from the address specified in the location register. $L opens and displays the contents of the location register.

    Open and display the location register:

    _____

    _____

****

```
$L 000400 <NEW LINE>
```

The command $L opens the location register and displays its contents. Your program will start running from the address specified in the location register.

The starting address of PAGE_ONE should be just above Page Zero (0 to 377) at address 400.

Try it on your system.

Be sure to close the register.

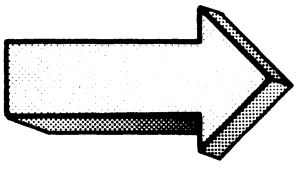
16. Now re-start PROG_ONE at the address contained in the location register. *Briefly* describe the anticipated response:

****

```
$R

                     *
                     *
                     *
                     *
                     *
                     *
                     *
                     *
                     *
                     *
                     *
                     *
                     *
                     *
                     *
                     *
                     *
                     *
                     *
                     *
                  THE MP/OS! HAS LANDED

 )
```

"$R" runs the program from the address in the location register.
Do it on your system.
Note that after the program runs, the Debugger terminates. BYE off your system.

This concludes the Search and Display Lab Exercise. Shut down your system and continue with the third segment of Module Five on the following page of your Student Guide.



BREAKPOINTS

Abstract

This segment discusses breakpoints, break proceed counters, conditional breakpoints, and accumulator displays.

Objectives

Upon completion of this segment you will be able to:

1. Set, display, and delete breakpoints in a given program.

2. Display the contents of the four accumulators, the stack register and the frame register.

3. Open, display, and modify the contents of an accumulator, stack register, and frame register.

4. Execute a given program with breakpoints and monitor the accumulators, stack registers, and frame register at each break.

5. Re-start a given program from a breakpoint.

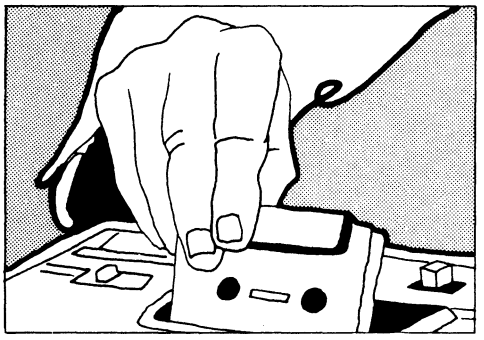
6. Set a conditional breakpoint and execute a given program with the conditional breakpoint.

7. Identify the definition of breakpoint and conditional breakpoint.

8. Match the following commands with their functions:
   a) $A
   b) n$A (n=0 to 5)
   c) $B
   d) address $B
   e) $D
   f) n$D (n=0 to 15)

Directions

Turn to figure 5-71 on the next page of your Student Guide and listen to the tape for segment three of Module Five.



1

```
) DEBUG DELTATWO )
#

$B
```

$B . . display the current breakpoints.

2

```
405$B

GOBAK+2$B
```

address $B . . . set a breakpoint at address.

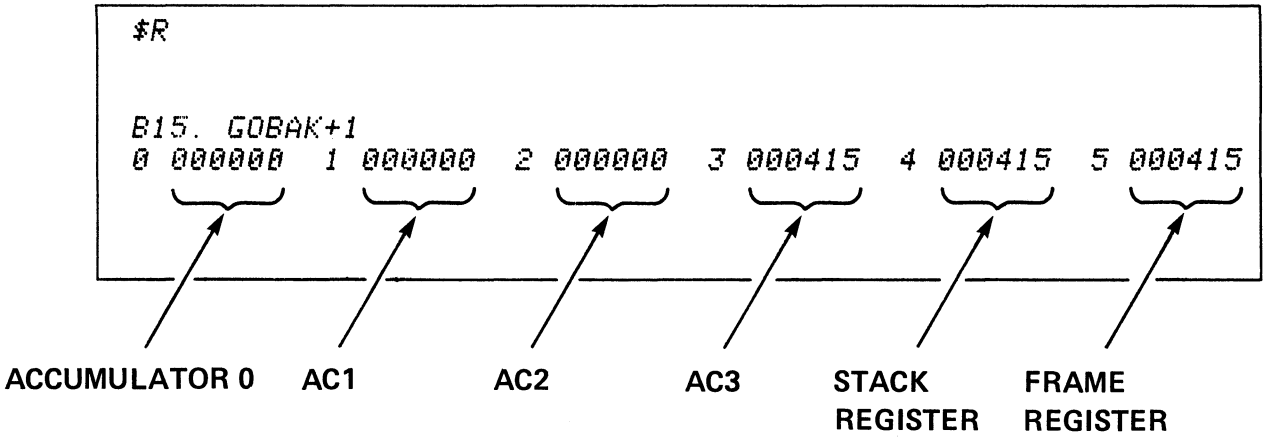
3

```
$B
B15. GOBAK+1
B14. GOBAK+2
```

Breakpoints are numbered 15. to 0.

BREAKPOINT COMMANDS

Figure 5-71



PROGRAM EXECUTION WITH A BREAKPOINT

Figure 5-72

```
$P

B14. GOBAK+2
0 000000  1 000000  2 000000  3 000415  4 000415  5 000415
```

$P . . . restart execution from current breakpoint

```
$P

)
```

PROGRAM EXECUTION FROM BREAKPOINTS

Figure 5-73

1

```
400$B
401$B
402$B
403$B
404$B
405$B

$B
B15.  START
B14.  START+1
B13.  START+2
B12.  START+3
B11.  GOBAK
B10.  GOBAK+1
```

2

```
14.$D

$B
B15.  START
B13.  START+2
B12.  START+3
B11.  GOBAK
B10.  GOBAK+1
```

n.$D . . . delete breakpoint n.

3

```
$D
$B
```

$D . . . delete all breakpoints.

DELETING BREAKPOINTS

Figure 5-74

1

```
400$B
401$B
402$B

$B
B15.  START
B14.  START+1
B13.  START+2
```

2

```
15.$Q 000001 <NEW LINE>

14.$Q 000001 <NEW LINE>

13.$Q 000001 <NEW LINE>
```

n.$Q . . . open and display the break proceed counter for breakpoint n.

3

```
15.$Q 000001  5 <NEW LINE>

15.$Q 000005 <NEW LINE>
```

BREAK PROCEED COUNTER . . . how many times to execute an instruction before stopping the program and re-entering the Debugger.

BREAK PROCEED COUNTER

Figure 5-75

1

```
$A
0 000000  1 000000  2 000000  3 000000  4 000415  5 000415
```

$A . . . display AC0, AC1, AC2, AC3, stack register and frame register.

2

```
0$A 000000<NEW LINE>
1$A 000000<NEW LINE>
2$A 000000<NEW LINE>
3$A 000000<NEW LINE>
4$A 000415<NEW LINE>
5$A 000415<NEW LINE>
```

n$A . . . open and display register n. (n is 0 to 5).

3

```
6$A?
```

ACCUMULATOR COMMAND

Figure 5-76

```
1  0$A 000000 33333<NEW LINE>
2  1$A 000000 55555<NEW LINE>
3  2$A 000000 99.<NEW LINE>
4  3$A 000000 GOBAK<NEW LINE>
5  4$A 000415 5+2<NEW LINE>
6  $A
   0 033333  1 055555  2 000143  3 000404  4 000007  5 000415
```

MODIFYING ACCUMULATORS AND REGISTERS

Figure 5-77

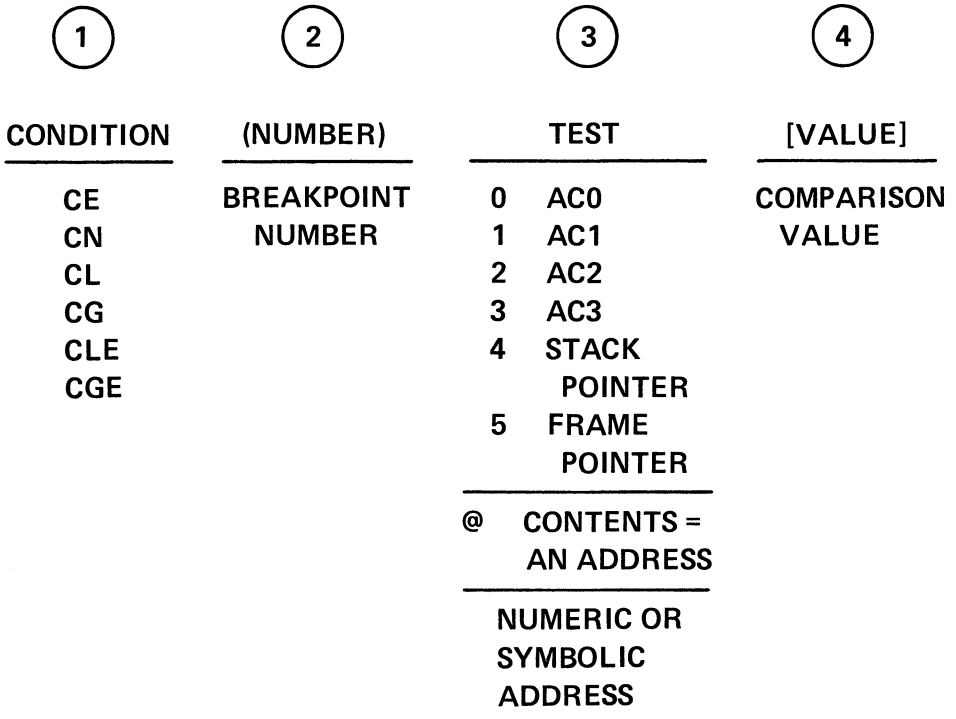


Figure 5-78

SYMBOL	MEANING
CE	Compare equal
CN	Compare not equal
CL	Compare less than
CG	Compare greater than
CLE	Compare less than or equal
CGE	Compare greater than or equal

CONDITIONAL TESTS IN
A CONDITIONAL BREAKPOINT COMMAND LINE

Figure 5-79

1
```
CE(15.)0[0]
```

Return control to the Debugger at breakpoint 15 if the contents of AC0 = 0.

2
```
CG(14.)1[177777]
```

Return control to the debugger at breakpoint 14 if the contents of AC1 are greater than minus one.

3
```
CLE(13.)@4[500]
```

Return control to the debugger at breakpoint 13 if the contents of the location pointed to by the stack pointer are less than or equal to five hundred.

Figure 5-80

1
```
CE(12.)GOBAK+3[6017]
```

Return control to the debugger at breakpoint 12 if the contents at address GOBAK+3 equal 6017.

2
```
CN(11.)410[13]
```

Return control to the debugger at breakpoint 11 if the contents of location 410 are not equal to 13.

3
```
CE(15.)400[LDA 0 GOBAK+5]
```

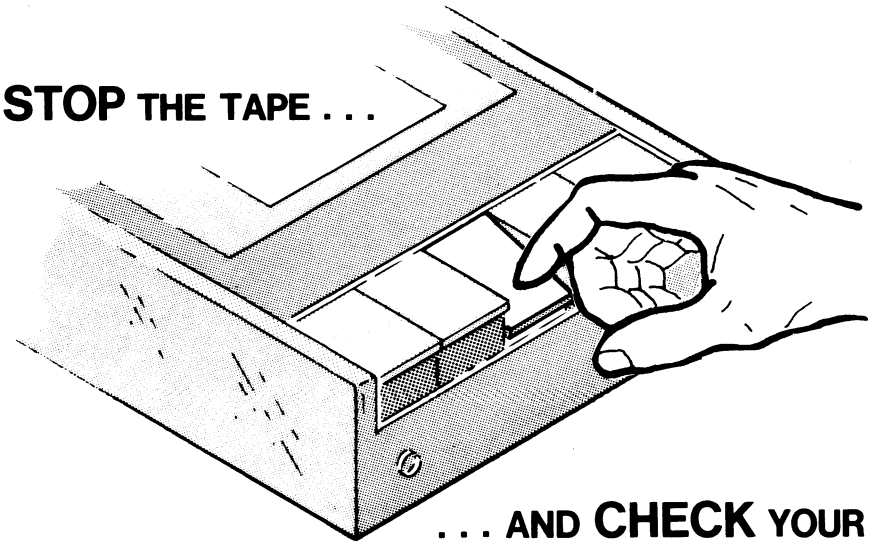
Return to the debugger at breakpoint 15 if location 400 is the instruction LDA 0, GOBAK+5.

Figure 5-81

TOPICS

- BREAKPOINTS
- BREAK PROCEED COUNTER
- CONDITIONAL BREAKPOINTS

NOW STOP THE TAPE . . .



. . . AND CHECK YOUR PROGRESS

BREAKPOINTS QUIZ

Circle the correct answers. Note that a question may have more than one correct answer.

1. When a program reaches the location of a (n) _____ , program execution stops and control transfers back to the Debugger.

   A. breakpoint
   B. break proceed counter
   C. accumulator
   D. mask register

2. A command to *display* all currently active breakpoints is:

   A. $A
   B. $B
   C. $C
   D. $D

3. A command to *set* a breakpoint at address 505 is:

   A. 505$B
   B. 505.$B
   C. B505$
   D. $B505

4. A command to *delete all* currently active breakpoints is:

   A. $A
   B. $B
   C. $C
   D. $D

5. The command to *delete* breakpoint 15. currently set at location 404 is:

   A. 15.$B
   B. 404.$B
   C. 15.$D
   D. 15$D

6. A command to *display* the four accumulators, the stack register and the frame register is:

   A. $A
   B. $B
   C. $C
   D. $D

7. A command to *open* and *display* the contents of accumulator 3 is:

   A. 3A
   B. 3$A
   C. 3A$
   D. $3A

8. AC2 contains 000000. A command sequence to open, display, insert 177777, and close accumulator 2 would look like:

   A. 2$A 177777 <NEW-LINE>
   B. 2$A 000000 177777
   C. 2$A 000000 177777 <NEW-LINE>
   D. 2$A 000000 I177777$ <NEW-LINE>

9. A command to open and display the location register is:

   A. $A
   B. $L
   C. $C
   D. $R

10. A command to restart program execution at the address contained in the location register is:

    A. $P
    B. $L
    C. $C
    D. $R

11. A command to restart program execution from the current breakpoint is:

    A. $P
    B. $L
    C. $C
    D. $R

12. The default value of a break proceed counter is initially:

    A. 0
    B. 1
    C. 177777
    D. −1

13. The break proceed counter for breakpoint 12. is set to 1. Set it to 5:

    A. 12.$A 000001 5 <NEW-LINE>
    B. 12.$B 000001 5 <NEW-LINE>
    C. 12.$Q 000001 5 <NEW-LINE>
    D. 12.$C 000001 5 <NEW-LINE>

14. Breakpoint #12's break proceed counter is set to 5. This means that breakpoint #12's location will execute_____ times before transferring control back to the Debugger.

    A. 0
    B. 1
    C. 5
    D. 12

15. The first breakpoint you assign is given the number:

    A. 0.
    B. 1.
    C. 15.
    D. 16.

16. Breakpoint 10. is assigned to address 600. A command to return control to the Debugger at breakpoint 10. if accumulator 0 equals 0 is:

    A. 0$A = 10 <NEW LINE>
    B. CE(10.) 0 [0]
    C. CE(0) 10. [0]
    D. CE(0) 0 [10.]

17. Breakpoint 13. is set at location 405. A command to return control to the debugger at breakpoint 13 if the valid symbol COUNT is less than 5 is:

    A. CL(13.) COUNT [5]
    B. 13.(COUNT) CL [5]
    C. CL(5.) COUNT [13.]
    D. 13.(CL) 5 [COUNT]

18. Breakpoint 2 is set at location 750. A command to return control to the Debugger at breakpoint 2 if the address pointed to by accumulator 3 is equal to 1 is:

    A. CE(2.) 3 [1]
    B. CE(2.) @3 [1]
    C. CE (1.) 2 [3]
    D. CE(750.) @2 [3]

Check your answers
on the following pages.

BREAKPOINTS
QUIZ ANSWERS

1. When a program reaches the location of a(n)_____, program execution stops and control transfers back to the Debugger.

   (A.) breakpoint (this assumes the break proceed counter was set to 1.)
   B. break proceed counter
   C. accumulator
   D. mask register

2. A command to *display* all currently active breakpoints is:

   A. $A displays AC0,1,2,3, stack register and frame register.
   (B.) $B
   C. $C opens & displays the carry register.
   D. $D deletes all current breakpoints.

3. A command to *set* a breakpoint at address 505 is:

   (A.) 505$B
   B. 505.$B No, sets a breakpoint at decimal 505.
   C. B505$ No, creates an error.
   D. $B505 No.

4. A command to *delete all* currently active breakpoints is:

   A. $A No, displays AC0,1,2,3, stack register and frame register.
   B. $B No, displays current breakpoints.
   C. $C No, opens & displays the carry register.
   (D.) $D

5. The command to delete breakpoint 15. currently set at location 404 is:

A.	15.$B	No, sets a breakpoint at decimal address 15.
B.	404.$B	No, sets a breakpoint at decimal location 404.
(C.)	15.$D	Yes, note the decimal notation.
D.	15$D	No, creates a "?" error.

6. A command to display the four accumulators, the stack register and the frame register is:

(A.)	$A	
B.	$B	No, displays breakpoints
C.	$C	No, displays carry register
D.	$D	No, deletes breakpoints.

7. A command to open and display the contents of accumulator 3 is:

A.	3A	No, does nothing
(B.)	3$A	Yes.
C.	3A$	No, yields "3A$?"
D.	$3A	No, gets a "?" and a "U".

8. AC2 contains 000000. A command sequence to open, display, insert 177777, and close accumulator 2 would look like:

A.	2$A 177777 <NEW-LINE>	No, skipped the current display
B.	2$A 000000 177777	No, forgot to close it.
(C.)	2$A 000000 177777 <NEW-LINE>	Yes
D.	2$A 000000 I177777$ <NEW-LINE>	No, the I...$ is an error.

9. A command to open and display the location register is:

A.	$A	No, displays AC0, 1,2,3, stack register, and frame register.
(B.)	$L	
C.	$C	No, opens & displays the carry register.
D.	$R	No, restarts program execution at address contained in location register.

10. A command to restart program execution at the address contained in the location register is:

A.	$P	No, restarts program execution from the current breakpoint
B.	$L	No, opens and displays the location register.
C.	$C	No, opens & displays the carry register.
(D.)	$R	Yes.

11. A command to restart program execution from the current breakpoint is:

(A.)	$P	
B.	$L	No, opens & displays the location register.
C.	$C	No, opens & displays the carry register.
D.	$R	No, restarts program execution from the address in the location register.

12. The default value of a break proceed counter is initially:

A.	0	
(B.)	1	Automatically set to 1 with the creation of each breakpoint
C.	177777	
D.	−1	

13. The break proceed counter for breakpoint 12. is set to 1. Set it to 5:

A.	12.$A 000001 5 <NEW-LINE>	No, gets an error for 12.$A.
B.	12.$B 000001 5 <NEW-LINE>	No, first sets a breakpoint at 1210.
(C.)	12.$Q 000001 5 <NEW-LINE>	Yes, now set to 5.
D.	12.$C 000001 5 <NEW-LINE>	No, gets "12.$C?"

14. Breakpoint #12's break proceed counter is set to 5. This means that breakpoint #12's location will execute_____ times before transferring control back to the Debugger.

A. 0
B. 1
(C.) 5
D. 12

15. The first breakpoint you assign is given the number:

A. 0.
B. 1.
(C.) 15. Breakpoints are numbered in reverse order, in decimal 15 to 0.
D. 16.

16. Breakpoint 10. is assigned to address 600. A command to return control to the Debugger at breakpoint 10. if accumulator 0 equals 0 is:

A. 0$A = 10 <NEW LINE> No, opens and displays AC0, then sets it to 10.
(B.) CE(10.) 0 [0] Yes. condition-breakpoint-AC-value.
C. CE(0) 10. [0] No, returns to Debugger at breakpoint 0, if address 10. contains a 0.
D. CE(0) 0 [10.] No, returns to Debugger at breakpoint 0, if address 0 contains a 10. However, breakpoints cannot be set at locations 0 or 1.

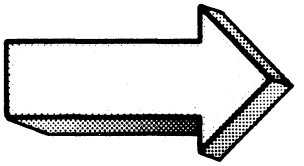
17. Breakpoint 13. is set a location 405. A command to return control to the debugger at breakpoint 13 if the valid symbol COUNT is less than 5 is:

(A.) CL(13.) COUNT[5]
B. 13. (COUNT) CL [5] No, "13.(" produces "?"
C. CL(5.) COUNT [13.] No, returns to Debugger at breakpoint 5. if address COUNT is less than a decimal 13.
D. 13.(CL) 5 [COUNT] No, "13.(" produces "?"

18. Breakpoint 2 is set at location 750. A command to return control to the Debugger at breakpoint 2 if the address pointed to by accumulator 3 is equal to 1 is:

A.	CE(2.) 3 [1]	No, returns to Debugger at breakpoint 2. if AC3 contains a 1.
(B.)	CE(2.) @3 [1]	
C.	CE(1.) 2 [3]	No, returns to Debugger at breakpoint 1. if AC2 has a 3.
D.	CE(750.) @2 [3]	No, the "750" produces an error.

A SCORE OF 16 CORRECT QUESTIONS OUT OF THE 18 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.



DEBUGGER LAB EXERCISE

Abstract

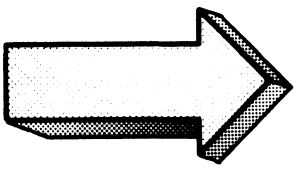
This lab covers all of the commands discussed in the three segments of Module Five. The lab is applicable for programmers of Assembly as well as MP/Fortran IV.

Directions

Once again, you will need PROG_ONE to complete questions in this lab. Preparation of PROG_ONE is detailed in the Operating Principles Lab Exercise. You are ready for this lab when PROG_ONE is bound with the libraries and Debugger into the program file PROG_ONE.PR.

As always:

1. Cover the answers,
2. Read the question,
3. Write the answer,
4. Check the answer,
5. Enter the answer on your system.



Additional information on the Debugger may be found in the *MP/OS Utilities Reference Manual.*

****

Note for MP/FortranIV programmers:

For your convenience, the command sequence for preparing a Fortran program (e.g. MONEY):

1. XEQ FORT4/L=MONEY.LS MONEY

   Compile MONEY. Make MONEY.LS the compiler listing and MONEY.SR the compiler output.

2.)XEQ MASM/PS=FORT4.PS/F/L=MONMASM.LS MONEY.

   Assemble MONEY.SR and output MONEY.OB. Equivalence internal values (/F). Make MONMASM.LS the assembler listing (extremely helpful).

3.) XEQ BIND/U/D/L=MONEYMAP MONEY FORT4.LB

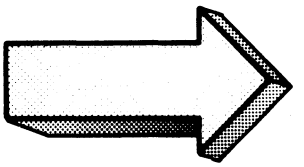
   Bind MONEY.OB with FORT4.LB and the Debugger. Include the user symbols (/U). Make MONEYMAP the load map. MONEY.PR is the program file.

4.)DEBUG MONEY

   Invoke the Debugger with the MONEY.PR file.

****

When ready, start the Lab



1. Invoke the Debugger for PROG_ONE. Write in your command and anticipated response before checking the answer or entering it on your system.

_____

_____

****

```
) DEBUG PROG_ONE )
*
```

The star (*) is the Debugger's initial prompt.

Do it on your system.

2. Write the commands for displaying the location register, accumulators, stack register, and frame register. Estimate the responses:

_____

_____

_____

****

```
$L 000400<NEW LINE>

$A
0 000000  1 000000  2 000000  3 000000  4 000503  5 000503
```

$L opens and displays the location register (the program's starting address). The New-line closes the register. $A displays ACs 0 to 3, the stack register, and the frame register.

Do it on your system.

3. Write the commands to search through your program and display the instruction "LDA 2, 55":

****

```
2  $M 000000 177777 <NEW LINE>
   $M 177777 <NEW LINE>

1  $W 000000 LDA 2 55 <NEW LINE>
   $W 030055 <NEW LINE>

3  400<500$S
   MESSG+1   030055
```

First set the word register to the value to find (LDA 2,55). Either the instruction or its octal equivalent will do. Second, set the mask register to reject all other combinations. Then search the program area from 400 to 503.

Try it on your system.

The load instruction is at MESSG+1.

4. Write the commands to search through locations 3000 to 4000 to find the occurrences of the all-one bit pattern (177777):

_____

_____

_____

_____

_____

_____

****

2
```
$M 177777<NEW LINE>
```

1
```
$W 030055 177777<NEW LINE>
$W 177777 <NEW LINE>
```

3
```
3000<4000$S
3600   177777
3601   177777
```

First set the word register to 177777, the value to find. Next, set the mask register to the full mask (it was already set). Then search from 3000 to 4000.

Do it on your system.

Be sure to close the registers with new-line or line-feeds, depending on your console.

5. Write the commands to display all of the accumulators, the stack register, and the frame register. Then write the commands to open, display, and close the above words.

_____

_____

_____

_____

_____

_____

_____

_____

****

```
$A
0 000000  1 000000  2 000000  3 000000  4 000503  5 000503
0$A 000000 <NEW LINE>
1$A 000000 <NEW LINE>
2$A 000000 <NEW LINE>
3$A 000000 <NEW LINE>
4$A 000503 <NEW LINE>
5$A 000503 <NEW LINE>
```

$A displays the accumulators and registers.
n$A opens and displays the n accumulator or register.

Be sure to close each word.

Try it on your system.

6. Open and display AC3. Set it to 111111. Close it. Then re-set it to its original value. Show all:

_____

_____

_____

_____

****

```
3$A 000000 111111<NEW LINE>
3$A 111111 <NEW LINE>

3$A 111111 0 <NEW LINE>
3$A 000000 <NEW LINE>
```

3$A opens and displays accumulator 3.
AC3 contains all zeroes. Typing 111111 immediately after inserts this value.

Do it on your system.

7. Execute PROG_ONE. Briefly describe the anticipated responses:

____

____

____

****

```
$R

                          *
                          *
                          *
                          *
                          *
                          *
                          *
                          *
                          *
                          *
                          *
                          *
                          *
                          *
                          *
                          *
                          *
                          *
                          *
                          *
                      THE MP/OS! HAS LANDED

)
```

I'm impressed.

$R restarts program execution from the address in the location register.

Perform it on your system.

Note that the program terminates and CLI returns. Re-invoke the Debugger for PROG_ONE.

```
) DEBUG PROG_ONE )
*
```

Everything reverts to its default value.

8. Open, display, and close locations 400, ERTN+22, and CLEAN. Make the displays numeric, instruction, and ASCII:

_____

_____

_____

****

```
400/020434 ;LDA 0 ERTN+3   '!<34 ><NEW LINE>
ERTN+22/020052 ;LDA 0 \STBL+5   ' * <NEW LINE>
CLEAn/020411 ;LDA 0 ERTN+2   '!<11 ><NEW LINE>
```

400/ opens location 400 and displays its contents (numeric 020434). ";" translates this to the LDA instruction.

Numeric is the default display format. Instruction format is produced by the semi-colon (;) and ASCII by the single quote (').

Try it.

The "*" is one of twenty printed as output of the program PROG_ONE.

9. Change location 455 from 24 to 2. Restart the program. Describe the anticipated response:

_____

_____

_____

_____

****

```
455/000024 2 <NEW LINE>

$R

                    *
                    *
              THE MP/OS! HAS LANDED

)
```

"455/" opens and displays the location. Insert the 2 immediately after the contents. Close the word.

$R runs the program with the modified count.

Do it!

Only two stars are printed because 455 has the loop counter. 24 octal is 20 decimal. We previously printed 20 stars, now only 2 print.

10. Re-invoke the Debugger for PROG_ONE and check the contents of location 455. Write it all down:

_____

_____

_____

****

```
) X/D PROG_ONE ↓
*
455/000024 <NEW LINE>
```

455/ opens location 455 and displays its contents.

Note that 455 reverted to its original setting of 24 (20 in decimal).

Do it on your system.

Modifications are temporary, they are re-set with each invocation of the Debugger.

11. Make LOOP+6 a conditional breakpoint that will return to the Debugger only when the loop counter (TWEN, at ERTN+24) is ten. Write it down first:

_____

_____

_____

_____

****

```
LOOP+6$B
$B
B15.  LOOP+6

CE(15.)ERTN+24[10]
```

First set the breakpoint at LOOP+6. Then define it as conditional. CE is the condition (equal), 15. is the breakpoint number, ERTN+24 (or 455) is the location to be tested, and 10 is the comparison value.

Enter this on your system.

12. Now restart program execution from the address in the location register. Describe what you think will occur:

_____

_____

_____

****

```
$R

                          *
                          *
                          *
                          *
                          *
                          *
                          *
                          *
                          *
                          *
                          *

B15  LOOP+6
0 000002  1 001102  2 000027  3 000503  4 000503  5 000503
```

$R starts PROG_ONE. Eleven stars are printed until TWEN (ERTN + 24) is down to 10. The Debugger returns. The breakpoint number and address are displayed. AC0, 1, 2, 3, the stack register, and frame register are displayed.

Try it on your system.

13. The Program has halted and the Debugger is in control. What are the contents of the location register and TWEN (ERTN + 24 or 455)?:

_____

_____

****

```
$L 000413 <NEW LINE>
ERTN+24/000010 <NEW LINE>
```

$L opens and displays the current location. We stopped at address 413, which is also referenced by LOOP+6. TWEN is not a valid symbol. ERTN + 24 or 455 open and display TWEN's value. The program stopped with TWEN equal to 10, as instructed.

Display these on your system.

Make sure you close both locations.

14. Re-start program execution. Describe what you think will happen:

_____

_____

_____

****

```
$P
     *
                    *
                    *
                    *
                    *
                    *
                    *
                    *
                    *
              THE MP/OS! HAS LANDED

)
```

$P restarts PROG_ONE from the breakpoint. The remaining stars (9) are printed, then the message

Do it on your system.

The remaining messages print and PROG_ONE terminates.

15. Invoke the Debugger for PROG_ONE for the last time. Set two breakpoints: one at LOOP+6 and the other at MESSG+1. Show all:

****

```
) XE/D PROG_ONE \
*
LOOP+6$B
$B
B15.  LOOP+6
MESSG+1$B
$B
B15.  LOOP+6
B14.  MESSG+1
```

LOOP+6$B sets a breakpoint at the location referenced by LOOP+6.

The $B command verifies the breakpoints by displaying their numbers and addresses.

Set it up on your system.

16. Make breakpoint 15 conditional. Return to the Debugger only when the loop counter (TWEN, located at ERTN+24) is equal to 5. Make breakpoint 14 unconditional:

****

```
CE(15.)ERTN+24[5]
```

CE sets the comparison to equal. We are stopping at breakpoing 15. ERTN+24 is the location tested. [5] specifies the comparison value.

Try it on your system.

Remember that 15. is in decimal notation.

17. Now start PROG_ONE. Describe what should occur: (Refer to your listing for aid.)

_____

_____

_____

****

```
$R
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *

B15. LOOP+6
0 000002  1 001102  2 000027  3 000503  4 000503  5 000503
```

$R starts PROG_ONE rolling.

Do it.

Fifteen stars (*) should print out before the Debugger returns and displays the breakpoint and accumulators.

Display the location register and loop counter:

```
$L 000413 <NEW LINE>

ERTN+24/000005 <NEW LINE>
```

18. Restart PROG_ONE. What will happen?

_____

_____

_____

****

```
$P
                        *
                        *
                        *
                        *
                        *

B14. MESSG+1
0 000002  1 001136  2 000027  3 000503  4 000503  5 000503
```

$P restarts execution from a breakpoint.

Try it.

The stars should finish before another halt, at breakpoint 14. What are the contents of the location register and loop counter now?

```
$L 000416 <NEW LINE>

ERTN+24/000000 <NEW LINE>
```

19. Before the "MP/OS. . ." message prints out, make a change. Modify "MP/OS" to "BARRY". Take it one step at a time. (Your listing helps here):

_____

_____

_____

_____

_____

_____

_____

_____

_____

****

```
ERTN+40/046520 'MP "BA <NEW LINE>

ERTN+41    027517 '/O "RR <NEW LINE>

ERTN+42    051441 'S! "Y! <NEW LINE>

ERTN+43    020110 <NEW LINE>
$'
ERTN+40<ERTN+42$S
ERTN+40  BA
ERTN+41  RR
ERTN+42  Y!
```

A search of your program file or the listing shows the text message "MP/OS" starts at ERTN + 40. Open it. Display it in ASCII. Then type "BA" to change MP to BA. Close the location and continue with the remaining locations.

Perform this operation on your system.

20. Last one. Run PROG_ONE. What will occur now?

_____

_____

_____

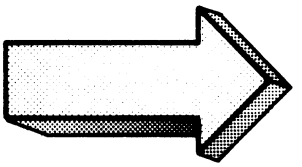
****

```
$P
                    THE BARRY! HAS LANDED

)
```

$P runs PROG_ONE. The modified message is printed. The program terminates and CLI returns.

This concludes the Debugger Lab Exercise and Module Five. At this point you should be able to invoke the Debugger and use it to interactively debug a given program.

Now proceed to MODULE SIX.



MODULE SIX
SYSTEM MAINTENANCE

MODULE SIX

SYSTEM MAINTENANCE

Abstract

This module is divided into three segments:

- Disk initialization - software formatting for disc media utilizing the DINIT utility program
- FIXUP - software repair of disc media utilizing the FIXUP utility program.
- MOVE - transfer of files from one directory to another utilizing the MOVE utility program.

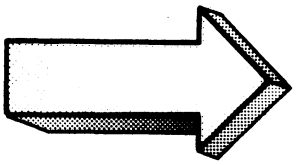
Objectives

Upon completion of this module you will be able to:

1. Execute the disc initializer utility to:
   a) software format a disc or diskette;
   b) install system programs on a disc or diskette

2. Execute the FIXUP utility to software repair a disc or diskette.

3. Use the MOVE utility to transfer files from one given directory to another.



DISK INITIALIZATION

Abstract

The Disk Initializer utility program (DINIT) is used to build an MP/OS file structure on a disk or diskette and to optionally install three system programs. This segment discusses the procedures involved in disk initialization under MP/OS.

Objectives

Upon completion of this segment you will be able to:

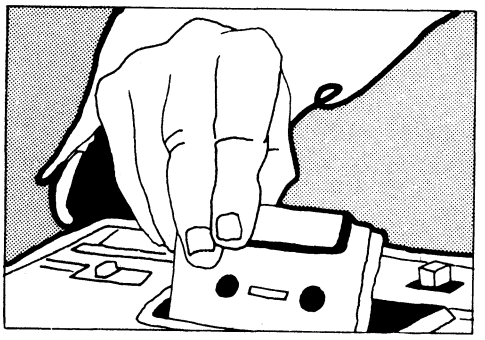
1. Identify the functions of DINIT;

2. Initialize (software format) a disk or diskette.

3. Install system files on an initialized disk or diskette.

Directions

Turn to Figure 6-1 on the next page of the Student Guide and listen to the audio-tape for the first segment of Module Six.



DISK INITIALIZER

- builds storage format or structure
- pre-requisite for data storage
- number of bad blocks
- location of bad blocks
- location of free space
- name of all files
- location of all files
- install device bootstrap loader
- install MP/OS
- install FIXUP.

DINIT

Figure 6-1

1. LOAD THE DISK OR DISKETTE

2. SOFTWARE FORMAT THE DISK

3. INSTALL SYSTEM FILES

DINIT STEPS

Figure 6-2

```
) XEQ  DINIT/SWITCHES
  XE
  X              /D    ALWAYS REBUILD
                       LABEL BLOCKS

                 /V    ASK VERIFICATION OF
                       FILE INSTALLATION
```

CLI COMMAND LINE TO INVOKE DINIT

Figure 6-3

```
) K DINIT⟩
MP/OS  DISK INITIALIZER  REV 1.00

Disk unit name ?
```

DINIT CALL AND TITLE MESSAGE

Figure 6-4

VALID DISK UNIT NAMES

@DPX(0-7)	6038 diskette
* @DPY(0-7)	60 XX quad density diskette
@DPD(0-7)	60XX cartridge disk
@DPH(0-7)	12.5 MB disk

```
1   Disk unit name ? XXXXX⟩

    Possible disk names :

            @DPX
            @DPH
            @DPD
            @DPY

2   Disk unit name ? @DPX⟩

    Unit # required after disk name

3   Disk unit name ? @DPH1⟩
    Error: File does not exist


    Initialize another disk ? YES⟩

    Disk unit name ?
```

*Informational purposes only.

DINIT DIALOGUE

Figure 6-5

```
') XEQ DINIT/U⟩
MP/OS  DISK INITIALIZER  REV 1.00

1   Disk unit name ? @DPX1⟩

2   MP/OS  format the disk ? YES⟩

    Formatting destroys previous disk structure,

3           Disk I.D.: "NEWSCRATCH"

    Continue formatting disk ? YES⟩
4   Disk I.D. (0 to 15 chars) ? SCRATCH⟩

    The following patterns are available:

            #1 - 155555
            #2 - 133333
5           #3 - 066666
            #4 - 000000
            #5 - 177777

    Run which patterns ? 4,5⟩

6     -- Running pattern #4  (000000)
      -- Running pattern #5  (177777)

7   Bad Blocks = 0
8   Maximum number of files ? 400⟩
      Rounded up to 1016.

9   ** Disk is software formatted **

10  Install a bootstrap ? YES⟩
    Bootstrap pathname (NL for BOOTDPX.SA) ? ⟩
    ** Disk bootstrap installed **

11  Install FIXUP ? YES⟩
    FIXUP pathname (NL for FIXUP.SA) ? ⟩
    ** FIXUP installed **

12  Install  MP/OS ? YES⟩
    System pathname (NL for MP/OS .SY) ?⟩
    **  MP/OS installed **


13  Initialize another disk ? NO⟩

    )
```

DINIT DIALOGUE

Figure 6-6

```
) MOUNT @DPX1↓
SCRATCH
) DISK/F @DPX1↓
@DPX1
                  517   Blocks available
                   99   Blocks in use
                 1016   Files can be created
                    2   Recoverable errors

) FILESTATUS/AS @DPX1:+↓

)
```

ACCESSING AN INITIALIZED DISKETTE

Figure 6-7

```
) XEQ DINIT ↓
MP/OS  DISK INITIALIZER  REV 1.00

Disk unit name ? @DPX1 ↓

MP/OS  format the disk ? YES ↓
Disk I.D. (0 to 15 chars) ? BLANKDISK ↓

The following patterns are available:

        #1 - 155555
        #2 - 133333
        #3 - 066666
        #4 - 000000
        #5 - 177777

Run which patterns ? 1,2 ↓

  -- Running pattern #1  (155555)
  -- Running pattern #2  (133333)

Bad Blocks = 0
Maximum number of files ? 50 ↓
  Rounded up to 1016.

** Disk is software formatted **

Install a bootstrap ? NO ↓

Install FIXUP ? NO ↓

Install  MP/OS  ? NO ↓


Initialize another disk ? NO↓

)
```

INITIALIZING A BLANK DISK

Figure 6-8

```
) MOUNT @DPX1:
BLANKDISK
)



) DISK/B @DPX1:
@DPX1
                309248   Bytes available
                  6144   Bytes in use
                     3   Recoverable errors

)



) DISK/F @DPX1:
@DPX1
                   604   Blocks available
                    12   Blocks in use
                  1016   Files can be created
                     3   Recoverable errors

)
```

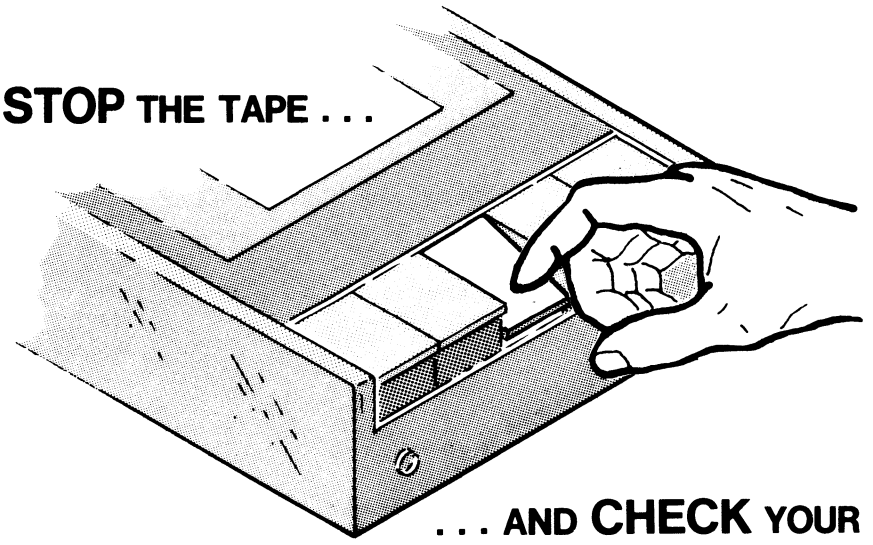
ACCESSING A "BLANK" DISKETTE

Figure 6-9

TOPICS

- DINIT
- MP/OS UTILITIES REFERENCE MANUAL
- INVOKE
- SURFACE ANALYSIS
- SYSTEM INSTALLATION

NOW STOP THE TAPE . . .



. . . AND CHECK YOUR PROGRESS

DINIT QUIZ

Circle the correct answer. A question may have more than one correct answer.

1. A command for invoking the Disk Initializer is:

   A. DINIT ↵
   B. X DKINIT ↵
   C. DISK ↵
   D. XEQ DINIT ↵

2. The "/V" switch on the DINIT command requests DINIT to ask whether you want the:

   A. default MP/OS
   B. default bootstrap loader
   C. default FIXUP
   D. default DINIT.

3. The "/D" switch on the DINIT command directs the disk initializer to:

   A. request confirmation before loading the default FIXUP.
   B. destroy the previous disk structure without confirmation.
   C. destroy the previous disk structure, but only after confirmation.
   D. display the DISKSTATUS after initialization.

4. The Disk initializer is a:

   A. pre-requisite for data storage.
   B. surface analyzer.
   C. disk structure formatter
   D. software formatter.

5. The Disk initializer records:

   A. the number of bad disk blocks.
   B. the location of bad disk blocks.
   C. the number of available blocks.
   D. the location of available blocks.

6. DINIT may be requested to install:

   A. FIXUP
   B. Bootstrap loader
   C. MP/OS
   D. no files.

7. Valid disk unit names may include:

   A. DPX1
   B @DPX
   C. @DPX1:
   D. @DPX1

8. The maximum number of files is allocated in multiples of:

   A. 10
   B. 1000
   C. 1016
   D. 32,576

9. A valid disk I.D. is:

   A. SCRATCHDISKFOUR
   B. DISK
   C. SCRATCH_DISKFOUR
   D. 4DISK

10. The three files that may be installed by DINIT (FIXUP, BOOT, and MP/OS) occupy about:

    A. 3 blocks.
    B. 12 blocks.
    C. 100 blocks.
    D. 1016 blocks.

11. You decide to forego installation of the three files (FIXUP, BOOT, MP/OS). DINIT uses up about:

    A. 0 blocks.
    B. 12 blocks.
    C. 100 blocks.
    D. 1016 blocks.

Check your answers
on the following pages.

DINIT QUIZ
ANSWERS

1. A command for invoking the Disk Initializer is:

   A. DINIT ↵
      Error: unknown command or macro.
   B. X DKINIT ↵
      Program name: DKINIT.PR error: file does not exist.
   C. DISK ↵
      No, this is the abbreviated DISKSTATUS command.
   (D.) XEQ DINIT ↵

2. The "/V" switch on the DINIT command requests DINIT to ask whether you want the:

   A. default MP/OS
      Always asked, with or without the "/V" switch.
   (B.) default bootstrap loader
   (C.) default FIXUP
   D. default DINIT.
      No, not installed with DINIT. Use MOVE or COPY

3. The "/D" switch on the DINIT command directs the disk initializer to:

   A. request confirmation before loading the default FIXUP.
      No, this occurs with "/V"
   (B.) destroy the previous disk structure without confirmation.
   C. destroy the previous disk structure, but only after confirmation.
      No, this occurs when the "/D" switch is left off.
   D. display the DISKSTATUS after initialization.
      No, requires the DISK command.

4. The Disk initializer is a:

(A.) pre-requisite for data storage.
(in an MP/OS environment)
(B.) surface analyzer.
(determines bad blocks)
(C.) disk structure formatter
(D.) software formatter
(C & D are equivalent)

5. The Disk initializer records:

(A.) the number of bad disk blocks.
(B.) the location of bad disk blocks.
(C.) the number of available blocks.
(D.) the location of available blocks.

6. DINIT may be requested to install:

(A.) FIXUP
(B.) Bootstrap loader
(C.) MP/OS
(D.) no files.
(answer "no" to A,B, & C)

7. Valid disk unit names may include:

A. DPX1
Valid name requires the "@" prefix to distinguish it as a device.
B. @DPX
disk requires unit #
C. @DPX1:
Identifies the directory on @DPX1.
(D.) @DPX1
O,K. for the diskette in the secondary drive.

8. The maximum number of files is allocated in multiples of:

   A. 10
   B. 1000
   (C.) 1016
   D. 32,576

      Maximum number that can be requested. Provides excellent use of disk space but slowest access times.

9. A valid disk I.D. is:

   (A.) SCRATCHDISKFOUR

      0 to 15 characters

   (B.) DISK

      Not a reserved name.

   C. SCRATCH_DISKFOUR

      Too long.

   D. 4DISK

      Cannot begin with a digit.

10. The three files that may be installed by DINIT (FIXUP, BOOT, and MP/OS) occupy about:

    A. 3 blocks.
    B. 12 blocks

       12 blocks are used up without the 3 files.

    (C.) 100 blocks

       3 files require about 87, Formatter sets up about 12.

    D. 1016 blocks

       Maximum file count allocated in multiples of 1016.

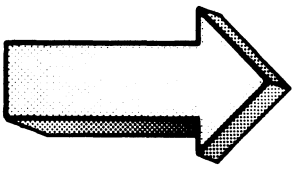
11. You decide to forego installation of the three files (FIXUP, BOOT, MP/OS). DINIT uses up about:

    A. 0 blocks.
    (B.) 12 blocks.

       For tracking bad blocks, free space, root directory, etc.

    C. 100 blocks.
    D. 1016 blocks.

A SCORE OF 9 CORRECT QUESTIONS OUT OF THE 11 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.



DINIT LAB

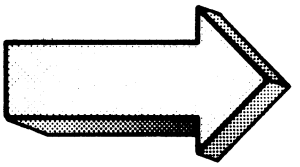
Abstract

This exercise takes you through a disk initialization sequence.

Directions

You need the program DINIT.PR on your system to complete this exercise. The CLI FILESTATUS command will verify the existence of the DINIT program. You also need a second device to load the disc medium to be initialized. Load the device and you are ready for the lab. As always:

1. Cover the answers;
2. Read the question;
3. Write the answer;
4. Check the answer;
5. Enter the response on your system.



1. Write the command to invoke the disk initializer and install the default files. Show the anticipated responses:

_____

_____

_____

****

```
) XEQ DINIT)
MP/OS  DISK INITIALIZER  REV 1.00

Disk unit name ?
```

XEQ DINIT invokes the Disk Initializer utility program.

The question specifies no switches.

Try it on your system.

2. Insert a valid disk unit name for your DINIT. What happens next?

_____

_____

_____

_____

****

```
Disk unit name ? @DPX1

MP/OS  format the disk ? YES
Disk I.D. (0 to 15 chars) ?


MP/OS  format the disk ? YES
```

@DPX1 is valid for our example system, configured with a dual diskette drive.

Enter the appropriate response for your system. (See figure 6-5 for assistance or force DINIT help by answering with a new-line. It will give you a list of valid names.)

If you get any errors, check the spelling of the disk unit name.

Answer "Yes" to "Format the Disk?"

3. Give your new medium the name "TEST" when DINIT asks for the I.D. Describe what follows:

_____

_____

_____

_____

_____

_____

_____

_____

****

```
Disk I.D. (0 to 15 chars) ? TEST
The following patterns are available:

        #1 - 155555
        #2 - 133333
        #3 - 066666
        #4 - 000000
        #5 - 177777

Run which patterns ?
```

The Disk may go nameless (0 characters) or be identified in up-to 15 characters.

Enter it on your system.

The bit patterns for surface analysis follow the Disk I.D.

4. Use patterns "155555" and "177777" for surface analysis. What should occur?

_____

_____

_____

_____

_____

****

```
Run which patterns ? 1 5 \

  -- Running pattern #1  (155555)
  -- Running pattern #5  (177777)

Bad Blocks = 0
Maximum number of files ?
```

The system reports the pattern being run. When done, it reports the number of bad blocks.

Try it on your system.

This takes a little time, so be patient. DINIT displays the pattern it is using. If your media has too many bad blocks, DINIT will issue an error message.

5. Request 2000 files as your maximum. What happens next?

_____

_____

_____

_____

****

```
Maximum number of files ? 2000
  Rounded up to 2032.

** Disk is software formatted **

Install a bootstrap ?
```

The maximum number of files is set at 2032.

Try it.

You are O.K. if DINIT displays the message "Disk is software formatted".

6. Install all three system files. Show the dialogue:

****

```
Install a bootstrap ? YES )
** Disk bootstrap installed **

Install FIXUP ? YES )
** FIXUP installed **

Install MP/OS  ? YES )
System pathname (NL for MP/OS.SY) ? )
** MP/OS  installed **


Initialize another disk ? NO )

)
```

Respond with YES for each question.

Do it on your disk.

We do not want to initialize another disk so answer no to the last question and return to CLI.

7. Access your newly initialized media and determine the available space:

_____

_____

_____

_____

_____

_____

_____

****

```
) MOUNT @DPX1 \
TEST
) DISK/B @DPX1 \
@DPX1
                264192   Bytes available
                 51200   Bytes in use
                     1   Recoverable errors

) DISK/F @DPX1 \
@DPX1
                   516   Blocks available
                   100   Blocks in use
                  2032   Files can be created
                     1   Recoverable errors

)
```

MOUNT @DPX1 brings the secondary diskette on-line and ready for software access.

Perform it on your system.

You can also "DIR" to @DPX1:, copy files to it, and in general use @DPX1: as necessary.

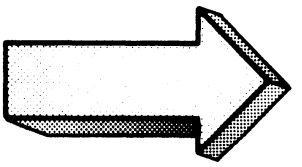
8. End of the line. Dismount the medium and BYE off your system.

****

```
) DISMOUNT @DPX1 ↓
) BYE ↓
MP/OS  CLI Terminating

System shutdown
056715
!
```

Do it.

This concludes the DINIT Lab Exercise. Shut down your system. Continue with the next segment of Module Six on the next page of your Student Guide.



MOVE UTILITY

Abstract

The MP/OS MOVE utility program provides a method for transferring files from one directory to another. The MOVE utility is an excellent means of backing-up your files.

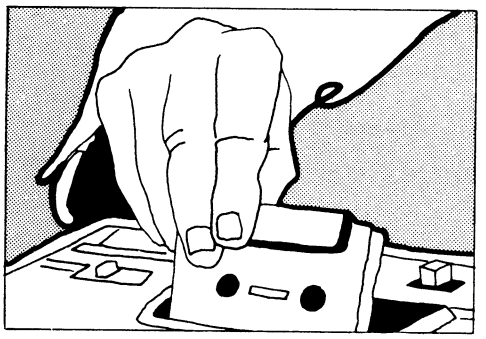
Objectives

Upon completion of this segment, you will be able to:

1. Write and execute the MOVE command line for solving a given file transfer problem.
2. Identify the function of the MOVE command switches.
3. Back-up a series of files.

Directions

Turn to Figure 6-18 on the next page of the Student Guide and listen to the audiotape for the second segment of Module Six.



```
XEQ MOVE/SWITCHES          DIRECTORY          FILE(S)
                                |                 |
XE              /L      DEFAULT:DESTINATION   FILES TO
X               /V                            BE MOVED
                /L=file                       DEFAULT # (ALL)
                /DELETE
                /RECENT
                /AFTER/TLM=DD-MMM-YY:hh:mmm:ss
                /BEFORE/TLM=DD-MM-YY:hh:mmm:ss
                /FROM
```

MOVE COMMAND LINE SYNTAX

Figure 6-18

SWITCH	FUNCTION
/FROM	Move files from the specified directory to the current directory
/DELETE	If "FILE" exists in the destination directory, delete it and then move "FILE" there from the source directory.
/L	List names of files moved at the printer
/L=LIST-FILE	List names of files moved in "LIST-FILE"
/RECENT	If "FILE" already exists in the destination directory move "FILE" only if the version in the source directory is more recent.
/V	List file names at the users console as each file is moved.
/AFTER/TLM=DD-MMM-YY:HH:MM:SS	Move only files last modified after time and date specified.
/BEFORE/TLM=DD-MMM-YY:HH:MM:SS	Move only files last modified before time and date specified.

MOVE COMMAND SWITCHES

Figure 6-19

1

```
) XEQ MOVE TWODIR SAMPLE↓

)
```

RESULT

SAMPLE ⟶ TWODIR

2

```
) XEQ MOVE/V TWODIR SAMPLE↓
@DPX0:SAMPLE

)
```

SAMPLE ⟶ TWODIR
CONSOLE DISPLAY

/V . . . list moved files on the console.

3

```
) XEQ MOVE/L TWODIR SAMPLE ↓

)
```

SAMPLE ⟶ TWODIR
LPT DISPLAY

/L . . . list moved files on line printer.

MOVE EXAMPLES

Figure 6-20

RESULT

1

```
) X MOVE/L=SAMPLE.LS TWODIR SAMPLE ↓

)
```

SAMPLE→TWODIR
SAMPLE.LS on DISK

/L = filename . .. list moved files on disk file "filename".

2

```
) X MOVE/V/L=SAMPLE.LS TWODIR SAMPLE ↓
@DPX0:SAMPLE

)

) TYPE SAMPLE.LS ↓
@DPX0:SAMPLE

)
```

SAMPLE→TWODIR
SAMPLE.LS on DISK
CONSOLE DISPLAY

3

```
) X MOVE/V/FROM TWODIR SAMPLE ↓
@DPX1:TWODIR:SAMPLE

)
```

SAMPLE→@DPX0:
CONSOLE DISPLAY

/FROM . . . move files from specified directory to working directory

MOVE EXAMPLES

Figure 6-21

1

```
) XEQ MOVE/U/DELETE TWODIR SAMPLE↓
@DPX0:SAMPLE

)
```

RESULT

NEW SAMPLE→TWODIR
CURRENT SAMPLE DELETED
CONSOLE DISPLAY

/DELETE . . . delete files with conflicting names in destination directory.

2

```
) X MOVE/U/RECENT TWODIR SAMPLE↓
@DPX0:SAMPLE

)
```

MOST RECENT
SAMPLE → TWODIR
OLDER SAMPLE DELETED
CONSOLE DISPLAY

NO MESSAGE, NO MOVE

/RECENT . . . retain only most recent file if names conflict.

3

```
) X MOVE/U/AFTER/TLM=1-JUN-79 TWODIR↓
@DPX0:?MSG
@DPX0:WHEN.CLI
@DPX0:LIST
@DPX0:?SWAP_1
@DPX0:LOGON.CLI
@DPX0:SAMPLE
@DPX0:TWOSAMPLE
@DPX0:SAMPLE.LS

)
```

FILES MODIFIED SINCE
JUNE 1, '79 → TWODIR

CONSOLE DISPLAY

/AFTER . . . move files modified after specified time.

MOVE EXAMPLES

Figure 6-22

RESULT

```
) X MOVE/V/BEFORE/TLM=30-MAY-79:12 TWODIR↵
@DPX0:CLI.PR
@DPX0:CLI.OL
@DPX0:ERMES
@DPX0:BOOTDPX.SA
@DPX0:BOOTDPD.SA
@DPX0:DINIT.PR
@DPX0:SYSCALL.PR
@DPX0:FIXUP.PR
@DPX0:FIXUP.SA
@DPX0:FDISP.PR
@DPX0:MICRON.OL
@DPX0:MOVE.PR
@DPX0:MICRON.SY

)
```

FILES MODIFIED BEFORE MAY 30, '79 ⟶ TWODIR

CONSOLE DISPLAY

/BEFORE . . . move files modified before time specified.

```
) X MOVE/V TWODIR  F+ ↵
@DPX0:FIXUP.PR
@DPX0:FIXUP.SA
@DPX0:FDISP.PR

)
```

ALL FILES ⟶ TWODIR

CONSOLE DISPLAY

```
) XEQ MOVE TWODIR SAMPLE↵
Warning: File already exists@DPX1:TWODIR:SAMPLE
```

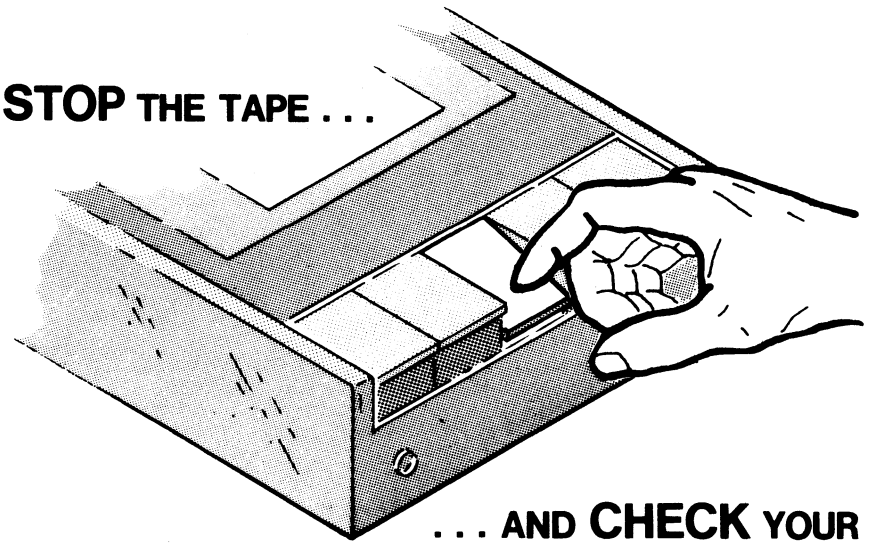
MOVE EXAMPLES

Figure 6-23

TOPICS

- MOVE UTILITY
- INVOKE – XEQ MOVE
- OPTIONAL SWITCHES
- SOURCE DIRECTORY
- DESTINATION DIRECTORY

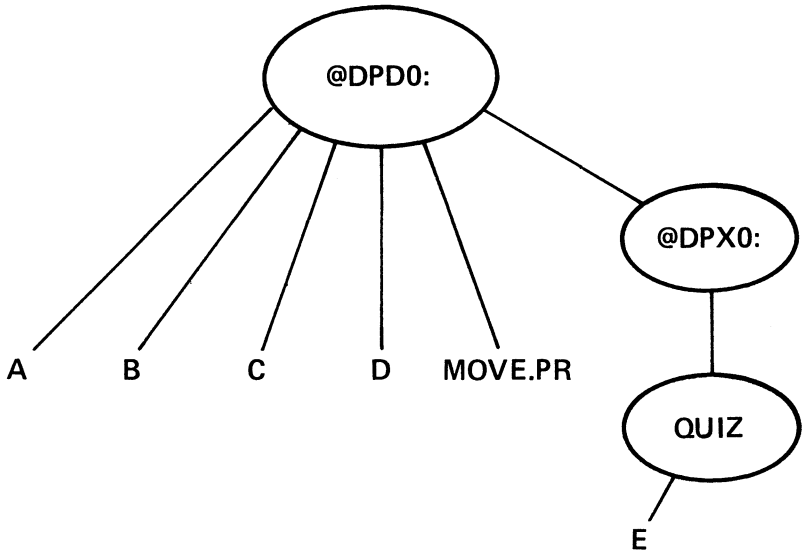
NOW STOP THE TAPE . . .



. . . AND CHECK YOUR PROGRESS

MOVE UTILITY QUIZ

Given the following file structure:



where A, B, C, D, E are non-directory files and @DPD0:, @DPX0:, and QUIZ are directory files. All 3 directories are on the searchlist. @DPD0: is the working directory unless stated otherwise.

Circle the correct command line for the following transfers. A question may have more than one correct answer. The questions are *not* cumulative. The file structure is always as shown above for *each* question.

1. The command line for moving file A to directory QUIZ is:

   A. XEQ MOVE A QUIZ↵
   B. XEQ MOVE QUIZ A↵
   C. XEQ MOVE @DPD0: A↵
   D. XEQ MOVE @DPX0:QUIZ↵

2. Move to QUIZ all @DPD0: files that have been modified (edited, created, compiled, etc.) since January 1 of 1979.

   A. X MOVE/AFTER/TLM QUIZ A, B, C, D ↵
   B. X MOVE/AFTER/TLM=1-JAN-79 QUIZ ↵
   C. X MOVE/BEFORE/TLM=31-DEC-78:12:00:00 QUIZ A, B, C, D ↵
   D. X MOVE/BEFORE/TLM=1-JAN-79 QUIZ # ↵

3. Move A and B to QUIZ. Delete any files named A and B in QUIZ as part of the move.

   A. X MOVE A, B QUIZ/DELETE ↵
   B. XE MOVE/DELETE QUIZ ↵
   C. XEQ MOVE/DEL QUIZ A, B ↵
   D. XEQ/DEL MOVE QUIZ A, B ↵

4. Back-up all files in @DPD0: to QUIZ. Print a list of the files on the console.

   A. X MOVE/FROM/V QUIZ # ↵
   B. X MOVE/ALL/V QUIZ ↵
   C. X MOVE/RECENT/V QUIZ A, B, C, D. ↵
   D. X MOVE/V QUIZ ↵

5. Transfer all files in @DPD0: that have not been modified since April 15, 1979. Move them to QUIZ and print a listing on the line printer.

   A. MOVE/L/BEFORE/TLM=15-APR-79 QUIZ ↵
   B. X MOVE/L/BEFORE/TLM=15:APR:79 QUIZ # ↵
   C. X MOVE/BEFORE/TLM=APR-15-79 QUIZ ↵
   D. X MOVE/L/BEFORE/TLM=15-APR-79:24 QUIZ ↵

6. Move E to @DPD0: . Type the name of the moved file on the console and the line printer.

   A. X MOVE/V/L QUIZ E ↵
   B. X MOVE/V/L/FROM QUIZ E ↵
   C. X MOVE/V/L/FROM QUIZ ↵
   D. X MOVE/L/V @DPD0: E ↵

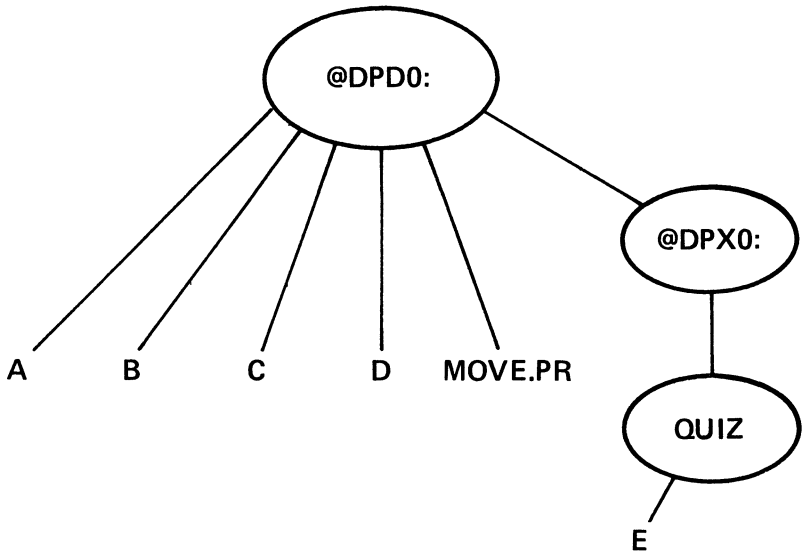
7. Back-up all files in @DPD0: to QUIZ. In case of conflict with files in QUIZ, keep the files that were last updated.

   A. X MOVE/AFTER QUIZ↓
   B. XEQ MOVE/RECENT QUIZ # ↓
   C. XEQ MOVE/RECENT QUIZ↓
   D. X MOVE/BEFORE QUIZ↓

Check your answers
on the following pages.

MOVE UTILITY QUIZ
ANSWERS

Given the following file structure:



where A, B, C, D, E are non-directory files and @DPD0:, @DPX0:, and Quiz are directory files. All 3 directories are on the searchlist. @DPD0: is the working directory unless stated otherwise.

Circle the correct command line for the following transfers. A question may have more than one correct answer. The questions are *not* cumulative. The file structure is always as shown above for each question.

1. The command line for moving file A to directory QUIZ is:

   A. XEQ MOVE A QUIZ ↵
      Target directory and filename are switched.
   (B.) XEQ MOVE QUIZ A ↵
      O,K. since QUIZ is on the searchlist.
   C. XEQ MOVE @DPD0: A ↵
      No, wrong destination directory.
   (D.) XEQ MOVE @DPX0:QUIZ A ↵
      Full pathname for destination directory.

2. Move to QUIZ all @DPD0: files that have been modified (edited, created, compiled, etc.) since January 1 of 1979.

   A. X MOVE/AFTER/TLM QUIZ A, B, C, D ↵
      Wrong time.

   (B.) X MOVE/AFTER/TLM=1-JAN-79 QUIZ↵
      Yes. When filenames are not stated, default of *all* is assumed.

   C. X MOVE/BEFORE/TLM=31-DEC-78:12:00:00 QUIZ A, B, C, D↵
      No, wrong time and switch.

   D. X MOVE/BEFORE/TLM=1-JAN-79 QUIZ # ↵
      Same as C.

3. Move A and B to QUIZ. Delete any files named A and B in QUIZ as part of the move.

   A. X MOVE A, B QUIZ/DELETE ↵
      No, all sorts of errors.

   B. XE MOVE/DELETE QUIZ↵
      Close, but moves A, B, C, and D.

   (C.) XEQ MOV/DEL QUIZ A, B ↵
      DELETE switch may be abbreviated.

   D. XEQ/DEL MOVE QUIZ A, B ↵
      No, switch is in the wrong place.

4. Back-up all files in @DPD0: to QUIZ. Print a list of the files on the console.

   A. X MOVE/FROM/V QUIZ # ↵
      No, moves E to @DPD0: which is the wrong direction.

   B. X MOVE/ALL/V QUIZ↵
      No, error: unknown switch specified

   (C.) X MOVE/RECENT/V QUIZ A, B, C, D. ↵
      Yes, this works. If there are any name conflicts, the most recent is kept.

   (D.) X MOVE/V QUIZ↵
      Yes, if the filenames are not specified, *all* is assumed.

5. Transfer all files in @DPD0: that have *not* been modified since April 15, 1979. Move them to QUIZ and print a listing on the line printer.

   A. MOVE/L/BEFORE/TLM=15-APR-79 QUIZ ↵
      No, need XEQ.
   B. X MOVE/L/BEFORE/TLM=15:APR:79 QUIZ # ↵
      No, wrong separators in the date field.
   C. X MOVE/BEFORE/TLM=APR-15-79 QUIZ ↵
      No, date recorded incorrectly.
   (D.) X MOVE/L/BEFORE/TLM=15-APR-79:24 QUIZ ↵
      Moves all files modified on or before April 15.

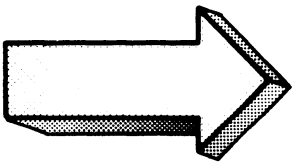
6. Move E to @DPD0: . Type the name of the moved file on the console and the line printer.

   A. X MOVE/V/L QUIZ E ↵
      Tries to move E to QUIZ, file already exists there.
   (B.) X MOVE/V/L/FROM QUIZ E ↵
      Yes, QUIZ becomes the source directory, @DPD0: is the target.
   (C.) X MOVE/V/L/FROM QUIZ ↵
      Yes, E is the only file in QUIZ.
   D. X MOVE/L/V @DPD0: E ↵
      No, @DPD0: is the working directory.

7. Back-up all files in @DPD0: to QUIZ. In case of conflict with files in QUIZ, keep the files that were last updated.

   A. X MOVE/AFTER QUIZ ↵
      Moves all files, but does not handle conflicting names.
   (B.) XEQ MOVE/RECENT QUIZ #↵
      The "#" template means "all".
   (C.) XEQ MOVE/RECENT QUIZ ↵
      No files named so "all" default is assumed.
   D. X MOVE/BEFORE QUIZ ↵
      Moves all files except those modified anytime today.

A SCORE OF 6 CORRECT QUESTIONS OUT OF THE 7 QUESTIONS INDICATES MASTERY LEVEL. REVIEW THE QUESTIONS YOU MAY HAVE MISSED. BE CERTAIN THAT YOU UNDERSTAND THE CORRECT ANSWERS. THEN CONTINUE WITH THE NEXT SEGMENT IN THE STUDENT GUIDE.



MOVE UTILITY

LAB EXERCISE

Abstract

This lab provides practice in the use of the MOVE utility for transferring files from one directory to another.

Directions

1. You will need the file MOVE.PR for completion of this lab. The CLI FILESTATUS command will help you find it.

2. Set the system time to 11 o'clock. A reminder is shown in the screen below:

```
) TIME 11:00:00 ↓
)
```

3. Create a directory named MOVEDIR. Make MOVEDIR the working directory. Create three empty files: HOUSE, HOME, HOTEL, as shown below:

```
) CREATE/DIR MOVEDIR ↓
) DIR MOVEDIR ↓
) CREATE HOUSE ↓
) CREATE HOME ↓
) CREATE HOTEL ↓
)
```

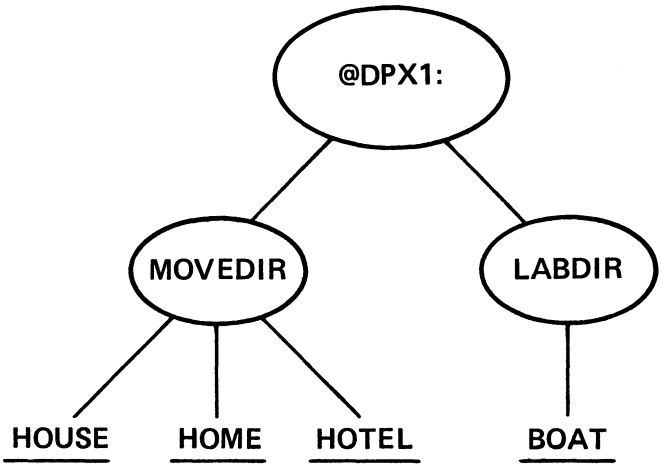
4. Return to the first working directory. Create a directory named LABDIR. Make LABDIR the working directory. Create an empty file named BOAT.

```
) DIR ^ ↓
) CREATE/DIR LABDIR ↓
) DIR LABDIR ↓
) CREATE BOAT ↓
)
```

5. Make MOVEDIR the working directory for the start of the lab exercise. Add LABDIR and MOVEDIR to the searchlist.

```
) DIR MOVEDIR↓
)
) SEARCHLIST @DPX0:, @DPX1:, @DPX1:MOVEDIR, @DPX1:LABDIR ↓
) SEARCH ↓
@DPX0:,@DPX1:,@DPX1:MOVEDIR,@DPX1:LABDIR
)
```

The answers in this lab were worked out on a secondary diskette drive (@DPX1:). The complete file structure looks like this:



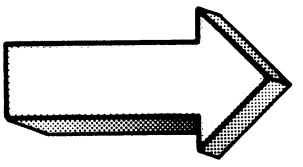
MOVE.PR is in @DPX0: on the primary diskette drive.

6. Note the modification times for the newly created files:

```
) FI/AS
DIRECTORY @DPX1:MOVEDIR

  HOUSE                TXT     9-AUG-79  11:00:30          0
  HOME                 TXT     9-AUG-79  11:00:35          0
  HOTEL                TXT     9-AUG-79  11:00:40          0
)

) FI/AS @DPX1:LABDIR:+
DIRECTORY @DPX1:LABDIR

  BOAT                 TXT     9-AUG-79  11:01:38          0
)
```

You are now ready for the lab. Remember:

1. Cover the answers,
2. Read the question,
3. Write the answer,
4. Compare the answers,
5. Enter the sequence on your system.



1. Transfer HOUSE to the LABDIR directory. Verify the transfer by printing a list of the files moved on the console.

_____

_____

_____

****

```
) XEQ MOVE/V LABDIR HOUSE )
@DPX1:MOVEDIR:HOUSE

)
```

The /V lists the name of the file that is moved. Note that /V does not list the new file's pathname, it shows the pathname of the moved file.

LABDIR is the destination directory. HOUSE is the file to move.

Try it on your system.

If you get an error, check your spelling and the entire command line.

2. Check the modification time of the moved file. Has it changed since HOUSE was created? Write the command and anticipated response:

_____

_____

_____

_____

****

```
) FI/TLM LABDIR:HOUSE )
DIRECTORY @DPX1:LABDIR

  HOUSE                  11:06:45
)
```

The /TLM switch displays the time of last modification. Modification (creation) occurs with a move and the TLM is updated.

Do it on your system.

3. Move all three files in MOVEDIR to LABDIR. If there are any filename conflicts, keep the file that was last modified. Verify the moves on the console:

_____

_____

_____

_____

****

```
) K MOVE/V/RECENT LABDIR H+)
@DPX1:MOVEDIR:HOME
@DPX1:MOVEDIR:HOTEL

)
```

/RECENT says that in the case of similar filenames in the source and destination directories, move the file only if the version in the source directory (MOVEDIR) is more recent than the version in the destination directory (LABDIR).

/V lists the moved filenames on the console.

Try the command on your system.

4. Which files are now in LABDIR. Are they the most recent? Write the command and anticipated response:

_____

_____

_____

_____

_____

_____

_____

_____

****

```
) FI/TLM LABDIR:+ )
DIRECTORY @DPX1:LABDIR

  BOAT                    11:01:38
  HOUSE                   11:06:45
  HOME                    11:09:04
  HOTEL                   11:09:21
)
```

The /TLM switch displays the time of last modification.

Try it on your system.

Compare these times with the MOVEDIR files:

```
) FI/TLM MOVEDIR:+ )
DIRECTORY @DPX1:MOVEDIR

  HOUSE                   11:00:30
  HOME                    11:00:35
  HOTEL                   11:00:40
)
```

Delete LABDIR's HOUSE and HOME (Keep HOTEL).

```
) DELETE/V/C @DPX1:LABDIR:HOUSE @DPX1:LABDIR:HOME
@DPX1:LABDIR:HOUSE? YES↓
Deleted @DPX1:LABDIR:HOUSE
@DPX1:LABDIR:HOME? YES↓
Deleted @DPX1:LABDIR:HOME
)
```

5. Move all three "H" files from MOVEDIR to LABDIR. Delete any files in LABDIR that conflict with the transferring files.

_____

_____

_____

_____

_____

****

```
) X MOVE/V/DELETE LABDIR H+↓
@DPX1:MOVEDIR:HOUSE
@DPX1:MOVEDIR:HOME
@DPX1:MOVEDIR:HOTEL

)
```

/DELETE directs MOVE.PR to delete any conflicting files in the destination directory (LABDIR) before transferring the file from the source directory (MOVEDIR).

Do it on your system.

The only conflict in filenames involves HOTEL. Don't forget the XEQ or its abbreviation or else you may get:

```
) MOVE/V/DELETE LABDIR H+↓.
Error: Unknown command or macro
MOVE/V/DELETE,LABDIR,H+
)
```

6. Which files now exist in LABDIR? What are their TLM's?

****

```
) FI/TLM LABDIR:+)
DIRECTORY @DPX1:LABDIR

  BOAT                    11:01:38
  HOUSE                   11:13:32
  HOME                    11:13:49
  HOTEL                   11:14:05
)
```

Do it on your system.

Note that a newer version of HOTEL, the conflicting file, was moved into LABDIR.

7. What happens to the source files involved in a MOVE? Are they deleted? Modified? Updated? Show the MOVEDIR files. What should occur?

_____

_____

_____

_____

_____

_____

_____

****

```
) FI/TLM MOVEDIR:+ ↓
DIRECTORY @DPX1:MOVEDIR

  HOUSE                 11:00:30
  HOME                  11:00:35
  HOTEL                 11:00:40
)
```

Try it on your system.

The source files are neither deleted, nor modified, nor updated.

Delete the files in LABDIR

```
) DIR LABDIR ↓
) DELETE/U + ↓
Deleted BOAT
Deleted HOUSE
Deleted HOME
Deleted HOTEL
)
```

8. Make MOVEDIR the working directory again. Move all MOVEDIR files modified before 11 o'clock today to LABDIR.

_____

_____

_____

****

```
) DIR MOVEDIR )
) XEQ MOVE/V/BEFORE/TLM=11:00:00 LABDIR )

)
```

/V asks for a display of moved files.
/BEFORE/TLM = says to move files modified before eleven o'clock. No date parameters are specified so Today is accepted as the default.

Execute the sequence on your system.

Nothing moves because all MOVEDIR files were modified (created) after 11 o'clock.

9. Move all MOVEDIR files modified after eleven o'clock today. Show the command and the anticipated response:

_____

_____

_____

_____

_____

_____

****

```
) XEQ MOVE/V/AFTER/TLM=11:00:00 LABDIR HO+)
@DPX1:MOVEDIR:HOUSE
@DPX1:MOVEDIR:HOME
@DPX1:MOVEDIR:HOTEL

)
```

LABDIR is the target directory. MOVEDIR is the source directory. HO+ requests all files beginning with the characters H O. No date is coded so today is the default.

Do it on your system.

/V provides the display. All three HO files are moved because they were all created after 11 o'clock

Delete the MOVEDIR files:

```
) DEL/V/C + )
HOUSE? YES)
Deleted HOUSE
HOME? YEP)
Deleted HOME
HOTEL? YUP)
Deleted HOTEL
)
```

10. MOVEDIR is the working directory. Move the LABDIR files to MOVEDIR. Write the names of the transferred files in MOVELIST.

_____

_____

****

```
) XEQ MOVE/L=MOVELIST/FROM  LABDIR  H+ )
)
```

/L = MOVELIST sets up a disc file with the names of the transferred files.

/FROM sets LABDIR as the source directory and MOVEDIR as the destination directory. All the H files are moved.

Do it.

11. Last one. Type MOVELIST. What do you think it contains?

_____

_____

_____

_____

_____

****

```
) TYPE MOVELIST
@DPX1:LABDIR:HOUSE
@DPX1:LABDIR:HOME
@DPX1:LABDIR:HOTEL

)
```

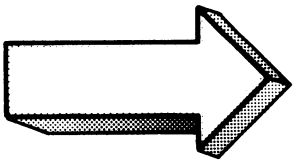
MOVELIST lists the source filenames. The list grows each time MOVELIST is used in the /L= switch.

Do it.

Clean up your system by deleting the files and directories created for this lab.

```
) DIR ^
) DELETE/DIR LABDIR MOVEDIR
)
```

This concludes the Move Utility Lab Exercise. At this point you should be able to use the Move Utility and options for file transfers. Now proceed to the next segment of Module Six.



FIXUP

Abstract

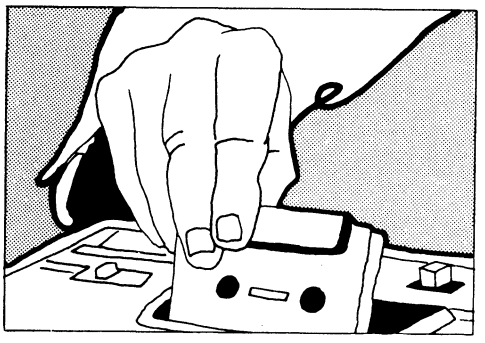
The FIXUP.PR program is used for software-repairing a disk that was not properly DISMOUNTed due to user error or a system failure. This segment describes the requirements and procedures involved in running FIXUP.

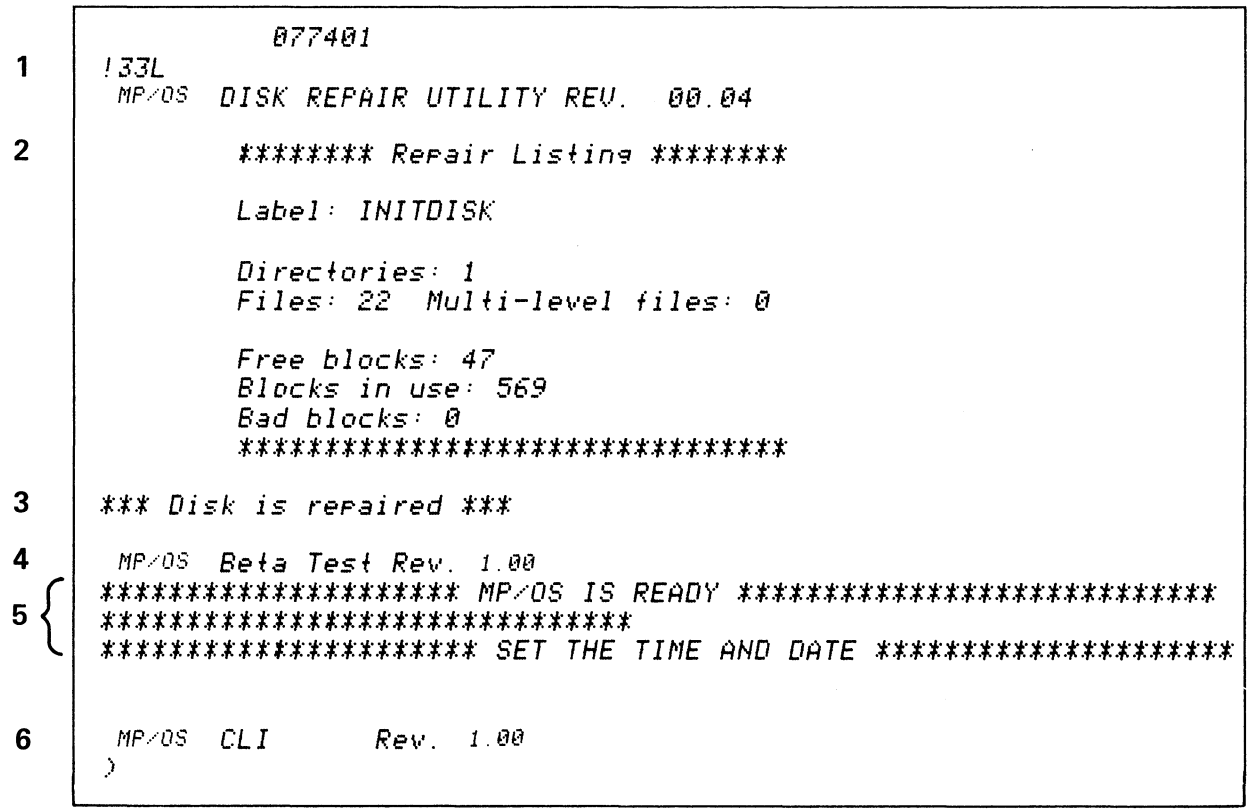
Objectives

Upon completion of this segment you will be able to run FIXUP to repair a disk or diskette.

Directions

Turn to figure 6-73 on the next page of the Student Guide and listen to the audiotape for the third segment of Module Six.





FIXUP ON A SYSTEM DISK

Figure 6-46

```
1   ) X FIXUP⏎

    MP/OS  DISK REPAIR UTILITY REV.  1.00

2   List filename (NL for @TTO) ? ⏎

3   Disk name ? DPX1⏎


4   Possible disk names :

            @DPX
            @DPD
            @DPY
            @DPH


5   Disk name ? @DPX ⏎
    File does not exist
6   Disk name ? @DPX1 ⏎

7           ******** Repair Listing ********

            Disk: @DPX1         Label: BLANKDISK

            Directories: 1
            Files: 0            Multi-level files: 0

            Free blocks: 599
            Blocks in use: 17
            Bad blocks: 0
            ********************************

8           *** Disk is repaired ***

9   Repair another disk ? YES⏎
```

FIXUP ON A NON-SYSTEM DISKETTE

Figure 6-47

```
) MOUNT @DPX1⟩
BLANKDISK
) DISK/F @DPX1⟩
@DPX1
                    599   Blocks available
                     17   Blocks in use
                   1016   Files can be created
                      9   Recoverable errors

)
```

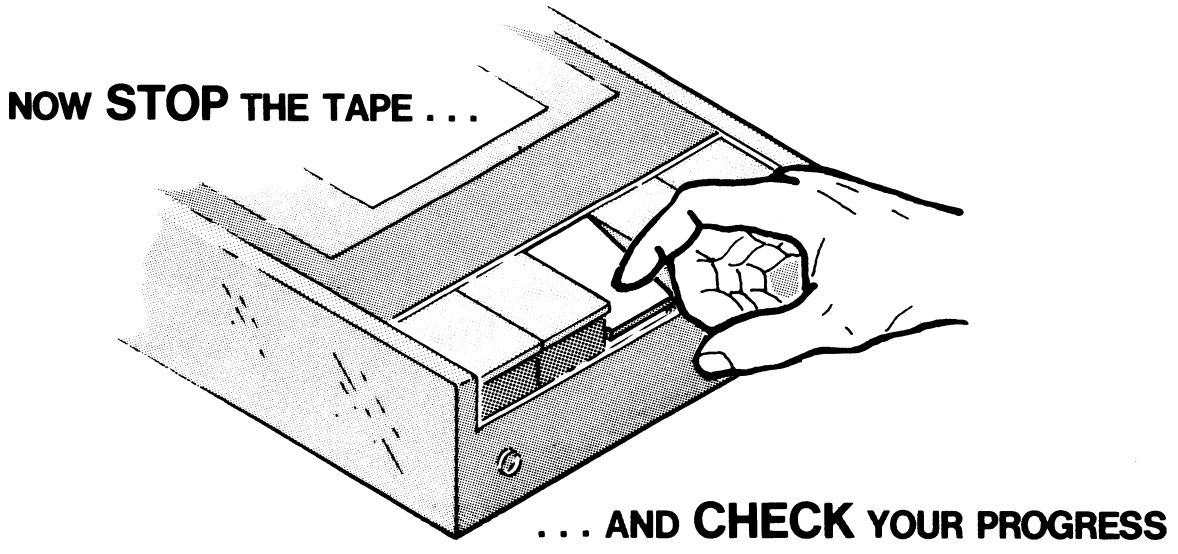
MOUNT diskname . . . bring a disk device on.

MOUNTING A REPAIRED DISK

Figure 6-48

TOPICS

- FIXUP
- DEVICE NAMES
- SYSTEM DISK vs. NON-SYSTEM DISK



FIXUP LAB EXERCISE

Abstract

This lab covers the procedures involved in repairing improperly dismounted disk media.

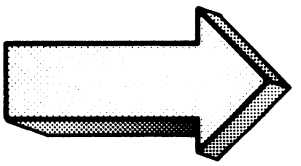
Directions

You will need the following files to complete this lab:

FIXUP.SA. . . .the stand-alone FIXUP program, on your system disk or diskette.
FIXUP.PR. . . .the user-invoked FIXUP program on your system disk or diskette.

In addition, you will need at least two disk devices. The examples in this lab were run on a dual diskette system, so make the appropriate adjustments for your system.

On to the lab.



1. Bring up your system. Describe what occurs:

_____

_____

_____

_____

_____

_____

****

```
!33L
MP/OS              Rev. 1.00
*** HELLO! MP/OS IS READY ***
*** SET THE SYSTEM TIME AND DATE ***


MP/OS CLI          Rev. 1.00
)
```

Do it on your system.

Our example has a LOGON.CLI macro that executes before the CLI message is displayed. Use the appropriate device code for your console debug load command.

2. Move the computer ON/OFF switch to the OFF position, then turn it back ON. Boot the system again. Describe the action:

_____

_____

_____

****

```
) 133166
!33L
MP/OS DISK REPAIR UTILITY REV.   1.00

        ******** Repair Listing ********

        Label: SYSDISK

        Directories: 1
        Files: 20  Multi-level files: 0

        Free blocks: 22
        Blocks in use: 594
        Bad blocks: 7
        ********************************

*** Disk is repaired ***

MP/OS              Rev. 1.00
*** HELLO! MP/OS IS READY ***
*** SET THE SYSTEM TIME AND DATE ***


MP/OS  CLI       Rev. 1.00
)
```

The OFF/ON action sends you back to CONSOLE DEBUG.

Do it!!

This time, FIXUP runs first, then MP/OS comes in. This occurs only with the system disk. Non-system disks must have FIXUP run explicitly from CLI.

3. Load the secondary disk device. Run FIXUP on it. Send the listing to the console. Describe what should occur:

_____

_____

_____

****

```
) XEQ FIXUP↓

MP/OS  DISK REPAIR UTILITY REV.   1.00

List filename (NL for @TTO) ? ↓

Disk name ? @DPX1 ↓

        ******** Repair Listing ********

        Disk: @DPX1          Label: SCRATCH

        Directories: 1
        Files: 0             Multi-level files: 0

        Free blocks: 517
        Blocks in use: 99
        Bad blocks: 0
        ******************************

        *** Disk is repaired ***

Repair another disk ? NO ↓

)
```

This repair requires your assistance.

Do it on your system.

Our diskette is named SCRATCH.

4. Bring the repaired disk on-line and run a DISKSTATUS on it. Does it compare with the FIXUP listing?

_____

_____

_____

****

```
) MOUNT @DPX1 )
SCRATCH
) DISK/F @DPX1 )
@DPX1
                    517   Blocks available
                     99   Blocks in use
                   1016   Files can be created
                      3   Recoverable errors

)
```

MOUNTing brings the diskette on-line. The system responds with the diskette's I.D. when successfully mounted.

Try it on your system.

Note the comparison between the FIXUP repair listing and the DISKSTATUS listing.

5. Put the secondary device off-line and shut down the system:

_____

_____

_____

****

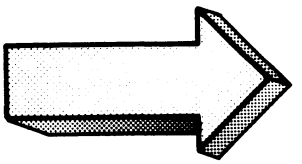
```
) DISMOUNT @DPX1)
) BYE
MP/OS  CLI Terminating

System shutdown
055251
!
```

Do it.

This concludes the FIXUP Utility Lab Exercise. At this point you should be able to use FIXUP to software-repair an improperly dismounted disc device.

This also concludes Module Six.

The following appendices have useful information and exercises on device operations, error codes, and related documentation.



APPENDIX A
DEVICE OPERATIONS

DEVICE OPERATIONS

Abstract

This unit is divided into three step-by-step exercises:

- Power-up and on-line of your processor and Dasher terminal.
- Power-up, load, and on-line your disc and diskette subsystems.
- Bootstrap MP/OS.

EXERCISE 1

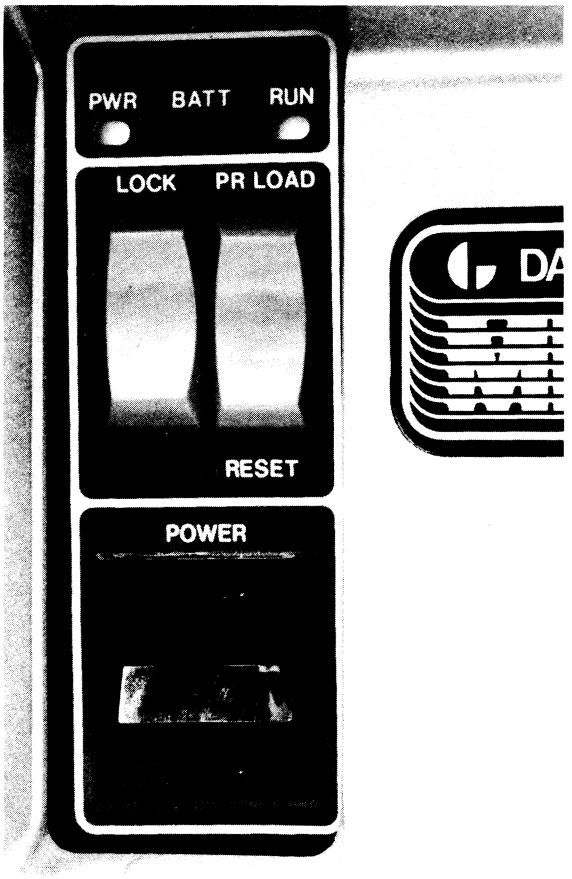
Abstract

In this exercise you will go through the basic operations required for powering-up and setting on-line your processor and Dasher terminal.

Directions

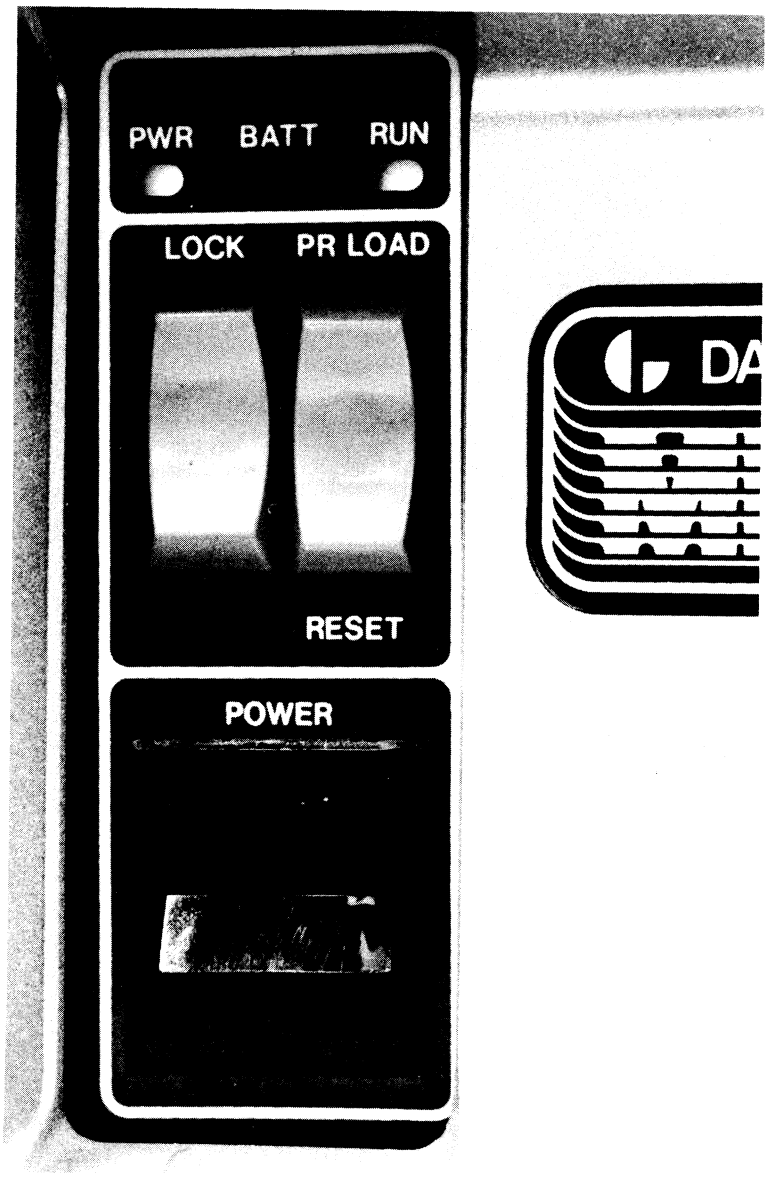
Follow the step-by-step procedures stated below. Skip those devices not included in your system.

SPU



1. Push the POWER switch on your MP control panel to the ON position.

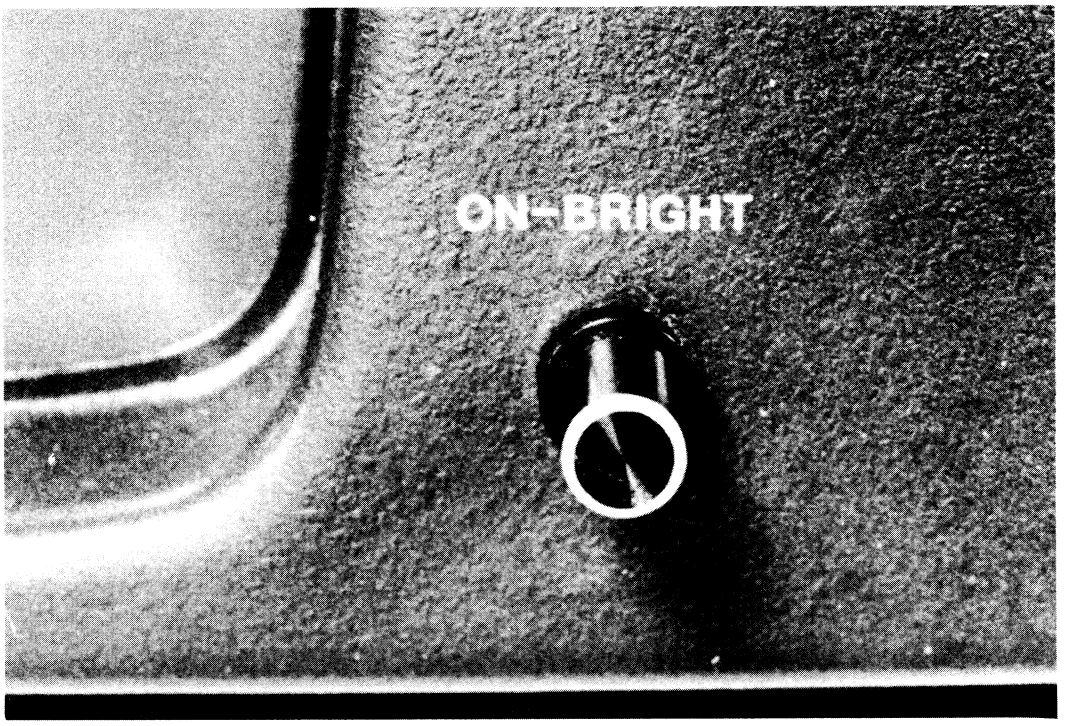
Figure A-1



2. Observe the status of the POWER indicator. It should be lit. The computer is now ON.

Figure A-2

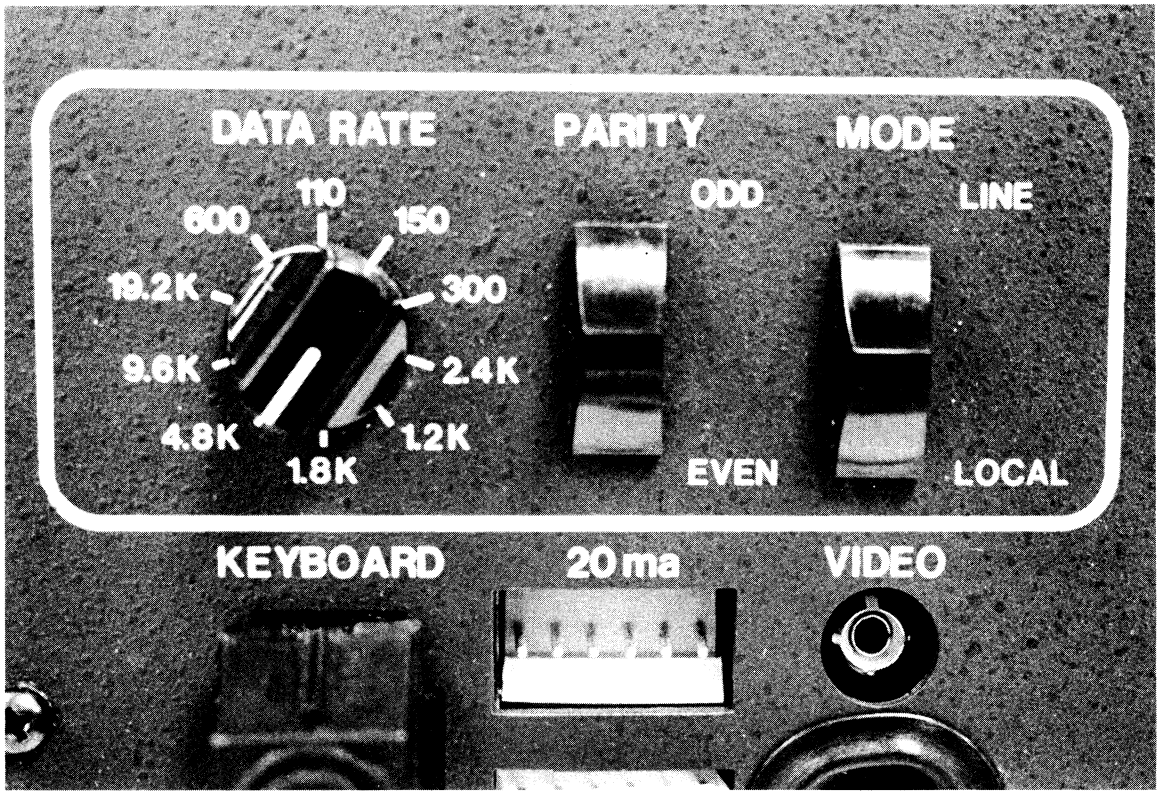
DASHER DISPLAY



3. Pull the Dasher POWER control out. Wait twenty seconds and observe the screen.

4. Swivel the unit around so that you can access the rear switches.

Figure A-3



5. Push the MODE switch to the LINE position. This sets the Dasher on-line.

6. Push the PARITY switch to the EVEN setting.

7. Turn the DATA RATE rotary switch to setting appropriate for your system.

Figure A-4

8. Swivel the Dasher around so that it faces forward again. Observe the screen. If you have the console debug option, your screen should display the prompt and blinking cursor.

9. Locate the ALPHA LOCK on the Dasher keyboard. Press the key so that uppercase entries are transmitted to the terminal.

Your Dasher is now ready to fly.

Work through the following exercises for each device in your system.

Figure A-5

EXERCISE 2

Abstract

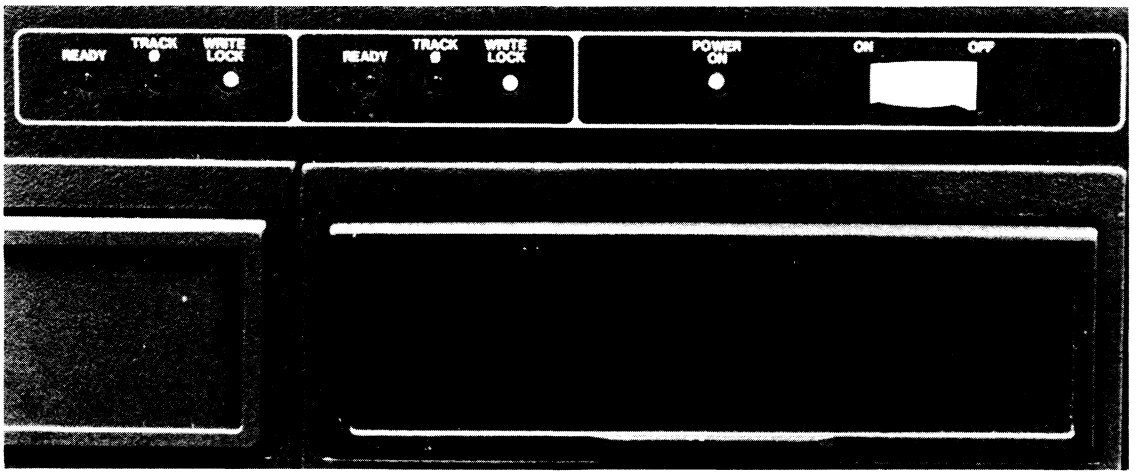
In this exercise you will power-up, load, and set on-line your disc and diskette devices.

Directions

DISKETTE SUBSYSTEM

microNOVA computers are supported by diskette subsystems which provide 157K words (single drive) or 315K words (dual-drive) of on-line storage. The diskette subsystem contains an integral data channel controller and 10-foot cable to computer-based or stand-alone card-frame systems. The controller is contained within the diskette drive unit, and is cabled to the external I/O bus. The external I/O bus may still be propagated beyond the diskette unit, consistent with I/O bus configuration specifications. Diskette medium is compatible and transferable to NOVA 3 based diskette subsystems. The diskette subsystem is packaged in a rack-mountable 19"W x 7"H x 22"D chassis. Multiple controllers, each supporting one or two drives, can be chained on a single system.

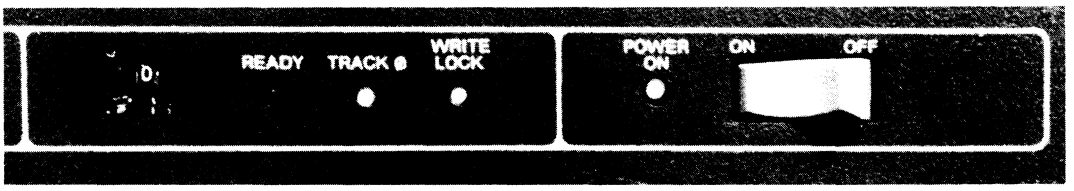
Model 6038	One-drive system (157K words)
Model 6039	Two-drive system (315K words)
Model 1098A	Carton of 10 diskettes



1. Press the POWER switch to the ON position.

2. Observe the POWER indicator. It should now be lit along with the WRITE LOCK indicator.

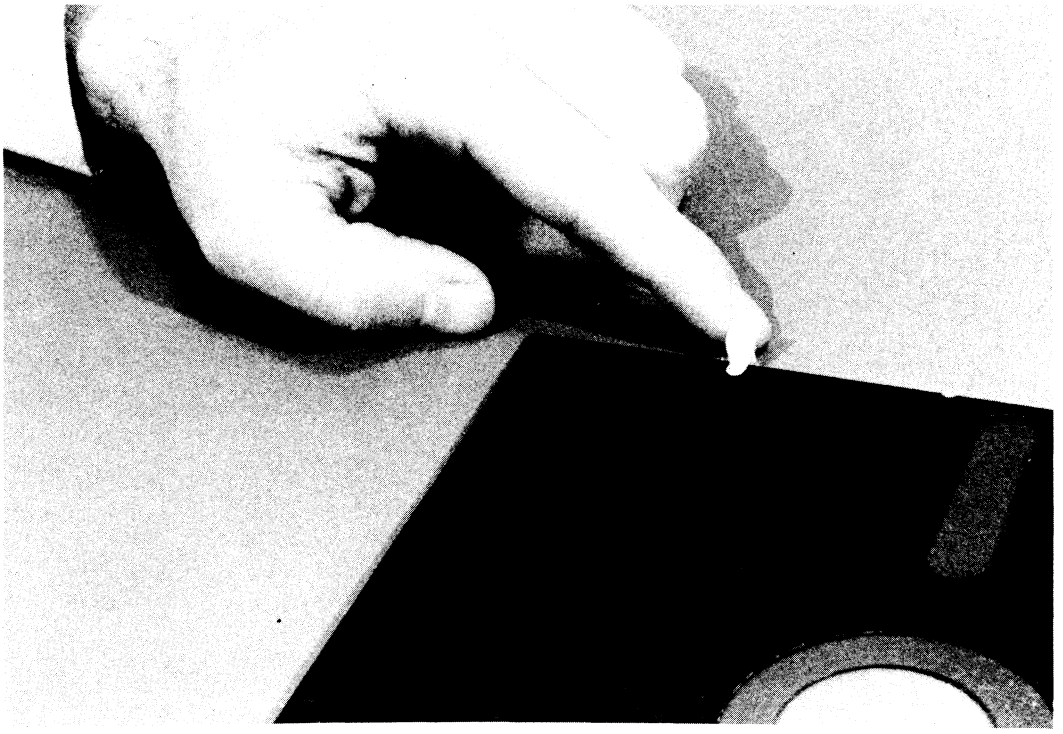
Figure A-6



3. Select your primary diskette drive by turning the drive select wheel to "0".

4. Observe the TRACK 0 indicator. It should be lit for the drive you selected as your primary drive.

Figure A-7



5. If you intend to write on your diskette, then place a piece of tape over the WRITE LOCK hole. If you do not intend to write on the diskette, then leave the hole uncovered.

Figure A-8

6. Depress the door latch on the diskette drive. Slide the diskette into the drive as shown. Press the door shut.

7. Observe the READY light. It should now be lit. If you did not tape the WRITE LOCK hole, the WRITE LOCK light should now be lit. If the light is not lit, then you will be able to read and write to the diskette.

Your diskette is now ready to fly.

Figure A-9

DISC SUBSYSTEM

The Model 6095 Cartridge Disc Subsystem offsers a single 10MB disc drive. The disc drive contains a single spindle for one fixed 5MB and one removable 5MB cartridge disc. The subsystem controller interfaces the disc drive to any microNOVA computer and mounts at the rear of the drive (no I/O slot in computer).

Each of the four available disc surfaces in the 10MB disc drive is accessed by a read-write head. The head positioning system accurately positions the four heads in unison at any of 408 cylinders. The average positioning time is 38 milliseconds with track-to-track and full stroke positioning of 8 and 70 milliseconds respectively.

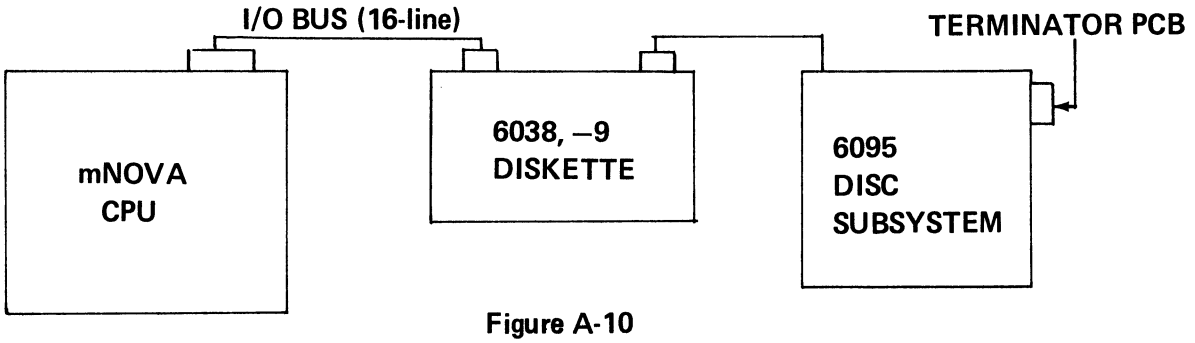


Figure A-10

It is also suggested that since the data is fully buffered, the disc drive be placed on the bus as a low priority device (electrically "distant" from the CPU). This is to insure that high speed devices that require minimum data channel latency receive priority on the I/O bus.

The disc drive operates at 2400 RPM and transfers data at 312,000 bytes per second.

Media interchange between various 6095 subsystem drives is a normal design specification, and should be routine if drives are properly maintained.

Data General does not guarantee interchange of media between Model 6095-type drives and Model 4234-type 10MB drives. The user may, in fact, find that the media is interchangeable in most cases, but may not rely upon it as a design specification.

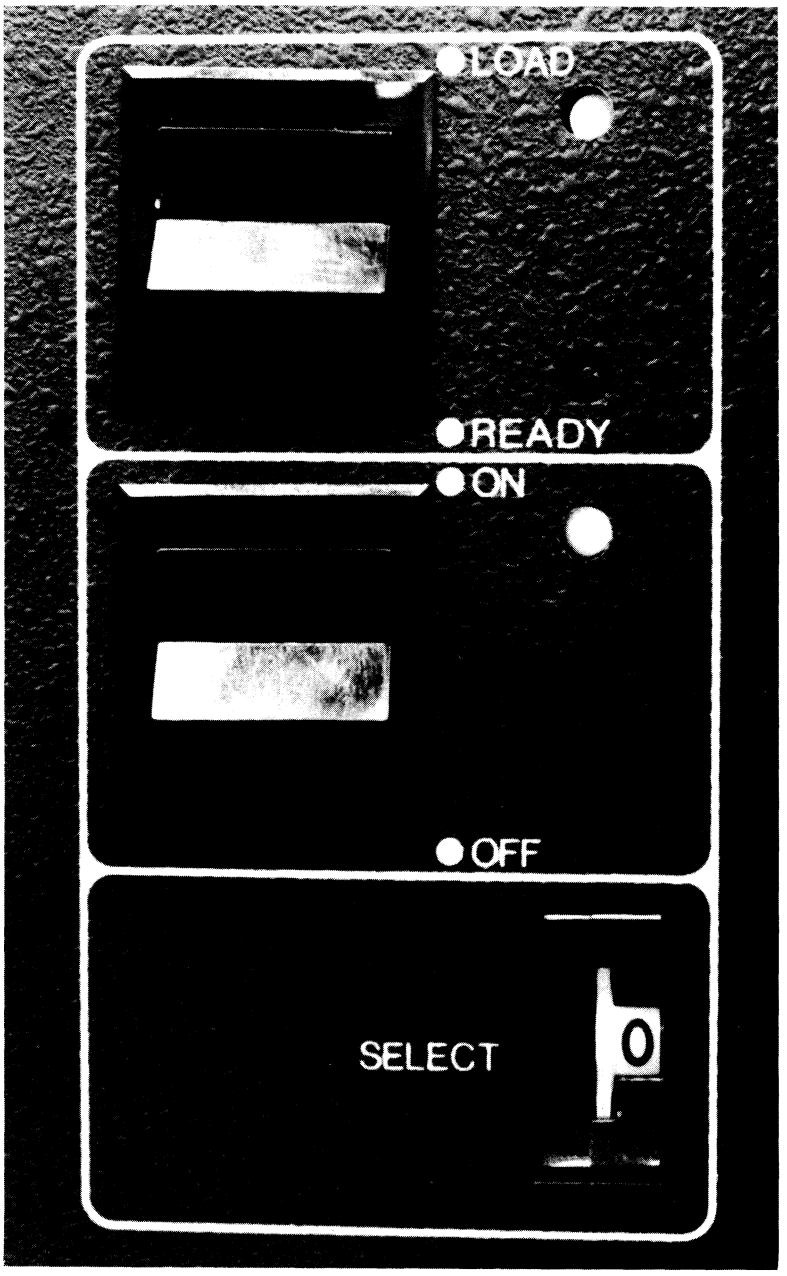
The 6095 disc subsystem operates exclusively with microNOVA systems (e.g., not on NOVA/ECLIPSE). However, 6095 cartridges are physically and format compatible with 6045 (6050) NOVA/ECLIPSE systems (media interchange compatible). The 6095 has approximately 95% parts commonality with existing 10MB subsystems and therefore, all applicable drive specifications (such as environment, power requirements, etc.) will be identical to the 6045, 6050 systems. The major difference is with the "disc cable interface board" subassembly which is now replaced with a new integrated controller board subassembly. This controller functionally equates to the 6051 controller PCB in the NOVA/ECLIPSE environment, and is required due to the nature of the microNOVA I/O bus structure. The controller supports only one drive and does not support dual porting.

The 6095 subsystem is identical to the 6050 drive in terms of rack mounting requirements.

A major advance achieved in the DGC cartridge disc subsystems is the compact, easily accessible packaging of the drive and power supply. The drive requires only 10½ inches of vertical rack space and the power supply is accessed by an innovative drop-down subassembly. Further, the drive design results in lower power dissipation (500 watts maximum) than with previously offered models.

The microNOVA 10MB disc controller is an 8" x 15" printed circuit board which is installed at the rear of the drive. The controller is programmed using the basic I/O commands for the microNOVA processor. The data transfer takes place independently through the data channel.

Model 6095 DG/Disc Subsystems are self-contained units that connect to any microNOVA computer via the 16-line I/O bus. Any other device that connects to the microNOVA I/O bus (e.g. diskette, DG/DAC, etc.) may be daisy-chained off the 10MB subsystem. Make sure that the terminator PCB is present on the device furthest from the CPU.



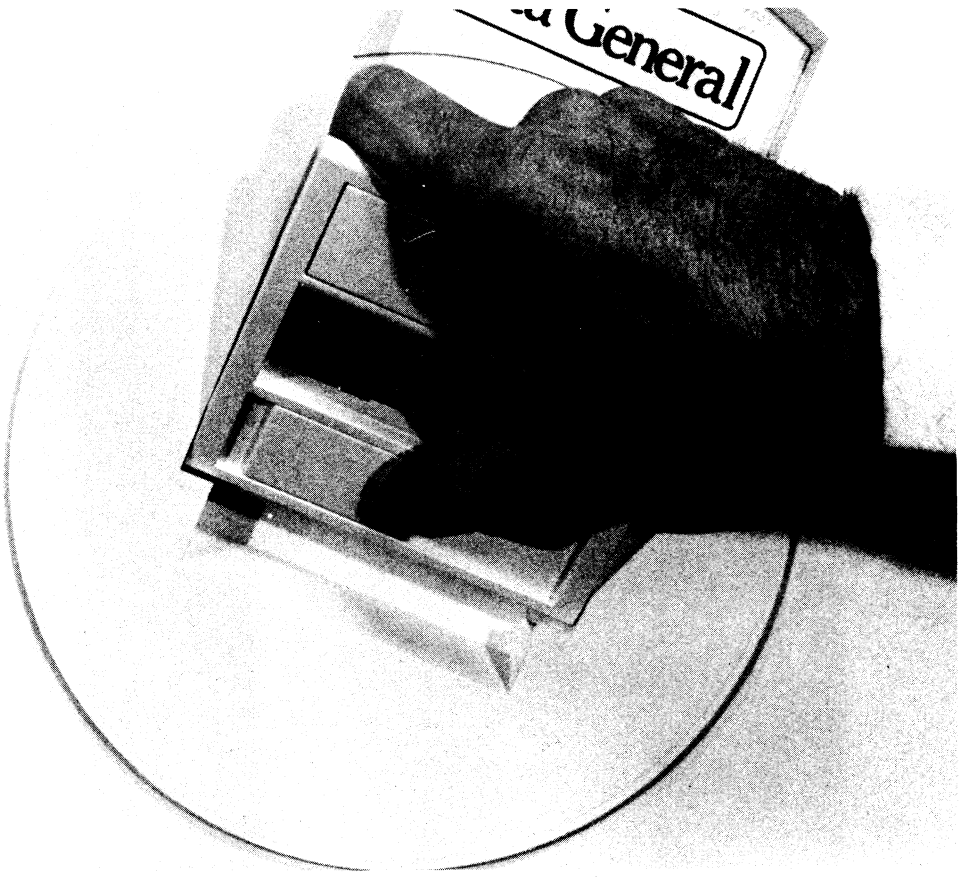
1. Push the POWER switch to the ON position. This should cause the POWER and LOAD indicators to light.

2. Press the LOAD switch to the LOAD position.

Figure A-11

3. Release the drive latches and slide the disc drive out.

Figure A-12

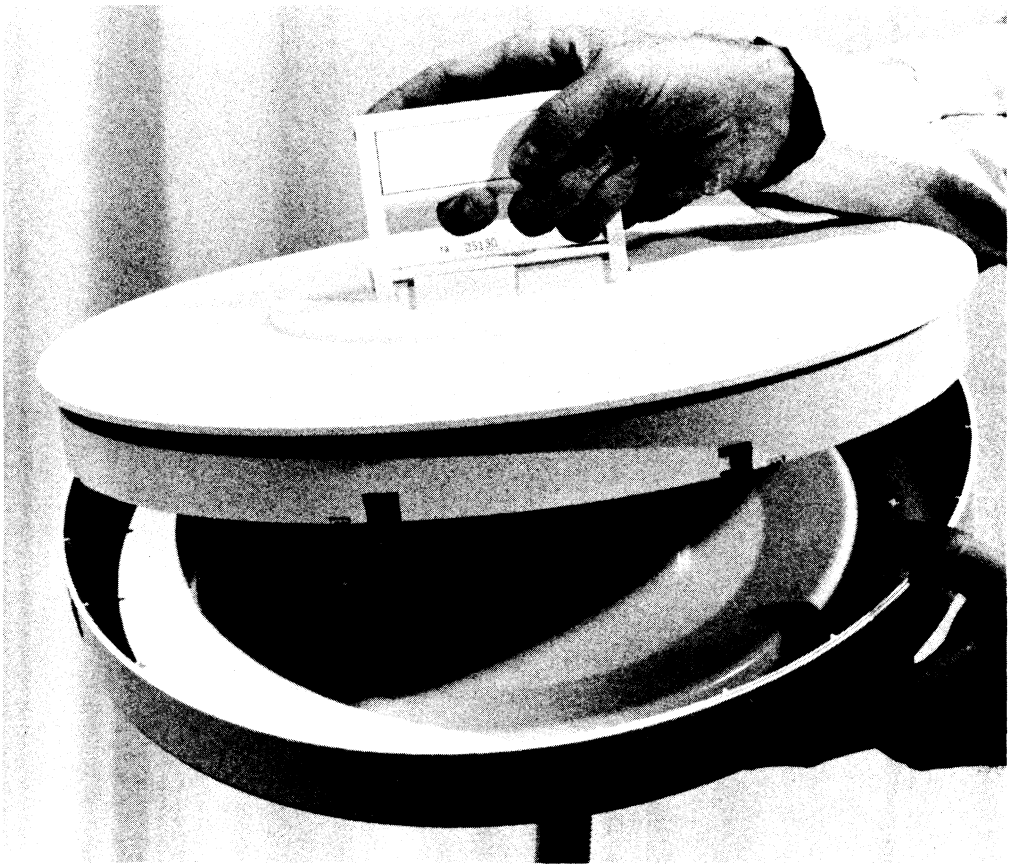


4. Grasp the disc cartidge handle in your right hand. Support the bottom dust cover in your left hand.

5. Press your right thumb against the release switch at the base of the handle.

6. Push the switch to the left and raise the handle. This will release the bottom dust cover.

Figure A-13



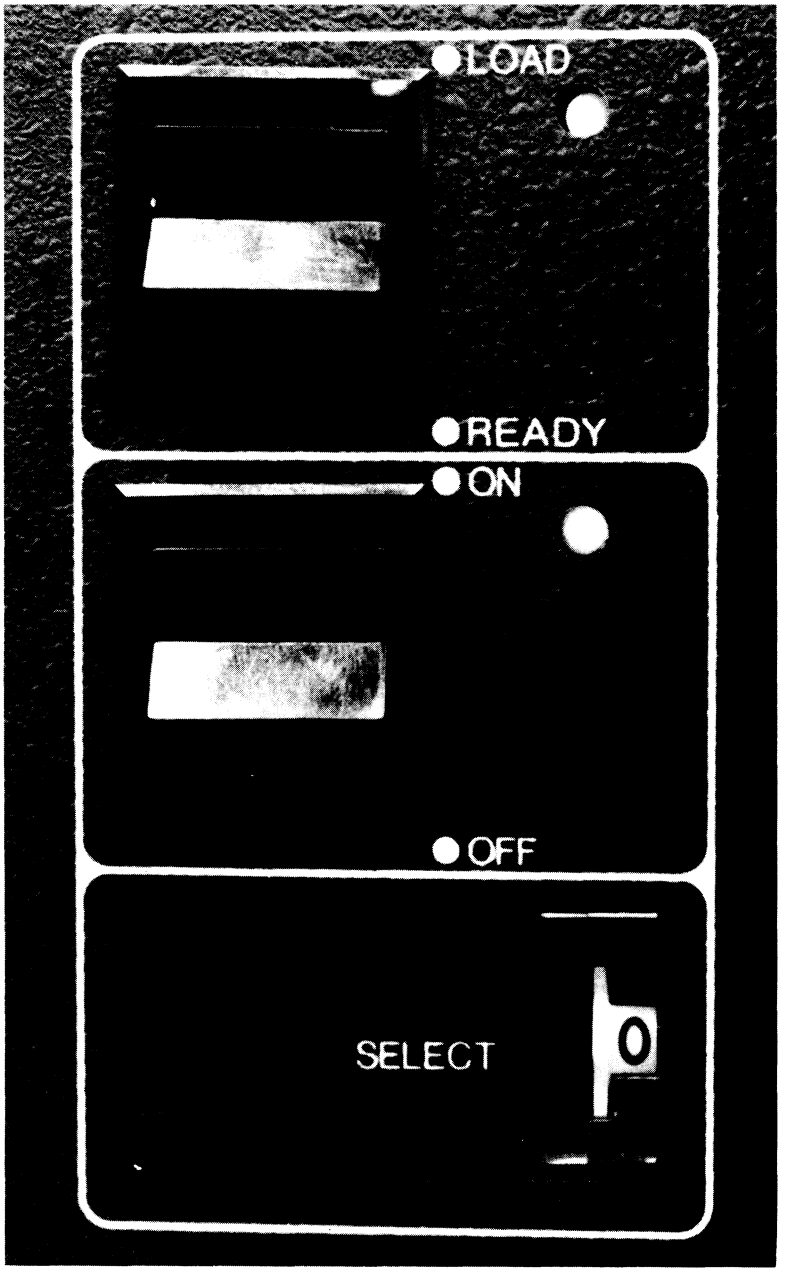
7. Lift the cartidge straight out of the dust cover.

8. Place the cartidge into the drive cavity. The sliding latch is towards the front of the drive.

9. Lower the cartidge handle until it is flat on the cartidge. Place the dust cover over the cartidge, making sure it fits into the cavity properly.

10. Slide the drive into the cabinet until the front panel latches engage the drive into place.

Figure A-14



11. Turn the LOAD/READY switch to READY. When the READY indicator goes on, the drive is on-line with the processor and is ready to receive commands.

Figure A-15

EXERCISE 3

Abstract

This exercise describes the procedures for booting MP/OS from disc devices.

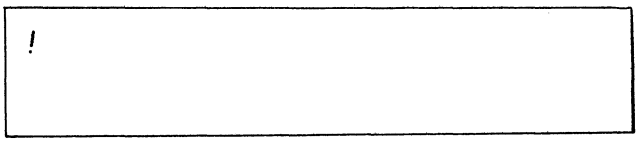


Figure A-16

1. When the disk has been installed and is on-line, you can bootstrap the operating system. On your terminal you will notice some numbers, followed by an exclamation mark (!). This is a prompt, informing you that the system is waiting for you to enter a command.

Disk(ette) Drive	Code	Name
10 Mbyte disk	27	DPD
0.3 Mbyte diskette*	33	DPX
12.5 Mbyte disk	26	DPH
1.2 Mbyte diskette	–	DPY

Device codes for disk drives

2. Before you can enter the bootstrapping command, you must find out what the device code is for your particular disk drive. A device code is a form of identification number. The default device codes for the disk drives are shown in the Table.

```
!100027L
```

BOOT FROM A DISC
Figure A-17

```
!100033L
```

BOOT FROM A DISKETTE
Figure A-18

```
!100026L
```

BOOT FROM A FIXED DISC
Figure A-19

3. You now type the bootstrapping command appropriate for your system. The command has the following format:

   1000*device_code*L

   where the 1 signifies you are talking to a high speed (data channel) device; the *device_code* number identifies the disk drive; and the L indicates that the MP/OS System is to be loaded into the computer's memory.

4. You will now see the MP/OS and CLI title messages.

   NOTE: *If you have a Model 6038/39 diskette drive, you only type:*

   33L

   i.e. you omit the 1000 *before the device code.*

```
MP/OS     Rev.  1.00
MP/OS CLI Rev.  1.00
)
```

Figure A-20

The right parenthesis is followed by a blinking underscore called a cursor*. Your MP/OS system is now up, with the parenthesis indicating that the Command Line Interpreter (CLI) is running and waiting for you to type in commands.

**If you have a hard-copy terminal instead of a display terminal, you will not have a cursor.*

APPENDIX B
ERROR CODES

FORTRAN COMPILER ERROR MESSAGES

N – Means that the syntax error is not necessarily fatal.

C – Means the scan of the statement is continued if the error is a syntax error. The continued scan applies only to syntex errors; errors at a different level may or may not allow the scan to continue.

In FORMAT statements, the error is generally fatal. In declaration statements, if a conflict occurs, the last declaration for the identifier is ignored.

CODE		MEANING
00		Working space exhausted. Fatal, but compiler continues.
01		Multiply-defined parameter.
02	N	Mixed precision operands.
03	N	Unknown statement type.
04	N	Something other than blanks at statement end.
05		Syntax error in DATA variable list.
06		Syntax error in DATA literal list.
07		Syntax error in statement function.
10	C	Missing integer in FORMAT.
11		Error in parameter list of CALL.
12		Array identifier not followed by a left or right parenthesis or comma.
13		Illegal element in expression.
14		Improper use of array name.
16		Missing operator.
17		Illegal sequence of adjacent operators.
20		Illegal element/operator when "(" or literal or variable expected.
21		Premature statement end for an IF.
22		Trailing period (.) missing in operator such as .EQ.
23		Illegal continuation line (follows comment or has label).
24		Period (.) not followed by letter or number.
25	C	Format error.
26	C	Format error after repeat count. (Errors 25 and 26 together indicate an illegal character. These errors may repeat on one statement.)
27		Abnormal end to FORMAT statement.
30		Expression didn't close at end of statement.
31		Multiply-defined error.
32		Variably-dimensioned array is not a dummy.
33		Variable list longer than value list in DATA.
34	C	Identifier in more than one type declaration.
35		Unclosed DO loop in program.
36		COMMON variable previously declared EXTERNAL, subprogram or dummy.
37	C	Dummy identifier predefined.
40	C	Dimension error.
41		Improper statement terminating DO loop.
42	C	Variable dimension for main program array.
43		Array size is greater than 32K.
44		Parentheses don't close before statement end.
45		Expected numeric operand for unary minus.

46		Expected logical operand for .NOT.
47	N	Illegal operand types for current operator.
50	C	DATA statement error; types don't match.
51		Both members of EQUIVALENCE pair in COMMON.
52		Beginning of COMMON extended by EQUIVALENCE.
53		Irrecoverable format error.
54		Statement function name in conflict with previous declaration.
55		Multiply-defined dummy identifier in statement function.
56	C	Too few subscripts in DATA or EQUIVALENCE.
57	C	Subscripts out of bounds in DATA or EQUIVALENCE.
60	C	Format syntactical structure of statement is in error, punctuation is missing, or an identifier is of the wrong variety.
61		Undefined label.
62		Attempt to load or store external or array.
63		Array element can't be specified for a dummy array.
64	C	Identifier in EXTERNAL previously declared in other than type declaration.
65	C	A variable dimension is not a dummy.
66		Variable on DATA list not in labeled COMMON.
67		Two variables, neither in COMMON, are EQUIVALENCEd.
70		A subscript is not type integer.
71		Wrong number of arguments for reserved name function.
72		Wrong type of arguments for a reserved name function.
73	N	Non-digit in label field.
74	N	Carriage return in label field.
75		Improper statement in block data subprogram.
76	N	Unreferenced label.
77		Stack variable referenced in statement function.
100	C	Variable stack has no room for all run-time variables.
101		Undeclared identifier in statement function expression.
102		RETURN statement in main program.
103		Abnormal return in function subprogram.
104		$ followed by something other than a digit.
105		End of file without END.
106		Wrong number of subscripts.
107		Illegal use of statement function name.
111		Hollerith constant not ended at statement end.
112	C	Truncated integer with magnitude greater than 2**15-1.

114 C	Exponent error in REAL variable.
115 C	Exponent error in DOUBLE PRECISION variable.
116 C	Illegal character for FORTRAN statement.
117	Statement function lacks argument.
120	Literal error of one of the following types: (a) two operands not both literals, (b) two literals of different types, or (c) source line is (%it("literal, literal_operator") where: %it ("literal_operator") is not a right parenthesis.
121	Attempted EQUIVALENCE to dummy argument.
122	Error in CHANTASK statement.
123	ID is used as an array element in DATA statement, but not declared array.
124	Illegal literal value for step count of a DO index.
125	Illegal complex relational tests.
126	Missing "," in I/O list.
127	Extraneous ",".
130	Illegal variable name.
131	Subscripted variable in DO list.
140-160	Compiler errors for debugging only.

FATAL SYSTEM ERRORS

Because MP/OS shares its address space with the user, it must be extremely suspicious of the integrity of its code and data. An errant user program can easily destroy critical portions of the operating system. The system must also be able to respond meaningfully in the presence of hardware faults. The system periodically checksums its code to detect over-writes and automatically retries failing device operations whenever necessary. Sometimes all this is to no avail and the only option is to report a fatal error.

On an uncrecoverable error, the following will be printed:

Fatal error #:: A B C D E F

- # Is the number of the fatal error.
- A Is the AC0.
- B Is AC1.
- C Is AC2.
- D Is AC3.
- E Is the stack pointer.
- F Is the frame pointer.

These registers have differing interpretations depending on the fatal error number. This writeup is intended as a brief guide to MP/OS users as to the meaning of those numbers. When reporting an error, you should always include the complete text of the fatal error message. In addition, as noted in the following table, sometimes you should include the contents of locations 5, 6, and 7.

Fatal error 0 - An internal system call has returned an error and there is no way to recover from it. For instance, in interrupt servicing, an IUNPEND, ID destined to wake up the task pended on this interrupt could conceivably return "INVALID TASK IDENTIFIER" if the user wrote on the system database where this information is stored.

AC0 contains a MP/OS error code.

Fatal error 1 - System checksum error. When the system has idle time, it checksums its pure area. If this changes, someone has stored into system space. No registers are meaningful here.

Fatal error 2 - The system is doing a JMP 0. The register set is needed in an STR as well as the contents of locations 5, 6, and 7. This is almost always an indication that a critical database has been corrupted.

Fatal error 3 - To perform a boot, the system must shut itself down. After this shutdown has proceded to dismount devices, etc. an error cannot be tolerated.

AC0 contains a MP/OS error code.

Fatal error 4 - The system was unable to clear an interrupting device for which there was no interrupt handler.

AC1 is the device code of the offending device.
If AC1 is −1 then a powerfail has occurred.

Fatal error 5 - The system received an interrupt from an "impossible" device (one whose device code is not in the range 0 to 76).

AC1 is the device code of the offending device.

Fatal error 6 - The system was unable to refresh CLI.PR. The register set is immaterial.

Fatal error 7 - The system encountered an error while attempting to read in a system overlay (or release one). The most probable cause is an unrecoverable device error while reading the system master device.

Fatal error 10 - A resource inconsistency was detected in the system. This can occur when the user program has written into system impure space. The register set and locations 5, 6, and 7 are needed for the STR. Note that the fatal error code number is in octal.

Fatal error 11 - System internal error. Again, the full state is needed for the STR.

APPENDIX C
RELATED DOCUMENTATION

Related Courses

MP/PASCAL PROGRAMMING SELF-STUDY COURSE. Audio-tape format. Written for the programmer responsible for applications and system maintenance. Familiarity with a high-level language is suggested but not required.

microNOVA/MP-SERIES HARDWARE MAINTENANCE SELF-STUDY COURSE.

Related Manuals

The list that follows gives a brief description of each of the other manuals which describe Microproducts and the MP/OS system.

- *An Introduction to Microproducts and MP/OS* describes the hardware and software in general terms, to give an overview of your MP/Computer and its capabilities.

- *Microproducts Hardware Systems Reference* gives a detailed functional description of the Microproducts line of Microcomputers and related peripherals, board by board.

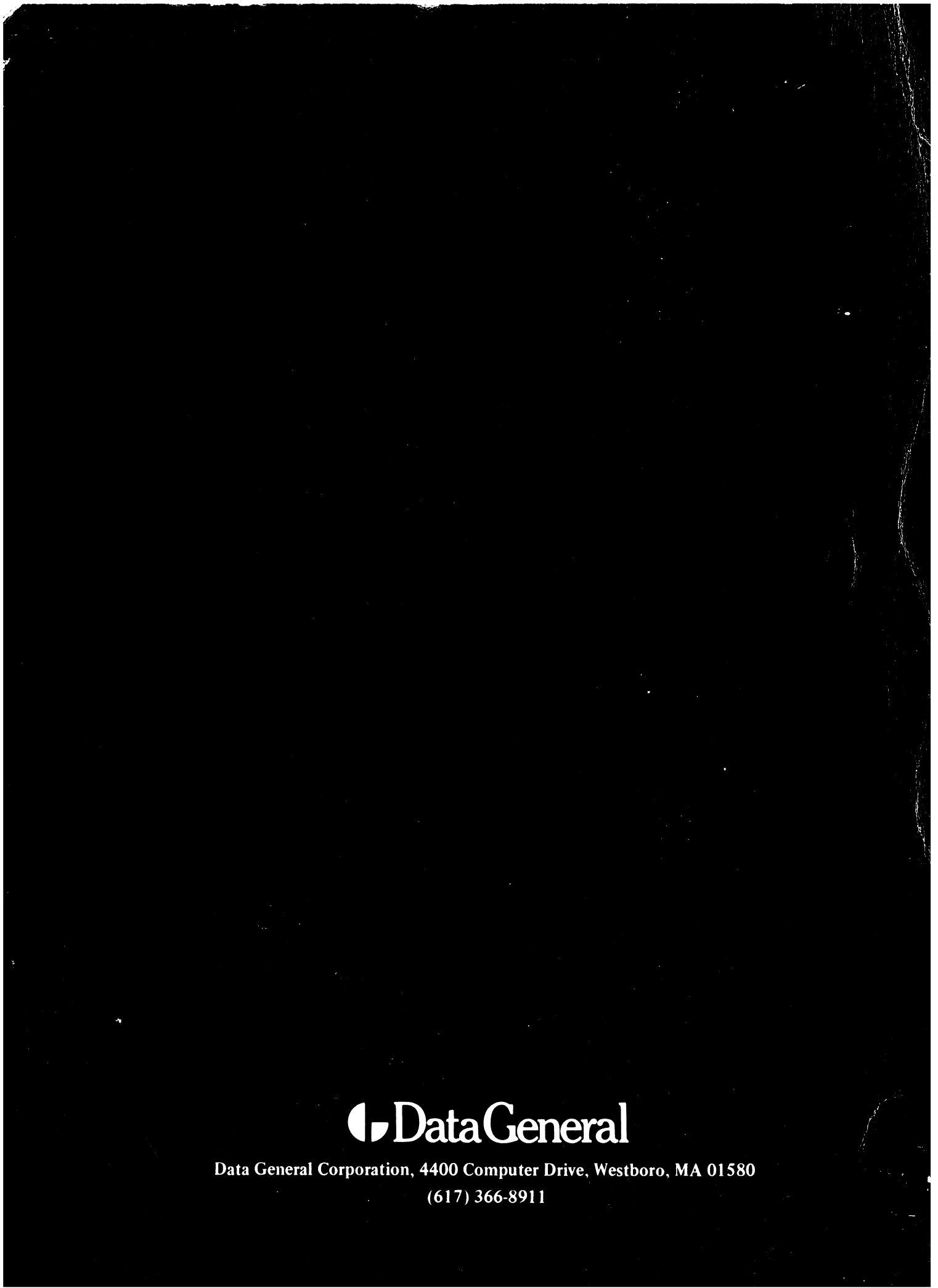
- *Learning to Use the MP/OS Operating System* should be read by anyone who has never used an MP/Computer. It introduces the MP/OS file system and the Command Line Interpreter. A console session gives you step by step hands-on experience with your new MP/Computer.

- *Assembly Language Programmer's Reference* is the main source of information for the assembly language programmer. It describes the instruction sets of MP/Computers in detail, and gives details of the Macroassembler and operating system.

- *MP/OS Utilities Reference* describes the utility programs available with the MP/OS system.

- *MP/Fortran IV Programmers' Reference* covers all of the features of this powerful high level language.

- *MP/Pascal Programmers Reference* describes Data General's extended version of PASCAL.



Data General Corporation, 4400 Computer Drive, Westboro, MA 01580
(617) 366-8911