Tektronix®
COMMITTED TO EXCELLENCE

This manual supports the following TEKTRONIX products:

8560 Options	Products
1A	8560B01
1B	8560B02
1C	8560B04
1G	8560B10
1J	8560B15
1K	8560B16
1L	8560B17
1M	8560B18

This manual supports the following software modules:

TEKTRONIX B Series Assembler Version 1 (8560)
TEKTRONIX B Series Linker Version 1 (8560)
TEKTRONIX B Series LibGen Version 1 (8560)

These modules are compatible with:

TNIX Version 1 (8560)

PLEASE CHECK FOR CHANGE INFORMATION AT THE REAR OF THIS MANUAL.

8500
MODULAR MDL SERIES
8560 HOST SPECIFICS USERS MANUAL
for
B Series Assemblers

Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077

070-3944-00
Product Group 61

Serial Number _____

First Printing DEC 1981

LIMITED RIGHTS LEGEND

Software License No._____

Contractor: Tektronix, Inc.
Explanation of Limited Rights Data Identification Method
Used: Entire document subject to limited rights.

Those portions of this technical data indicated as limited rights data shall not, without the written permission of the above Tektronix, be either (a) used, released or disclosed in whole or in part outside the Customer, (b) used in whole or in part by the Customer for manufacture or, in the case of computer software documentation, for preparing the same or similar computer software, or (c) used by a party other than the Customer, except for: (i) emergency repair or overhaul work only, by or for the Customer, where the item or process concerned is not otherwise reasonably available to enable timely performance of the work, provided that the release or disclosure hereof outside the Customer shall be made subject to a prohibition against further use, release or disclosure; or (ii) release to a foreign government, as the interest of the United States may require, only for information or evaluation within such government or for emergency repair or overhaul work by or for such government under the conditions of (i) above. This legend, together with the indications of the portions of this data which are subject to such limitations shall be included on any reproduction hereof which includes any part of the portions subject to such limitations.

RESTRICTED RIGHTS IN SOFTWARE

The software described in this document is licensed software and subject to **restricted rights**. The software may be used with the computer for which or with which it was acquired. The software may be used with a backup computer if the computer for which or with which it was acquired is inoperative. The software may be copied for archive or backup purposes. The software may be modified or combined with other software, subject to the provision that those portions of the derivative software incorporating restricted rights software are subject to the same restricted rights.

Products of Tektronix, Inc. and its subsidiaries are covered by U.S. and foreign patents and/or pending patents.

TEKTRONIX, TEK, SCOPE-MOBILE, and  are registered trademarks of Tektronix, Inc. TELEQUIPMENT is a registered trademark of Tektronix U.K. Limited.

Printed in U.S.A. Specification and price change privileges are reserved.

Section 8B

8560 HOST SPECIFICS

TABLES

ILLUSTRATIONS

Section 8B

8560 HOST SPECIFICS

This section is designed to be inserted into Section 8 of the 8500 Series B Assembler Core Users Manual. This Host Specifics section provides information that applies specifically to the 8560 Multi-User Software Development Unit.

Throughout the section, "this manual" refers to the 8500 Series B Assembler Core Users Manual.

The examples given in this section are for the 8086/8088 microprocessor, but apply equally well to other microprocessors.

ASCII codes are given in hexadecimal only.

NOTE

The TNIX operating system does **not** *support Series A software.*

TNIX OPERATING SYSTEM FEATURES

Filenames

TNIX filenames may contain up to 14 characters. If you enter a filename longer than 14 characters, the excess characters are discarded. Uppercase and lowercase characters are considered **different**. A filename may be made up of any ASCII characters except for NULL (ASCII code 00) and slash (ASCII code 2F). You should also avoid using any of the following special characters in filenames until you are aware of each character's significance:

```
( ) * ? \ ' " $ /
\ < > ; [ ] & ?
```

Refer to the discussion under the heading "Interpretation of Special Characters" in Section 5 (The TNIX Shell) of your 8560 MUSDU System Users Manual for a better understanding of these characters.

In addition, you should avoid using the **dash** (-) as the **first** character of a filename. The dash is used in command names as a command option designator and has special meaning to the linker, library generator and many TNIX commands.

Filespecs

A filespec is a sequence of filenames, separated by slashes, that defines the directory path of a file. Each filename in the filespec identifies a directory along the path; the final filename represents the file itself.

Command Names

A command name is a word that represents a command. TNIX command names must be entered exactly as specified in the 8560 MUSDU System Users Manual. The TNIX operating system differentiates between upper and lower case characters in command lines.

Redirection of Standard Output

The **-l** command option is used with the linker or the library generator to write a listing file to standard output. Standard output is normally assigned to the terminal (**/dev/tty***nn*, where *nn* is the terminal number), but may be redirected using a greater-than sign (>). For example:

```
link -l -O prog.obj sub.obj -o load >lnkl
```

With this invocation, the listing file will be written to a file named **lnkl**, rather than to the terminal. See the discussion "Output Redirection" in Section 5 (The TNIX Shell) of your 8560 MUSDU System Users Manual for more information about redirecting output.

SOFTWARE INSTALLATION

This procedure describes how to copy the software for the 8086/8088 Series B assembler from the assembler installation flexible disk to the 8560 system fixed disk.

NOTE

The linker and library generator software is already present on your 8560 system fixed disk.

To complete this installation procedure you need the following items:

- an 8560 Multi-User Software Development Unit
- an 8086/8088 assembler software installation flexible disk

This installation procedure adds about 5 files to your 8560 system fixed disk (device **/dev/hd0**).

This installation procedure will work regardless of which Series B assembler you are installing. See Table 8B-1.

Install the Software

NOTE

You **must** *have super-user* **status** *(for example, be logged in as* **root**) in order to install software.

Now place your 8086/8088 assembler installation flexible disk in drive 0 of the 8560 flexible disk device (device name is **/dev/fd0**) and close the drive door.

The **install** command installs the assembler software; simply type:

```
# install <CR>
```

Your assembler software is installed once the TNIX operating system responds with its prompt. The dollar sign ($) is the standard prompt; however, for the superuser, the standard prompt is the pound sign (#).

After entering the **install** command, wait for the superuser prompt to return. This indicates that your assembler software has been installed. You may now remove your assembler flexible disk and:

- install more software on your 8560 system fixed disk, or
- continue with the 8086/8088 Assembler Demonstration Run that follows in this section.

You can install more than one assembler on your 8560 system fixed disk.

ASSEMBLER SOFTWARE SELECTION

To tell the TNIX operating system which assembler software to use, you must set the **uP** TNIX shell environment variable to the proper assembler name.

For example, in order to select the 8086/8088 assembler software, the following command line would be entered:

```
$ uP=8086;export uP <CR>
```

The assembler name. See Table 8B-1.

NOTE

You need only enter the ";**export uP***" part of the indicated command line* **once.** *Each succeeding time, you may enter just "***uP***=micro", where "micro" is the appropriate assembler software choice (from Table 8B-1).*

You may alternatively select other assembler software, but only if such software has been installed.

Table 8B-1 shows what assembler names may be used in order to select the desired assembler software.

Table 8B-1
Assembler Selection by Setting
the uP Shell Variable

Assembler Selected	Name(s) to Use in uP Declaration
68000	68000
Z8001/Z8002	Z8000 Z8001 Z8002 z8000 z8001 z8002
8086/8088	8086 8087 8088
6809	6809
Z80A	Z80 z80
8080A/8085A	8080 8085
6800/6801/6802	6800 6801 6802 6803
8048/8021/8041A/8022	8048 MCS84 mcs48 8021 8041 8022

Notice in Table 8B-1 that most assemblers may be selected by more than one name. For instance, the 8086/8088 assembler software may be selected using any of the following names: **8086**, **8087**, or **8088**.

ASSEMBLER INVOCATION

The assembler is invoked by the operating system command **asm**. The **asm** command has the following syntax:

asm [object] [listing] source...

object—filespec of object file to be produced
listing—filespec of listing file to produced
source—filespec(s) of source file(s) to be assembled

Assembler invocation is discussed in detail in The Assembler section of this manual.

LINKER INVOCATION

The linker is invoked by the operating system command **link**. The **link** command has the following syntax:

link command-option...

Linker invocation is discussed in detail in The Linker section of this manual.

LIBGEN INVOCATION

The library generator (LibGen) is invoked by the operating system command **libgen**. The **libgen** command has the following syntax:

libgen command-option...

LibGen invocation is discussed in detail in The Library Generator section of this manual.

NOTE

LibGen creates some temporary files during execution. These files are deleted when LibGen terminates normally. However, if LibGen terminates abnormally (fatal error, pressing the CTRL-C (control-C) key, system crash), these temporary files remain on the 8560 system fixed disk. The temporary file names all begin with ###**.lib.tmp**. *The TNIX command that would be entered to delete all such files is:*

```
rm ###.lib.tmp*
```

DEMONSTRATION RUN

Introduction

This demonstration run shows you how to enter, modify, assemble, link, and load a simple 8086/8088 program and subroutine. If you are using a different microprocessor, you must change the microprocessor dependent instructions to similar instructions that are valid for the selected microprocessor.

The purpose of this demonstration is to give you the basic information and experience you will need to begin using the assembler, linker, and library generator. For your convenience, the sample program and subroutine are short and trivial. Only a few features of the assembler and linker are demonstrated, and the library generator is not discussed.

To perform this demonstration run you need the following items:

- an 8560 Multi-User Software Development Unit
- an 8540 Integration Unit **or** an 8550 Microcomputer Development Lab

Demonstration Run Hardware Configurations

This discussion assumes that the 8560 and 8540/8550 units are already set up and running. The hardware configuration assumed depends on whether you use an 8540 or an 8550 with the 8560. The following two paragraphs describe the two hardware configurations that may be used in this demonstration run.

Using an 8540 with the 8560. The cable from the terminal is connected to a terminal I/O port (port numbers 0–7) on the 8560. The High Speed Interface (HSI) Cable is connected between the HSI port on the 8540 and an HSI I/O port on the 8560. When you connect the 8540 to the 8560, set the jumpers on the 8560 I/O Adapter board to the HSI position.

Using an 8550 with the 8560. The cable from the terminal is connected to a terminal I/O port (port numbers 0–7) on the 8560. The RS-232-C cable is connected between the REMOTE port on the 8550 and a High Speed Interface (HSI) I/O port on the 8560. When you connect the 8550 to the 8560, jumpers on the 8560 I/O Adapter board **must** be set to the RS-232-C position in the 8560.

For more information on these and other hardware configurations, refer to the following discussions in your 8560 MUSDU System Users Manual: "TERM MODE" in Section 7 (Intersystem Communication) and "Intersystem Communication" in Section 2 (Operating Procedures).

8560 to 8540/8550 Communications

In order to communicate from the 8560 Multi-User Software Development Unit to either an 8540 Integration Unit **or** an 8550 Microcomputer Development Lab, you must inform the 8560 which 8560 I/O port (channel) the 8540/8550 is plugged into. The form of the command line to accomplish this is:

```
IU=n; export IU      [needed each time you log in]
```

Notice that *n* is a number from 0 to 7 specifying which 8560 I/O port that the 8540/8550 is plugged into. This allows 8540/8550 operating system commands to be sent through the 8560 to the 8540/8550.

NOTE

If you have connected an 8550 to your 8560 and have set the jumpers on the 8560 I/O Adapter board to the **RS-232-C** *position, you must enter the following command (***superusers** *only) to enable HSI protocol on an RS-232-C Line.*

```
stty IU >/dev/ttyn    [needed only once as long as
                       the TNIX system is up]
```

*Again, the port number (0–7) is specified by n. Some I/O ports may have already been set for High Speed Interface (HSI) protocol by your system manager. The I/O protocol may be defined for each 8560 I/O port by the file /***etc/ttys.**

It may be helpful to refer to the heading "TTYS(5)" in Section 5 (File Formats and Conventions) in your 8560 MUSDU System Reference Manual.

Once communication is established between the 8560 and an 8540/8550, you may enter an 8540/8550 command and it will be executed by the 8540/8550.

Command Line Conventions

In this demonstration run, each command line that is to be typed in by **you** will use the following conventions:

- Underlined—Underlined characters in a command line must be entered from your system terminal. Those characters not underlined are system output.
- <CR>—Each command line ends with an end-of-line character. The end-of-line character used here is the carriage return (ASCII code 0D). When a carriage return is to be entered, the symbol <**CR**> is used. This is **not** to be confused with the end-of-line character (the line feed, ASCII code 0A) used to end each line in any file stored on the 8560 system fixed disk.

Preparation

To accomplish this demonstration run you should have a basic understanding of the 8560 Multi-User Software Development Unit and the TNIX Text Editor. If you need to review how the 8560 and its editor work, refer to Sections 1 (Learning Guide) and 4 (The TNIX Editor) of the 8560 MUSDU System Users Manual.

You will need about 60 minutes to complete this demonstration run.

NOTE

If you have not installed the 8086/8088 assembler software yet, then do so before continuing this demonstration run.

Install the appropriate assembler software. Select the 8086/8088 assembler software by setting the uP TNIX shell environment variable to the proper assembler name (See Table 8B-1).

For example, to select the 8086/8088 assembler software, enter:

```
$ uP=8086; export uP <CR>
```

The assembler name. See Table 8B-1.

Now enter the following command to create the new directory, **asm.demo**:

```
$ mkdir asm.demo <CR>
```

Now cause **asm.demo** to become the working directory using the following command:

```
$ cd asm.demo <CR>
```

Examine the Sample Subroutine and Main Program

Figure 8B-1 lists the 8086/8088 subroutine and program you will enter, assemble, link, and load in the demonstration run. Similar instructions for any other microprocessor may be substituted without changing the validity of the demonstration run.

Subroutine OUTSUB:

```
        TITLE   'SAMPLE SUBROUTINE'
        NAME    SUBSMOD
        GLOBAL  PORTN,OUTSUB
        SECTION SUBS1
; SUBROUTINE OUTSUB -- OUTPUTS A CHARACTER
OUTSUB  OUT     #PORTN,AL       ; OUTSUB STARTS HERE.
        RETS                    ; RETURN TO PROGRAM.
        END
```

Main Program:

```
        GLOBAL  PORTN,OUTSUB
PORTN   EQU     15H             ; PORT = 15H
START   MOVB    AL,#'?'         ; CHARACTER = '?'
        CALLS   OUTSUB,OUTSUB   ; SEND '?' TO PORT 15
        HLT                     ; ... AND STOP.
        END     START
```

Fig. 8B-1. Source code for sample subroutine and main program.

Subroutine OUTSUB outputs a single ASCII character to an I/O port specified by PORTN. The main program specifies an I/O port and a character and calls OUTSUB to send the character to that port. The subroutine and main program are discussed in more detail later in this section.

The subroutine performs a trivial task: it outputs the ASCII character stored in the 8-bit accumulator AL to an I/O port specified by the symbol PORTN.

NOTE

For the 8086/8088 microprocessor, the value for PORTN was chosen arbitrarily. If your I/O ports are configured differently, you must alter this demonstration program before you can run it successfully on your system.

The main program places a character in the accumulator, calls the subroutine to send the character to the I/O port, and then halts.

You can think of the subroutine as a carefully prepared component of a major programming project. The main program can be viewed as a quickly written test for the subroutine.

Assembly Language Statements

An assembler source module is made up of assembly language statements. There are three types of assembly language statements:

- An **assembly language instruction** is translated by the assembler into a machine instruction.
- An **assembler directive** indicates a special action to be taken by the assembler. Assembler directives define data items, constants, and variables; provide information to the linker; control macros and conditional assembly; and specify options for the assembler and linker listings.
- A **macro invocation** is replaced by the statements of the macro it invokes. (Macro invocations are not discussed in this demonstration.)

Each assembly language statement has four fields. Each field may vary in width, and certain fields may be blank. However, the fields always occur in the following order:

1. The **label** field. The label field always begins in column 1 of the statement. The label allows the statement to be referenced by other statements. The label usually represents the address of the instruction or data item represented by the statement.
2. The **operation** field. The word in the operation field indicates the type of action to be taken by the assembler. The word may be an assembler directive, an assembler mnemonic, or the name of a macro. If the word is an assembler mnemonic, the assembler translates the statement into a machine instruction.
3. The **operand** field. The operand field completes the assembly language statement. Most assembler directives and assembler instructions contain one or more operand expressions. The type and number of operands depend on the operation.
4. The **comment** field. Comments are used for program documentation only; they have no effect on assembly. A semicolon (;) indicates that the remainder of the line is a comment. A comment may follow the operand field, or may begin with a semicolon in column 1 and take up an entire source line.

Explanation of the Subroutine Source Code

The following text explains each statement in the sample subroutine (shown in Fig. 8B-1). The two statements preceding the END statement are 8086/8088 instructions. If you are using any processor **other** than the 8086/8088, substitute equivalent instructions for these two statements. The rest of the statements are assembler directives.

```
TITLE   'SAMPLE SUBROUTINE'
```

The phrase 'SAMPLE SUBROUTINE' will appear in the heading on each page of the assembler source listing.

```
NAME    SUBSMOD
```

When the subroutine is assembled, the resulting object module will be named "SUBSMOD".

```
        GLOBAL  PORTN,OUTSUB
```

PORTN and OUTSUB are declared as global symbols, since each symbol is given a value in one module and referred to in another module. OUTSUB is a **bound global** because it is defined in this module. PORTN is an **unbound global** because it is **not** defined anywhere in this module.

```
        SECTION SUBS1
```

Each object module is composed of one or more sections. The linker treats each section as a separate unit: sections from the same module may be placed in different areas of memory. The section in object module SUBSMOD will be called SUBS1. (If you were to add more sections to this source module, they might be called SUBS2, SUBS3, and so on.)

NOTE

The assembler directives SECTION, COMMON, and RESERVE each declare a different type of section, and may also specify restrictions on the relocatability of the section. When no restriction is specified, the section is given the default relocation type. The default relocation type is microprocessor dependent. For the 8086/8088, the default relocation type is alignment on 16-byte boundaries (ALIGN 16). See the Assembler Specifics section of this manual for the default relocation type for your microprocessor. The Linker section of this manual contains an explanation of the six attributes of a section: name, class name, section type, relocation type, size, and memory location.

```
; SUBROUTINE OUTSUB -- OUTPUTS A CHARACTER
```

This is a comment.

```
OUTSUB OUT     #PORTN,AL        ; OUTSUB STARTS HERE.
```

This 8086/8088 instruction outputs the contents of the 8-bit accumulator AL to an I/O port specified by PORTN. The symbol OUTSUB becomes defined as the address of this instruction, which is the first instruction in the subroutine.

```
        RETS                     ; RETURN TO PROGRAM.
```

This 8086/8088 instruction returns control to the calling program.

```
        END
```

This assembler directive marks the end of the source module.

Explanation of the Main Program Source Code

The following text explains each statement in the sample main program (shown in Fig. 8B-1). The program contains three assembler directives (GLOBAL, EQU, and END) and three 8086/8088 instructions (MOVB, CALLS, and HLT).

```
        GLOBAL  PORTN,OUTSUB
```

As in the subroutine, PORTN and OUTSUB are global symbols. However, in this module PORTN is a bound (defined) global while OUTSUB is an unbound (undefined) global. The GLOBAL statements allow the two modules to share the I/O port address and the address of the subroutine.

```
PORTN  EQU     15H              ; PORT = 15H
```

This assembler directive assigns the hexadecimal value 15 to the symbol PORTN. "PORTN" becomes synonymous with the constant 15H.

```
START  MOVB    AL,#'?'          ; CHARACTER = '?'
```

This 8086/8088 instruction loads the hexadecimal value 3F (the ASCII code for a question mark) into the 8-bit accumulator AL. This statement is given a label, "START", so the END statement may refer to it.

```
       CALLS   OUTSUB,OUTSUB    ; SEND '?' TO PORT 15
```

This 8086/8088 instruction transfers control to the instruction labeled OUTSUB in the segment of memory whose base address is specified by the address OUTSUB. The second operand of an intersegment CALLS must be the base address of the segment of memory in which the called subroutine resides. Since OUTSUB is the location of the first byte in the section SUBS1, OUTSUB defines the base address of the section. The subroutine sends the ASCII code for the question mark to I/O port 15H.

```
       HLT                      ; ... AND STOP.
```

Control returns from the subroutine to this 8086/8088 instruction. The HLT instruction halts program execution.

```
       END     START
```

This assembler directive terminates the source module and indicates that program execution should begin with the instruction labeled "START". START is called the **transfer address**. The transfer address is passed through the assembler and linker to the 8540/8550 operating system **lo** and **g** commands.

Notice that this program source module does not contain a TITLE, NAME, or SECTION directive. The following default conditions result:

- No special title appears in the page heading of the source listing.
- The object module is called *NONAME*.
- The one section in *NONAME* is given a default name, section type, and relocation type.

Naming Files

This demonstration run produces several files. To give each file a name that reflects its contents and importance, we suggest you use the following file naming standards:

- The first part of the file name is an optional descriptive name followed by a period.
- The last part of the file name is a 3-or 4-character identifier that reflects the file type.

The following files will be produced:

File Name	Description	How Created
sub.asm	*sub*routine *asse*mbler source file	by you
sub.obj	*sub*routine *obj*ect file	by assembler
sub.asml	*sub*routine *asse*mbler *l*ist file	by assembler
prog.asm	*prog*ram *asse*mbler source file	by you
prog.obj	*prog*ram *obj*ect file	by assembler
prog.asml	*prog*ram *asse*mbler *l*ist file	by assembler
load	Program *load* file	by linker
lnkl	Program *link*er *l*isting file	by linker

Create the Subroutine Source File

The TNIX Text Editor, **ed**, helps create and modify source files. See Section 4 (The TNIX Editor) in your 8560 MUSDU System Users Manual for a more complete explanation of **ed**.

How to Correct Typing Mistakes in the Editor

Before you begin to input text, you need to know how to correct your mistakes on the input line. You may correct only the line you are currently editing. Once you have entered a carriage return to end the line, no further corrections can be made to that line while in input mode.

Deleting Characters One-by-One During Input Mode. The **BACKSPACE** key on a CRT terminal cancels the character above the cursor and moves the cursor one space to the left. The same thing results when a CTRL-H (control-H is the default TNIX erase character command) is entered.

Deleting an Entire Line During Input Mode. If you are entering a line of text and decide for some reason (bad text perhaps) that you would like to replace the **entire** line and you have **NOT** entered the carriage return yet, you may enter a CTRL-U (control-U is the default TNIX kill line command) to delete the line. TNIX recognizes a CTRL-U entry as a command to delete the entire input buffer entry. Thus after entering a CTRL-U, simply continue typing as though you had never entered the bad text line.

If you have typed in the bad text line and entered the carriage return, you may replace the bad line by using the following procedure:

```
Old-line-you-just-typed <CR>      [Your current text line]
. <CR>                            [Terminate input mode]
c <CR>                            [Delete bad line and enter input mode]
type-in-new-line <CR>             [Text is appended to line preceding deleted line]
... continue with text input
```

Correcting Errors in a Line That Has Already Been Entered. The general command procedure is:

```
.  <CR>                          [Terminate input mode]
/error/  <CR>                    [Search for string "error"]
line-with-error                  [Display line with string "error"]
s/error/the-correction/p  <CR>   [Substitute "error" with "the-correction"]
line-with-the-correction         [Corrected line is displayed]
```

These commands:

- Terminate input mode.
- Search for the first occurrence of a line with the character string **"error"**. This line becomes the current line and is echoed to the system terminal.
- Substitute the characters **"error"** with the characters **"the-correction"**. The corrected line is then displayed.

Start Editing

The TNIX command **ed** invokes the TNIX Text Editor (named **ed**). The filename that follows the command indicates the file to be edited. In this demonstration, the name **sub.asm** is used. The extension (**asm**) indicates an assembler source file.

The editor does not respond with a prompt after performing the entered command. Rather, **ed** responds by moving the cursor to the beginning of the next line after displaying the appropriate information (if any) for that command. Enter the following command line to begin the editing session that creates **sub.asm**, the subroutine source file:

```
$ ed sub.asm <CR>
```

After you enter this command, the editor responds:

```
?sub.asm
```

The editor is now waiting for a command. Enter input mode using the **a** command and then type in the subroutine.

NOTE

Be sure to misspell **GLOBAL** *in the third line of text. This deliberate typing error will be used to illustrate features of the assembler and editor.*

```
a <CR>
        TITLE 'SAMPLE SUBROUTINE' <CR>
        NAME    SUBSMOD <CR>
        GLOABL  PORTN,OUTSUB <CR>
        SECTION SUBS1 <CR>
; SUBROUTINE OUTSUB -- OUTPUTS A CHARACTER.  <CR>
OUTSUB OUT     #PORTN,AL        ; OUTSUB STARTS HERE.  <CR>
        RETS                    ; RETURN TO PROGRAM.  <CR>
        END <CR>
. <CR>
```

Insert mode is terminated by typing a single period followed by a carriage return (. <CR>) on an empty line. Typing the period directs the editor to enter **command** mode. This is signified by the cursor moving to the next line. You may now enter your next editor command.

Examining the Program Text Entered in the File. The text that you just entered is stored in the editor workspace. To display the workspace contents from beginning to end, enter the following command:

```
1,$p <CR>
        TITLE 'SAMPLE SUBROUTINE'
        NAME    SUBSMOD
        GLOABL  PORTN,OUTSUB
        SECTION SUBS1
; SUBROUTINE OUTSUB -- OUTPUTS A CHARACTER.
OUTSUB OUT     #PORTN,AL        ; OUTSUB STARTS HERE.
        RETS                    ; RETURN TO PROGRAM.
        END
```

Examine the listing of the text. If you discover an error, don't panic, you will soon learn how to correct the misspelling of **GLOBAL** and you may apply it to any misspellings of your own.

Now enter the following commands to copy the text in the workspace out to the new source file and end the editing session:

```
w <CR>
265
q <CR>
$
```

Thus we see that

- The **w** command writes the contents of the editor workspace to the file (**sub.asm**) you specified in the **ed** command line (**ed sub.asm**).
- The **265** indicates the number of characters that are in the file you have just produced.
- The **q** command closes the editing session and returns control to the TNIX operating system.

The TNIX prompt (**$**) indicates that you have exited the editor. You may now enter your next TNIX command.

Assemble the Subroutine and Examine Any Errors

The **asm** command invokes the assembler and specifies the source file(s) to be assembled and the object and listing files to be produced. To scan source file **sub.asm** for errors, enter the following command:

```
$ asm "" "" sub.asm <CR>
```

Omitting the filespecs of the object and listing files has two advantages:

1. The assembler runs faster because it produces no object code or listing.
2. The **asm** command line is shorter.

You may want to omit these filespecs from your **asm** command line whenever you suspect that your source code contains errors.

The assembler responds as follows on your system terminal:

```
ASM 8086/8088 Vxx.xx-xx Copyright (C) 1981 Tektronix, Inc.
*****Pass 2
     3 00000000 00000000              GLOABL  PORTN,OUTSUB
*** ASM: 107(E)  Undefined opcode "GLOABL"
     6 00000000 E600      OUTSUB OUT      #PORTN,AL      ; OUTSUB STARTS HERE.
*** ASM:  21(E)  Undefined operand
    8 Lines Read
    8 Lines Processed
    2 Errors
```

The assembler's response can be interpreted as follows:

```
ASM 8086/8088 Vxx.xx-xx Copyright (C) 1981 Tektronix, Inc.
```

The assembler announces itself as it begins executing. The assembler reads through your source file twice. The first time through (Pass 1), the assembler makes a list of symbols that appear in the source code and tries to assign an address or value to each symbol.

```
*****Pass 2
```

The assembler begins its second pass through your source file. During Pass 2 the assembler produces the object and listing files and displays error messages and statistics.

```
     3 00000000 00000000              GLOABL  PORTN,OUTSUB
*** ASM: 107(E)  Undefined opcode "GLOABL"
```

The assembler cannot translate the above statement because "GLOABL" is not an 8086/8088 mnemonic, an assembler directive, or the name of a macro. The erroneous source line and the error message would appear in the listing (if any) just as they appear on the system terminal. The three numbers to the left of the statement will be explained when you examine an assembler listing later in this demonstration run.

```
     6 00000000 E600          OUTSUB OUT      #PORTN,AL    ; OUTSUB STARTS HERE.
*** ASM:  21(E)  Undefined operand
```

Because the assembler did not understand the GLOABL statement, it does not know that PORTN is a global symbol. The assembler expects PORTN to be defined in this module.

```
8 Lines Read
8 Lines Processed
2 Errors
```

These lines summarize the assembler's activities. There are eight lines of code in your source file. The number of lines read differs from the number of lines processed only in programs that contain macros or conditional assembly.

The two errors, already discussed, produced the two undefined symbols GLOABL, and PORTN.

Correct the Error in the Subroutine Source Code

Both errors detected by the assembler arose from the misspelling of "GLOBAL" in line 3 of the source file, **sub.asm**. Invoke the editor so that you may correct the misspelling:

```
$ ed sub.asm <CR>
265
```

Enter the following command line. This command searches for the character string **"GLOABL"** and then displays the first line that contains the string.

```
/GLOABL/ <CR>
        GLOABL  PORTN,OUTSUB
```

Now the workspace pointer is at the line you want to modify. Use the **s** command to reverse the letters "A" and "B" in "GLOABL":

```
s/AB/BA/p <CR>
        GLOBAL  PORTN,OUTSUB
```

The modified line is displayed.

As before, enter the following commands to copy the edited text in the workspace out to the source file (**sub.asm**) and end the editing session:

```
w <CR>
265
q <CR>
$
```

Re-Assemble the Subroutine

Enter the following command to create an object file (**sub.obj**) and an assembler listing file (**sub.asml**) from the subroutine source file:

```
$ asm sub.obj sub.asml sub.asm <CR>
ASM 8086/8088 Vxx.xx-xx Copyright (C) 1981 Tektronix, Inc.
*****Pass 2
    8 Lines Read
    8 Lines Processed
    0 Errors

$
```

This time the assembler finds no errors.

Examine the Subroutine Listing

In order to examine the assembler listing stored on the file **sub.asml**, copy the file to your line printer:

```
$ lp1r sub.asml <CR>
```

If you have no line printer, use the **cat** command to list the file on your system terminal. (Remember that you may use CTRL-S to suspend and CTRL-Q to resume display on the system terminal.)

```
$ cat sub.asml <CR>
```

Figure 8B-2 shows the listing of the sample subroutine.

```
ASM       8086/8088 SAMPLE SUBROUTINE                          Page   1
Vxx.xx-xx  (8560)                                    dd-mmm-yy/xx:xx:xx

     2                                NAME    SUBSMOD
     3                                GLOBAL  PORTN,OUTSUB
     4                                SECTION SUBS1
     5                          ; SUBROUTINE OUTSUB -- OUTPUTS A CHARACTER.
     6  00000000 E600 RU        OUTSUB OUT     #PORTN,AL    ; OUTSUB STARTS HERE.
     7  00000002 CB                     RETS                ; RETURN TO PROGRAM.
     8                                END


ASM       8086/8088 SYMBOL TABLE                               Page   1
Vxx.xx-xx  (8560)                                    dd-mmm-yy/xx:xx:xx


Section = SUBS1, Aligned to 00000010, Size = 00000003


OUTSUB------------00000000 G


Section = %SUBOBJ, Aligned to 00000010, Size = EMPTY



Unbound Globals

PORTN-------------00000000


   8 Lines Read
   8 Lines Processed
   0 Errors
```

Fig. 8B-2. Assembler listing for the sample subroutine.

The command **asm sub.obj sub.asml sub.asm** produces this listing file from the subroutine source file. The command **lp1r sub.asml** copies the listing file to the line printer.

Every assembler listing has two parts: the source listing and the symbol table. Each page of the listing begins with a standard page heading.

The Source Listing

Page 1 of your listing contains the source listing. The heading includes the words "SAMPLE SUBROUTINE", which you supplied with the TITLE directive.

Each line of the source listing may contain the following information:

1. the line number (decimal);
2. the location counter (hexadecimal) of the object code generated (if any);
3. the assembled object code (hexadecimal) or result value;
4. a line indicator (indicates macro expansion, string substitution, relocation, text from include file, or unbound global reference);
5. the source statement.

If any statement contains an error, the appropriate error message appears directly after the statement.

Examine each line of your source listing:

- Line 1 (The TITLE directive) is not printed because it is a listing control directive.
- Lines 2, 3, 4, and 8 are assembler directives that produce no object code. The information they provide is stored in special areas of the object module.
- Line 5 is a comment.
- Lines 6 and 7 are 8086/8088 assembly language instructions:

  The 8086/8088 instruction OUT #PORTN,AL produces the two-byte machine instruction E600. E6 is the hexadecimal opcode for the OUT instruction. The dummy value 00 is used for the I/O port number until the linker supplies a value for PORTN. The machine instruction E600 is stored in bytes 00000000 and 00000001 of section SUBS1. **RU** indicates a relocation and unbound global symbol.

  The 8086/8088 instruction RETS produces the one-byte machine instruction CB, which is stored in byte 00000002 of section SUBS1.

  If you are not using an 8086/8088 microprocessor, similar statements should be substituted for these two statements.

The Symbol Table

Page 2 of your listing contains the symbol table, which indicates the value and type of each symbol in your source code.

The assembler symbol table is divided into the following categories:

- Scalars (numeric values other than addresses)
- Strings and macros
- Sections (and addresses within each section)
- Unbound globals
- Undefined symbols
- Statistics

The statistics at the bottom of the symbol table are the same statistics that appeared on the system terminal when the assembler finished execution.

Create the Main Program Source File

Now that you have created, corrected, and assembled the sample subroutine, it is time to create the main program that uses the subroutine. Enter the following command to begin the editing session that creates the main program source file, **prog.asm**:

```
$ ed prog.asm <CR>
?prog.asm
```

The editor is now waiting for a command. Enter input mode using the **a** command and then type in the main program. This time, however, don't include any typing errors.

```
a <CR>
        GLOBAL  PORTN,OUTSUB <CR>
PORTN   EQU     15H             ; PORT = 15H <CR>
START   MOVB    AL,#'?'         ; CHARACTER = '?'  <CR>
        CALLS   OUTSUB,OUTSUB   ; SEND '?' TO PORT 15 <CR>
        HLT                     ; ... AND STOP.   <CR>
        END     START <CR>
. <CR>
```

As before, enter the following commands to copy the text in the workspace out to the source file (**prog.asm**) and return control to the TNIX operating system.

```
w <CR>
242
q <CR>
$
```

Assemble the Main Program

Enter the following command line to create an object file (**prog.obj**) and a listing file (**prog.asml**) from the main program source file:

```
$ asm prog.obj prog.asml prog.asm <CR>
ASM 8086/8088 Vxx.xx-xx Copyright (C) 1981 Tektronix, Inc.
*****Pass 2
    6 Lines Read
    6 Lines Processed
    0 Errors

$
```

The main program contains no errors.

Examine the Main Program Listing

Copy the assembly listing to the line printer or to the system terminal:

```
$ lp1r prog.asml <CR>

or

$ cat prog.asml <CR>
```

```
ASM       8086/8088                                          Page   1
Vxx.xx-xx  (8560)                                    dd-mmm-yy/xx:xx:xx

  1                                GLOBAL  PORTN,OUTSUB
  2                 15     PORTN   EQU     15H             ; PORT = 15H
  3  00000000 B03F         START   MOVB    AL,#'?'         ; CHARACTER = '?'
  4  00000002 9A000000 R           CALLS   OUTSUB,OUTSUB   ; SEND '?' TO PORT 15
     00000006 00
  5  00000007 F4                   HLT                     ; ... AND STOP.
  6                  0            END     START


ASM       8086/8088 SYMBOL TABLE                             Page   1
Vxx.xx-xx  (8560)                                    dd-mmm-yy/xx:xx:xx

Scalars

PORTN-------------00000015


Section = %PROGOBJ, Aligned to 00000010, Size = 00000008

START-------------00000000


Unbound Globals

OUTSUB------------00000000



  6 Lines Read
  6 Lines Processed
  0 Errors
```

Fig. 8B-3. Assembler listing for the sample main program.

The command **asm prog.obj prog.asml prog.asm** produces this listing file from the main program source file. The command **lp1r prog.asml** copies the listing file to the line printer.

Compare the listing of the sample main program (Fig. 8B-3) with the listing of the sample subroutine (Fig. 8B-2).

The Source Listing

Page 1 of your assembler listing contains the source listing. Notice that there is no user-defined title for the program listing: the source code did not contain a TITLE directive.

Examine each line of the program source listing:

- Line 1 declares PORTN and OUTSUB as global symbols. The GLOBAL statement produces no object code.
- Line 2 is an EQU statement that assigns the value 15 (hexadecimal) to the symbol PORTN. The symbol PORTN and its value are stored in the global symbol block of the program object module. At link time the value of PORTN will be substituted into the OUT instruction in the subroutine.
- Line 3 is an 8086/8088 assembly language instruction. MOVB AL,#'?' generates the machine instruction B03F. B0 is the opcode for "MOVB to AL" and 3F is the ASCII code for the question mark. The machine instruction B03F is stored in bytes 00000000 and 00000001 of the main program.
- Line 4 is an 8086/8088 assembly language instruction. CALLS OUTSUB,OUTSUB generates the machine instruction 9A00000000 in bytes 00000002-00000006 of the main program. 9A is the opcode for the CALLS instruction. The zeroes represent dummy values for OUTSUB that will be adjusted at link time.
- Line 5 is an 8086/8088 assembly language instruction. HLT produces the one-byte machine instruction F4 in byte 00000007 of the main program.
- Line 6 is an END statement that specifies that the transfer address is START, the address of the MOVB instruction. The transfer address will be adjusted if this section of object code is not loaded at the beginning of memory.

The Symbol Table

1. The scalars table (which lists every scalar in the symbol list and the value associated with each scalar) lists the scalar PORTN. The value of PORTN is 00000015 (hexadecimal).
2. The strings and macros table is again omitted because it is empty.
3. Because the main program source code contains no SECTION directive, the section produced by this assembler run is given the following default attributes:
   - name: %PROGOBJ (derived from the name of the object file);
   - section type: SECTION;
   - relocation type: aligned on a 16-byte boundary.

   Section %PROGOBJ contains eight bytes of code. START is the address of the first byte of the section.
4. OUTSUB is the only unbound (undefined) global symbol in the main program.

The statistics at the bottom of the symbol table are the same statistics that appeared on the system terminal when the assembler finished execution.

Link the Object Modules

Now both the subroutine and the main program have been translated into machine language. In order for the subroutine and main program object modules to communicate with each other, they must be linked. The linker performs the following tasks in creating a load file of executable object code:

- It locates each section in the specified object files to a block of memory.
- It adjusts addresses to reflect relocation of sections.
- It resolves global references.

Enter the following command to create a load file (**load**) and a linker listing file (**lnkl**) from your two object files:

```
$ link -0 prog.obj sub.obj -o load -l >lnkl <CR>
```

The linker responds as follows:

```
Tektronix Linker  Vxx.xx-xx  (8560)
Copyright (C) 1981 Tektronix, Inc.
```

The listing file is written to **lnkl** .

Examine the Linker Listing

Copy the linker listing file to the line printer or system terminal:

```
$ lp1r lnkl <CR>
```

or

```
$ cat lnkl <CR>
```

Figure 8B-4 shows the linker listing.

```
Tektronix        8086/8088 Linker Vxx.xx-xx  (8560)          Page 1
For load

MODULE AND FILE MAP:

 LINK FILES:
    *NONAME*            prog.obj
    SUBSMOD             sub.obj


Tektronix        8086/8088 Linker Vxx.xx-xx  (8560)          Page 2
For load

MEMORY AND SECTION MAP:

NONAME:
            0 -   FFFFF

    %PROGOBJ              0-       7       8 SECTION ALIGNED  *NONAME*
    SUBS1                10-      12       3 SECTION ALIGNED  SUBSMOD
```

Fig. 8B-4. Linker listing. (part 1 of 2)

```
Tektronix        8086/8088 Linker Vxx.xx-xx  (8560)           Page 3
For load

MODULE AND SECTION MAP:

MODULE(S) IN LINK FILE(S):
  MODULE:  *NONAME*

    SECTION:   %PROGOBJ          0-       7       8 SECTION ALIGNED

  MODULE:  SUBSMOD

    SECTION:   SUBS1            10-      12       3 SECTION ALIGNED

       OUTSUB ---------10


Tektronix        8086/8088 Linker Vxx.xx-xx  (8560)           Page 4
For load

GLOBAL SYMBOL LISTING:

%PROGOBJ                 0    *NONAME*
ENDREL                  13    **********
OUTSUB                  10    SUBSMOD
                              *NONAME*
PORTN                   15    *NONAME*
                              SUBSMOD
SUBS1                   10    SUBSMOD


Tektronix        8086/8088 Linker Vxx.xx-xx  (8560)           Page 5
For load

STATISTICS:

Number of warning errors:      0
Number of errors:      0

Transfer address:        0


Load file is not relinkable
Load file is not useable for symbolic debugging
```

Fig. 8B-4. Linker listing. (part 2 of 2)

The command **link -O prog.obj sub.obj -o load -l >lnkl** produces this linker listing file. The command **lp1r lnkl** copies the listing file to the line printer.

The standard linker listing contains six parts:

1. The COMMAND LOG lists command processing errors and those command options read from a linker command file. If a command file is not used and there are no errors, the command log does not appear.
2. The MODULE AND FILE MAP (page 1) lists all the modules linked into the load file.
3. The MEMORY AND SECTION MAP (page 2) provides the following information:
   - The name of each logical memory area and its memory range.
   - The name of each section located within each logical memory area.
   - The module name for each section.
4. The MODULE AND SECTION MAP (page 3) provides the following information:
   - The name of each module linked.
   - The name of each section within each module.
   - The global symbols defined in each section.
5. The GLOBAL SYMBOL LISTING (page 4) lists the value assigned to each global symbol and the name of the module in which it was defined.
6. The STATISTICS (page 5) gives the number of errors, the transfer address, and whether the load module is relinkable or usable by the symbolic debugger. Extra lines may follow a particular global symbol line and simply list any modules that reference that same global symbol.

Load the Executable Object Code into 8540 or 8550 Memory

This demonstration run assumes that you have plugged your 8540 or 8550 into High Speed Interface (HSI) I/O port number **3** (on the 8560), type the following command in order to inform the 8560 which HSI I/O port the 8540 or 8550 is plugged into.

```
$ IU=3; export IU <CR>
$ stty IU >/dev/tty03 <CR>    [enter only if using the 8550]
```

The terminal you are using will now behave as though it were connected to the 8540/8550 that is connected to I/O port number **3** (except that you cannot go into local mode).

Before you load your object program into 8540 or 8550 memory (depending on which unit you are using), use the 8540/8550 command **f** to fill the beginning of 8540/8550 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the end of your code. Enter the following command to fill memory locations 000000-00001F with zeros:

```
$ f 0 01F 00 <CR>
```

Now copy the executable object code from the load file into 8540 or 8550 program memory:

```
$ lo <load <CR>
```

Bytes 000000-000012 of program memory now contain the 11 bytes of machine language that form the executable program.

The 8540/8550 command **d** displays the contents of a specified section of memory. Each byte is displayed as a two-digit hexadecimal number and as the ASCII character it represents (if any). Enter the following command to display the contents of memory locations 000000–00001F:

```
$ d 0 1F <CR>
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000000 B0 3F 9A 00 00 01 00 F4 00 00 00 00 00 00 00 00  .?..............
```

address of first byte displayed — **main program** — **memory not affected by the lo command** — **corresponding ASCII characters**

```
000010 E6 15 CB 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

subroutine — **memory not affected by the lo command** — **corresponding ASCII characters**

The main program occupies bytes 00000000–00000007. The subroutine occupies bytes 00000010–00000012.

Compare the relocatable object code produced by the assembler with the executable object code produced by the linker. (The addresses and object bytes adjusted by the linker are underlined.)

	Relocatable Object Code (from assembler listings)		Executable Object Code (from the Dump command)	
Source Code	**Address**	**Object Code**	**Address**	**Object Code**
MOVB AL, '?'	00000000	B03F	00000000	B03F
CALLS OUTSUB,OUTSUB	00000002	9A00000000	00000002	9A<u>00000100</u>
HLT	00000007	F4	00000007	F4
OUT PORTN,AL	00000000	E600	<u>00000010</u>	E6<u>15</u>
RETS	00000002	CB	<u>00000012</u>	CB

Note the adjustments made by the linker:

- The subroutine is relocated. It now occupies memory bytes 000010–000012.
- The address of the subroutine is substituted into the CALLS instruction.
- The I/O port number is substituted into the OUT instruction.

Enter the following command to establish the parent directory as the current directory.

```
$ cd ..  <CR>
```

".." causes the parent directory of asm.demo to become the working directory.

Summary of Demonstration Run

Enter the **ls** command to list the files you have created:

```
$ ls asm.demo <CR>
lnkl
load
prog.asm
prog.asml
prog.obj
sub.asm
sub.asm#
sub.asml
sub.obj
$
```

Recall the eight files you have created in this demonstration run:

- the two source files (**sub.asm** and **prog.asm**) you created using the editor;
- the two object files (**sub.obj** and **prog.obj**) and the two listing files (**sub.asml** and **prog.asml**) generated by the assembler;
- the load file (**load**) and the listing file (**lnkl**) generated by the linker.

When you corrected the misspelling of GLOBAL in your source file, the editor retained the original file **sub.asm** as a backup file named **sub.asm#**.

You have now finished the demonstration run. It showed how to:

- create a source file using the editor;
- create an object file from a source file using the assembler;
- create a load file from object files using the linker;
- copy the load file into memory, using the lo command;
- interpret listings generated by the assembler and linker.

NOTE

For the 8086/8088 assembler: if you wish to execute the demonstration program, then you must substitute a port number valid for your system into the definition of the variable PORTN. If the first few locations in memory are used for system vectors, then you must use the linker relocation capabilities to move the demonstration program to user memory.

If you would like to execute and monitor this program, you must first select the 8086 emulator software. This may be done using the 8540/8550 **sel** command. For example, in order to select the 8086 emulator software, the following command line would be entered:

```
$ sel 8086 <CR>
```

The emulator name.

Refer to the heading, "Demonstration Run", in Section 1 (Learning Guide) of the 8560 MUSDU System Users Manual to learn how to execute and monitor the program you have loaded into memory.

If you prefer, you can leave all the files you have created for future reference. However, if you want to delete these files, you can do so with one command:

```
$ rm -r asm.demo <CR>
```

This command deletes all the files in the directory **asm.demo** *and* the directory.

Section 9

ASSEMBLER SPECIFICS

Processor-specific information is contained in the Assembler Specifics supplement that accompanies each assembler. Each supplement is designed as a subsection to this manual.

The Assembler Specifics supplements are numbered as if they were separate sections of this manual. For example, the 8086/8088 supplement is labeled "Section 9A", and the first illustration is numbered "Fig. 9A-1". Similarly, other supplements are labeled Sections 9B, 9C, etc. Figures, pages, and tables are numbered accordingly.

Each subsection presents the following information:

- A brief summary of the processor's registers and addressing modes.
- A list of notational conventions used to describe the instruction set.
- The microprocessor instruction set in a notation acceptable to the given assembler.
- A list of reserved words for the given assembler.
- The page size for the assembler, as defined in the Linker section of this manual.
- Any processor-specific assembler error messages.
- Any irregularities that should be noted.

Each supplement has its own table of contents.