SOURCER™

COMMENTING DISASSEMBLER

SOURCER™

V COMMUNICATIONS

EDITION
September 1989

Part Number S0989-164
Printed in the United States

PUBLISHED BY
V COMMUNICATIONS, INC.
3031 Tisch Way, Suite 802
San Jose, CA 95128
(408) 296-4224

ACKNOWLEDGEMENTS
Product Development: Geoff Caras, Frank Van Gilluwe
Sourcer was written entirely in assembly language.
Documentation: Lisa Caras

Preface: Before You Begin

Sourcer™ is both very powerful and straightforward to use. Depending on the complexity of the task you have at hand, you may need very little information about Sourcer to get started using it. The information program on the Sourcer distribution disk contains all the information you need to install and get started using Sourcer.

To run this program, insert the Sourcer distribution diskette into your diskette drive, change to that drive, and enter readme. Then select any of the options in the menu. For example, enter 1 for a quick introduction to Sourcer.

Before you continue, use the information program to install Sourcer on your hard disk or to copy it to a backup diskette. Enter 2 from the main menu and then follow the instructions the program presents.

When you need more information, turn to this manual. It contains a more detailed description of Sourcer basics and complete instructions for using more advanced features.

Table of Contents

Chapter 1: Using Sourcer

Sourcer is a multi-pass, commenting disassembler. It creates documented listings and source code directly from memory and from standard COM, EXE, and other binary files. In many cases the code that Sourcer generates looks better and is easier to read than the original code. There are many applications for Sourcer. For example, you might use it to:

- Clarify undocumented code.
- Recreate lost source code.
- Easily modify programs without original source code.
- Document programs in English.
- Simplify maintenance of poorly documented programs.
- Solve bugs in application programs.
- Support programs no longer supported by their publishers.

In addition to a binary program file, you can use a *definition file* as input to Sourcer. A definition file is a text file that presets Sourcer options, defines memory segments, specifies comments, and controls data types and options for the program you are analyzing. For complete instructions for using definition files, see Chapter 4, "Using Definition Files."

Installing Sourcer

Use the Sourcer information program (readme) to install Sourcer on your hard disk or a backup diskette. To start this program enter the following DOS command:

```
readme
```

Follow the instructions provided by the program.

Starting Sourcer

To start Sourcer, change to the drive and directory in which you installed it. At the DOS prompt, enter:

```
sr
```

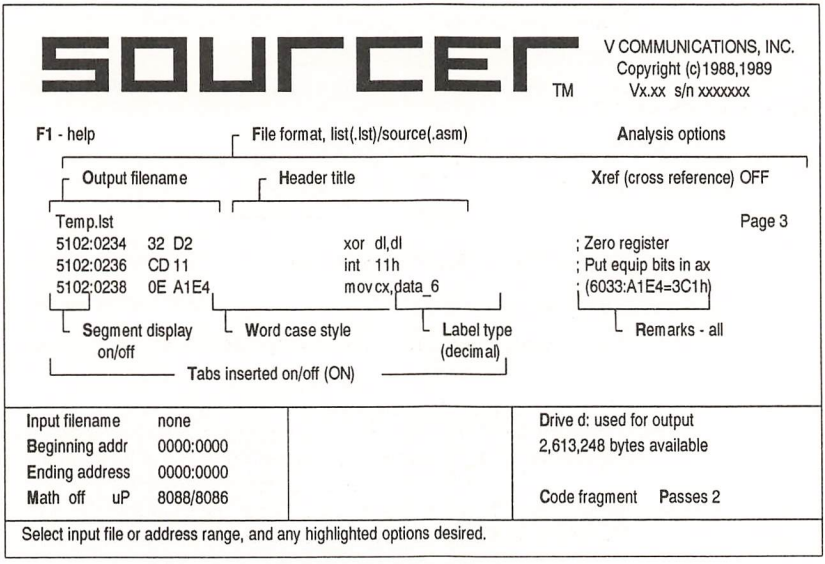
To load a file as Sourcer starts, enter:

```
sr filename
```

where *filename* is the filename or pathname for the file you want to load.

The Sourcer Main Menu appears. The top part of the Main Menu contains output formatting commands and four sample output lines. The position of the commands shows how they affect the output. The first letter of each command is highlighted. To select a command, press the keyboard key that matches the highlighted letter. For a description of each output formatting command, see Chapter 8, "Command Reference."

The lower part of the screen contains main processing options and a message box. The commands in this area let you control how Sourcer processes a file. To select a command, press the keyboard key that matches the highlighted letter. The message box displays instructions and informational messages.



Loading an Input File

To load an input file, press **I** from the Main Menu and then enter the name of the file. Sourcer loads the file, determines its beginning and ending addresses, determines its file type, and sets the processing options.

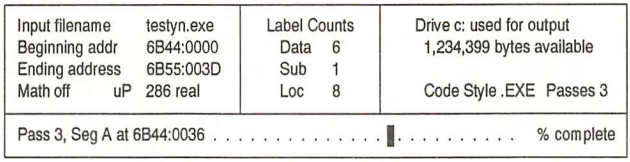
Before processing a file, you can change the output formatting options or processing options. For instructions, see Chapter 3, "Customizing Sourcer Operation."

Processing a File

To process a loaded file, choose an output file format (either listing or source code) by pressing **F** until Sourcer selects the format you want. To specify an output filename, press **O** and enter the filename. The filename can include a drive letter and a path. (Don't enter a filename extension; Sourcer automatically

appends an extension depending on the file type.) Press **G** to begin processing the file.

Sourcer displays progress status in the lower part of the screen. The following example shows status information for the demonstration program TESTYN.EXE.



The message box shows the current pass number, segment name, segment value, and offset. The sliding bar shows progress within each pass and segment. The label counts area shows the number of data, subroutine, and location references found as the analysis progresses.

When Sourcer finishes processing the file, it writes the output file and quits. A listing file has the name *filename*.LST where *filename* is the output filename you specified (or the name of the input file if you did not specify an output filename). Because the listing is 132 columns wide, you must view it with a viewing program or an editor that displays 132 columns or lets you scroll from left to right. For recommended programs, see Appendix D, "Other Useful Programs." For instructions on reading a listing file, see Chapter 2, "Reading a Sourcer Listing." A source code file has the name *filename*.ASM. Source code output is normally 80-columns wide.

In addition to a listing or source code file, Sourcer creates a definition file named *filename*.SDF. This file contains the settings you used to process the file and range and reference definitions created by Sourcer. You can modify this file, rename it, and use it to process the input file again. For a complete description of definition files and their use, see Chapter 4, "Using Definition Files."

Quitting Sourcer

Sourcer quits automatically when it finishes processing a file. To quit Sourcer at any other time, press Q or Esc.

An Example

The distribution diskette contains a sample test program, TESTYN.EXE. This program is designed to be used in a batch file. It checks for Y (yes) or N (no) input, and returns an error code 1 for Y, 0 for N, or beeps when the user presses any other key. The following example shows how it might be used.

Example 1-1: Batch File Containing TESTYN.EXE

```
ECHO Do you want to system format diskette A (Y/N)?
TESTYN
IF NOT ERRORLEVEL 1 GOTO EXIT
FORMAT A: /S /V
:EXIT
```

You can use Sourcer to create a commented source file or a listing of TESTYN.EXE. On the DOS command line, enter:

```
sr testyn.exe
```

To generate a listing using the standard option settings, press **G**. Sourcer writes the listing to a file named TESTYN.LST.

To generate source code instead of a listing, press **F** from the main menu to switch the file format to source code. Notice that the output filename extension changes and the listing formatting options disappear. Press **G** to begin processing. Sourcer writes the output source code to the file TESTYN.ASM.

The distribution diskette contains a definition and a remarks file for TESTYN. These files add descriptive labels and comments to the Sourcer output. To run Sourcer using the information in these files enter the following command on the DOS command line:

```
sr testyn
```

When you don't supply a filename extension for an input file, Sourcer looks first for a definition file (with the extension .DEF). In this case, the definition file TESTYN.DEF instructs Sourcer to load TESTYN.EXE and sets the processing options. Press **G** to

process TESTYN.EXE. Then examine the output file, TESTYN.LST, to see how the definition file added descriptive labels and comments.

Chapter 2: Reading a Sourcer Listing

A listing file contains memory locations, hex instructions, labels, code, and comments. The following illustration shows a partial listing.

Example 2-1: Sample Partial Listing

```
Line 1  6159:0031  3C 6E          cmp   al,6Eh          ; 'n'
     2  6159:0033  74 08          je    loc_3          ; Jump if equal
     3  6159:0035  9A 616A:0002   call  far ptr sub_1
     4  6159:003A  EB EB          jmp   short loc_1
     5  6159:003C             loc_2:
     6  6159:003C  B0 01          mov   al,1
     7  6159:003E  EB 02          jmp   short loc_4
     8  6159:0040             loc_3:
     9  6159:0040  32 C0          xor   al,al          ; Zero register
    10  6159:0042             loc_4:
    11  6159:0042  B4 4C          mov   ah, 4Ch        ; 'L'
    12  6159:0044  CD 21          int   2ih            ; DOS Services ah=function 4Ch,
    13                                                 ;  terminate with al=return code
    14  6159:0046  000B[00]       db    11 dup (0)
    15                        testyn  endp
```

The first two hex words contain the segment and offset where the instruction appears in memory.

Hex representations of the instructions appear next. Byte references consist of two digits, word references consist of four digits, double-word (dword) references consist of eight digits, and segment offset references consist of two sets of four digits with a colon separating them. In memory, the hex representations occur with the low byte first. Sourcer converts the byte order of references to make them more readable. For example, line 3 in the preceding example contains the hex instructions 9A

616A:0002. In memory, the actual byte order is 9A, 02, 00, 6A, 61.

To reduce the length of listings, duplicate data items use a condensed format. In the hex instruction for duplicate data, the first word indicates the number of duplicate bytes. The duplicate data value, enclosed in brackets, follows it. In line 14 of the example, the value 0 occurs 11 times.

Occasionally extra data bytes of value 0 appear at the end of a segment such as in line 14 of the example. The linker inserts these 0-value bytes into multiple-segment programs to fill the bytes to the segment boundary. They do not affect the operation of the program, and you can remove them if you want.

In code, Sourcer appends an h to hex numbers. Otherwise, numbers are decimal. The listing suppresses leading zeros in hex numbers except when the value of the first significant digit is A through F.

Equates

When it encounters an external data reference, Sourcer converts the reference to an equate and places at the beginning of the listing. An assembler can use the equate to recreate the same address when you reassemble the code.

Sourcer appends an e to internally generated equates. When the program uses equate data items internally, a segment override tells the assembler that the item is a data reference. The assembler will determine whether the segment override is actually necessary.

Example 2-2: Equates

```
= 0044   data_1e   equ   44h      ; (0000:0044=0FE23h)
= 0046   data_2e   equ   46h      ; (0000:0046=0F000h)
= FF79   data_3e   equ   0FF79h   ; (6159:FF79=0A5A5h)
= 923E   data_4e   equ   923Eh    ; (??:923E)
```

In the preceding example, Sourcer created the first two equates because the segment used with data_1e and data_2e was set to 0--outside the program's range.

The third equate in the example might occur when an indexed instruction has an opcode such as data_3e[si]. In this example,

data_3e has a value of FF79h--outside the range of this program. If analysis option U is off (see page 28), Sourcer displays index values from -64h to +64h as numbers in the opcode field rather than as references; they do not appear as equates.

If Sourcer determines that a segment reference cannot be valid, it replaces the segment with ??. Such a replacement occurs, for example, when an instruction attempts to write into ROM. In that case, Sourcer uses an appropriate segment if it can determine one, otherwise, it lists the segment as ??.

Prefixes

Prefixes are associated with instructions. Sourcer will not allow invalid prefixes. Nor will it allow multiple prefixes of the same type in a single instruction. It addition, it allows 80386 and 80486 prefixes only when you select the 80386 or 80486 processor. For clarity, Sourcer uses special symbols to denote prefixes as shown in the following table.

Prefix Type	Symbol
Segment	:
32-bit operand and address size override	|
Repeat	/
Floating point wait	-
Lock	>

The following example shows several prefix instructions.

Example 3-3: Prefixes

```
Line 1  6159:0123  26: 89  0018   mov   cx,cs:data_35        ; (6159:0618=0)
     2  6159:0127  8D 7F  44     les   di,ds:data_2e[bx]    ; (0000:0144=23Dh) load 32 bit ptr
     3  6159:012A  FC            cld                      ; Clear direction flag
     4  6159:012B  F3/ A4        rep   movsb              ; Rep while cx>0 Mov [si] to es:[di]
     5  6159:012D  93- D9 EB     fldpi                    ; Push Pi onto stack
```

The first line contains the CS segment override prefix (26h). On the second line, the segment prefix is included in the opcode area so the assembler will handle the equate data item as a memory reference and not generate a segment override instruction.

The fourth line shows a repeat prefix in the hex instruction area (F3h). The fifth line shows the math co-processor instruction's wait prefix (93h).

Cross-references

When the cross reference option is on and the number of passes is five or more, Sourcer inserts cross-reference information into the listing file during the final pass. Cross-references appear at the point where the cross-referenced item is defined making it easy to see which routines and functions use the referenced item.

Sourcer cross-references data, locations, and subroutines. Using analysis options, you can turn on and off each type of cross-reference. For instructions, see page 27. The following listing shows sample cross-references.

Example 3-4: Cross-references

```
                        ;*****************************************
                        ;                SUBROUTINE
                        ;
                        ;       Called from:  62AA:0071, 04AF, 0604, 0A65, 1C7E
                        ;                     6093:003B, 6093:209A
                        ;*****************************************

                        sub_23  proc    far
62AA:0000  52                   push    dx
62AA:0001  BA 03C8              mov     dx,3C8h
62AA:0004  B0 03                mov     al,3
62AA:0006                       loc_147:                        ; xref 62AA:0013, 006B
62AA:0006  40                   inc     ax
62AA:0007  EE                   out     dx,al                   ; port 3C8h, VGA pel address
```

Cross-references to an item in the same segment always appear before cross-references to an item outside the segment. In the example, five locations within the current segment and two locations outside the current segment reference sub_23. Two items within the current segment reference the location loc_147.

On rare occasions, Sourcer may encounter a cross-reference during the final pass, after writing the referenced item to the listing file. In such cases, the item will not have a cross-reference even though it is referenced.

A list of external references appears at the end of the listing file. It includes items such as the program entry point (which has the label start), interrupt entry points created within the program (typically RAM resident programs), and entry points defined by the optional Pre-Processor programs (such as the BIOS Pre-Processor).

Use the definition file created by Sourcer, *filename*.SDF, to look up items in the listing. For a complete description of the definition file format, see Chapter 4, "Using Definition Files."

Indexed Jumps and Calls

To insure a high degree of output accuracy, Sourcer makes as few assumptions as possible. Clarifying indexed call and jump instructions, however, does require some assumptions.

When Sourcer encounters an indexed call or jump, it uses the simulation data to determine the pointer value. The length and bounds of the index jump table are not explicitly defined, so Sourcer uses a separate table analyzer to determine them. Sourcer is usually able to accurately delineate an indexed jump table within 3 passes.

The following sample listing contains an indexed call.

Example 3-5: Indexed Call

```
6159:0073  F6 E3          mul     bl                  ; ax = reg * al
6159:0075  05 0080        add     ax,80h
6159:0078  8B F0          mov     si,ax
6159:007A  FF 1C          call    dword ptr [si]      ;*(616F:0080=8Ch) 3 entries
6159:007C  B4 4C          mov     ah,4Ch              ; 'L'
6159:007E  CD 21          int     21h                 ; DOS Services  ah=function 4Ch,
                                                      ; terminate with al=return code
6159:0080  008C 6159  data_1  dw      offset sub_1, seg sub_1
6159:0084  008F 6159  data_2  dw      offset sub_2, seg sub_2
6159:0088  0092 6159  data_3  dw      offset sub_3, seg sub_3

                      ;*****************************************
                      ;                SUBROUTINE
                      ;*****************************************

                      sub_1   proc    far
6159:008C  B0 04              mov     al,4
6159:008E  C3                 ret
                      sub_1   endp
```

Chapter 3: Customizing Sourcer Operation

This chapter explains how to use Sourcer processing and analysis options to customize its operation. It describes how to

- Set the code style.
- Set the number of passes.
- Set the microprocessor filter.
- Set analysis options.
- Use command-line options.
- Set output options.

Setting the Code Style

Sourcer automatically sets the code style when you load a file. To change the code style, press **C** from the main menu until you see the code style you want to use. The following table describes code style options.

Code style	Description
Fragment	Use when analyzing only a portion of a program. You cannot assemble the generated code.
COM	Use for a single-segment program.
EXE	Use for multiple-segment programs, BIOS ROMs, and COM files with an EXE header.
Zero start	Use for files that start at offset 0.

Device driver	Use for files that have a device driver header with strategy and command entry points.
Overlay	Use for an overlay, starting at offset 0, to another program.

Fragment Code Style

When you choose the fragment code style, Sourcer interprets only explicitly referenced items as data and assumes all other items to be code. Use this option for files that contain program fragments or self-modifying code. If you use another code style for such a file, Sourcer interprets items not referenced within the program as data.

COM Code Style

Use COM code style for single-segment programs less than 64 Kb in length. DOS allows programs defined as COM to have only one segment. However, with clever programming practice, a COM program can access other data segments. Sourcer resolves these data accesses correctly and generates a single segment program meeting DOS COM requirements.

EXE Code Style

DOS allows EXE-type programs to have one or more segments. However, most EXE programs use less than 4 segments. Sourcer supports programs that have up to 250 segments. Also use EXE code style when analyzing multiple segments in RAM and/or ROM. (You must use a definition file to analyze RAM and ROM segments. For instructions, see Chapter 4 "Using Definition Files.")

Zero Start Code Style

Use the zero start code style for COM-style programs that start with zero offset from the beginning of the file. Only a few internal DOS programs use this format; they typically have a .SYS or .BIN file extension.

Device Driver Code Style

When the code style is device driver, Sourcer determines the special entry points for the strategy and command sections and creates labels that reference these sections. When you load a file with the extension .COM, .SYS, .DRV, or .BIN, Sourcer checks for the reserved device driver header. If found, Sourcer automatically sets the code style to device driver.

Overlay Code Style

If a file extension is an overlay type (for example, .OVR, .OVL, .OV1, and such), Sourcer sets the code style to overlay. The overlay style operates in the same way as the Zero Start code style. Overlays that contain code typically start at an offset of zero, but the program that calls the overlay determines its exact usage. The overlay can have entry points that only the parent program can determine as it runs, or it can contain only data for the parent program.

Setting the Number of Passes

You can instruct Sourcer to analyze your code in two to nine passes. Each pass improves the accuracy of the generated code. Five passes are usually sufficient. However, programs with large, indirectly addressed data areas may benefit from additional passes. Sourcer may interpret such data areas as code in earlier passes. By analyzing the code in each successive pass, it can determine invalid code structures and mark sections for conversion to data during subsequent passes.

When the code style is set to any style other than fragment, Sourcer sets the number of passes to five. Program fragments usually require fewer passes; when the code style is set to fragment, Sourcer sets the number of passes to two. To set the number of passes, press **P** from the Main Menu until Sourcer displays the number you want.

Setting the Microprocessor Filter

The microprocessor filter setting determines which instruction set Sourcer uses. You achieve the best results by setting the microprocessor filter to match the microprocessor for which the

program you are analyzing was written. To maintain compatibility across computers in the PC family, most programs are written for the 8088. If you are not sure for which microprocessor the program was written, set the microprocessor filter to 8088/8086. 8088/8086 is the default microprocessor filter. Any computer can use any filter with one exception: Some 8088-based computers may not be able to use V20/V30 instruction set filter.

> **Important:** Using the V20/V30 instruction set filter on some 8088-equipped machines may cause the system to halt. Some non-Intel 8088 parts generate a halt instruction instead of ignoring invalid instructions.

To change the instruction set, press **U** from the Main Menu until the menu displays the instruction set you want. To use math instructions, press **M** until the Main Menu displays Math on. The following table describes each microprocessor filter.

Filter:	Description:
8088/8086	Base level instruction set.
V20/V30	Similar to the 8088/8086 set. Adds some unique instructions and some 80186 instructions. Unique V20/V30 instructions are not supported by all assemblers. Few programs, except those available from NEC, use these instructions.
80188/80186	Uses 80186 instruction set.
286 real	Uses 80286 instruction set in real mode.
286 protected	Uses the 80286 instruction set in protected mode (uses the entire instruction set).
386 real	Uses 80386 instruction set in real mode.
386 protected	Uses the 80386 instruction set in protected mode (uses the entire instruction set).
486 real	Uses 80486 instruction set in real mode (includes math microprocessor instructions).
486 protected	Uses the 80486 instruction set in protected mode (uses the entire instruction set).

The 8088/8086 instruction set is a subset of the other instruction sets. Sourcer supports real and protected mode 80286, 80386, and 80486 instructions. Most DOS programs operate in real

mode; real mode uses a subset of the entire instruction set. Protected mode uses the entire instruction set. For more information about the 80386 and 80486 instruction sets, see Appendix C, "Using 80386 and 80486 Instruction Sets."

Setting Analysis Options

Analysis options give you additional control over how Sourcer processes a file. Because the default analysis options produce the best results in most cases, you may only need to set analysis options for special needs.

You can use analysis options to:

- Change the way Sourcer separates data from code.
- Select the assembler type (The default is Microsoft Macro Assembler (MASM) 5.0/5.1, Turbo Assembler (TASM), and OPTASM).
- Prevent graphics characters in output.
- Turn off sections of the analysis.
- Make selective cross references.

To set analysis options, press **A** from the Main Menu. The Analysis Options Menu appears. In the Analysis Options Menu, a letter (A through Z) represents each analysis option. When an option is on, the letter is uppercase and highlighted. When an option is off, the letter is lowercase. To turn an option on or off, press the corresponding keyboard key. For example, to turn option E on or off, press **E**. This section describes each option in detail. For a summary of each option, see page 83.

Options A through C: Inaccessible Code Areas

Normally, Sourcer interprets inaccessible code areas as data. By turning on options A through C, you can instruct Sourcer to interpret the bytes following specific instructions as code. Setting the code style to fragment is similar to turning on options A, B, and C. For more information on code styles, see page 13.

Option A

Turn on this option to remain in code mode instead of switching to data mode after a jump instruction. The following two examples show the results of turning on and off option A. The listing in Example 3-1 was produced with option A off; after it encounters a jump instruction that is not followed by an entry point (or label), Sourcer switches to data mode. The listing in Example 3-2 was produced with option A turned on; after the jump instruction, Sourcer remains in code mode--even when it finds no way to access the code area. (For more information about inaccessible code areas, see page 73).

Example 3-1. Option A Off (Default)

```
6D8C:0032  3C 6E              cmp   al,6Eh                    ; 'n'
6D8C:0034  74 12              je    loc_3                     ; Jump if equal
6D8C:0036  9A 6D9D:0000       call  far sub_1                 ; (6D9D:0000)
6D8C:003B  EB EB              jmp   loc_28
6D8C:003D  B0 01 EB 09 32 C0  db    0B0h, 1, 0EBh, 9, 32h, 0C0h
```

Example 3-2: Option A On

```
6D8C:0032  3C 6E              cmp   al,6Eh                    ; 'n'
6D8C:0034  74 12              je    loc_3                     ; Jump if equal
6D8C:0036  9A 6D9D:0000       call  far sub_1                 ; (6D9D:0000)
6D8C:003B  EB EB              jmp   loc_28
6D8C:003D  B0 01              mov   al,1
6D8C:003F  EB 09              jmp   loc_4
6D8C:0041  32 C0              xor   al,al                     ; Zero register
```

Option B

Turn on this option to remain in code mode instead of switching to data mode after a return instruction (RET).

Option C

Turn on this option to remain in code mode instead of switching to data mode after an interrupt return instruction (IRET).

Options D and E: DS and ES Values

Normally, Sourcer sets the DS and ES simulator values to default values similar to the DOS defaults before disassembling a program. Options D and E change the default DS and ES values.

Option D

When Sourcer loads an EXE file, it sets the DS and ES simulator registers to the code segment less 10h. DOS does the same when it loads a file in order to run it. When you turn on option D, Sourcer sets the DS and ES defaults to the code segment. The following two examples show the results of turning on and off option D. Both show the beginning of the code. In Example 3-3, option D is turned off and Sourcer sets the default DS segment to cs-10h. In Example 3-4, option D is turned on and Sourcer sets the default DS segment to the same value as the code segment; in this example, turning on option D links the data reference to the wrong point.

Example 3-3: Option D Off (Default)

```
6D8C:0000                  start:
6D8C:0000  8B  16 0038          mov     dx,data_4       ; (6D7C:0038=1F3Eh)
```

Example 3-4: Option D On

```
6D8C:0000                  start:
6D8C:0000  8B  16 0038          mov     dx,data_21      ; (6D8C:0038=6134h)
```

Important: You must turn on option D before loading an EXE file. Otherwise, it has no effect.

Option E

Turning on option E instructs the code simulator to ignore changes to the DS and ES simulation registers. You can use a definition file to set the register values. For a complete description of a definition file, see Chapter 4, "Using Definition Files." The following two examples show the results of turning on and off option E. Example 3-5 shows a listing produced with

option E turned off. Normal simulator operation changes the value of DS to 40h. Example 3-6 shows a listing produced with option E turned on. In this case, the DS value is unchanged. As a result, the data reference is incorrect. You might turn on option E when you are analyzing self-modifying code and want to force the DS and ES segment registers to specific values given in a definition file.

Example 3-5: Option E Off (Default)

```
6D9D:0000  8B 0E 0040      mov   cx,40h
6D9D:0004  0E              push  cx
6D9D:0005  1F              pop   ds
6D9D:0006  88 16 001E      mov   dx,keybd_queue     ; (0040:001E=0)
```

Example 3-6: Option E On

```
6D9D:0000  8B 0E 0040      mov   cx,40h
6D9D:0004  0E              push  cx
6D9D:0005  1F              pop   ds
6D9D:0006  88 16 001E      mov   dx,data_13     ; (6D9D:001E=144Fh)
```

Option F: Terminated Subroutines

When option F is turned off, Sourcer identifies subroutines that do not return to the point from which they were called. After the call instruction, Sourcer switches from code to data mode. Turn on option F to prevent Sourcer from detecting such terminated subroutines. The following two examples show the results of turning on and off option F. Example 3-7 shows a listing produced with option F off. Because it encountered an invalid instruction after the subroutine call, Sourcer marks the subroutine as terminated. Example 3-8 shows the listing produced when option F is on; the subroutine is not marked as terminated.

Example 3-7: Option F Off (Default)

```
6A65:00F6  8B D1           mov   dx,cx
6A65:00F8  E8 0017         call  sub_3          ; Sub does not return here
6A65:00FB  B0 07 32 DB 64 65  db   0B0h, 7, 32h, 0DBh, 64h, 65h
```

Example 3-8: Option F On

```
6A65:00F6  8B D1          mov   dx,cx
6A65:00F8  E8 0017        call  sub_3
6A65:00FB  B0 07          mov   al,7
6A65:00FD  32 DB          xor   bl,bl
6A65:00FF  64 65          db    64h, 65h
```

Options G and H: Indexed Data Handling

In some programs, indexed data (for example, [si+9Ah]) can create an incorrect data reference (in the example, at ds:9Ah). Such incorrect references are more likely to occur in EXE files than in COM files. By turning on option G or option H, you force these references to equates.

Option G

Forces indexed data references 0h to 200h to equates. Turn on option G instead of option H if indexed data items are appearing as data from 0h to 200h.

Option H

Forces indexed data references 0h to 100h to equates. By default, option H is turned on.

The following examples show the results of turning on and off option G and H. In Example 3-9, option G is off and option H is on; the listing references data_19 at offset 183h. In Example 3-10, option G is on and option H is off; a simple hex offset, [bx+183h], replaces the data reference. In Example 3-11, both options are off, and all offset values are treated as data references.

Example 3-9: Option G Off and Option H On (Default)

```
6ED8:0077  8A 8C 0004  mov  cl,byte ptr [si+4]       ; (6EDB:0004=0)
6ED8:007F  8A Af 0183  mov  ch,byte ptr data_19[bx]  ; (6ED8:0183=0)
```

Example 3-10: Option G On and Option H Off

```
6ED8:0077  8A 8C 0004  mov  cl,byte ptr [si+4]       ; (6EDB:0004=0)
6ED8:007F  8A Af 0183  mov  ch,byte ptr [bx+183h]    ; (6ED8:0183=0)
```

Example 3-11: Option G and Option H Off

```
6ED8:0077  8A 8C 0004  mov  cl,byte  ptr  data_2[si]    ; (6EDB:0004=0)
6ED8:007F  8A Af 0183  mov  ch,byte  ptr  data_19[bx]   ; (6ED8:0183=0)
```

Option I: Graphics Characters

Turning on option I prevents Sourcer from using graphics characters in the output file. Turn on this option when your printer does not support the graphics characters. The following two examples show the result of turning on and off option I.

Example 3-12: Option I Off (Default)

```
;
;                    SUBROUTINE
;
beeper      proc    far
```

Example 3-13: Option I On

```
;=============================================================
;                    SUBROUTINE
;=============================================================
beeper      proc    far
```

Option J: Interrupt Vector Entry Points

Turn on option J to prevent Sourcer from detecting interrupt vector entry points into the program. Example 3-14 shows a listing produced with option J off. The simulator detects the loading of an interrupt vector and generates a procedure label for the interrupt.

Example 3-14: Option J Off (Default)

```
6F84:000A                int_36h_entry   proc    far
6F84:000A 50                             push    ax
6F84:000B B0 02                          mov     al,2
                                         .
                                         .
                                         .
6F84:0141 BA 000A                        mov     dx,0Ah
6F84:0144 B8 2536                        mov     ax,2536h
6F84:0147 CD 21                          int     21h          ; DOS Services ah=function 25h
                                                              ; set intrpt vector al to ds:dx
6F84:0149 B8 0040                        mov     ax,40h
```

Option K: Absolute Location References

Turn on option K to prevent Sourcer from writing absolute location reference comments for jumps and calls. The following two examples show the results of turning on and off option K. In Example 3-15, option K is off and the listing contains comments to show the destination of calls and jumps. In Example 3-16, option K is on and the listing does not contain such comments.

Example 3-15: Option K Off (Default)

```
6D8C:0036  9A  6D9D:0000   call  far ptr beeper          ; (6D9D:0000)
6D8C:003B  EB  EB          jmp   short begin_program    ; (0028)
```

Example 3-16: Option K On

```
6D8C:0036  9A  6D9D:0000   call  far ptr beeper
6D8C:003B  EB  EB          jmp   short begin_program
```

Option L: RETF and RETN Instructions

Turn on this option to prevent Sourcer from using the instruction mnemonics RETF/RETN for return far and near. Turn on this option if you are using MASM version 4.0 or another assembler that does not support RETF and RETN. The following two examples show the results of turning on and off option L.

Example 3-17: Option L Off (Default)

```
6D9D:002B  CB                retf                : return far
```

Example 3-18: Option L On

```
6D9D:002B  CB                ret                 : return far
```

Option M: MASM 4.0 Compatibility

Turn on this option if you intend to reassemble the generated code with MASM 4.0. For certain instructions, code originally assembled with MASM 4.0 (or an earlier assembler) uses two bytes where MASM 5.0 (and later assemblers) use only one byte. When option M is off, Sourcer checks for these old-style instructions and preserves the number of bytes in the original file by inserting NOP instructions. When option M is on, Sourcer does not fix the old-style instructions. The following two examples show the result of turning on and off option M. In Example 3-19, option M is off and Sourcer inserts a NOP instruction that was not in the original code. Without the NOP instruction, all newer assemblers will produce an instruction that uses one byte less than the original program. The NOP instruction insures that the reassembled program has the same number of bytes as the original program. In Example 3-20, option M is on, and Sourcer does not insert a NOP instruction.

Example 3-19: Option M Off (Default)

```
F000:8830  81 0E 0415 0008   or      chan_io_size_,8     ; (0000:0415=80h)
                             nop                         ;*Fixup for MASM (M)
```

Example 3-20: Option M On

```
F000:8830  81 0E 0415 0008   or      chan_io_size_,8     ; (0000:0415=80h)
```

Option N: ASSUME Statements

Turn on this option to prevent Sourcer from placing extra ASSUME statements in code to insure reassembly. By default,

this option is turned off. The following two examples show the results of turning on and off option N. In Example 3-21, option N is off (normal operation), and Sourcer inserts the ASSUME statements necessary for reassembly. In Example 3-22, option N is on, and Sourcer does not insert ASSUME statements. Without the ASSUME statements, the generated code may not reassemble properly.

Example 3-21: Option N Off (Default)

```
F000:814E  33 C0          xor     ax,ax                   ; Zero register
F000:8150  8E D8          mov     ds,ax
                          assume  ds:seg_a
F000:8152  FF 2E 0467     jmp     dword ptr @gen_io_ptr_  ; (0000:0467=3)
```

Example 3-22: Option N On

```
F000:814E  33 C0          xor     ax,ax                   ; Zero register
F000:8150  8E D8          mov     ds,ax
F000:8152  FF 2E 0467     jmp     dword ptr @gen_io_ptr_  ; (0000:0467=3)
```

Option O: Indexed Jumps and Calls

Turn on this option to prevent Sourcer from analyzing indexed jump and indexed call instructions. By default, this option is turned off. The following two examples show the result of turning on and off option O. In Example 3-23, option O is off and Sourcer performs its normal analysis of indexed jumps and calls. In Example 3-24, option O is on and Sourcer does not make any indexed jump or call linkages nor does it create a table of offsets.

Example 3-23: Option O Off (Default)

```
6A65:613F  FF A4 6147          jmp  word ptr data_5[si]  ;*(6A65:6147=614Fh)  3 entries
6A65:6143                loc_3:
6A65:6143  B4 4C               mov  ah,4Ch
6A65:6145  CD 21               int  21h                  ; DOS Services  ah=function 4Ch
                                                         ;  terminate with al=return code
6A65:6147  61 4F         data_5  dw   offset loc_4       ; Data table (indexed access)
6A65:6149  61 57         data_6  dw   offset loc_5
6A65:614B  61 64         data_7  dw   offset loc_6
```

Example 3-24: Option O On

```
6A65:613F  FF A4 6147          jmp  word ptr data_5[si]     ;*(6A65:6147=614Fh)
6A65:6143              loc_3:
6A65:6143  B4 4C               mov  ah,4Ch
6A65:6145  CD 21               int  21h                     ; DOS Services  ah=function 4Ch
                                                            ;  terminate with al=return code
6A65:6147  61 4F       data_5  dw   614Fh                   ; Data table (indexed access)
6A65:6149  57 61 4B 61         db   57h, 61h, 4Bh, 61h
```

Option P: Index Register

When option P is turned off (the default), Sourcer ignores the index register value when it is less than 180h. Turn on this option to use any value in the simulation index register (BX, BP, SI, or DI) for indexed jumps and calls. Only in very rare cases do you need to turn on option P. The following two examples show the result of turning on and off option P. In Example 3-25, option P is off and Sourcer correctly identifies the table at data_5. In Example 3-26, option P is on and Sourcer builds the table at data_5+si. The resulting code incorrectly references the table at 613F+23h (6162h).

Example 3-25: Option P Off (Default)

```
6S65:613C  BE 0023             mov  si,23h
6A65:613F  FF A4 6147          jmp  word ptr data_5[si]     ;*(6A65:6147=614Fh)  3 entries
6A65:6143              loc_3:
6A65:6143  B4 4C               mov  ah,4Ch
6A65:6145  CD 21               int  21h                     ; DOS Services  ah=function 4Ch
                                                            ;  terminate with al=return code
6A65:6147  614F        data_5  dw   offset loc_4            ; Data table (indexed access)
6A65:6149  6157        data_6  dw   offset loc_5
6A65:614B  6164        data_7  dw   offset loc_6
```

Example 3-26: Option P On

```
6S65:613C  BE 0023             mov  si,23h
6A65:613F  FF A4 6147          jmp  word ptr data_5[si]     ;*(6A65:6147=614Fh)  1 entry
6A65:6143              loc_3:
6A65:6143  B4 4C               mov  ah,4Ch
6A65:6145  CD 21               int  21h                     ; DOS Services  ah=function 4Ch
                                                            ;  terminate with al=return code
6A65:6147  614F        data_5  dw   614Fh                   ; Data table (indexed access)
6A65:6149  57 61 4B 61         db   57h, 61h, 4Bh, 61h
```

Options Q through T: Cross-references

Options Q through T set cross-reference options. Turning on Xref in the Main Menu turns on options Q, R, and S.

Option Q

Turn on this option to cross-reference all data references.

Option R

Turn on this option to cross-reference all location references. The following two examples show the result of turning on and off option R.

Example 3-27: Option R Off

```
6D8C:003D             loc_2:
6D8C:003D  B0 01                mov     al,1
6D8C:003F  EB 02                jmp     short exit      ; (0043)
6D8C:0041             loc_3:
6D8C:0041  32 C0                xor     al,al           ; Zero register
```

Example 3-28: Option R On

```
6D8C:003D             loc_2:                                  ; xref 6D8C:0030
6D8C:003D  B0 01                mov     al,1
6D8C:003F  EB 02                jmp     short exit      ; (0043)
6D8C:0041             loc_3:                                  ; xref 6D8C:0034
6D8C:0041  32 C0                xor     al,al           ; Zero register
```

Option S

Turn on this option to cross-reference all subroutine references.

Option T

Turn on this option to prevent Sourcer from cross-referencing locations that are less than 128 bytes away from each other.

Option U: Indexed Data Values

Turn on option U to prevent Sourcer from interpreting indexed data values -64h through +64h as offsets. When option U is on, Sourcer creates data references. In most cases, numbers from -64h to 64h are adjustments to an index and creating a data item in low memory may obstruct code in the same area. The following two examples show the result of turning on and off option U.

Example 3-29: Option U Off (Default)

```
F000:8731    C6  44 11 F2          mov    byte ptr [si+11h],0F2h
```

Example 3-30: Option U On

```
F000:8731    C6  44 11 F2          mov    data_2[si],0F2h
```

Option V: Near Jump Fixup

Turn on option V to prevent Sourcer from fixing near jumps. Some older assemblers use a near jump (3 bytes) where they could have used a short jump (2 bytes). When option V is turned off, Sourcer inserts a NOP after the jump. The resultant code, if reassembled on a newer assembler, maintains the same number of instruction bytes as the original code.

Option W: Warning Comments

Turn on option W to prevent Sourcer from adding warning comments to the output file. Sourcer inserts a warning comment when it detects an indexed jump or call, identifies a register jump or call, or inserts a fixup instruction (see option M on page 24. Warning comments begin with a semicolon and an asterisk (;*). If you plan to reassemble the code, you should check each line marked with a warning comment. For more information, see Chapter 7, "Reassembling Generated Source Code." The following two examples show the result of turning on and off option W.

Example 3-31: Option W Off (Default)

```
6A65:613F   FF A4 6147          jmp   word ptr data_5[si]      ;*(6A65:6147=614Fh)  3 entries
```

Example 3-32: Option W On

```
6A65:613F   FF A4 6147          jmp   word ptr data_5[si]      ; (6A65:6147=614Fh)  3 entries
```

Option X: Final Pass References

When option X is turned on, Sourcer creates references during its last pass as it writes the file to disk. References created during the last pass usually become undefined when the code is reassembled. By default, option X is turned off. The following two examples show the result of turning on and off option X. In Example 3-33, the last pass did not define loc_27. Sourcer converts loc_27 into equivalent data bytes so that the code can be reassembled and inserts a warning comment. When you encounter such a comment, try disassembling the code again using more passes or check whether the area is actually code. Also, some compilers produce code with invalid jumps that never execute.

Example 3-33. Option X Off (Default)

```
F000:8140   E9 2101     ;*      jmp   loc_27          ;*
F000:8140   E9 01 21            db    0E9h, 1, 21h
F000:8144   33 ED               xor   bp,bp           ; Zero register
```

Example 3-34. Option X On

```
F000:8140   E9 2101             jmp   loc_27
F000:8144   33 ED               xor   bp,bp           ; Zero register
```

Command Line Options

You can set additional Sourcer features when you start Sourcer from the DOS command line. Use the following format to set a command line option:

sr *option option* ...

where *option* is one of the command line options described in the following table.

Option	Description
-d	Do not look for a system defaults definition file. For more information, see Chapter 5, "Setting System Defaults."
filename	Specifies the file to load during startup. If you do not provide a filename extension, Sourcer tries to locate the file in the current directory by appending the following filename extensions, in order: .DEF (definition file), .EXE, and .COM. If it cannot find the file, it starts without loading a file.
-n	Process display information through the DOS interrupt instead of writing directly to the screen. Use this option when your system does not support one of the IBM display standards.
-s*xxxx*	Changes the video segment to *xxxx* where *xxxx* is a 4-digit hex number. Use this option when your system does not support one of the IBM display standards and the video buffer is set to a value other than B000h or B800h. If you use the -N option, Sourcer ignores this option.
-v	Prevents Sourcer from using VGA/EGA-specific features and from changing the color palette. Use this option if your system has an EGA or VGA card and screen problems occur while using Sourcer.

For example, to run Sourcer with the file TESTYN.EXE, ignoring the system defaults, on a system that does not have an IBM-standard display, enter:

```
sr testyn.exe -d -n
```

You can preset the command line options by setting the DOS environment variable VCOM. For example, to prevent Sourcer from using EGA/VGA features enter:

```
set vcom = -v
```

To permanently set this environment variable, include a command such as the preceding one in your AUTOEXEC.BAT file.

Setting Output Options

To control the format of the output file, you can use Main Menu output formatting commands, analysis options, and definition file options. In addition, you can write an output file directly to a printer. This section describes how to set output options.

Output Formatting Commands

Output formatting commands appear in the upper portion of the Main Menu along with four sample output lines that show their effect. The available output options change when you change the output file format. For a description of each command, see page 81.

Analysis Options for Output

Analysis option I lets you turn on and off graphics characters in the output file. Turn off this option to use graphics characters, and turn it on to prevent Sourcer from using graphics characters. For a complete description of on analysis option I, see page 22.

Definition File Output Options

The Vertical lines option in a definition file gives you additional control over output format. It lets you change the number of lines per page and the page width in output files. For a complete description of the Vertical lines definition file option, see page 41.

Writing the Output File to a Printer

To write the Sourcer output file directly to your printer instead of to disk, load the file to process and then change the output filename to PRN.

Chapter 4: Using Definition Files

Definition files let you preset Sourcer commands and options, define multiple memory segments, add your own comments to generated code, and control data types and options for the program you are analyzing. A definition file is a text file with a special format. This chapter explains how to create and use definition files. It describes how to:

- Use the definition file format.
- Use definition files generated by Sourcer.
- Use the template definition file: SAMPLE1.DEF.
- Load a definition file.
- Specify control information in section one.
- Specify range definitions in section two.
- Specify reference definitions and comments in section three.

Definition File Structure

Each definition file contains a control information section, a range definition section, and a reference definition section. The control information section (section one) presets Main Menu commands and analysis options. The range definition section (section two) defines segment ranges. The reference definition section (section three) defines and labels code items and assigns comments to them.

In each section, data lines begin with a non-space character in column one and comment lines begin with a space in column one. At least one comment line separates sections, and the reference definition section can have comments between entries.

Sourcer ignores all comment lines when it processes the definition file. The following example shows an abbreviated version of the file TESTYN.DEF.

Example 4-1: TESTYN.DEF

```
                SOURCER DEFINITION FILE - TESTYN

        Section 1  CONTROL INFORMATION

uP                      = 8088
Input filename          = testyn.exe
Header                  = TESTYN with definition file
Xref                    = ON

        Section 2  RANGE DEFINITION

  begin    end      default        seg    seg
  seg:off  off      ds    es       type   size
  -------  ----     ----  ----     ----   ----
none                                        ; comments in this area ignored

        Section 3  REFERENCE DEFINITIONS


  ========== Subroutines ==========
  seg:off      type & options   label            comments
  -------      --------------   --------------   --------------
seg_c:0000     sub, far, c 02   beeper           ; beep subroutine (3rd segment)


  ========== Locations ==========
  seg:off      type & options   label            comments
  -------      --------------   --------------   --------------
seg_a:0028     loc, c 04        begin_program
seg_a:0043     loc              exit


  ========== Data Items ==========
  seg:off      type & options   label            comments
  -------      --------------   --------------   --------------
seg_c:002C     da, r 0C         yes_no_message
seg_c:0038     dw, c 6          speaker_port
seg_c:003A     dw, c 7          on_off_time
seg_c:003C     dw, c 8          beep_cycles
```

Using Definition Files Generated by Sourcer

Each time you run Sourcer, it creates a definition file containing all command and option settings, segment range definitions, and all locations, subroutines and other items that it identified. A Sourcer-generated definition file uses the standard definition file format. It has the output filename you specified and the extension .SDF.

You may want to use a Sourcer-generated definition file as the basis for your own file. For example, you might want to edit it to add your own labels and comments to the code. To use a Sourcer-generated definition file as input to Sourcer, you must change its name so it has a filename extension of .DEF.

For example, you might start Sourcer, load the file MYPROG.EXE, specify a listing output file with the name MYPROG.LST, and then generate the listing. In addition to the listing file, Sourcer creates the definition file MYPROG.SDF. To create an input definition file from MYPROG.SDF, copy it to a file with the .DEF filename extension. For example:

```
copy  myprog.sdf  myprog.def
```

Edit MYPROG.DEF to add your own labels and comments to the definition file.

The optional Pre-Processor programs also create definition files to use as input to Sourcer.

Using the Template Definition Files

The SAMPLE1.DEF and SAMPLE2.DEF files (on the Sourcer distribution disk) are templates for your definition files. SAMPLE1.DEF contains comments that divide sections and describe the format for each section. SAMPLE2.DEF is similar to SAMPLE1.DEF except that it contains fewer descriptive comments. To create your own definition files, copy one of these template files and edit the copy.

Loading a Definition File

To load a definition file, include the definition file name in the command you use to start Sourcer. For example:

```
sr testyn.def
```

You do not need to supply the .DEF filename extension. However, all definition files must have a .DEF extension.

If the definition file does not specify an input filename and you want to use one, press I from the Main Menu and enter a filename. When you load the input file, Sourcer asks which segment ranges to keep. Press 1 to only keep only the segments for the new file, press 2 to keep both the segments for the new file and the segments defined in the definition file, or press 3 to keep only the definition file segments. No matter which option you choose, Sourcer uses the labels from the definition file. It also creates references as needed when they are not supplied in the definition file.

Specifying Control Information in Section One

The first section of a definition file contains control information that presets commands and options.

Each line in this section assigns a value to a command or option using the format:

control_option = value

where ***control_option*** is one of the predefined options for this section and ***value*** is the value assigned to it. (Not all options have values.) You can abbreviate a control option by using its shortest unique form; Sourcer ignores other characters until it encounters the equal sign.

If a definition file contains no control information, the control information section must contain the following line:

```
none
```

Control Options

This section describes (in alphabetical order) each of the predefined control options and its possible values.

Analysis options =
Sets analysis options. List all analysis options you want to preset after the equal sign. Use uppercase to turn on an option and lowercase to turn off an option. For a detailed description of the analysis options, see Chapter 3, "Customizing Sourcer Operation." For example, to turn on options A, B, C, and G and

turn off options N, X, and Y, use the following line in your definition file:

```
analysis options = A B C G n x y
```

Begin segment =
Specifies the memory location at which to load an input file. Use this option to make sure that the input file loads at the same location each time you process it. Its value must be a 4-digit hex number and must be equal to or greater than the location that Sourcer would normally use (if you loaded the file without using this option). To determine the beginning segment for EXE files, subtract 10h from the location at which you want the code to begin. For example:

```
Begin segment = 97F0
```

Code style =
Specifies the code style. Its value can be one of the following styles: COM, EXE, fragment, overlay, zero start, or device driver. For a complete description of code style options, see page 13. For example, to set the code style to EXE, use the following line in your definition file:

```
code style = exe
```

If you specify an input filename in your definition file, it must precede the code style.

Drive =
Sets the disk drive for output. Its value can be any disk drive letter from A to Z. For example, to write the output files to drive C, use the following line in your definition file:

```
drive = c
```

File format =
Sets the output file format. Set it to ASM for source code or LST for a listing. (LST is the default.) For example, to generate source code, use the following line in your definition file:

```
file format = asm
```

Go

Instructs Sourcer to begin disassembling the input file immediately after loading the definition file. This option lets you run Sourcer in a batch mode. If the output file exists, Sourcer will still ask whether it is ok to overwrite the file.

Header =

Specifies a header text string (up to 32 characters long) for the listing. If used, this option must appear after the Input filename option. For example:

```
Header = Test run number 2
```

Input filename =

Names the input file. You can specify a drive and/or a complete pathname. The process of loading an input file automatically sets other options. Thus, the following options must appear after Input filename option in the definition file:

- Code style
- Header
- Output filename
- Passes
- Xref

For example, to load the file named TESTYN.EXE, use the following line in your definition file:

```
input filename = testyn
```

Keep segments =

In special cases, you can force Sourcer to use selected segment ranges. The following table shows possible values.

Value:	Description:
FILE	Use segments from the input file only.
DEF	Use segments from the range definition section of the definition file only.
BOTH	Use segments from both the input file and the definition file. This value is the default.

In most cases, leave Keep segments set to its default of BOTH. Sourcer will automatically resolve differences between the segments specified when a file is loaded and any entries in the range definition section of the definition file. When you use the Keep segment option, it must precede the Input filename option.

Regardless of the Keep segments value, Sourcer sets the seg_a to seg_z type segment variables when loading a file. You can use these variables throughout the definition file. For more information about segment variables, see page 43.

For example, to use segments from the range definitions section of the definition file only, use the following line:

```
keep segments = def
```

Label type =
Specifies the label type. The following table shows possible values and an example of their output:

Value:	Example:
decimal	data_123
zero fill	data_0123
segment & offset	d_7B23_08AB
letter seg & offset	data_c_08AB

The default label type is decimal.

For example:

```
Label type = letter seg & offset
```

Math on
Instructs Sourcer to use the math coprocessor instruction set (8087, 80287, or 80387). The 80486 microprocessor has math capabilities built in. When you set the processor type to 80486, Sourcer automatically supports 80486 math instructions.

Output filename =
Specifies the output filename. You can specify a drive and/or a complete pathname. You do not need to supply a filename extension; Sourcer appends either .LST (for a listing file) or .ASM (for a source code file) to the filename you specify. If you

do not specify an output file name in the definition file, Sourcer uses the definition file name and appends to it the extension .LST or .ASM. For example:

```
output filename = temp3
```

Passes =
Sets the number of passes. Its value can be any number from 2 to 9. For example:

```
passes = 4
```

Remarks =
Sets the type of automatic comments to include in the output file. The following table lists and describes possible values.

Value:	Description:
all	Show all comments.
except data	Show all comments except data value comments.
except interrupt	Show all comments except interrupt and I/O comments.
except others	Show only data, interrupt, and I/O comments.
data	Show only data value comments.
interrupt	Show only interrupt and I/O comments.
others	Show all comments except data, interrupt, and I/O comments.
none	Show no automatic comments.

For example, to include all comments except data value comments, use the following line in your definition file:

```
remarks = except data
```

Segment display off
Turns off the display of segments in the listing file. Omit this option to display segments.

Tabs off
Instructs Sourcer to output spaces instead of tabs. Unless this option appears in the definition file, Sourcer uses tabs.

uP =
Selects the a microprocessor instruction set. Possible values are 8088, V20/V30, 80186, 80286, P80286 (protected mode), 80386, P80386, 80486, P80486, AUTO, and MAX. AUTO sets the instruction set to the microprocessor on which Sourcer is running. MAX operates the same as AUTO, but uses protected modes. The default is 8088. For example:

```
up = P80846
```

Vertical lines =
Sets the number of lines per page and the width compression for listings. The default number of lines per page is 59. You can specify any number from 10 to 255. Set this option to 0 to suppress page breaks and page headers in listings.

Listings usually require 132 columns to show hex information, code, and comments. 80-column printers may require you to use a condensed printing mode. Laser printers may require you to print in landscape mode or to print using a character set that allows 132 columns per line. Consult your printer manual for the proper setup.

Using the Vertical lines option, you can set one of four levels of width compression. The following table describes each level.

	Characters Per Line	
Level	**ASM File**	**LST File**
c0	100	132
c1	92	124
c2 (the default)	84	116
c3	76	108

Tabs must be ON to allow compression. (See page 41.) For example, a definition file might contain one of the following lines:

```
Vertical = 76
Vertical = 59, c3
```

Word upper
Specifies uppercase characters in output. Omit this option to use lowercase characters.

Xref =
Turns cross-references on or off. Turning cross-references on sets the number of passes to 5. For example, use the following line to turn on cross-references:

```
Xref = on
```

Control Information Example

The following example shows a control information section.

Example 4-2: Control Information

```
        Section 1  CONTROL INFORMATION
Analysis options        = B
uP                      = 8088
Input filename          = testyn
Code style              = exe
Tabs off
Xref                    = on
```

This example turns on analysis option B, sets the microprocessor instruction set to 8088/8086, and loads the file TESTYN.EXE. It also sets the code style to EXE, turns off tabs, and turns on cross-references.

Specifying Range Definitions in Section Two

The second section of a definition file defines ranges of bytes for Sourcer to process. In addition to specifying the beginning and ending bytes in a range, the definition file also specifies the range's default DS and ES segments, its segment type, and its size. A definition file can contain a maximum of 250 ranges (including segments generated by loading a file, see page 38 and "Segment Variables," next).

You can use range definitions to analyze ROM or RAM areas, to redefine a loaded file's environment, or to divide a single segment into data and code areas. Use range definitions to identify

32-bit and 16-bit segments when using an 80386 or 80486 microprocessor filter.

Segment Variables

When Sourcer loads an input file from a definition file, it identifies up to 250 segments and loads the segment values into variables. You can use segment variables in place of numeric segment values anywhere in the definition file. The segment variables let the definition file be independent of the memory locations in which the program is actually loaded. The segment variable labels always match those used in the listing. Sourcer assigns variable names to segments in the order that it encounters them. It uses the following naming scheme.

Segment Order	Variable Name
1st segment through 26th segment	seg_a to seg_z
27th segment through 52nd segment	seg_aa to seg_zz
53rd segment through 78th segment	seg_ba to seg_bz
.	
.	
.	
235th segment through 250th segment	seg_ia to seg_io

For a list of segment variable abbreviations used by earlier releases of Sourcer, see page 102.

Range Definition Entries

Each entry in the range definitions section defines a single range. It consists of a single line containing the following items:

- Beginning segment and offset.
- Ending offset.
- Default DS and ES values.
- Segment type.
- Optional segment size (either 16 or 32 bits).
- Optional comments (ignored by Sourcer).

The sample definition files and the definition file templates, SAMPLE1.DEF and SAMPLE2.DEF, contain header comments that label the components of a range definition. Range definitions can appear in any order. If your definition file contains no range definitions, it must contain the following line in section two:

```
none
```

This section describes each component of the range definition.

Beginning Segment and Offset
The beginning segment and offset defines the beginning of the range.

Ending Offset
The ending offset is the last byte in the range. If Sourcer processes an instruction near or at the end of a range, it includes in the range all bytes required for that instruction, even if the range extends beyond the ending offset you specified.

Default DS & ES
The default DS and ES values are the values of the DS and ES registers at the beginning of the range and at any external entry points within the range. Set these values to 0000 for data and stack ranges.

Segment type
You can assign to a range any one of the following segment types.

Type	Description
AUTO	Range contains both code and data. Sourcer will fully resolve references. (Used in most cases unless other types are specified.)
CODE	Range contains code. Unless items are directly referenced as data elsewhere in the input file, Sourcer identifies them as code within the CODE range.
DATA	Range contains data only. Sourcer does not process the range as code.

STACK	Range contains a stack. Sourcer does not process the range as code.
RAUTO	Range contains code and data in ROM. Sourcer treats the range the same as an AUTO type range, except that it detects attempts to write into the range and marks them as a bad data segment access.
RCODE	Range contains code in ROM. Sourcer treats the range the same as a CODE type range, except that it detects attempts to write into the range and marks them as a bad data segment access.

Segment Size

You can specify either a 16-bit or a 32-bit segment size for the range by setting the segment size to either USE16 (for a 16-bit segment) or USE32 (for a 32-bit segment). If you do not specify a segment size, Sourcer assumes the size is USE16.

Comments

You can enter comments in the final column. Each comment must begin with a semicolon (;). Range definition comments do not appear in the output file.

Range Definition Example

The following example shows a range definitions section.

Example 4-3: Range Definitions Section

```
 begin      end     default        seg    seg
 seg:off    off     ds     es      type   size
 -------    ----    ----   ----    ----   ----
0040:0000   00FF    0000   0000    data   use16   ; BIOS data area
seg_a:0000  0009    seg_a  seg_a   code   use16   ; beginning of program (code)
seg_a:000A  0026    0000   0000    data   use16   ; program (data)
seg_a:0027  0051    seg_a  seg_a   code   use16   ; program (code)
seg_b:0000  0179    0000   0000    stack  use16   ; program's stack area
```

The first range defines the data area used by the BIOS. The segment type specifies data only. For data ranges, the default

DS and ES have no meaning, so you should set them both to 0000.

The next three ranges are all within the first segment, represented by the segment variable seg_a. (Sourcer determines segments when it loads the input file.) When ranges are consecutive within the same segment, Sourcer does not generate extra ORG statements at the range boundaries. The example predefines the code and data areas rather than allowing Sourcer to automatically determine them.

The last segment range uses the second segment (seg_b) and defines the size of the stack (180h bytes).

NOTE: If it has loaded a file, Sourcer compares the range definitions you specified in the definitions file with the ranges it established when it loaded the program. If there are any overlaps, it uses the ranges defined in the definition file. Use the Keep segments control option (described on page 38) to control handling of range definitions.

Specifying Reference Definitions in Section Three

The reference definition section lets you define the components of the input file by creating specific references to data, locations, or subroutines. You can use it to

- Add descriptive labels to code.
- Set data items to a specific type and length.
- Force items to be data or code.
- Identify external entry points.
- Insert your own comments in the code.
- Fine tune a Sourcer output file.

Reference Definition Entries

Each one-line entry in the reference definition section defines a single reference. A reference consists of the following items:

- Segment and offset.
- Item type.
- Reference options.
- Reference label (optional).
- Comment (optional).

Reference definitions can appear in any order. However, to avoid duplicate entries, we recommend ordering them by segment and offset. All values should be in hex. If your definition file contains no reference definitions, it must contain the following line in section three:

```
none
```

The sample definition files and the definition file templates, SAMPLE1.DEF and SAMPLE2.DEF, contain header comments that label the components of a reference definition. This section describes each component of the range definition.

Segment and Offset

The segment and offset is the absolute location of the referenced item. If the control information section of the definition file loads the input file, you should use segment variables instead of hex values for segments. For a complete description of segment variables, see page 43.

Item Type

Assign one of the following types to the item (the last three types have abbreviations in addition to their standard labels).

Type:	Description:
DB	Data byte.
DW	Data word.
DD	Double word.
DA	ASCII data byte (text).

DS	Data structure. See the example on page 53.
SUB	Subroutine reference.
LOC	Location reference.
FORCED	Forced function.

Each item type has its own set of reference options.

Data Options. The reference definition for a data item can specify one or more of the following data options.

Option:	Description:
, C *xxx*	Insert comment; *xxx* is a hex number from 1 to 3FF and references a comment stored in a separate remarks file.
, DUP	Duplicate bytes or words. Use only with DB or DW types.
, EQU	Force the data reference to an equate (DW type only).
, INDEX	Replace an index with the associated label. See the example on page 53 for its usage.
, MULTI	Multiple defined label. Code generated using this option will not reassemble.

The following DW type options specify an offset to another reference:

, OSN	Offset Subroutine Near (dw offset sub_*xxx*)
, OSF	Offset Subroutine Far (dw offset sub_*xxx*, seg sub_*xxx*)
, OLN	Offset Location Near (dw offset loc_*xxx*)
, OLF	Offset Location Far (dw offset loc_*xxx*, seg loc_*xxx*)
, ODN	Offset Data Near (dw offset data_*xxx*)
, ODF	Offset Data Far (dw offset data_*xxx*, seg data_*xx*)
, R *xxxx*	Repeat the data type *xxxx* times, where *xxxx* is a number from 1 to FFFFh (the default is 1). For example, dw, R 4 indicates

	8 bytes. If text data (type DA) has no R *xxxx* option, Sourcer automatically sets the length of the text to the first dollar sign ($) or 0 it finds.
, UNUSED	If item is never referenced, then do not use.
, SEG	Use segment name as value (DW type only). (dw seg_b)

Location Options. A reference definition for a location item can specify one or more of the following options.

Option:	Description:
, C *xx*	Insert comment where *xx* is a hex number from 1 to FF representing a comment in a separate remarks file.
, EXT	Create an external entry point into the program. Sourcer resets the simulation registers to 0 and sets the DS, ES, and SS segments to their default values.
, FAR	Create a far procedure and reset simulation registers to 0. Sets DS, ES, and SS to their defaults.
, NEAR	Create a near procedure and reset simulation registers to 0. Sets DS, ES, and SS to their defaults.
, UNUSED	If item is never referenced, do not use it.

Subroutine Options. A reference definition for a subroutine item can specify one or more of the following options.

Option:	Description:
, C *xx*	Insert comment where *xx* is a hex number from 1 to FF representing a comment in a separate remarks file.
, FAR	Define subroutine as FAR. If not specified, Sourcer automatically determines whether the subroutine is near or far based on usage.
, TERM	Subroutine terminates (it does not return to caller). After each call to the subroutine, Sourcer switches to data mode.

, UNUSED	If item is never referenced, then do not use it.

Forced Function Options. Use the following options to specify forced functions.

Option:	Description:
, CODE	Switch to code mode without changing segment size.
, CODE16	Switch to 16-bit code mode and change segment size to 16 bits. Sourcer remains in 16-bit code mode until the end of the segment or until encounters a CODE32 option.
, CODE32	Switch to 32-bit code mode and change segment size to 32 bits. Sourcer remains in 32-bit code mode until the end of the segment or until encounters a CODE or CODE16 option. The microprocessor instruction set should be 80386 or 80486 to use CODE32.
, DATA	Switch to data mode.
, ENDP	Force an ENDP instruction to appear after this instruction if inside a subroutine or major procedure.
, AX=*xxxx*	Change the value of a simulation register to *xxxx* (a 4-digit hex value). You can use this option to set the following 16-bit simulation registers: AX, BX, CX, DX, BP, SI, DI, SP, DS , ES, or SS.
, EAX=*xxxxxxxx*	Change the value of a simulation register to *xxxxxxxx* (an 8-digit hex value). You can use this option to set the following 32-bit simulation registers: EAX, EBX, ECX, EDX, EBP, ESI, EDI.
, EA*x*	Selects LEA instruction's reference item type. By default, Sourcer selects an absolute number (LEA DI,DS:[234h]). Use this option to select a different reference. It has the following forms: EAS - for subroutine. EAL - for location.

	EAD - for data. To use a segment other than the current one, you can append a segment value (See the example on page 56).
, O*x*	Converts immediate value to offset at the referenced instruction. This option has the following forms: OS - for offset to subroutine. OL - for offset to location. OD - for offset to data item. To use a segment other than the current one, you can append a segment value (See the example on page 56).
, REG	Displays all internal simulation registers.

The following options control forced table analysis for indexed call and jump instructions.

, SN	Table offset sub_*xx*
, SF	Table offset sub_*xxx*, seg sub_*xxx*
, LN	Table offset loc_*xx*
, LF	Table offset loc_*xxx*, seg loc_*xxx*

Use the format seg_a:xxxx f, sn 1234:22 34 where 1234:22 is the position of the subroutine table with 34h entries.

Label Field

In this optional field, you can enter a label up to 15 characters long. Do not enter a comma before the label or include a comma within it. Forced functions ignore the label field.

If you do not enter a label, Sourcer ignores the remainder of the line. Thus you cannot enter comments for unlabeled items in the definition file. You can, however, enter comments for unlabeled items in a remarks file. (See Chapter 6, "Adding Comments to the Output File.")

Comment Field

In this field, you can enter a short comment (64 characters or less). Your comment appears in listings when the item is

defined. If a semicolon appears in the comment, Sourcer ignores the remainder of the line. Type a semicolon at the beginning of comments that you want to appear only in the definition file.

Forced functions ignore the comment field.

Reference Definition Examples

The examples in this section show how to use the data reference section.

Simple Data References

The following example shows how to create simple data references.

Example 4-4: Data References

```
seg:off     type & options    label            comments
-------     --------------    --------------   --------------------
0000:0020   dw,  Unused       vector_8h_off
0000:0022   dw,  Unused       vector_8h_seg
0040:001E   dw,  R 10         keybd_buffer     ; 16 words (10h)
0040:0049   db                video_mode       Current video mode
seg_c:0025  dw,  C 4E         speaker_port     ; Comment # 4E
```

In the example, the first two data items reference the interrupt 8 vector offset and segment. The type is set to word (DW), and the item will be removed if it is never referenced or used within the program.

The third and fourth items reference the BIOS data area. The keyboard buffer is comprised of 16-word entries (10h). The video mode is a single byte. The comment Current video mode appears where the video mode is defined in Sourcer output files.

The fifth item, speaker_port, is a data word in the third segment of the loaded program at an offset of 25h. In listings, comment number 4Eh from the associated remarks file appears where the item is defined. For more information on remarks files, see Chapter 6, "Adding Comments to the Output File."

Special Data Items and Options

The following example explains the usage of special options and more complex data functions.

Example 4-5: Special Data Items and Options

Line	seg:off	type & options	label	comments
1	seg_c:0006	dw, Index	index_six	
2				
3	seg_c:0200	ds, R 3		; Data Structure (3 times)
4		dw	handle	; 1 word
5		da, R 0C	filename	; 12 character string
6				
7	seg_c:0224	dw, OSF	sub_pointer	; Offset to a subroutine
8				
9	seg_c:0228	db, DUP, R 8	temp_buffer	; Duplicate bytes
10				
11	0000:0449	db, Multi	video_mode	; Multi-defined
12	0040:0049	db	video_mode	; Primary definition

The first item overrides an index value of 6 in the third segment. References of the type [si+6] will appear as index_six[si]. This function is primarily used with a Pre-Processor. At location 6, the item index_six will be defined as an equate rather than a data word.

The third line defines a structure that repeats 3 times. A data structure can contain any number of data items within the same segment up to a total of 64 Kb. The structure in the example contains a single word (line 4) and an ASCII string of 12 characters (line 5). The structure concludes when Sourcer encounters a blank line or another segment and offset in column 1.

Line 7 indicates the data item is a pointer to a far subroutine. Line 9 creates duplicate data bytes, 8 bytes long. Sourcer does not check whether the bytes are identical, but lists only the first byte as duplicated.

The following listing section shows the result of using the definition file example with a program. Note how the data structure expands into 3 sets of data.

Example 4-6: Sample Program Listing

```
6159:0200  0005                  handle      dw  5
6159:0202  74 65 73 74 61 32     filename    db  'testa100.com'
           31 31 2E 63 6F 6D
6159:020E  0006                  handle1     dw  6
6159:0210  74 65 73 74 61 33     filename1   db  'testa200.com'
           31 31 2E 63 6F 6D
6159:021C  0000                  handle2     dw  0
6159:021E  00 00 00 00 00 00     filename2   db  0, 0, 0, 0, 0, 0
           00 00 00 00 00 00                 db  0, 0, 0, 0, 0, 0
6159:0224  07FA 6159             sub_pointer dw  offset sub_23, seg sub_23
6159:0228  0008[FF]              temp_buffer db  8 dup (0FFh)
```

Location References

The next example shows how to define location references with descriptive labels.

Example 4-7: Location Items

```
seg:off     type & options    label            comments
-------     --------------    --------------   --------------------
seg_a:0030  loc               begin_program
seg_a:0422  loc,  FAR         int_9_keyboard   ; Far procedure
seg_a:053E  loc,  C 1E        error_handler    ; Comment # 1E
```

The first entry is typical of most location references. It defines a label begin_program in the first segment at offset 30h. The second item int_9_keyboard uses the FAR option to indicate that the reference is a far procedure and to reset Sourcer's internal simulator. The last line references the location comment number 1Eh in the associated remarks file. In a listing, the comment appears before the label. (For more information on using the remarks file, see Chapter 6, "Adding Comments to the Output File.")

Subroutine References

The next example defines subroutine references.

Example 4-8: Subroutine References

```
seg:off     type & options    label          comments
-------     --------------    -------------  --------------------
seg_a:10E2  sub               clear_buffer
seg_a:119B  sub, Far          beep_once      ; Far subroutine
seg_a:133D  sub, Far, C 24    boop_twice     ; Far & use comment 24
seg_a:1C71  sub, TERM         out_string     ; Terminated subroutine
```

Most subroutine references use a format similar to the first entry. Sourcer automatically determines whether the subroutine is near or far based on usage. The second entry forces the subroutine to be defined as a far procedure. The third entry combines the FAR option and the COMMENT option. In listings, comment 24h appears in the subroutine header block.

The last entry uses the TERM function to define a terminated subroutine. When Sourcer encounters a call to a terminated subroutine, it indicates that the subroutine will not return to the next instruction and switches from code to data mode.

The following example shows part of the listing generated from the reference definitions in Example 4-8. On the first line, it shows a subroutine defined as terminated.

Example 4-9: Terminated Subroutine

```
6A65:00E2  E8 168F              call    out_string              ; (1C71) Sub does not return here
6A65:00E5  43 6F 70 79 72 69    db      'Copyright (c) 1988,1989', 0
6A65:00EB  67 68 74 20 28 63
6A65:00F1  29 20 31 39 38 38
6A65:00F7  2C 31 39 38 39 00
6A65:00FD                   loc_11:
6A65:00FD  8B D1                mov     dx,cx
```

Forced Functions

The following example shows the use of forced functions.

Example 4-10: Forced Functions

```
Line  seg:off      type & options      label        comments
      -------      --------------      -----------  ----------------
   1  seg_a:0CB1   forced, code                      ; Force to code mode
   2  seg_a:0D22   forced, data                      ; Force to data mode
   3  seg_a:0D9A   forced, endp                      ; Indicate end procedure
   4
   5  seg_a:0FF6   forced, ds=seg_c                  ; set DS to 3rd segment
   6  seg_a:16C9   forced, es=40                     ; set es to 40h
   7  seg_a:1CD5   forced, registers                 ; display registers
   8
   9  seg_a:1714   forced, eas                       ; LEA ptr to subroutine
  10  seg_a:188C   forced, eal seg_c                 ; LEA ptr to location
  11
  12  seg_a:1AB8   forced, od                        ; immediate to offset data
  13  seg_a:1C32   forced, os F000                   ; immediate to sub offset
  14
  15  seg_a:1E97   forced, sn seg_a:32 2             ; sub near index table
  16  seg_a:1F92   forced, lf 1234:4C2 0             ; location far index table
```

The first entry sets the system to process code at location seg_a:CB1. The second line forces the system to process data at seg_a:D22. The third line issues an ENDP directive after the instruction at seg_a:D9A.

The fifth and sixth lines set the DS and ES simulation registers. You can use this function to change an incorrect simulation register. Use an entry similar to the seventh line to show the contents of the simulation registers in the listing.

The LEA instruction opcode at seg_a:1714 is set to a subroutine. If a subroutine does not exist in the same segment, Sourcer creates one. The tenth line specifies that the LEA instruction at seg_a:188c should reference a location in the 3rd segment (seg_c).

When an immediate word occurs in most instructions, Sourcer cannot determine whether the word is a value or an offset to another reference. Sourcer treats the immediate word as a value. This practice allows accurate reassembly as long as bytes are not added or deleted before the referenced item. Lines 12 and 13 convert the immediate value into an offset. The instruction at seg_a:1AB8 has an immediate field. The entry at line 12 converts the field to an offset of a data item relative to the CS

segment. The next entry converts the immediate field at the instruction seg_a:1C32 to an offset of a subroutine, relative to the F000h segment.

The last two entries demonstrate the forced index function. When Sourcer cannot automatically determine whether an indexed jump or call instruction requires it to determine a table of pointers, this function forces it to create a table of pointers. The example first specifies the type of table (sn, sf, In, or If). Next it gives the segment:offset of a table of pointers. Finally, it provides the number of entries in the table (1 to FFh). If you use a value of 0, Sourcer determines the number of table entries.

Forced Function Indicator

The listing file denotes forced functions by inserting a period between the instruction offset and the hex instruction code. If a forced function was specified in the definition file but a period does not appear in the listing, the function was not used. Sourcer does not use a forced function if it is inappropriate or if the definition file entry contained the wrong segment:offset address.

The second line in the following example shows a forced function in a listing file.

Example 4-11: Forced Function

```
62AA:0000  52          push  dx
62AA:0001  .BA 03C8    mov   dx,offset loc_234
```

Chapter 5: Setting System Defaults

The file SDEFAULT.DEF contains default settings for Sourcer Main Menu commands and analysis options. You can edit this file to customize Sourcer defaults.

SDEFAULT.DEF is a definition file (for a complete description of the definition file format, see Chapter 4, "Using Definition Files.") Sourcer uses it only if you do not specify another definition file when you start Sourcer. After starting Sourcer, you can use the Main Menu commands to change settings.

The control information section (section one) in the SDEFAULT.DEF file presets Sourcer Main Menu commands and analysis options. For information on setting options in this section, see page 36. The range definition section does not apply to an SDEFAULT.DEF file. It must contain the single line:

```
none
```

The reference definition section (section three) of the standard SDEFAULT.DEF file contains labels for commonly used BIOS values. These entries are used only if the input program references BIOS data items. Only referenced data items appear in the Sourcer-generated definition file (SDF file).

The following example shows part of the standard SDEFAULT.DEF file.

Example 5-1: SDEFAULT.DEF File

```
SOURCER DEFAULT DEFINITION FILE

Section 1: CONTROL INFORMATION

input filename = ?   (? = use command line entry if provided)

Section 2: RANGE DEFINITION

begin     end       default       seg     seg
seg:off   off       ds    es      type    size
-------   ----      ----  ----    ----    ----
none

Section 3: REFERENCE DEFINITIONS

Subroutines
seg:off   type & options   label          comments
-------   --------------   --------------   --------------

Locations
seg:off   type & options   label          comments
-------   --------------   --------------   --------------

Data Items
seg:off   type & options   label          comments
-------   --------------   --------------   --------------
0040:0000 dw, Unused       @rs232_port_1    ; Start of primary BIOS RAM
0040:0002 dw, Unused       @rs232_port_2    ; These entries are optional and
0040:0004 dw, Unused       @rs232_port_3    ; will only show up if referenced
0040:0006 dw, Unused       @rs232_port_4    ; by the program under analysis
0040:0008 dw, Unused       @prn_port_1
0040:000A dw, Unused       @prn_port_2
```

When you start Sourcer without specifying a definition file, it loads the SDEFAULT.DEF file in the current directory. If there is no SDEFAULT.DEF file, it uses settings preset in the program without issuing a warning message. If you need to use different sets of Sourcer defaults, you can create SDEFAULT.DEF files in different directories or on different disks.

To start Sourcer without using SDEFAULT.DEF from a directory that contains an SDEFAULT.DEF file, use the following command:

```
sr -d
```

Chapter 6: Adding Comments to the Output File

Using definition files, you can insert comments into the Sourcer output file. Each comment is associated with a specific data, location, or subroutine reference item in the definition file's reference definition section. Comments appear in listings and in source code files. Depending on the type of item it references and the method you used to specify a comment, it either appears to the right of the referenced item or immediately before the referenced item.

There are two ways to associate comments with a data, location, or subroutine reference. The easiest way is to include the comments in a reference definition entry in the definition file. Use this method for comments that are 64 characters (the maximum allowed) or less in length. The following example shows a comment associated with a data reference using this method. The comment in line 4, Current video mode, appears to the right of the data reference in the output file. The comment on line 3 does not appear in the output file. Sourcer ignores any characters after a semicolon in a definition file line.

Example 6-1: Definition File Comment

```
seg:off     type & options      label              comments
-------     --------------      --------------     ----------------
0000:0020   dw,   Unused        vector_8h_off
0000:0022   dw,   Unused        vector_8h_seg
0040:001E   dw,   R 10          keybd_buffer       ; 16 words (10h)
0040:0049   db                  video_mode         Current video mode
```

If a comment is longer than 64 characters, you can store it in a remarks file associated with the definition file. All comments in the remarks file are numbered. You use the comment number to associate it with a data, location, or subroutine reference in the

definition file. The following example adds one line to the previous example. The code C 4E in the last line links the data reference to a comment in the associated remarks file.

Example 6-2: Remarks File Comment

```
seg:off     type & options    label           comments
-------     --------------    --------------  ----------------
0000:0020   dw,  Unused       vector_8h_off
0000:0022   dw,  Unused       vector_8h_seg
0040:001E   dw,  R 10         keybd_buffer    ; 16 words (10h)
0040:0049   db                video_mode      Current video mode
seg_c:0025  dw,  C 4E         speaker_port    ; Comment # 4E
```

You can use both commenting methods with a single input file. Sourcer checks the definition file first. If the definition file contains a comment for a particular item, it does not use the remarks file comment for that item.

Using a Definition File to Add Comments

The easiest way to add a comment is to include it in the definition for a data, location, or subroutine reference in the reference definitions section (section three) of the definition file. For more information about adding comments to definition files, see Chapter 4, "Using Definition Files." Data and location comments specified in the definition file appear to the right of the referenced item in output files. Subroutine comments appear before the subroutine reference.

Using a Remarks File to Add Comments

When comments are longer than 64 characters, you must store them in a remarks file. The remarks file must reside in the same disk and directory as the corresponding definition file. It must have the same filename as the definition file with the extension .REM instead of .DEF. A remarks file consists of three sections--one for each type of comment.

The file TESTYN.REM is a sample remarks file for use with the demonstration definition file TESTYN.DEF and the input file TESTYN.EXE. You may want to examine this file as you create your own remarks files.

Data Remarks Section

The data remarks section contains comments associated with data items. Each comment must begin with a three digit hex reference number, followed by a space and then the comment text. Comment reference numbers can range from 001 to 3FF (for a total of 1023 comments). Comments can consist of multiple lines. However each line cannot consist of more than 34 characters. The following line must conclude the data remarks section:

```
end
```

The following illustration shows a sample data remarks section.

Example 6-3: Data Remarks Section

```
--------- Data Remarks ---------

001     This is remark number 1
002     This remark continues onto three
          lines, with the 2nd & 3rd lines
          indented 2 spaces

003
        Lines starting with a graphics
        character will be placed prior
        to the line that the data item
        comment appears on.

        A line feed will automatically
        be placed prior to and after
        the graphics character area.

        this line is left of data item 3
end
```

The Location Remarks Section

The location remarks section contains comments associated with location items. In the output file, location comments appear before the location label. Sourcer automatically inserts a line feed before the comment. If you want a line feed to occur after the comment, you must include it as part of the comment in the remarks file.

Each comment in the location remarks section must begin with a two-digit hex reference number, followed by two spaces and then the comment text. Comment reference numbers can range

from 01 to FF (for a total of 255 comments). Comments can contain graphics characters and consist of multiple lines. However, each line cannot consist of more than 76 characters. The following line must conclude the location remarks section:

```
end
```

The following illustration shows a sample location remarks section.

Example 6-4: Location Remarks Section

```
---------- Location Reference Remarks ----------
01  This is a location reference remark

02  ═══════════| GRAPHICS |═══════════

    ■ All graphics characters are allowed for special highlighting

    ■ Line feeds can be inserted for extra clarity

03  Text of each remark begins in column 5 in this file
end
```

Subroutine Remarks Section

The subroutine remarks section contains comments associated with subroutine items. In the output file, subroutine comments appear before the subroutine procedure. Sourcer automatically inserts a line feed before the comment. If you want a line feed to occur after the comment, you must include as part of the comment in the remarks file. If the subroutines uses the standard header, Sourcer inserts the comment into the header.

Each comment in the subroutine remarks section must begin with a two-digit hex reference number, followed by two spaces and then the comment text. Comment reference numbers can range from 01 to FF (for a total of 255 comments). Comments can contain graphics characters and consist of multiple lines. However, each line cannot consist of more than 76 characters. The following line must conclude the subroutine remarks section:

```
end
```

The following illustration shows a sample subroutine remarks section.

Example 6-5: Subroutine Remarks Section

```
---------- Subroutine Reference Remarks ----------
01 TESTA SUBROUTINE
02                              BEEPER SUBROUTINE

   This subroutine generates a short beep.  Note that the sound and
   duration are uP speed dependent.


   Registers used:  al,bx,cx,dx,ds

end
```

Remarks File Notes

- Each remark can consist of up to 64,000 characters.
- The maximum remarks file size is the smaller of 32 Mb or 64,000 lines.
- Sourcer automatically inserts a semicolon before a comment in an output file; you do not need to include the semicolon in the comment text in the remarks file.
- To create a blank line with no semicolon, enter a return without characters on or before column 3. If Sourcer detects any characters (including a single space), it writes a semicolon to the output file.

Chapter 7: Reassembling Generated Source Code

This chapter gives tips for:

- Generating code that is most compatible with a given assembler.
- Checking a generated source file before reassembling it.
- Correcting reassembly errors.
- Changing generated source code.

Generating Code for Assembler Compatibility

Different assemblers generate code that is functionally the same. However, the actual instructions used may vary. The 8086 family of processors have a number of instructions that perform the same task, but use different opcodes. Some assemblers don't always use the most compact instruction.

The following examples compare code produced by MASM 4.0 and MASM 5.1.

Example 7-1: MASM 4.0 Code

```
612A:0123 81 E3 0030          and         bx,30h
```

Example 7-2: MASM 5.1 Code:

```
612A:0123 83 E3 30            and         bx,30h
```

MASM 4.0 produces a valid instruction, but does not use the most compact form. MASM 5.0 and 5.1, however, use a more compact instruction to perform the same function. In these examples, the same instruction produces different results. In addition, there is no way to instruct the assembler to reassemble code in exactly the same way that it was assembled or compiled.

Sourcer has two automatic fixup routines that insure the same functional operation as the original code by maintaining the same number of bytes assembled. Sourcer cannot insure the assembler you are using will create a program that is identical byte-for-byte with the original program. The remainder of this section describes assemblers that have been tested with Sourcer and gives recommended options for each one. For a complete description of Sourcer analysis options, see Chapter 3, "Customizing Sourcer Operation."

Microsoft Macro Assembler (MASM) version 4.0 or earlier

When generating code to be reassembled with MASM 4.0 (or earlier versions), turn on Sourcer analysis options L and M. If the original program was assembled with a later version of MASM or another optimizing compiler and you disassemble it using MASM 4.0 settings, Sourcer produces code that may have more bytes than the original. If the original source code had both far and near returns in the same procedure, MASM 4.0 cannot reassemble it. MASM 4.0 and earlier versions do not support unique 80386/80387 or 80486 instructions. Contact Microsoft to upgrade to a later version of MASM for a nominal fee.

Microsoft Macro Assembler (MASM) versions 4.5 and later

When disassembling code created with MASM version 4.5 or later, leave analysis options L, and M off (the default settings). Sourcer uses MASM features to produce phase locked code. MASM 5.1 has several bugs that may make it impossible to reassemble the code if it was not originally assembled using MASM 5.1. MASM 5.1 and earlier versions do not support unique 80486 instructions.

Turbo Assembler (TASM)

When disassembling code generated with TASM, leave analysis options L and M off (the default settings). Sourcer uses features of this assembler to produce phase locked code. When reassembling Sourcer-generated code with TASM, use the TASM option for compatibility with MASM 5.1 (default TASM operation). TASM 1.0 does not support unique 80486 instructions.

OPTASM versions 1.0 and 1.5

When reassembling Sourcer-generated code, use the OPTASM /m command line option (for MASM-compatible code). This option turns off optimizations that can change the phase lock by using more compact instructions than the original code. Use this option even if the Sourcer input file was originally created with OPTASM. OPTASM 1.0 and 1.5 do not support unique 80386/80387 and 80486 instructions.

A86 version 3.0 and later versions

To generate code for assembly with A86, turn on Sourcer analysis option L. This assembler is not fully compatible with Sourcer output. A86 may report errors that you'll have to fix by editing the generated source code. Also, A86 uses different opcodes from other assemblers that are functionally the same. Thus, it is difficult to compare the original and A86-generated hex files. A86 does not support unique 80286/80287, 80386/80387, and 80486 instructions.

Reassembly Checklist

Sourcer was designed to make reassembly easy and relatively trouble-free. However, Sourcer must make some assumptions as it analyzes an input file. As a result there are some things you should check before reassembling a generated file. In addition, some warnings or errors may occur when you reassemble the file.

Everyone has different objectives when reassembling generated code. For example, if you need to thoroughly understand a program and plan to make extensive modifications, it may take more work to reassemble it. If a specific problem exists and a

minor patch will fix it (for example, changing bytes to NOP instructions or changing a specific instruction), you may be able to skip some of the steps described in this section.

Sourcer automatically inserts an asterisk (*) on lines where it made assumptions that could cause reassembly problems. Use an editor or list program, to search for ;* to examine each potential problem. It is usually easier to examine problems when the listing contains cross-references. (For instructions on including cross-references in the listing, see page 10.)

The remainder of this section list specific problems for which you should check and gives recommended solutions.

Indexed Jumps and Calls

Sourcer automatically builds a data table for referenced calls and jumps. In a few cases, Sourcer's internal simulation may not link to the correct table. The most common indicator of this problem is a comment that indicates only one entry. For example:

Example 7-3: Incorrect Table

```
call dword_ptr data_123 [BX]          ;* (63A2:078E =(0CDh) 1 entry
```

Check that the table appears valid, and that the proper number of entries were found. If necessary, use a definition file to replace Sourcer's automatic analysis with forced table analysis. Use the SN (sub near), SF (sub far), LN (loc near), and LF (loc far) options. (See page 50.) The automatic analysis stops after 50 entries to prevent possible errors from creating a very large table.

Register Jumps and Calls

To learn the destination of a register jump or call, you must analyze previous code. In some cases the original program may contain a table of jumps or calls instead of using an indexed jump or call. An instruction preceding the jump or call loads the address into the register (see the following example). However, without having encountered the subsequent register jump or call, Sourcer may interpret the address as data.

Example 7-4: Register Jump

```
mov     cx,90Eh
jmp     cx
```

To change an immediate value to an offset to the desired reference, make a definition file entry that points to the beginning of the instruction (in the example, mov cx,9DEh). Use the forced function code OL for a jump and OS for a subroutine.

If the program sets up a data table, you can set the entries in the table to offsets to the jump locations or subroutines. Use a definition file entry that points to the beginning of the table. Define the data words as offsets to the locations or subroutines. For example to define a table of *xx* location offsets, use the codes:

```
DW, OLN, R xx Table_start
```

For a table of subroutine offsets, use OSN in place of OLN.

Immediate Fields within an Instruction

Sourcer may not determine whether an immediate field within an instruction is simply a numeric value or an offset to a data, subroutine, or location reference. An immediate field could also be an offset to a data item in another segment (in which case the current segment register is unrelated to the instruction containing the immediate field). The binary file does not contain information specifying the use of the immediate field. The following example shows an immediate field.

Example 7-5: Immediate Fields

```
mov  si,1C24h
```

If you reassemble the file and the instructions remain in the same position as the original code, you will have no problems with immediate fields. Sourcer has several methods of insuring that the instructions remain in the same position. However, if you modify the generated code you may need to determine whether each immediate field is an offset or offset type (data, location, or subroutine) and in which segment the item resides.

You can use a definition file to force an immediate value into an offset. If the item is not in the same segment as the instruction, include the segment in the definition file entry. Use the forced

function codes OS, OL, and OD. For their description, see page 50.

Incorrect Code or Data Determination

Although the binary file does not distinguish code from data, Sourcer's multiple-pass method of disassembling a file usually results in a very accurate determination of code and data items.

Occasionally, code appears as data or data appears as code in a generated file. A section of data might happen to look like code, or code might be referenced as data (as is self-modifying code). Unless you plan to modify the generated file, you may not need to correct such problems.

To correct inaccurate code and data designations, you can use a definition file to force an area of the program to be data or code only. (See the following instructions.) You can also set the code style to fragment or turn on analysis options A through C to change some data areas to code. (See page 17.)

Using Segment Ranges to Specify Data and Code Areas

In the range definitions section (section two) of a definitions file, you can specify a range of bytes to treat as data or code. For example, the definition file for the program TESTYN.EXE might have the following ranges defined. (In this case, Sourcer does not need the range definitions in order to accurately handle the data and code ranges. They are shown as an example only.)

Example 7-6: Definition File, Section 2

```
 begin      end      default        seg    seg
 seg:off    off    ds      es      type   size
 -------    ----   ----    ----    ----   ----
seg_a:0000  0001   seg_a   seg_a   code   use16   ; first jump instruction is code
seg_a:0002  0026   0000    0000    data   use16   ; forced data area
```

Since the two segment ranges are contiguous, Sourcer treats them as a single segment with forced code and data sections.

Specifying Forced Data Areas

You can also use data references in section three of the definition file to specify data items. When Sourcer encounters a single

data reference at the beginning of a data area, it remains in data mode until it encounters a location label or subroutine.

Example 7-7 shows a line that specifies a data area beginning with a data byte at offset 2. Example 7-8 shows a line that specifies a data area without including a label or setting a specific type of data.

Examples 7-7: Data Type and Label

```
seg:off      type & options   label            comments
-------      -------------   --------------   ----------------
seg_a:0002   db              data_start_area
```

Example 7-8: Forced Data

```
seg:off      type & options   label            comments
-------      -------------   --------------   ----------------
seg_a:0002   forced, data
```

Specifying Forced Code Areas

Use a location label or forced function code to switch back to code mode. The following examples show two ways to force the file back to code mode at offset 27h.

Example 7-9: Location Reference

```
seg:off      type & options   label            comments
-------      -------------   --------------   ----------------
seg_a:0027   loc             begin_code
```

Example 7-10: Forced Function

```
seg:off      type & options   label            comments
-------      -------------   --------------   ----------------
seg_a:0027   forced, code
```

Inaccessible Code

When an area of code is never accessed, Sourcer assumes it is data. Code that was originally produced by a compiler some-

times has inaccessible (and unnecessary) code fragments. The better the compiler is at optimizing, the fewer inaccessible code fragments there will be.

External Entry Points

Sourcer normally determines external entry points into a program. The file type (and the EXE and device driver file header) provides external entry point information. In special cases, Sourcer cannot determine entry points from the input file. You must predefine the entry points, or Sourcer cannot accurately process the file. For example, disassembling a BIOS requires predefined entry points. The optional BIOS Pre-Processor program, available from V Communications, predefines BIOS entry points for you.

Terminate and Stay Resident (TSR) programs also require predefined entry points. When installed, TSR programs create additional external entry points to interlink the system interrupts. Sourcer's internal simulator can detect these entry points if they are created through the standard DOS interface. Sourcer generates a special label to indicate interrupt entry points; it has the format int_*xx*h_entry where *xx* is the interrupt number. If a TSR program doesn't use the DOS convention, Sourcer won't be able to determine external entry points.

When Sourcer is unable to determine entry points, you can predefine them in a definition file. The predefined external entry points can also create a near or far procedure, if necessary. Predefined interrupt vectors from TSR programs, for example, should include a far procedure call. The following example shows a far procedure entry point definition.

Example 7-11: Entry Point

seg:off	type & options	label	comments
seg_a:0422	loc, far	int_9_keyboard	; Far procedure

Reassembly Errors

Even if you carefully prepared the generated code file, some errors may occur during reassembly. If many errors occur, most of them are probably caused by a few key problems. This sec-

tion suggests actions you can take to correct the most common errors.

Block Nesting Errors

Sourcer correctly identifies procedures and subroutines in modular programs and (in most cases) non-modular programs. In some cases, it is impossible to correctly define the endpoint of a procedure. For example, a subroutine that does not exit or that jumps into a previous subroutine may fail to have a properly placed end of procedure (ENDP) statement. An assembler will generate an error for such a subroutine. You must manually insert an ENDP statement at the correct point in the source code file or specify a forced ENDP function at the correct point in the definition file and disassemble the code again.

Syntax Error (RETN/RETF)

MASM version 4.0 and older assemblers do not support the RETN and RETF instructions. Rerun Sourcer with analysis option L turned on.

Phase Errors Between Passes

Phase errors are difficult to debug, because an error message does not appear when the actual error occurs. The error occurs at some point between the message and the last location or subroutine reference.

To correct phase errors, rerun MASM with the /d command option. This option generates a listing for both assembly passes. (Disregard the errors that MASM finds during the first pass; they are normal.) For each instruction above the phase error, compare the hex instruction bytes in the pass 2 listing with the same instruction bytes in the pass 1 listing. You will find one instruction that has a 1 byte difference between the two passes. In the example below MASM codes a 4 byte move instruction in pass 1, but corrects its error in pass 2, using a 3 byte move instruction. Correcting the instruction creates a phase error.

Example 7-12: Phase Error Pass 1 Listing

```
002A  8B  1E    0000  U                    mov          bx,seg_c
testxxx.ASM(40): error A2009: Symbol not defined:SEG_C
002E  8E  C3                               mov          es,bx
0030  E8  0000  U                          call         sub_2
testxxx.ASM(42): error A2009: Symbol not defined:SUB_2
0033                      testa:
```

Example 7-13: Phase Error Pass 2 Listing

```
002A  BB  ----  R    mov            bx,seg_c
002D  8E  C3         mov            es,bx
002F  E8  0052  R    call           sub_2
                          testa:
testxxx.ASM(43): error A2006: Phase error between passes
```

In the example, MASM cannot handle a valid 8088 instruction. You can use TASM or OPTASM to reassemble the source code. You may also be able to correct the problem by changing the instructions so MASM will accept them. In the example, you could replace the problem instruction (mov bx, seg_c) with the following instructions:

```
mov     ax,seg_c
xchg    ax,bx
```

Unlike the original code which does not affect the AX register, the replacement instructions assume that it is safe to change the AX register.

You can correct other phase errors in a Sourcer definition file by forcing the offending instruction to data mode and forcing the next instruction back to code mode. Then rerun Sourcer and reassemble the output file.

Illegal Size for Operand Error

This error should not occur, because Sourcer locks the data types prior to its last pass. If it does occur, perhaps the instruction should be data. Also, check that Sourcer used at least 5 passes to generate the source code. To eliminate this error, edit the Sourcer output file so it has the proper PTR override (for example, WORD PTR DATA_217 or use a definition file entry to change the instruction to data bytes.

Changing Generated Source Code

If you are making small changes to a program, avoid adding or removing bytes from the program. To remove code, replace each byte of instruction code with a NOP instruction. The following examples show code before and after removing an instruction.

Example 7-14: Original Code

```
mov     al,1
call    sub_2
or      al,4
```

Example 7-15: Code after Removing an Instruction

```
mov     al,1
nop
nop
nop
or      al,4
```

The example uses three NOP instructions to remove the CALL instruction, because the CALL instruction was three bytes long.

If you need to add instructions, be extremely careful not to change indexed references to data items. If the original code used an offset to a data item, your changes may cause the program to handle the offset as a number. As long as you make changes and additions after indexed data references, you should encounter no problems.

For example, if a program consists of separate data and code areas with the data area first, changes you make in the code area will not affect indexed data references. You can add new code or delete code with little risk. On the other hand, if you modify items in the data area, you would make all indirect references to data items inaccurate.

Patching Programs

Once you've determined the location to change, you can patch a program instead of reassembling it. Patching a small program can be quicker than reassembling it. Patching is also useful for programs too large for your assembler.

To patch a COM-type program, use the DOS DEBUG program to change the appropriate bytes and write the new file.

To patch an EXE-type program, you must insure that you change bytes at the correct point without changing the EXE file header.

1. First, copy the program file to a file that has the same file-name and the extension .TMP.

2. Load the new file into DEBUG. At the DEBUG prompt, enter:

   ```
   -d 108
   ```

3. The word at offset 108h is the size of the header in paragraphs. Remember to convert the two bytes DEBUG shows to a word. For example, if the value DEBUG shows 00 01, use the word 0100h. Add a 0 nibble to convert to bytes. If the value is 0020, the header is 200h bytes long.

4. Add the header size in bytes plus 100h to determine point in memory where the program begins.

5. Determine the location of the bytes to change by adding the value computed in step 4 to the actual location you want to change in the first segment.

   If you want to change an item in a later segment, also add the difference between the first segment and the item's segment. The .SDF file shows the values associated with each segment name.

   Before you make changes, check the bytes you are changing against a listing to make sure that you calculated the correct location.

6. Make the necessary changes.

7. Write the file back to disk.

8. Change the file extension back to .EXE.

The following example shows how to calculate a patch location.

Example 7-16: Planning for a 1-byte Patch at Offset 45h in seg_c

```
word offset at 108h in DEBUG = 20h (or 200h header bytes)
seg_a = 7200h
seg_c = 7250h

                                  bytes to                          patch
header        +   PSP   +   seg_c from seg_a        +   offset

(20h * 10h)   +   100h  +   ((7250h-7200h)*10h)     +   45h      = 845h
```

The result, is 845h, is the offset at which to make the change. This calculation is based on the DS segment value created by DEBUG when it loads the file. You can view the bytes with the **D** command, or alter bytes with the **E** command.

The **U** (unassemble) command, uses the CS segment register, not the DS register. To unassemble at this location, view the present DS register with the **R** command, and set the CS register to the same value using the **RCS** command. For more information about using DEBUG, see the DOS documentation.

Chapter 8: Command Reference

Main Menu Commands

General Commands

F1	Help	Displays command summaries in the Help screen.
G	Go	Starts processing the input file or address range. Operates after you enter a valid file or address range.
Q	Quit	Quits Sourcer; available at all times while Sourcer is running.
Esc	Quit	Quits Sourcer; available at all times while Sourcer is running.

Output Formatting Commands

F	File format	Selects output file format: either listing or source code.
H	Header title	Lets you enter a title (up to 32 characters long) to display on each page of a listing file. A default title appears after you load an input file.

L	Label type	Selects a label type and shows an example of its format.
O	Output filename	Sets the output filename. The default is the input filename or temp (when there is no input file).
R	Remarks	Selects automatic comment types.
S	Segment on/off	Turns segment display on or off in the listing file.
T	Tabs on/off	Selects either tabs or spaces for use in the output file. Tabs reduce the required file space by half.
W	Word case	Specifies either uppercase or lowercase instructions and labels in the output file.
X	Xref on/off	Turns automatic cross-referencing on or off.

Processing Commands

A	Analysis Options	Displays Analysis Options Menu. (For a detailed description of each option, see page 17. For a summary of each option, see page 83.)
B	Beginning address	Lets you enter the address at which to begin processing.
C	Code style	Sets the code style.
D	Drive	Lets you specify the disk drive to use for output.
E	Ending address	Lets you enter the ending address of the first segment at which to stop processing.

I	Input filename	Lets you name an input file for processing.
M	Math	Turns the math co-processor filter on or off.
P	Passes	Sets the number of passes from 2 to 9.
U	micro-processor	Sets the microprocessor filter.

Analysis Options

Press A from the Main Menu to display the Analysis Options Menu. You turn options on and off by pressing the corresponding keyboard key. When the option letter appears in uppercase, the option is on. When the letter appears in lowercase, the option is off. This section gives the effects of turning on an option. For a complete description of each option, see page 17.

A	Instructs Sourcer to remain in code mode after a jump instruction instead of switching to data mode.
B	Instructs Sourcer to remain in code mode after a return instruction instead of switching to data mode.
C	Instructs Sourcer to remain in code mode after an interrupt return instead of switching to data mode.
D	Sets the DS and ES defaults to the code segment.
E	Instructs the code simulator to ignore changes to the DS and ES simulation registers.
F	Prevents Sourcer from detecting terminated subroutines.
G	Forces indexed data references 0h to 100h to equates.

H	Forces indexed data references 0h to 200h to equates.
I	Prevents Sourcer from using graphics characters in the output file.
J	Prevents Sourcer from detecting interrupt vector entry points into the program.
K	Prevents Sourcer from writing absolute location reference comments for jumps and calls.
L	Prevents Sourcer from using the instruction mnemonics RETF/RETN for return far and near. Turn on this option if you are planning to reassemble generated code with MASM version 4.0 or earlier.
M	Prevents Sourcer from fixing references that appear as 16-bit displacements in the binary file. Turn on this option if you are planning to reassemble generated code with MASM version 4.0 or earlier.
N	Prevents Sourcer from placing necessary ASSUME statements in code.
O	Turn on this option to use the index register (BX, BP, SI, or DI) as is for indexed jumps and calls. When option P is turned off, Sourcer ignores the index register value when it is less than 180h.
Q	Instructs Sourcer to cross-reference all data references.
R	Instructs Sourcer to cross-reference all location references.
S	Instructs Sourcer to cross-reference all subroutine references.
T	Prevents Sourcer from cross-referencing locations less that 128 bytes away from each other.
U	Prevents Sourcer from interpreting indexed data values -64h through +64h as offsets.

Prevents Sourcer from fixing near jumps.

Prevents Sourcer from adding warning remarks.

Allows Sourcer to create references during the final pass.

Chapter 9: Troubleshooting

This chapter suggests actions you should take to correct problems you encounter while running Sourcer. If you encounter a problem when reassembling generated code, see Chapter 7, "Reassembling Generated Source Code."

General Problems

This section describes problems that occur without Sourcer warning messages.

Cannot Access Sourcer
Check whether the file SR.EXE is in the current directory or on the current diskette.

System Locks Up While Processing A File
Check the microprocessor filter. The V20/30 filter can cause some 8086-based systems to halt. (See page 15.) Also, try removing other memory-resident programs.

Screen is Difficult to Read or the Color Changes
If you are using a CGA card with a composite monochrome monitor enter the following command at the DOS prompt:

```
mode bw80
```

If you are using a EGA or VGA adapter that is not fully IBM-compatible, start Sourcer with the -v command line option. (See page 30.)

If your system does not support one of the IBM display standards, start Sourcer with the -n command line option. (See page 30.)

Listing Is Difficult to Read
A listing file (.LST) is 132 columns wide, so you must view it with an editor or listing program that lets you view all 132 columns or scroll left and right. For a list of recommended programs, see Appendix D, "Other Useful Programs." For instructions on changing the format of the listing, see page 31. A source code (.ASM) file usually fits in 80 columns.

"Reassembly Not Recommended" Appears in Output File
The definition file used the multi-defined option. Code generated with this option will not reassemble correctly. To reassemble the file, remove multi-defined labels from the definition file and generate a new assembly file.

Sourcer Messages

This section lists each Sourcer message (in numerical order) and give suggestions for correcting problems.

Message 1: File does not have a binary extension type.
The input file must be a binary file with one of the following filename extensions: .COM, .DRV, .EXE, .SYS, .BIN, .OVR, .OVL, or .OV1 to .OV4. If the file is a binary file, but has a different extension, rename it so that it has a valid extension.

Message 2: WARNING: Output file already exists. Overwrite (Yes/No) ?
The specified output file already exists. Press **Y** to overwrite it, or press **N** to enter a new filename.

Message 3: WARNING: Packed file, conversion quality low.
The input file is packed, making jump references inaccurate, data areas compressed, and multiple segments unresolvable. Output results will generally be inaccurate.

DOS does not perform file unpacking. However, the packer program attaches an unpacking routine to the end of the packed file. There are a number of packer programs and there is no standard packing technique. Sourcer generates a START: label at the beginning of the unpack routine. However, when multiple-segment EXE programs are packed, all of the

relocation information is moved from the header to the unpacker routine which makes it impossible for Sourcer to determine how segments are separated.

If possible, use the unpacked version of the input file or convert the packed file to an unpacked EXE file. UNPACKER, available from V Communications, can convert most packed files into their original unpacked format.

Message 4: WARNING: Cross-reference table full, not all xrefs identified.

The cross reference table is full; references made later in the program are not included. To avoid filling the cross-reference table, use analysis options Q through T to select only a subset of all cross-references. For more information, see page 27.

If your system has 640 Kb of memory and has no RAM-resident programs and if the file to analyze is over 220 Kb, Sourcer reduces the size of the reserved cross-reference table to fit the file the available memory.

Message 5: WARNING: Keep function ignored and set to both.

The Keep option must occur before the Input filename option in a definition file. Sourcer ignores the Keep option specified in the definition file and sets its value to BOTH. BOTH is the appropriate value for most input files. (See page 38.)

Message 6: WARNING: Hidden EXE file, output quality low.

This message appears when a file with the extension .COM is really a multi-segment EXE file. Such files contain an EXE converter (at the end). DOS loads the file as a COM file. However, when the program starts, the EXE converter actually starts first to support the relocation information of the original EXE program. Use the EXE version or convert to EXE before using Sourcer. COM2EXE, included in the UNPACKER set of utilities, can convert these multi-segment COM files to EXE files.

Only a few IBM/Microsoft supplied programs use this multi-segment COM format. In DOS 3.30 the following programs are multi-segment COM files: CHKDSK, DEBUG, EDLIN, FDISK, FORMAT, GRAFTABL, GRAPHICS, PRINT, RECOVER, and SYS.

Message 10: Symbol table full.
Sourcer reserves 192 Kb for symbol space and uses variable-length records for maximum compression. To allow more entries in the tables, you can

- Set the Label type to seg_offset or ltrseg_off (Press **L** from the Main Menu).
- Move comments in section three of the definition file into a remarks file. (See Chapter 6, "Adding Comments to the Output File.")
- Eliminate entries in section three of the definition file.
- If possible, use a smaller file or address range.
- Add the following line to your definition file:

  ```
  Just skip relocations = 1
  ```

  This line instructs Sourcer to skip loading EXE relocation entries to save symbol space in the location table. It must appear in the definition file before the Input filename option. A Sourcer-generated definition file (SDF file) never includes this option.

Message 11: Disk full.
Clear sufficient disk space for the output file or select an option that uses less space (for example, output tabs instead of spaces). The output file is correct up to the point where the disk became full. You can also send output directly to a printer. For instructions, see page 32.

Message 12: Output file disk operation failed.
An error occurred while Sourcer was writing the output file. Check that the disk is operating properly. If you directed the Sourcer output to a diskette, you must not remove the diskette while Sourcer is running.

Message 13: Sourcer CANCELLED - No files created or processed
Appears when you press the **Q** or **Esc** key to quit Sourcer.

Message 14: Sourcer CANCELLED - Partial file processed.
See message 13. Sourcer may have created part of the output file.

Message 15: File not found or path is invalid.
Retry using a valid filename and/or path.

Message 16: File access denied.
The specified file is not accessible. Retry with an accessible file.

Message 17: Insufficient memory to load the file.
To provide additional memory, you can:

- Remove any RAM-resident programs.
- Add main memory if your system has less than 640 Kb.
- Reduce the input program size (usually difficult to do).
- Set the CONFIG.SYS file statements FILES and BUFFERS to lower numbers.
- Switch to an earlier version of DOS (saves up to 50 Kb).

Only EXE type files can extend beyond the 64 Kb segment boundary. Files with other extensions generate this error message if the file size is greater than 65,534 bytes.

Sourcer uses DOS to load the file. In some cases it may be possible to add 64 Kb of additional memory from A000h to AFFFh if unused by the video adapter (CGA, MDA or Hercules). To address this area, you usually need a special memory card.

Some files store data in overlays at the end of the executable code. In such cases, amount of memory required to load the file may be less than the total file size reported by DOS.

In some cases, you can reduce the size of an EXE file to fit into memory. Copy the file and use DEBUG to reduce the size of the duplicate. The following example shows how to change a 375 Kb EXE file (MYFILE.EXE) to a 200 Kb EXE file (NEWFILE.EXE). This method works very well if the end of the EXE file is simply buffer space used by the program but not directly referenced. In some cases, however, the resulting program can be unclear.

Example 9-1: Reducing File Size

```
Original file size:   375,776 bytes  = 5BBE0h  =      2DEh   * 200h + 1E0h  (EXE header format)
Planned file size:    200,000 bytes  = 30D40h  =      186h   * 200h + 13Fh  (EXE header format)
```

1. Copy the EXE file (MYFILE.EXE) to a new file (NEWFLE.TMP).

   ```
   COPY MYFILE.EXE NEWFILE.TMP
   ```

2. Start DEBUG with the new file loaded.

   ```
   DEBUG NEWFILE.TMP
   ```

3. Use the -r command to display the registers.

   ```
   AX=0000   BX=0005   CX=BBE0   DX=0000 SP=E12B BP=0100   SI=0000   DI=0000
   DS=1DC4   ES=1DC4   SS=1DC4   CS=1DC4   IP=0100 NV UP EI PL NZ NA P0 NC
   1DC4:0100 4D              DEC   BP
   ```

4. The BX register contains the upper 16 bits of the file size. Change it to 3 (for a new size of 30D40h).

   ```
   -rbx
   BX 0005
   :3
   ```

5. The CX register contains the lower 16 bits of the file size. Change it to D40h.

   ```
   -rcx
   CX BBE0
   :d40
   ```

6. Display the EXE file header.

   ```
   -d 100 L6
   1DC4:0100 4D 5A E0 01 DE 02
   ```

   The word at offset 102h (01E0h) is the file size remainder. The word at offset 104h (02DEh) is the file size divided by 200h.

7. Change the remainder to 13Fh.

   ```
   -e 102
   1DC4:0102 0F.3F 01.1
   ```

8. Change the file size divided by 200h to 786h.

   ```
   -e 104
   1DC4:0104 DE.86 02.1
   ```

9. Write the file to disk.

   ```
   -w
   Writing 30D40 bytes
   ```

10. Quit DEBUG.

    ```
    -q
    ```

11. Rename the new file so it has a .EXE extension.

    ```
    RENAME NEWFILE.TMP NEWFILE.EXE
    ```

This method works only to shorten files (you must keep the first part and discard the last part), because the relocation information is not easily adjustable.

Message 18: Unable to load file.
Retry with a valid file.

Message 19: Cannot open output file.
Output file may already exist and is set to read only.

Message 20: Not a valid EXE file header.
The EXE file header is corrupt and cannot be processed. View the EXE header with the DOS EXEMOD utility.

Message 21: Table error, contact customer service
Sourcer detected an internal program error. Contact V Communications Customer Service.

Message 22: Internal fatal error
Sourcer detected an internal program error. Contact V Communications Customer Service.

Message 23: Bad opcode processed: *xx*h at *segment:offset*. Press any key.
Sourcer encountered an internal error. Note the opcode and contact V Communications Customer Service.

Message 24: Invalid begin segment value in def file at line number *xx*.
The value of the Begin segment control option must be greater than or equal to the value of the location at which Sourcer would normally load the file. The number you entered is either too small or not a valid hex number. (See page 37.)

Message 31: Incorrect entry in definition file at line number *XXXX*
The loaded definition file has an error, such as a non-hex digit in a number or an incorrectly formatted line. Check the definition file.

Message 32: Definition file disk operation failed.
Sourcer could not read the definition file. Check that the definition file is valid.

Message 33: Cannot open definition file.
Sourcer could not open the specified definition file. Check whether the file exists in the specified subdirectory (or the default subdirectory).

Message 34: WARNING Could not find or load .rem file.
The remarks file associated with the definition file of the same filename could not be found. The remarks file should be in the same subdirectory or disk as the definition file. If not supplied, Sourcer can proceed without it, but special remarks and comments will not appear in the output.

Message 35: Cannot open the remark file.
Sourcer found the remarks file, but could not open it. Check whether the remarks file is in the same directory or on the same diskette as the specified definitions file.

Message 36: Incorrect entry in remark file at line number *XXXX*.
The entry on the specified line does not follow the standard format and cannot be processed. Entries must begin with a hex number, there must be no tabs in entries, and each section must have an END statement.

Message 37: Too many remarks in remark file. Stopped at line number *XXXX*.
See Chapter 6, "Adding Comments to the Output File," for the maximum number of comments allowed in each remarks file section.

Message 39: Too many data entries created in definition file.
Reduce the number of data entries, reduce the number of times a structure repeats, and/or move any comments from within the definition file into a remarks file.

Appendix A: About Sourcer

How Sourcer Works

When you press **G** to begin processing a file, Sourcer loads the input file into memory and determines the appropriate segment ranges. During the first pass, it identifies most of the location, data, and subroutine references. Sourcer assumes that both data and code areas are code until proven otherwise and determines whether duplicate bytes should be considered data.

Sourcer sorts symbol tables before the start of each successive pass and removes invalid references flagged during the previous pass. Each subsequent pass analyzes the code and data references to resolve the data and code areas. The final pass determines necessary assembler directives, formats each line, and determines appropriate comments.

The internal code simulator keeps track of all registers and maintains a separate stack for the program. The simulator insures that proper data segments are specified when more than one is in use. The simulator also keeps track of detailed comments, identifies I/O ports, and resolves indexed calls and jumps.

Simulation proceeds linearly through the code. The simulator ignores jumps and skips instructions that attempt to write into memory. However, it processes instructions that read from memory. Special handling of the CS register insures full support for simulation of ROM and RAM memory locations.

When it encounters a subroutine call, Sourcer makes a snapshot of the simulator registers and saves it with the subroutine. When it returns to that subroutine, Sourcer restores the saved registers as if the subroutine were called with the proper regis-

ter values. Sourcer uses a similar process to save the values in the ES and DS segment registers when a subroutine exits.

Files on the Sourcer Distribution Disk

File name	Description
README.BAT	Batch file that starts up the Sourcer information program. The information program installs Sourcer, explains Sourcer basics, and gives corrections and additions to the information in this manual.
SR.EXE	Sourcer program file.
SDEFAULT.DEF	Definition file that sets system defaults.
TESTYN.EXE	Sample program file.
TESTYN.DEF	Definition file for the sample program TESTYN; adds descriptive labels and links remarks in the file TESTYN.REM.
TESTYN.REM	Remarks file for the sample program TESTYN.
SAMPLE1.DEF	Blank definition file with detailed comments on commands and file format; use a copy of this file to create your own definition file.
SAMPLE2.DEF	Blank definition file without the instructional comments in SAMPLE1.DEF
INFOSR.COM	Sourcer information program. Use README.BAT to run.

Hardware Requirements

Sourcer is compatible with the IBM PC, XT, AT, and PS/2 computers, all systems that are 100% IBM compatible, and many not-so-compatible systems that run PC-DOS or MS-DOS. This section provides specific hardware requirements.

RAM

To run Sourcer, your system must have at least 448 Kb of RAM. You need additional memory to load a file for processing. For example, a 40 Kb file requires an additional 40 Kb of memory.

Display

Sourcer supports all standard displays, including MDA, CGA, EGA, VGA, and Hercules (in MDA mode). If your display does not support one of these standard, start Sourcer with the -n option to use DOS for all video functions:

```
sr -n
```

If you are using a monochrome composite monitor with a CGA card, run the DOS mode program at the DOS prompt to set monochrome operation:

```
mode bw80
```

Disk Drives

You do not need a hard disk to process smaller files. If you have limited disk storage (for example, no hard disk) be sure to use tabs in the Sourcer output file instead of spaces. (A file using spaces can be double the size of one using tabs.). When a disk becomes full, Sourcer stops processing, closes the file, and informs you that the disk is full. You can also send output directly to a printer. For instructions, see page 32.

Sourcer Specifications

Sourcer Memory:

Under 400 Kb (does not include DOS).

Computers Supported:

IBM PC, XT, AT, and PS/2; IBM-compatible systems; and most systems that can run PC-DOS or MS-DOS versions 2.0 through 4.

Instruction Sets:

8088/8086, 8087, V20/V30, 80186/80188, 80286, 80287, 80386, 80387, and 80486. Supports the 80286, 80386, and 80486 in real and protected modes.

Input File Types:
Binary files, COM and BIN type files, multi-segment files, EXE type files (up to 250 segments, unpacked), device drivers, program overlays, system files, SYS type files, any RAM or ROM address within the first 1 Mb of memory, multiple segments.

Input File Size:
Up to 260 Kb (on a system with 640 Kb RAM, no RAM-resident programs, and DOS 3.0).

Output:
Listing or source code file and generated definition file.

Symbol Tables:
Separate, RAM-based symbol tables for data, locations, subroutines, and segments.

Symbol Table Space:
192 Kb reserved within Sourcer.

On-line Help:
One screen.

Error Handling:
Errors identified by number and description. Manual contains suggested actions for each message.

Other Notes:
Single user license with no copy protection.

Feature Summary

Inputs

- Disassembles binary, EXE, COM, SYS, BIN, device driver, overlay, and multi-segment files.
- Handles up to 250 segments.
- Resolves most segment overlap packing from C compilers.

- Disassembles any RAM or ROM address in first 1 Mb of memory.

Output

- Source code or listing.
- Uppercase or lowercase.
- Selectable comments.
- Selectable label styles.
- Output sent to disk or printer.

Code Support

- Selectable instruction sets: 8088, 80186, 80286, 80386, 80486, & V20/V30.
- Math co-processor instruction set support: 8087, 80287, and 80387.
- Unique labels for location references, data references, sub-routine references, segments, equates, procedures, loops, interrupt entries, and other items.

Assembler Directives

- Inserts assembler directives as required for reassembly.
- Supports SEGMENT, ENDS, PROC, ENDP, END, ASSUME, PAGE, ORG, EQU, Processor type, PTR overrides, and other directives.
- Determines data types, bytes, words, double words, text strings, duplicate bytes and resolves overlaps.

Comments

- All infrequently used and complicated instructions.
- Interrupts including BIOS calls, DOS calls, EMS 4.0, and most common adapter cards for PC and PS/2 machines.
- Input and output port numbers and functions.
- Segment, offset, and value for referenced data item.
- Program flow.

Special Features

- Output appears in the same style as original assembly source code.
- Internal program simulation resolves data items across segment boundaries.
- Automatically analyzes indexed jump and call instructions.
- Definition file facility allows adding comments and descriptive labels, setting defaults, specifying segments, and fine tuning results.
- Finds and identifies TSR entry points.
- Cross-references.
- Uses correct mnemonic for multiple-mnemonic instructions.
- Automatically detects device drivers and finds entry points.
- Automatically corrects for assembler limitations.
- Displays progress on screen.
- Provides descriptive labels for commonly referenced BIOS data items.

Compatibility with Earlier Versions

If you have been using an earlier version of Sourcer, there are several changes you should note.

Version 1.72 and Earlier

If you have been using version 1.72 of Sourcer, you may need to modify your definition files. This section describes changes that affect definition files.

Control Information: Section One

Analysis options M and N have different functions. In definition files, use the Math on control option instead of the M option. (See page 39.) The earlier function of the N option is no longer needed, because Sourcer automatically uses a remarks file whenever remarks file comments are specified in the definition file.

The functions of analysis options K and L are now the opposite of their functions in earlier versions of Sourcer. By default,

option K is off and the Sourcer listing contains comments for jumps and calls. Turn on option K to prevent Sourcer from listing these comments. (See page 23.) Option L is also off by default, and Sourcer uses the RETF (return far) and RETN (return near) instructions. Turn on this option to prevent Sourcer from using these instructions if you are using MASM 4.0 or an earlier assembler that does not support them. (See page 23.)

Note: If the code you are disassembling was originally processed by an optimizing compiler or assembler (such as MASM 5.0 or later) and you set Sourcer analysis options for compatibility with MASM 4.0, Sourcer cannot generate code that is identical to the original. If you normally use MASM 4.0 to reassemble generated code, you might consider upgrading to MASM 5.0 or later.

The Keep statements statement is no longer necessary. By default, Sourcer compares segment table entries loaded from the input file with range definitions in section two of the definition file. If there are any differences, it uses the definition file entries. (See page 38.)

The Lower case off option has been changed to Word upper. (See page 42.)

The Remarks option has seven options for turning on and off automatic comments. (See page 40.)

Range Definitions: Section Two

Sourcer now supports up to 250 segment variable names. Output files use full variable names instead of abbreviations. For more information, see page 43.

Reference Definitions: Section Three

You can now specify comments to be included in output files as part of a reference definition entry. Comments must appear in the last column of the entry. A comment preceded with a semicolon (;) appears only in the definition file. Otherwise, comments appear in output files. If there are comments for an item in both the reference definition and the remarks file, Sourcer uses the comment in the reference definition. For more information, see Chapter 6, "Adding Comments to the Output File."

The EXT (external) location option has changed. It resets the simulator and produces an external entry comment. However, it no longer generates a far procedure. Use EXT for external entry points only. To generate a far procedure, use the FAR option. For more information, see page 49.

Version 1.9 and Earlier

If you have been using version 1.9 or an earlier version of Sourcer, you should note the following changes:

- Sourcer now supports the 80386 and 80486 instruction sets and the 80387 math instruction set. (See page 15.)
- Each time you process a file, Sourcer automatically generates a definition file containing command and option settings, segment range definitions, and all locations, subroutines, and other items it identified. This file has the output filename and the extension .SDF. Use it as a reference when you read a listing, or use it as the basis of your own definition file. (See page 35.)
- There are several new definition file features:
  - You can specify 16- or 32-bit segment ranges. (See page 45.)
  - The DA (ASCII data) item type can automatically determine text string length. (See page 47.)
  - Labels are optional.
  - Definition files can use full segment variable names. (See page 43.) The following table compares full variable names with the old-style abbreviations. The abbreviations are still supported.

Segment Variable Names	Abbreviation
seg_a to seg_z	sega to segz
seg_aa to seg_az	sgaa to sgaz
seg_ba to seg_bz	sgba to sgbz
.	
.	
.	
seg_ia to seg_iz	sgia to sgiz

  - You can specify a memory location at which to load the input file. (See page 37.)
  - Fixed field formats are eliminated.

- Sourcer now analyzes code in two to nine passes. (See page 15.)
- Sourcer now supports displays that are not IBM compatible. (See page 30.)
- You can set Sourcer command line options in the DOS environment. (See page 30.)
- Sourcer is Desqview compatible.

Appendix B: Product Support and Update Policy

Customer Support

If you have a problem with Sourcer, gather the following information before contacting V Communications Customer Service:

- Program version, and serial number. Both appear on the diskette label and in the upper right corner of the screen when you are running Sourcer.
- Computer name and model number.
- Printed listing (if the problem is in an output file).

You can contact V Communications Customer Service at:

V Communications, Inc.
3031 Tisch Way, Suite 802
San Jose, CA 95128
(408) 296-4224

Product Update Policy

Registered Sourcer users can purchase a software update for major new releases for a nominal fee. The update includes a new program diskette and manual. Contact V Communications Customer Service for update information and fees.

Appendix C: Using 80386 and 80486 Instruction Sets

The 80386 and 80486 instruction sets have both 16- and 32-bit forms of many instructions. In addition, the segment size attribute of a code segment can be 16 bits (in real or protected mode) or 32 bits (in protected mode only). When you use protected mode, Sourcer must be able to determine the segment size attribute in order to correctly disassemble a file. For example, the following instruction is coded differently depending on the mode of operation:

```
mov ax, 1234h
```

When the segment size attribute is 16 bits, the instruction moves the value 1234h to the AX register. However, when the segment size attribute is 32 bits, the instruction moves the value *xxxx*1234h to the EAX register, where *xxxx* is the 16-bits following 1234h. In both cases, the opcodes are the same. To properly disassemble the coded instruction, Sourcer must know the segment size attribute.

There are three ways Sourcer can determine the segment size attribute:

- In some cases, Sourcer can automatically determine the segment size attribute. The BIOS interrupt 15h, service 89h is the most common method of switching between 16- and 32-bit modes. In most cases, Sourcer can use this interrupt to determine the segment size attribute.
- You can specify 16- and 32-bit segment ranges in the range definition section of a definition file. (See page 45.)
- You can use the CODE16 and CODE32 forced function options in the reference definition section of a definition file to force Sourcer to switch between 16- and 32-bit modes. (See page 50.)

The 80386 and 80486 instruction sets are very complex, and you may require additional information to determine when a program switches modes. It can take experienced assembly-language programmers months of full-time work to fully understand the subtle complexities of these microprocessor instruction sets. For sources of information about these microprocessors, see Appendix E, "Reference Materials."

Appendix D: Other Useful Programs

This appendix contains a list of other programs you might find helpful while using Sourcer. You may find other programs, not listed here, to be helpful as well.

File Viewers and Editors

The following table contains a list of file viewers and editors that support 132-column (or longer) lines.

Program	Publisher	Notes
InView	V Communications 3031 Tisch Way, Suite 802 San Jose, CA 95128 800-662-8266	File viewer. Very fast. Capable of displaying full 132 columns.
Brief	Solution Systems 541 Main St., Suite 541 S. Weymouth, MA 02190 (617) 337-6963	Editor. Displays 80 columns. Allows left and right scrolling.
Multi-Edit	American Cybernetics, Inc. 1228 N. Stadem Dr. Tempe, AZ 85281 (602) 968-1945 inside AZ 800-221-9280 outside AZ	Editor. Displays 80 columns. Allows left and right scrolling
PC-Write	Quicksoft 219 First N. #224 Seattle, WA 98109 (206) 282-0452	Editor. Shareware. Displays 80 columns. Allows left and right scrolling.

Assemblers

Program	Publisher	Notes
Macro Assembler/2	IBM Corporation Contact IBM Authorized Dealer	Not tested
MASM 5.1	Microsoft 16011 Northeast 36th Way Redmond, WA 98073 800-426-9400	Good documentation
OPTASM	SLR Systems 1622 N. Main St. Butler, PA 16001 800-833-3061	Very fast, no phase errors
TASM	Borland 1800 Green Hills Rd. Scotts Valley, CA 95066 800-331-0877	Very fast, no phase errors

Other Useful Products

Program	Publisher	Notes
ASMtool	V Communications 3031 Tisch Way, Suite 802 San Jose, CA 95128 800-662-8266	Assembly program analyzer and flowcharter
UNPACKER	V Communications 3031 Tisch Way, Suite 802 San Jose, CA 95128 800-662-8266	Assembly programming utilities (UNPACK, etc.)

Appendix E: Reference Materials

Assembly Language

Jourdain, Robert. *Programmer's Problem Solver for the IBM PC, XT & AT*. New York: Prentice Hall Press, 1986.

Lafore, Robert. *Assembly Language Primer For the IBM PC & XT*. New York: New American Library, 1984.

Miller, Alan R. *Assembly Language Techniques for the IBM PC*. Berkeley, CA: Sybex Inc., 1986.

BIOS

Personal System/2 and Personal Computer BIOS Interface Technical Reference. IBM Corporation, 1988. To order contact IBM Corporation, PO Box 2009, Racine, WI 53404; 800-465-1234. (Order number 68X2341).

DOS and Assembly-language Programming

Angermeyer, John and Kevin Jaeger. *MS-DOS Developer's Guide*. Indianapolis, IN: Howard W. Sams & Co. 1986.

Duncan, Ray. *Advanced MS-DOS Programming*. Redmond, WA: Microsoft Press, 1988.

Duncan, Ray. *MS-DOS Encyclopedia*. Redmond, WA: Microsoft Press, 1988.

Instruction Sets and Microprocessors

iAPX 86/88, 186/188 User's Manual. Intel Corporation, 1986. To order contact Intel Corporation, PO Box 58130, Santa Clara, CA 95052; 800- 548-4725. (Order number 210911-003).

80286 and 80287 Programmer's Reference Manual. Intel Corporation, 1987. To order contact Intel Corporation, PO Box 58130, Santa Clara, CA 95052; 800-548-4725. (order number 210498-004)

80386 Programmer's Reference Manual. Intel Corporation, 1986. To order contact Intel Corporation, PO Box 58130, Santa Clara, CA 95052; 800-548-4725. (order number 230985-001)

Intel i486 Microprocessor. Intel Corporation, 1989. To order contact Intel Corporation, PO Box 58130, Santa Clara, CA 95052; 800-548-4725. (order number 240440-001)

Video Systems

Wilton, Richard. *Programmer's Guide to PC & PS/2 Video Systems.* Redmond, WA: Microsoft Press, 1987.

Appendix F: Reference Tables

This appendix contains opcode decoder and character set tables.

Opcode Decoder Table

LOW

HI	0	1	2	3	4	5	6	7
0	add mb=mb+rb	add mw=mw+rw	add rb=rb+rmb	add rw=rw+rmw	add al=al+ib	add ax=ax+lw	push es	pop es
1	adc mb=mb+rb	adc mw=mw+rw	adc rb=rb+rmb	adc rw=rw+rmw	adc al=al+ib	adc ax=ax+iw	push ss	pop ss
2	and mb=mb,rb	and mw=mw,rw	and rb=rb,rmb	and rw=rw,rmw	and al=al,ib	and ax=ax,iw	es:	daa
3	xor mb=mb,rb	xor mw=mw,rw	xor rb=rb,rmb	xor rw=rw,rmw	xor al=al,ib	xor ax=ax,iw	ss:	aaa
4	inc ax	inc cx	inc dx	inc bx	inc sp	inc bp	inc si	inc di
5	push ax	push cx	push dx	push bx	push sp	push bp	push si	push di
6	pusha[1]	popa[1]	bound[1] rw md	arpl[2]	fs: ----[3] repnc --[v]	gs: ----[3] repc --[v]	Operand Size	Address Size
7	jo	jno	jb	jae	je / jz	jne / jnz	jbe	ja
8	Table S rmb=rmb,ib	Table S rmw=rmw,iw	Table S rmb=rmb,ib	Table S rmw=rmw,ib	test flg=rb,rmb	test flg=rw,rmw	xchg rmb,rmb	xchg rmw,rmw
9	nop	xchg cx,ax	xchg dx,ax	xchg bx,ax	xchg sp,ax	xchg bp,ax	xchg si,ax	xchg di,ax
A	mov al=mb	mov ax=mw	mov mb=al	mov mw=ax	movsb	movsw	cmpsb	cmpsw
B	mov al=ib	mov cl=ib	mov dl=ib	mov bl=ib	mov ah=ib	mov ch=ib	mov dh=ib	mov bh=ib
C	Table T[1] rmb,ib	Table T[1] rmw,ib	retn sp=sp+iw	retn	les	lds	mov rmb=ib	mov rmw=iw
D	Table T rmb,1	Table T rmw,1	Table T rmb,cl	Table T rmw,cl	aam	aad		xlat
E	loopnz	loopz	loop	jcxz	in al,ib	in ax,ib	out ib,al	out ib,ax
F	lock repne		repnz / / repz	rep / repe	hlt	cmc	Table U rmb	Table U rmw

Codes

mb = memory, byte
mw = memory, word
md = memory, double word
mq= memory, quad word
rb = register, byte
rw = register, word
rd = register, double word
rmb = register or memory, byte
rmw = register or memory, word
ib = immediate, byte
iw = immediate, word
flg= flags

LOW

HI	8	9	A	B	C	D	E	F
0	or mb=mb,rb	or mw=mw,rw	or rb=rb,rmb	or rw=rw,rmw	or al=al,ib	or ax=ax,iw	push cs	Table W[2]
1	sbb mb=mb-rb	sbb mw=mw-rw	sbb rb=rb-rmb	sbb rw=rw-rmw	sbb al=al-ib	sbb ax=ax-iw	push ds	pop ds
2	sub mb=mb-rb	sub mw=mw-rw	sub rb=rb-rmb	sub rw=rw-rmw	sub al=al-ib	sub ax=ax-iw	cs:	das
3	cmp mb=mb,rb	cmp mw=mw,rw	cmp rb=rb,rmb	cmp rw=rw,rmw	cmp al=al,ib	cmp ax=ax,iw	ds:	aas
4	dec ax	dec cx	dec dx	dec bx	dec sp	dec bp	dec si	dec di
5	pop ax	pop cx	pop dx	pop bx	pop sp	pop bp	pop si	pop di
6	push[1] iw	imul[1] rw=rmw*iw	push[1] ib	imul[1] rw=rmw*ib	insb[1]	insw[1]	outsb[1]	outsw[1]
7	js	jns	jp	jnp	jl	jge	jle	jg
8	mov rmb=rb	mov rmw=rw	mov rb=mb	mov rw=mw	mov rmw=rseg	lea flg=rw,rmw	mov rseg=rmw	pop mw
9	cbw	cwd	call far	wait	pushf	popf	sahf flg=ah	lahf ah=flg
A	test flg=al,ib	test flg=ax,iw	stosb	stosw	lodsb	lodsw	scasb	scasw
B	mov ax=iw	mov cw=iw	mov dw=iw	mov bw=iw	mov sp=iw	mov bp=iw	mov si=iw	mov di=iw
C	enter[1] iw,ib	leave[1]	retf sp=sp+iw	retf	int 3	int ib	into	iret
D	esc 0	esc 1	esc 2	esc 3	esc 4	esc 5	esc 6	esc 7
E	call near	jmp near	jmp far	jmp short al,dx	in ax,dx	in dx,al	out dx,al	out
F	clc	stc	cli	sti	cld	std	Table V rmb	Table V rmw

Notes

v V20/V30 only
1 V20/V30 & 80186 through 80486
2 80286 through 80486 only
3 80386 & 80486 only
4 80486 only

Tables S through V

3 bits from reg field, 2nd Opcode (md-reg-r/m)

Tbl	000	001	010	011	100	101	110	111
S	add	or	adc	sbb	and	sub	xor	cmp
T	rol	ror	rcl	rcr	shl / sal	shr	shl/sal[1]	sar
U	test	not	neg	mul	imul	div	idiv	
V	inc	dec	call near	call far	jmp near	jmp far		

Notes

v V20/V30 only
1 V20/V30 & 80186 through 80486
2 80286 through 80486 only
3 80386 & 80486 only
4 80486 only

Table W: 80286 through 80486 Multi-Byte Opcode Instructions (First Opcode is 0Fh)

HI \ LOW	0	1	2	3	4	5	6	7
0	Table X[2]	Table Y[2]	lar[2] rw,rmw	lsl[2] rw,rmw			clts[2]	
1								
2								
3								
4								
5								
6								
7								
8	jo[3]	jno[3]	jb[3]	jnb[3]	jz[3]	jnz[3]	jbe[3]	jnbe[3]
9	seto[3] rmb	setno[3] rmb	setb[3] rmb	setnb[3] rmb	setz[3] rmb	setnz[3] rmb	setbe[3] rmb	setnbe[3] rmb
A	push[3] fs	pop[3] fs		bt[3] rmw,rw	shld[3] rmw,rw,ib	shld[3] rmw,rw,cl		
B			lss[3] rw,mw	btr[3] rmw,rw	lfs[3] rw,mw	lgs[3] rw,mw	movzx[3] rw,rmb	movzx[3] rd,rmw
C	xadd[4] rmb,rb	xadd[4] rmw,rw					cmpxchg[4] rmb,rb	cmpxchg[4] rmw,rw
D								
E								
F								

LOW

HI	8	9	A	B	C	D	E	F
0	invd[4]	wbinvd[4]						
1								
2								
3								
4								
5								
6								
7								
8	js[3]	jns[3]	jp[3]	jnp[3]	jl[3]	jnl[3]	jle[3]	jnle[3]
9	sets[3] rmb	setns[3] rmb	setp[3] rmb	setnp[3] rmb	setl[3] rmb	setnl[3] rmb	setle[3] rmb	setnle[3] rmb
A	push[3] gs	pop[3] gs		bts[3] rmw,rw	shrd[3] rmw,rw,ib	shrd[3] rmw,rw,cl		imul[3] rw,rmw
B			Table Z[3]	btc[3] rmw,rw	bsf[3] rw,rmw	bsr[3] rw,rmw	movsx[3] rw,rmb	movsx[3] rw,rmb
C	bswap[4] ax	bswap[4] cx	bswap[4] dx	bswap[4] bx	bswap[4] sp	bswap[4] bp	bswap[4] si	bswap[4] di
D								
E								
F								

Notes

v V20/V30 only
1 V20/V30 & 80186 through 80486
2 80286 through 80486 only
3 80386 & 80486 only
4 80486 only

Tables X through Z

Bits 5, 4, 3 from reg field, 3rd Opcode (md-reg-r/m)

	000	001	010	011	100	101	110	111
X	sldt[2] rmw	str[2] rmw	lldt[2] rmw	ltr[2] rmw	verr[2] rmw	verw[2] rmw		
Y	sgdt[2] mq	sidt[2] mq	lgdt[2] mq	lidt[2] mq	smsw[2] rmw	invlpg[4] m	lmsw[2] rmw	
Z					bt[3] rmw,ib	bts[3] rmw,ib	btr[3] rmw,ib	btc[3] rmw,ib

Notes

v V20/V30 only
1 V20/V30 & 80186 through 80486
2 80286 through 80486 only
3 80386 & 80486 only
4 80486 only

Character Set Table

Hex	Dec	Control Codes	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec
0	000	NUL ^@	20	032	40	064 @	60	096 `	80	128 Ç	A0	160 á	C0	192 └	E0	224 α
1	001	SOH ^A	21	033 !	41	065 A	61	097 a	81	129 ü	A1	161 í	C1	193 ┴	E1	225 β
2	002	STX ^B	22	034 "	42	066 B	62	098 b	82	130 é	A2	162 ó	C2	194 ┬	E2	226 Γ
3	003	ETX ^C	23	035 #	43	067 C	63	099 c	83	131 â	A3	163 ú	C3	195 ├	E3	227 π
4	004	EOT ^D	24	036 $	44	068 D	64	100 d	84	132 ä	A4	164 ñ	C4	196 —	E4	228 Σ
5	005	ENQ ^E	25	037 %	45	069 E	65	101 e	85	133 à	A5	165 Ñ	C5	197 ┼	E5	229 σ
6	006	ACK ^F	26	038 &	46	070 F	66	102 f	86	134 å	A6	166 ª	C6	198 ╞	E6	230 µ
7	007	DEL ^G	27	039 '	47	071 G	67	103 g	87	135 ç	A7	167 º	C7	199 ╟	E7	231 τ
8	008	BS ^H	28	040 (	48	072 H	68	104 h	88	136 ê	A8	168 ¿	C8	200 ╚	E8	232 Φ
9	009	HT ^I	29	041)	49	073 I	69	105 i	89	137 ë	A9	169 ⌐	C9	201 ╔	E9	233 θ
A	010	LF ^J	2A	042 *	4A	074 J	6A	106 j	8A	138 è	AA	170 ¬	CA	202 ╩	EA	234 Ω
B	011	VT ^K	2B	043 +	4B	075 K	6B	107 k	8B	139 ï	AB	171 ½	CB	203 ╦	EB	235 δ
C	012	FF ^L	2C	044 ,	4C	076 L	6C	108 l	8C	140 î	AC	172 ¼	CC	204 ╠	EC	236 ∞
D	013	CR ^M	2D	045 -	4D	077 M	6D	109 m	8D	141 ì	AD	173 ¡	CD	205 ═	ED	237 φ
E	014	SO ^N	2E	046 .	4E	078 N	6E	110 n	8E	142 Ä	AE	174 «	CE	206 ╬	EE	238 ∈
F	015	SI ^O	2F	047 /	4F	079 O	6F	111 o	8F	143 Å	AF	175 »	CF	207 ╧	EF	239 ∩
10	016	DLE ^P	30	048 0	50	080 P	70	112 p	90	144 É	B0	176 ⁞	D0	208 ╨	F0	240 ≡
11	017	DC1 ^Q	31	049 1	51	081 Q	71	113 q	91	145 æ	B1	177 ▒	D1	209 ╤	F1	241 ±
12	018	DC2 ^R	32	050 2	52	082 R	72	114 r	92	146 Æ	B2	178 ▓	D2	210 ╥	F2	242 ≥
13	019	DC3 ^S	33	051 3	53	083 S	73	115 s	93	147 ô	B3	179 │	D3	211 ╙	F3	243 ≤
14	020	DC4 ^T	34	052 4	54	084 T	74	116 t	94	148 ö	B4	180 ┤	D4	212 ╘	F4	244 ⌠
15	021	NAK ^U	35	053 5	55	085 U	75	117 u	95	149 ò	B5	181 ╡	D5	213 ╒	F5	245 ⌡
16	022	SYN ^V	36	054 6	56	086 V	76	118 v	96	150 û	B6	182 ╢	D6	214 ╓	F6	246 ÷
17	023	ETB ^W	37	055 7	57	087 W	77	119 w	97	151 ù	B7	183 ╖	D7	215 ╫	F7	247 ≈
18	024	CAN ^X	38	056 8	58	088 X	78	120 x	98	152 ÿ	B8	184 ╕	D8	216 ╪	F8	248 °
19	025	EM ^Y	39	057 9	59	089 Y	79	121 y	99	153 Ö	B9	185 ╣	D9	217 ┘	F9	249 ∙
1A	026	SUB ^Z	3A	058 :	5A	090 Z	7A	122 z	9A	154 Ü	BA	186 ║	DA	218 ┌	FA	250
1B	027	ESC ^[	3B	059 ;	5B	091 [	7B	123 {	9B	155 ¢	BB	187 ╗	DB	219 ▉	FB	251 √
1C	028	FS ^\	3C	060 <	5C	092 \	7C	124 ¦	9C	156 £	BC	188 ╝	DC	220 ▄	FC	252 ⁿ
1D	029	GS ^]	3D	061 =	5D	093]	7D	125 }	9D	157 ¥	BD	189 ╜	DD	221 ▌	FD	253 ²
1E	030	RS ^^	3E	062 >	5E	094 ^	7E	126 ~	9E	158 ₧	BE	190 ╛	DE	222 ▐	FE	254 ■
1F	031	US ^_	3F	063 ?	5F	095 _	7F	127	9F	159 ƒ	BF	191 ┐	DF	223 ▀	FF	255

Glossary of Abbreviations

A/D	Analog to Digital
b&w	Black and White (display)
BIOS	Basic Input/Output System
bps	Bits Per Second
CGA	Color Graphics Adapter
CTS	Clear To Send (modem control)
D/A	Digital to Analog
DMA	Direct Memory Access (may refer to DMA IC)
DSR	Data Set Ready (modem control)
DTA	Disk Transfer Address (software pointer)
DTR	Data Terminal Ready (modem control)
ECC	Error Checking and Correction
EGA	Enhanced Graphics Adapter
EMS	Expanded Memory Specification
EPROM	Erasable Programmable Read Only Memory
FCB	File Control Block (data describing a file)
I/O	Input/Output
IC	Integrated Circuit
IMR	Interrupt Mask Register
int	Interrupt
IRQ	Interrupt Request

LAN	Local Area Network
LED	Light-Emmitting Diode
MDA	Monochrome Display Adapter
NMI	Non-Maskable Interrupt (interrupt 2)
PEL	Pixel (display)
PSP	Program Segment Prefix (program header for DOS)
POS	Programable Option Select (micro-channel)
POST	Power On Self Test
RAM	Random Access Memory
ROM	Read Only Memory (usually EPROMS)
RTC	Real Time Clock
RTS	Request To Send (modem control)
TSR	Terminate and Stay Resident (type of program)
VGA	Video Graphics Array/Adapter
uP	MicroProcessor

Index

License and Warranty Agreement

This software is protected by both United States copyright law and international treaty provisions. This means you must treat this software just like a book. Like a book, this software may be freely moved from machine to machine, so long as there is NO POSSIBILITY of the software being used at two different places at the same time. This software is NOT SHAREWARE, and may not be distributed by bulletin boards or other forms which violate V Communications' copyrights. The single exception for duplicating this software is for the sole purpose of backing up our software and protecting your investment from loss.

V Communications Limited Warranty

V Communications, Inc. warrants the diskettes on which the program is furnished to be free from defects in materials and workmanship under normal use, for a period of thirty (30) days from the date of delivery to you as evidenced by a copy of your receipt. In the event of notification within the warranty period of defects in the material or workmanship, V Communications will exchange the diskette. V Communications also warrants that the software conforms substantially with any V Communications' advertised claims at the date of sale, for a period of thirty (30) days from the date of delivery to you as evidenced by a copy of your receipt. In the event of notification within the warranty period of failure to meet any V Communications' advertised claims at the date of sale, you may return the program and all associated manuals in good condition to your place of purchase for a refund. IF YOU NEED TO EXCHANGE OR RETURN A PRODUCT TO V COMMUNICATIONS, CALL OUR CUSTOMER SERVICE DEPARTMENT TO OBTAIN A RETURN AUTHORIZATION NUMBER. The sole remedy for breach of this warranty shall be limited to replacement or refund and shall not encompass any other damages, including but not limited to loss of profit, and incidental, consequential, or other similar claims. Refunds do not include any portion of shipping and handling, or special shipping fees such as COD costs, etc.

Limitation of Remedies

V Communications, Inc. specifically disclaims all other warranties, expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose with respects to defects in the diskette or documentation, and the program license granted herein in particular, and without limiting operation of the program license with respect to any particular application, use, or purpose. In no event shall V Communications be liable for any loss of profit or any other damages, including but not limited to incidental, consequential, or other damages.

Complete Agreement

Having read this Agreement, you agree that no contrary or inconsistent statement, oral or written, has ben made by any person and that this is the complete and exclusive statement of the terms and conditions of the Agreement between us, and any prior proposal or statement, whether oral or written, is superseded. In the event of any dispute regarding the terms and conditions of the Agreement, you agree that the laws of the State of California will govern the interpretation of the Agreement.

V COMMUNICATIONS, INC.
3031 Tisch Way, Suite 802
San Jose, CA 95128
(408) 296-4224