TEXAS INSTRUMENTS

TMS470R1x Code Generation Tools

Release 1.20

Getting Started Guide

1997 ***Microcontroller Products***

TEXAS INSTRUMENTS

Printed in U.S.A., March 1997
M414003-9741 revision B

SPNU117B

TMS470R1x Code Generation Tools Getting Started Guide

Release 1.20

Literature Number: SPNU117B
Manufacturing Part Number: M414003-9741 revision B
March 1997

PRINTED WITH SOY INK™

TEXAS INSTRUMENTS

Printed on Recycled Paper

Preface

Read This First

About This Manual

The *TMS470R1x Code Generation Tools Getting Started Guide* tells you how to install release 1.20 of the TMS470R1x code generation tools on your system. It also provides the following information:

❑ Tells you how to set environment variables for parameters that you use often

❑ Gets you started using the compiler, linker, and assembler

❑ Provides a list of the media contents for your tools set, so you will know what information is associated with each file you have installed

❑ Details enhancements in this release and tells you where to find further information

❑ Describes how you can resolve problems that you may encounter on a PC™ running DOS (MS-DOS™ or PC-DOS™)

Notational Conventions

In this document, the following notational conventions are used:

❑ Program listings, program examples, and interactive displays are shown in a `special typeface`. Examples use a **`bold version`** of the special typeface for emphasis. Interactive displays use **`bold`** to distinguish commands that you enter from items that the system displays (such as prompts, command output, error messages, etc.). Some interactive displays use *`italics`* to describe the type of information that should be entered.

Here is a sample program listing:

```
0011  0005  0001         .field    1, 2
0012  0005  0003         .field    3, 4
0013  0005  0006         .field    6, 3
0014  0006               .even
```

Here is an example of a command that you might enter:

```
set PATH=c:\tool_dir;%PATH%
```

To change your path statement to use the tools, enter the command text as shown in bold and replace `tool_dir` with the name of your tools directory.

- In syntax descriptions, the instruction, command, or directive is in a **bold typeface** font and parameters are in an *italic typeface*. Portions of a syntax that are in **bold** should be entered as shown; portions of a syntax that are in *italics* describe the type of information that should be entered.

  Here is an example of a command that you might use:

  mkdir *tool_dir*

  In this example, you would type mkdir, as shown, and replace *tool_dir* with the name of your directory.

- Square brackets ([and]) identify an optional parameter. If you use an optional parameter, you specify the information within the brackets; you don't enter the brackets themselves. Here's an example of a command that has optional parameters:

  SET C_DIR=*pathname*$_1$ [;*pathname*$_2$ . . .]

  Setting the C_DIR environment variable allows you to specify one or more pathnames for the C compiler to search.

Related Documentation From Texas Instruments

The following books describe the TMS470R1x and related support tools. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477–8924. When ordering, please identify the book by its title and literature number.

TMS470R1x Assembly Language Tools User's Guide (literature number SPNU118) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the TMS470R1x devices.

TMS470R1x Optimizing C Compiler User's Guide (literature number SPNU119) describes the TMS470R1x C compiler. This C compiler accepts ANSI standard C source code and produces assembly language source code for the TMS470R1x devices.

TMS470R1x C Source Debugger User's Guide (literature number SPNU124) describes the TMS470R1x emulator and simulator versions of the C source debugger interface. This book discusses various aspects of the debugger interface, including window management, command entry, code execution, data management, and breakpoints. It also includes a tutorial that introduces basic debugger functionality.

TMS470R1x User's Guide (literature number SPNU134) describes the TMS470R1x RISC microcontroller, its architecture (including registers), ICEBreaker module, interfaces (memory, coprocessor, and debugger), 16-bit and 32-bit instruction sets, and electrical specifications.

Trademarks

DOS/4G is a trademark of Tenberry Software, Inc.

HP-UX, HP 9000 Series 700, and PA-RISC are trademarks of Hewlett-Packard Company.

IBM, PC, and PC-DOS are trademarks of International Business Machines Corp.

MS-DOS, Windows, and Windows NT are registered trademarks of Microsoft Corp.

OpenWindows, SunOS, and Solaris are trademarks of Sun Microsystems, Inc.

Pentium is a trademark of Intel Corporation.

SPARCstation is trademark of SPARC International, Inc., but licensed exclusively to Sun Microsystems, Inc.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

X Window System is a trademark of the Massachusetts Institute of Technology.

If You Need Assistance . . .

❑ **World-Wide Web Sites**

TI Online	http://www.ti.com
Semiconductor Product Information Center (PIC)	http://www.ti.com/sc/docs/pic/home.htm
Microcontroller Home Page	http://www.ti.com/sc/micro

❑ **North America, South America, Central America**

Product Information Center (PIC)	(972) 644-5580		
TI Literature Response Center U.S.A.	(800) 477-8924		
Software Registration/Upgrades	(214) 638-0333	Fax: (214) 638-7742	
U.S.A. Factory Repair/Hardware Upgrades	(281) 274-2285		
U.S. Technical Training Organization	(972) 644-5580		
Microcontroller Hotline	(281) 274-2370	Fax: (281) 274-4203	Email: micro@ti.com
Microcontroller Modem BBS	(281) 274-3700 8-N-1		

❑ **Europe, Middle East, Africa**

European Product Information Center (EPIC) Hotlines:

Multi-Language Support	+33 1 30 70 11 69	Fax: +33 1 30 70 10 32	Email: epic@ti.com
Deutsch	+49 8161 80 33 11 or +33 1 30 70 11 68		
English	+33 1 30 70 11 65		
Francais	+33 1 30 70 11 64		
Italiano	+33 1 30 70 11 67		
EPIC Modem BBS	+33 1 30 70 11 99		
European Factory Repair	+33 4 93 22 25 40		
Europe Customer Training Helpline		Fax: +49 81 61 80 40 10	

❑ **Asia-Pacific**

Literature Response Center	+852 2 956 7288	Fax: +852 2 956 2200

❑ **Japan**

Product Information Center	+0120-81-0026 (in Japan)	Fax: +0120-81-0036 (in Japan)
	+03-3457-0972 or (INTL) 813-3457-0972	Fax: +03-3457-1259 or (INTL) 813-3457-1259

❑ **Documentation**

When making suggestions or reporting errors in documentation, please include the following information that is on the title page: the full title of the book, the publication date, and the literature number.

Mail: Texas Instruments Incorporated
Technical Documentation Services, MS 702
P.O. Box 1443
Houston, Texas 77251-1443

Email: comments@books.sc.ti.com

Note: When calling a Literature Response Center to order documentation, please specify the literature number of the book.

Contents

Chapter 1

Setting Up the Code Generation Tools With DOS or Windows 3.1x

This chapter helps you install release 1.20 of the TMS470R1x code generation tools and set up your code-development environment on a 32-bit x86-based or Pentium PC running MS-DOS, PC-DOS, or Windows 3.1x. These tools include an optimizing C compiler and a full set of assembly language tools for developing and manipulating assembly language and object (executable) code.

The C compiler tools are composed of the following components:

❑ Parser
❑ Optimizer
❑ Code generator
❑ Interlist utility
❑ Library-build utility

The assembly language tools are composed of the following components:

❑ Assembler
❑ Archiver
❑ Linker
❑ Absolute lister
❑ Cross-reference lister
❑ Hex conversion utility

Topic **Page**

1.1 System Requirements

To install and use the code generation tools, you need the items listed in the following hardware and software checklists.

Hardware checklist

☐	**Host**	32-bit x86-based or Pentium™ based PC with an ISA/EISA bus
☐	**Memory**	4M–16M bytes of RAM plus 32M bytes of hard-disk space for temporary files and 4M bytes of hard-disk space for the code generation tools
☐	**Display**	Monochrome or color monitor (color recommended)
☐	**Required hardware**	CD-ROM drive
☐	**Optional hardware**	Microsoft™ compatible mouse

Software checklist

☐	**Operating system**	One of these operating systems: ❏ MS-DOS ❏ PC-DOS ❏ Windows 3.1x
☐	**CD-ROM**	*TMS470R1x Code Generation Tools*

Note: Memory Needed

The code generation tools, when installed on a PC, require at least 4M bytes of memory, but you can expect some performance problems when using only 4M bytes. (16M bytes is recommended.) You may want to free as much memory as possible before running the tools, especially if you have less than 16M bytes.

1.2 Installing the Code Generation Tools

This section helps you install the code generation tools on your hard-disk system. The code generation tools package is shipped on CD-ROM. The installation instructions vary according to your operating system.

Installing the tools on DOS systems

To install the tools on a DOS system, follow these steps:

1) Insert the *TMS470R1x Code Generation Tools* CD-ROM into your CD-ROM drive.
2) Change to the CD-ROM drive (where d: is the name of your CD-ROM drive):

   `d:` ↵

3) Enter the following command:

   `install` ↵

4) Follow the on-screen instructions.

If you choose not to have he environment variables set up automatically, you can set yp the environment variables in your autoexec.bat file. See Section 1.3, *Setting Up the Code Generation Environment*, on page 1-4, for more information.

Installing the tools on Windows 3.1x systems

To install the tools on a Windows 3.1x system, follow these steps:

1) Insert the *TMS470R1x Code Generation Tools* CD-ROM into your CD-ROM drive.
2) Start Windows 3.1x.
3) From the File menu, select Run.
4) In a dialog box, enter the following command (where d: is the name of your CD-ROM drive):

   `d:\setup.exe`

5) Click on OK.
6) Follow the on-screen instructions.

If you choose not to have the environment variables set up automatically, you can set up the environment variables in your autoexec.bat file. See Section 1.3, *Setting Up the Code Generation Environment*, on page 1-4, for more information.

1.3 Setting Up the Code Generation Environment

Before or after you install the code generation tools, you can define environment variables that set certain software tool parameters you normally use. An *environment variable* is a special system symbol that you define and associate to a string. A program uses this symbol to find or obtain certain types of information.

When you use environment variables, default values are set, making each individual invocation of the tools simpler because these parameters are automatically specified. When you invoke a tool, you can use command-line options to override many of the defaults that are set with environment variables.

The code generation tools use the following environment variables:

- A_DIR
- C_DIR
- C_OPTION
- TMP

By default, the installation program modifies your autoexec.bat file and sets up these environment variables:

```
set PATH=c:\tool_dir;%PATH%
set A_DIR=c:\tool_dir
set C_DIR=c:\tool_dir
```

If you choose not to have the environment variables set up automatically, you can modify your autoexec.bat file to include the set commands above.

In addition to setting up environment variables, you must modify your path statement. The following subsections describe how to modify your path statement and how to define the environment variables that the code generation tools use.

Identifying the directory that contains the executable files (PATH statement)

You must include the *tool_dir* directory in your PATH statement so that you can specify the assembler and compiler tools without specifying the name of the directory that contains the executable files.

- ❑ If you modify your autoexec.bat file to change the path information, add the following to the end of the PATH statement:

  ;c:\*tool_dir*

- ❑ If you set the PATH statement from the command line, enter the following:

  set PATH=c:\*tool_dir*;%PATH%

  The addition of **;%PATH%** ensures that this PATH statement does not undo the PATH statements in any other batch files (including the autoexec.bat file).

Identifying alternate directories for the assembler to search (A_DIR)

The assembler uses the A_DIR environment variable to name alternative directories for the assembler to search. To set the A_DIR environment variable, use this syntax:

set A_DIR=*pathname*$_1$[;*pathname*$_2$ . . .]

The *pathnames* are directories that contain copy/include files or macro libraries. You can separate the pathnames with a semicolon or with a blank. Once you set A_DIR, you can use the .copy, .include, or.mlib directive in assembly source without specifying path information.

If the assembler does not find the file in the directory that contains the current source file or in directories named by the –ioption (which names alternate directories), it searches the paths named by the A_DIR enviroment variable. For more information on the –i option, see the *TMS470R1x Assembly Language Tools User's Guide* or the *TMS470R1x Optimizing C Compiler User's Guide.*

Identifying alternate directories for the compiler to search (C_DIR)

The compiler uses the C_DIR environment variable to name alternative directories for the compiler to search. To set the C_DIR environment variable, use this syntax:

set C_DIR=*pathname*$_1$[;*pathname*$_2$ . . .]

The *pathnames* are directories that contain #include files or function libraries (such as stdio.h). You can separate the pathnames with a semicolon or with a blank. In C source, you can use the #include directive without specifying path information. Instead, you can specify the path information with C_DIR.

Setting default shell options (C_OPTION)

You might find it useful to set the compiler, assembler, and linker shell default options using the C_OPTION environment variable. If you do this, the shell uses the default options and/or input filenames that you name with C_OPTION every time you run the shell.

Setting the default options with the C_OPTION environment variable is useful when you want to run the shell consecutive times with the same set of options and/or input files. After the shell reads the command line and the input filenames, it reads the C_OPTION environment variable and processes it.

To set the C_OPTION environment variable, use this syntax:

set C_OPTION=*option*$_1$ [;*option*$_2$ . . .]

Environment variable options are specified in the same way and have the same meaning as they do on the command line. For example, if you want to always run quietly (the –q option), enable C source interlisting (the –s option), and link (the –z option), set up the C_OPTION environment variable as follows:

```
set C_OPTION=–qs –z
```

In the following examples, each time you run the compiler shell, it runs the linker. Any options following –z on the command line or in C_OPTION are passed to the linker. This enables you to use the C_OPTION environment variable to specify default compiler and linker options and then specify additional compiler and linker options on the shell command line. If you have set –z in the environment variable and want to compile only, use the –c option of the shell. These additional examples assume C_OPTION is set as shown above:

```
cl470 *c                ; compiles and links
cl470 –c *.c            ; only compiles
cl470 *.c –z lnk.cmd    ; compiles and links using a
                        ; command file
cl470 –c *.c –z lnk.cmd ; only compiles (–c overrides –z)
```

For more information about shell options, see the *TMS470R1x Optimizing C Compiler User's Guide*. For more information about linker options, see the *TMS470R1x Assembly Language Tools User's Guide .*

Specifying a temporary file directory (TMP)

The compiler shell program creates intermediate files as it processes your program. By default, the shell puts intermediate files in the current directory. However, you can name a specific directory for temporary files by using the TMP environment variable.

Using the TMP environment variable allows use of a RAM disk or other file systems. It also allows source files to be compiled from a remote directory without writing any files into the directory where the source resides. This is useful for protected directories.

To set the TMP environment variable, use this syntax:

set TMP=*pathname*

For example, to set up a directory named temp for intermediate files on your hard drive, enter:

```
set TMP=c:\temp ⏎
```

Resetting defined environment variables

The environment variables that you define remain set until you reboot the system. If you want to clear an environment variable, use this command:

set *variable name***=**

For example, to reset the A_DIR environment variable, enter:

```
set A_DIR ⏎
```

Verifying that the environment variables are set

To verify that the environment variables are set, open a DOS box and enter:

```
set ⏎
```

This command lists the path and environment variables and their current values.

1.4 Performance Considerations

You may notice a speed degradation when you use the code generation tools with DOS or Windows 3.1x. This speed degradation may occur when you use DOS with the tools to get appropriate host memory support.

If you encounter error messages when you use the tools on a PC with DOS, run PMINFO to determine your system configuration before you contact technical support. For more information about PMINFO, see Appendix A, *Troubleshooting DOS systems.*

1.5 Where to Go From Here

Your code generation tools are now installed on your DOS or Windows 3.1x system. Now you should do each of the following tasks:

- ❏ Go to Chapter 5, *Getting Started With the Code Generation Tools*. This chapter provides you with an overview of how to invoke and use the assembler, linker, and compiler.
- ❏ Read Chapter 6, *Release Notes*. This chapter explains the new features included in release 1.20 of the code generation tools.
- ❏ Use Appendix A, *Troubleshooting DOS Systems*, as necessary. This appendix lists kernel and DOS/4G™ error messages and explains how you can resolve the messages.

Chapter 2

Setting Up the Code Generation Tools With Windows 95 and Windows NT

This chapter helps you install release 1.20 of the TMS470R1x code generation tools and set up your code-development environment on a 32-bit x86-based or Pentium PC running Windows 95 or Windows NT™. These tools include an optimizing C compiler and a full set of assembly language tools for developing and manipulating assembly language and object (executable) code.

The C compiler tools are composed of the following components:

- ❑ Parser
- ❑ Optimizer
- ❑ Code generator
- ❑ Interlist utility
- ❑ Library-build utitlity

The assembly language tools are composed of the following components:

- ❑ Assembler
- ❑ Archiver
- ❑ Linker
- ❑ Absolute lister
- ❑ Cross-reference lister
- ❑ Hex conversion utility

Topic **Page**

2.1 System Requirements

To install and use the code generation tools, you need the items listed in the following hardware and software checklists.

Hardware checklist

☐	**Host**	32-bit x86-based or Pentium based PC with an ISA/EISA bus
☐	**Memory**	Minimum of 16M bytes of RAM plus 32M bytes of hard-disk space for temporary files and 4M bytes of hard-disk space for the code generation tools
☐	**Display**	Monochrome or color monitor (color recommended)
☐	**Required hardware**	CD-ROM drive
☐	**Optional hardware**	Microsoft compatible mouse

Software checklist

☐	**Operating system**	One of these operating systems: ❑ Windows 95 version 4.0 (or higher) ❑ Windows NT Workstation version 3.5.1 or 4.0
☐	**CD-ROMs**	*TMS470R1x Code Generation Tools*

2.2 Installing the Code Generation Tools

This section helps you install the code generation tools on your hard-disk system. The code generation tools package is shipped on CD-ROM. To install the tools on a PC running Windows 95 or Windows NT, follow these steps:

1) Insert the *TMS470R1x Code Generation Tools* CD-ROM into your CD-ROM drive.

2) Start Windows.

3) If you are running Windows 95, select Run from the Start menu.
   If you are running Windows NT, select Run from the File menu.

4) In the dialog box, enter the following command (where d: is the name of your CD-ROM drive):

   ```
   d:\setup.exe
   ```

5) Click on OK.

6) Follow the on-screen instructions.

If you choose not to have the environment variables set up automatically, you can set them up yourself in one of the following ways:

- ❑ If you are running Windows 95, you can set up the environment variables in your autoexec.bat file.

- ❑ If you are running Windows NT, you can set up the environment variables in the System applet of the Control Panel.

See Section 2.3, *Setting Up the Code Generation Environment,* on page 2-4, for more information.

2.3 Setting Up the Code Generation Environment

Before or after you install the code generation tools, you can define environment variables that set certain software tool parameters you normally use. An *environment variable* is a special system symbol that you define and assign to a string. A program uses this symbol to find or obtain certain types of information.

When you use environment variables, default values are set, making each individual invocation of the tools simpler because these parameters are automatically specified. When you invoke a tool, you can use command-line options to override many of the defaults that are set with environment variables.

The code generation tools use the following environment variables:

- ❑ A_DIR
- ❑ C_DIR
- ❑ C_OPTION
- ❑ TMP

By default, the installation program modifies your autoexec.bat file and sets up these environment variables:

```
set PATH=c:\tool_dir;%PATH%
set A_DIR=c:\tool_dir
set C_DIR=c:\tool_dir
```

These variables are set up in the registry under:

```
HKEY_CURRENT_USER\Environment
```

If you choose not to have the environment variables set up automatically, you can set them up yourself in one of the following ways:

- ❑ If you are running Windows 95, you can modify your autoexec.bat file to include the set commands above.
- ❑ If you are running Windows NT, you can set up the environment variables in the System applet of the Control Panel. Enter the same commands that you would enter on the command line in the System applet.

In addition to setting up environment variables, you must modify your path statement. The following subsections describe how to modify your path statement and how to define the environment variables that the code generation tools use.

Identifying the directory that contains the executable files (PATH statement)

You must include the *tool_dir* directory in your PATH statement so that you can specify the assembler and compiler tools without specifying the name of the directory that contains the executable files.

- ❑ You can change the path information in one of the following ways:
  - ■ If you are running Windows 95, modify your autoexec.bat file to change the path information by adding the following to the end of the PATH statement:

    `;c:\`*`tool_dir`*

  - ■ If you are running Windows NT, modify the System applet of the Control Panel to change the path information by adding the following to the end of the PATH statement:

    `;c:\`*`tool_dir`*

- ❑ If you set the PATH statement from the command line, enter the following:

  `set PATH=c:\``tool_dir`**`;%PATH%`**

  The addition of **;%PATH%** ensures that this PATH statement does not undo the PATH statements in any other batch files (including the autoexec.bat file).

Identifying alternate directories for the assembler to search (A_DIR)

The assembler uses the A_DIR environment variable to name alternative directories for the assembler to search. To set the A_DIR environment variable, use this syntax:

set A_DIR=*pathname*$_1$[;*pathname*$_2$ . . .]

The *pathnames* are directories that contain copy/include files or macro libraries. You can separate the pathnames with a semicolon or with a blank. Once you set A_DIR, you can use the .copy, .include, or .mlib directive in assembly source without specifying path information.

If the assembler does not find the file in the directory that contains the current source file or in directories named by the –i option (which names alternate directories), it searches the paths named by the A_DIR environment variable. For more information on the –i option, see the *TMS470R1x Assembly Language Tools User's Guide* or the *TMS470R1x Optimizing C Compiler User's Guide*.

Identifying alternate directories for the compiler to search (C_DIR)

The compiler uses the C_DIR environment variable to name alternate directories that contain #include files and function libraries. To set the C_DIR environment variable, use this syntax:

set C_DIR=*pathname*$_1$[;*pathname*$_2$ . . .]

The *pathnames* are directories that contain #include files or libraries (such as stdio.h). You can separate the pathnames with a semicolon or with a blank. In C source, you can use the #include directive without specifying path information. Instead, you can specify the path information with C_DIR.

Setting default shell options (C_OPTION)

You may find it useful to set the compiler, assembler, and linker default shell options using the C_OPTION environment variable. If you do this, the shell uses the default options and/or input filenames that you name with C_OPTION every time you run the shell.

Setting up default options with the C_OPTION environment variable is useful when you want to run the shell consecutive times with the same set of options and/or input files. After the shell reads the command line and the input filenames, it reads the C_OPTION environment variable and processes it.

To set the C_OPTION environment variable, use this syntax:

set C_OPTION=*option*$_1$ [*option*$_2$ . . .]

Environment variable options are specified in the same way and have the same meaning as they do on the command line. For example, if you want to always run quietly (the –q option), enable C source interlisting (the –s option), and link (the –z option), set up the C_OPTION environment variable as follows:

```
set C_OPTION=–qs –z
```

In the following examples, each time you run the compiler shell, it runs the linker. Any options following –z on the command line or in C_OPTION are passed to the linker. This enables you to use the C_OPTION environment variable to specify default compiler and linker options and then specify additional compiler and linker options on the shell command line. If you have set –z in the environment variable and want to compile only, use the –c option of the shell. These additional examples assume C_OPTION is set as shown above:

```
cl470 *.c               ; compiles and links
cl470 –c *.c            ; only compiles
cl470 *.c –z lnk.cmd    ; compiles and links using a
                        ; command file
cl470 –c *.c –z lnk.cmd ; only compiles (–c overrides –z)
```

For more information about shell options, see the *TMS470R1x Optimizing C Compiler User’s Guide*. For more information about linker options, see the *TMS470R1x Assembly Language Tools User’s Guide*.

Specifying a temporary file directory (TMP)

The compiler shell program creates intermediate files as it processes your program. By default, the shell puts intermediate files in the current directory. However, you can name a specific directory for temporary files by using the TMP environment variable.

Using the TMP environment variable allows use of a RAM disk or other file systems. It also allows you to compile source files from a remote directory without writing any files into the directory where the source resides. This is useful for protected directories.

To set the TMP environment variable, use this syntax:

set TMP=*pathname*

For example, to set up a directory named temp for intermediate files on your hard drive, enter:

```
set TMP=c:\temp ↵
```

Resetting defined environment variables

The environment variables that you define remain set until you reboot the system. If you want to clear an environment variable, use this command:

set *variable_name***=**

For example, to reset the A_DIR environment variable, enter:

```
set A_DIR= ↵
```

Verifying that the environment variables are set

To verify that the environment variables are set, open a DOS box and enter:

```
set ↵
```

This command lists the path and environment variables and their current values.

2.4 Where to Go From Here

Your code generation tools are now installed on your Windows 95 or Windows NT system. Now you should do each of the following tasks:

- ❑ Turn to Chapter 5, *Getting Started With the Code Generation Tools*. This chapter provides you with an overview of how to invoke and use the assembler, linker, and compiler.
- ❑ Read Chapter 6, *Release Notes*. This chapter explains the new features included in release 1.20 of the code generation tools.

Chapter 3

Setting Up the Code Generation Tools With SunOS

This chapter helps you install release 1.20 of the TMS470R1x code generation tools and set up your code-development environment on a SPARCstation running SunOS™ version 4.1.x (or higher) or Solaris™ version 2.5.x (or higher). These tools include an optimizing C compiler and a full set of assembly language tools for developing and manipulating assembly language and object (executable) code.

The C compiler tools are composed of the following components:

- ❑ Parser
- ❑ Optimizer
- ❑ Code generator
- ❑ Interlist utility
- ❑ Library-build utitlity

The assembly language tools are composed of the following components:

- ❑ Assembler
- ❑ Archiver
- ❑ Linker
- ❑ Absolute lister
- ❑ Cross-reference lister
- ❑ Hex conversion utility

Topic **Page**

3.1 System Requirements

To install and use the code generation tools, you need the items in the following hardware and software checklists.

Hardware checklist

☐	**Host**	SPARCstation compatible system with a SPARCstation 2 class or higher performance
☐	**Display**	Monochrome or color monitor (color recommended)
☐	**Disk space**	4M bytes of disk space
☐	**Required hardware**	CD-ROM drive
☐	**Optional hardware**	Mouse

Software checklist

☐	**Operating system**	SunOS version 4.1.x (or higher) or SunOS version 5.x (also known as Solaris 2.x) using an X Window System™ based window manager, such as OpenWindows™ version 3.0 (or higher).
☐	**Root privileges**	If you are running SunOS 4.1.x, 5.0, or 5.1, you *must* have root privileges to mount and unmount the CD-ROM. If you do not have root privileges, get help from your system administrator.
☐	**CD-ROMs**	*TMS470R1x Code Generation Tools*

3.2 Installing the Code Generation Tools

This section helps you install the code generation tools on your hard-disk system. The software package is shipped on a CD-ROM. To install the tools on a SPARCstation running SunOS or Solaris, you must mount the CD-ROM, copy the files to your system, and unmount the CD-ROM.

Note:

If you are running SunOS 4.1.x, 5.0, or 5.1, you *must* have root privileges to mount or unmount the CD-ROM. If you do not have root privileges, get help from your system administrator.

Mounting the CD-ROM

The steps to mount the CD-ROM vary according to your operating-system version:

❑ If you have SunOS 4.1.x, as root, load the CD-ROM into the drive and enter the following from a command shell:

```
mount –rt hsfs /dev/sr0 /cdrom ⏎
exit ⏎
cd /cdrom/sunos ⏎
```

❑ If you have SunOS 5.0 or 5.1, as root, load the CD-ROM into the drive and enter the following from a command shell:

```
mount –rF hsfs /dev/sr0 /cdrom ⏎
exit ⏎
cd /cdrom/cdrom0/sunos ⏎
```

❑ If you have SunOS 5.2 or higher:

- If your CD-ROM drive is already attached, load the CD-ROM into the drive and enter the following from a command shell:

  ```
  cd /cdrom/cdrom0/sunos ⏎
  ```

- If you do not have a CD-ROM drive attached, you must shut down your system to the PROM level, attach the CD-ROM drive, and enter the following:

  ```
  boot –r ⏎
  ```

  After you log into your system, load the CD-ROM into the drive and enter the following from a command shell:

  ```
  cd /cdrom/cdrom0/sunos ⏎
  ```

Copying the files

Be sure you are not logged on as root. After you mount the CD-ROM, you must create the directory that will contain the tools software and copy the software to that directory.

1) Create a tools directory on your hard disk. To create this directory, enter:

```
mkdir /your_pathname/tool_dir
```

2) Copy the files from the CD-ROM to your hard-disk system:

```
cp –r * /your_pathname/tool_dir
```

Unmounting the CD-ROM

You must unmount the CD-ROM after copying the files.

❑ If you have SunOS 4.1.x, 5.0, or 5.1, as root, enter the following from a command shell:

```
cd
umount /cdrom
eject /dev/sr0
exit
```

❑ If you have SunOS 5.2 or higher, enter the following from a command shell:

```
cd
eject
```

3.3 Setting Up the Code Generation Environment

Before or after you install the code generation tools, you can define environment variables that set certain software tool parameters you normally use. An *environment variable* is a special system symbol that you define and assign to a string. A program uses this symbol to find or obtain certain types of information.

When you use environment variables, default values are set, making each individual invocation of the tools simpler because these parameters are automatically specified. When you invoke a tool, you can use command-line options to override many of the defaults that are set with environment variables.

The code generation tools use the following environment variables:

- A_DIR
- C_DIR
- C_OPTION
- TMP

You can set up the environment variables on the command line or in your .login or .cshrc file (for C shells) or .profile file (for Bourne or Korn shells). To set up these environment variables in your system initialization file, enter the same commands that you would enter on the command line in the file.

In addition to setting up environment variables, you must modify your path statement. The following subsections describe how to modify your path statement and how to define the environment variables that the code generation tools use.

Identifying the directory that contains the executable files (path statement)

You must include the *tool_dir* directory in your path statement so that you can specify the assembler and compiler tools without specifying the name of the directory that contains the executable files.

❑ If you modify your .cshrc file (for C shells) or .profile file (for Bourne or Korn shells) to change the path information, add the following to the end of the path statement:

  /*your_pathname***/***tool_dir*

❑ If you set the path statement from the command line, use this format:

  ■ For C shells:

   set path=(/*your_pathname***/***tool_dir* **$path)**

  ■ For Bourne or Korn shells:

   PATH=/*your_pathname***/***tool_dir* **$PATH**

  The addition of **$path** or **$PATH** ensures that this path statement does not undo the path statements in the .cshrc or .profile file.

Identifying alternate directories for the assembler to search (A_DIR)

The assembler uses the A_DIR environment variable to name alternative directories for the assembler to search. To set the A_DIR environment variable, use this syntax:

❑ For C shells:

  setenv A_DIR "*pathname*$_1$[;*pathname*$_2$ . . .]**"**

❑ For Bourne or Korn shells:

  A_DIR="*pathname*$_1$[;*pathname*$_2$ . . .]**"**
  export A_DIR

(Be sure to enclose the directory names within quotes.)

The *pathnames* are directories that contain copy/include files or macro libraries. You can separate the pathnames with a semicolon or a blank. Once you set A_DIR, you can use the .copy, .include, or .mlib directive in assembly source without specifying path information.

If the assembler does not find the file in the directory that contains the current source file or in directories named by the –i option (which names alternate directories), it searches the paths named by the A_DIR environment variable.

For more information on the –i option, see the *TMS470R1x Assembly Language Tools User's Guide* or the *TMS470R1x Optimizing C Compiler User's Guide*.

Identifying alternate directories for the compiler to search (C_DIR)

The compiler uses the C_DIR environment variable to name alternate directories that contain #include files and function libraries. To set the C_DIR environment variable, use this syntax:

- For C shells:

  setenv C_DIR "*pathname*$_1$[**;***pathname*$_2$ . . .]**"**

- For Bourne or Korn shells:

  C_DIR="*pathname*$_1$[**;***pathname*$_2$ . . .]**"**
  export C_DIR

(Be sure to enclose the directory names within quotes.)

The *pathnames* are directories that contain #include files or libraries (such as stdio.h). You can separate pathnames with a semicolon or with blanks. In C source, you can use the #include directive without specifying path information. Instead, you can specify the path information with C_DIR.

Setting default shell options (C_OPTION)

You may find it useful to set the compiler, assembler, and linker default shell options using the C_OPTION environment variable. If you do this, the shell uses the default options and/or input filenames that you name with C_OPTION every time you run the shell.

Setting up default options with the C_OPTION environment variable is useful when you want to run the shell consecutive times with the same set of options and/or input files. After the shell reads the command line and the input filenames, it reads the C_OPTION environment variable and processes it.

To set the C_OPTION environment variable, use this syntax:

- For C shells:

  setenv C_OPTION "*option*$_1$ [*option*$_2$ . . .]**"**

- For Bourne or Korn shells:

  C_OPTION="*option*$_1$ [*option*$_2$ . . .]**"**
  export C_OPTION

(Be sure to enclose the options within quotes.)

Environment variable options are specified in the same way and have the same meaning as they do on the command line. For example, if you want to always run quietly (the –q option), enable C source interlisting (the –s option), and link (the –z option), set up the C_OPTION environment variable as follows:

❑ For C shells:

```
setenv C_OPTION ”–qs –z”
```

❑ For Bourne or Korn shells:

```
C_OPTION=”–qs –z”
export C_OPTION
```

In the following examples, each time you run the compiler shell, it runs the linker. Any options following –z on the command line or in C_OPTION are passed to the linker. This enables you to use the C_OPTION environment variable to specify default compiler and linker options and then specify additional compiler and linker options on the shell command line. If you have set –z in the environment variable and want to compile only, use the –c option of the shell. These additional examples assume C_OPTION is set as shown above:

```
cl470 *.c               ; compiles and links
cl470 –c *.c            ; only compiles
cl470 *.c –z lnk.cmd    ; compiles and links using a
                        ; command file
cl470 –c *.c –z lnk.cmd ; only compiles (–c overrides –z)
```

For more information about shell options, see the *TMS470R1x Optimizing C Compiler User’s Guide*. For more information about linker options, see the *TMS470R1x Assembly Language Tools User’s Guide*.

Specifying a temporary file directory (TMP)

The compiler shell program creates intermediate files as it processes your program. By default, the shell puts intermediate files in the current directory. However, you can name a specific directory for temporary files by using the TMP environment variable.

Using the TMP environment variable allows use of a RAM disk or other file systems. It also allows you to compile source files from a remote directory without writing any files into the directory where the source resides. This is useful for protected directories.

To set the TMP environment variable, use this syntax:

❑ For C shells:

setenv TMP ”*pathname***”**

❑ For Bourne or Korn shells:

TMP=”*pathname***”**
export TMP

(Be sure to enclose the directory name within quotes.)

For example, to set up a directory named temp for intermediate files, enter:

❑ For C shells:

```
setenv TMP ”/temp”
```

❑ For Bourne or Korn shells:

```
TMP=”/temp”
export TMP
```

Reinitializing your shell

When you modify your shell configuration file, you must ensure that the changes are made to your current session. Use one of the following commands to reread your system initialization file:

❑ For C shells:

```
source ~/.cshrc
```

❑ For Bourne or Korn shells:

```
source ~/.profile
```

Resetting defined environment variables

The environment variables that you define remain set until you reboot the system. If you want to clear an environment variable, use this command:

❑ For C shells:

unsetenv *variable_name*

❑ For Bourne or Korn shells:

unset *variable_name*

For example, to reset the A_DIR environment variable, enter one of these commands:

❑ For C shells:

```
unsetenv A_DIR
```

❑ For Bourne or Korn shells:

```
unset A_DIR
```

Verifying that the environment variables are set

To verify that the environment variables are set, enter:

```
set ↵
```

This command lists the path and environment variables and their current values.

3.4 Where to Go From Here

Your code generation tools are now installed. Now you should do each of the following tasks:

❏ Go to Chapter 5, *Getting Started With the Code Generation Tools*. This chapter provides you with an overview of how to invoke and use the assembler, linker, and compiler.

❏ Read Chapter 6, *Release Notes.* This chapter explains the new features included in release 1.20 of the code generation tools.

Chapter 4

Setting Up the Code Generation Tools on an HP Workstation

This chapter helps you install release 1.20 of the TMS470R1x code generation tools and set up your code-development environment on an HP 9000 Series 700™ PA-RISC™ computer with HP-UX™ 9.0x. These tools include an optimizing C compiler and a full set of assembly language tools for developing and manipulating assembly language and object (executable) code.

The C compiler tools are composed of the following components:

- ❑ Parser
- ❑ Optimizer
- ❑ Code generator
- ❑ Interlist utility
- ❑ Library-build utility

The assembly language tools are composed of the following components:

- ❑ Assembler
- ❑ Archiver
- ❑ Linker
- ❑ Absolute lister
- ❑ Cross-reference lister
- ❑ Hex conversion utility

Topic **Page**

4.1 System Requirements

To install and use the code generation tools, you need the items in the following hardware and software checklists.

Hardware checklist

☐	**Host**	An HP 9000 Series 700 PA-RISC computer
☐	**Display**	Monochrome or color monitor (color recommended)
☐	**Disk space**	4M bytes of disk space
☐	**Required hardware**	CD-ROM drive
☐	**Optional hardware**	Mouse

Software checklist

☐	**Operating system**	HP-UX 9.0x operating system
☐	**Root privileges**	Root privileges to mount and unmount the CD-ROM
☐	**CD-ROMs**	*TMS470R1x Code Generation Tools*

4.2 Installing the Code Generation Tools

This section helps you install the code generation tools on your hard-disk system. The software package is shipped on a CD-ROM. To install the tools on an HP workstation, you must mount the CD-ROM, copy the files to your system, and unmount the CD-ROM.

Note:

If you are running HP-UX 9.0x, you *must* have root privileges to mount or unmount the CD-ROM. If you do not have root privileges, get help from your system administrator.

Mounting the CD-ROM

As root, you can mount the CD-ROM using the UNIX™ mount command or the SAM (system administration manager):

❑ To use the UNIX mount command, enter the following from a command shell:

```
mount –rt cdfs /dev/dsk/your_cdrom_device /cdrom
exit
```

Make the hp directory on the CD-ROM the current directory. For example, if the CD-ROM is mounted at /cdrom, enter the following:

```
cd /cdrom/hp
```

❑ To use SAM to mount the CD-ROM, see *System Administration Tasks*, the HP documentation about SAM, for instructions.

Copying the files

After you mount the CD-ROM, log out as root and log back on as yourself. You must create the directory that will contain the tools software and copy the software to that directory.

1) Create a tools directory on your hard disk. To create this directory, enter:

```
mkdir /your_pathname/tool_dir
```

2) Copy the files from the CD-ROM to your hard-disk system:

```
cp –r * /your_pathname/tooldir
```

Setting up the software tools using a C shell

If you are using a C shell, enter the following:

```
setenv C_DIR "tool_dir" ↵
setenv A_DIR "tool_dir" ↵
set path=(tool_dir $path) ↵
```

You can move the **setenv** and **set path** commands into your .login or .cshrc file to avoid entering these commands each time you invoke a new shell.

Setting up the software tools using a Korn shell

If you are using a Bourne or Korn shell, enter the following:

```
C_DIR=tool_dir ↵
A_DIR=tool_dir ↵
PATH=tool_dir:$PATH ↵
```

You can move the environment variable instructions into your .kshrc file to avoid entering these commands each time you invoke a new shell.

Unmounting the CD-ROM

You must unmount the CD-ROM after copying the files. As root, enter the following from a command shell:

```
cd ↵
umount /cdrom ↵
exit ↵
```

4.3 Setting Up the Code Generation Environment

Before or after you install the code generation tools, you can define environment variables that set certain software tool parameters you normally use. An *environment variable* is a special system symbol that you define and assign to a string. A program uses this symbol to find or obtain certain types of information.

When you use environment variables, default values are set, making each individual invocation of the tools simpler because these parameters are automatically specified. When you invoke a tool, you can use command-line options to override many of the defaults that are set with environment variables.

The code generation tools use the following environment variables:

- A_DIR
- C_DIR
- C_OPTION
- TMP

You can set up the environment variables o the command line or in your .login or .cshrc file (for C shells) or .profile file (for Bourne or Korn shells). To set up these environment variables in your system initialization file, enter the same commands that you would enter on the command line in the file.

In addition to setting up environment variables, you must modify your path statement. The following subsections describe how to modify your path statement and how to define the environment variables that the code generation tools use.

Identifying the directory that contains the executable files (path statement)

You must include the *tool_dir* directory in your path statement so that you can specify the assembler and compiler tools without specifying the name of the directory that contains the executable files.

❑ If you modify your .cshrc file (for C shells) or .profile file (for Bourne or Korn shells) to change the path information, add the following to the end of the path statement:

  /*your_pathname***/***tool_dir*

❑ If you set the path statement from the command line, use this format:

  ■ For C shells:

    set path=(/*your_pathname***/***tool_dir* **$path)**

  ■ For Bourne or Korn shells:

    PATH=/*your_pathname***/***tool_dir* **$PATH**

  The addition of **$path** or **$PATH** ensures that this path statement does not undo the path statements in the .cshrc or .profile file.

Identifying alternate directories for the assembler (A_DIR)

The assembler uses the A_DIR environment variable to name alternative directories for the assembler to search. To set the A_DIR environment variable, use this syntax:

❑ For C shells:

  setenv A_DIR "*pathname*$_1$[;*pathname*$_2$ …]**"**

❑ For Bourne or Korn shells:

  A_DIR="*pathname*$_1$[;*pathname*$_2$ …]**"**
  export A_DIR

(Be sure to enclose the directory names within quotes.)

The *pathnames* are directories that contain copy/include files or macro libraries. You can separate the pathnames with a semicolon or a blank. Once you set A_DIR, you can use the .copy, .include, or .mlib directive in assembly source without specifying path information.

If the assembler does not find the file in the directory that contains the current source file or in directories named by the –i option (which names alternate directories), it searches the paths named by the A_DIR environment variable. For more information on the –i option, see the *TMS470R1x Assembly Language Tools User's Guide* or the *TMS470R1x Optimizing C Compiler User's Guide.*

Identifying alternate directories for the compiler (C_DIR)

The compiler uses the C_DIR environment variable to name alternate directories that contain #include files and libraries. To set the C_DIR environment variable, use this syntax:

- ❑ For C shells:

  setenv C_DIR "*pathname*$_1$[;*pathname*$_2$ . . .]**"**

- ❑ For Bourne or Korn shells:

  C_DIR="*pathname*$_1$[;*pathname*$_2$ . . .]**"**
  export C_DIR

(Be sure to enclose the directory names within quotes.)

The *pathnames* are directories that contain #include files or libraries (such as stdio.h). You can separate pathnames with a semicolon or with blanks. In C source, you can use the #include directive without specifying path information. Instead, you can specify the path information with C_DIR.

Setting default shell options (C_OPTION)

You may find it useful to set the compiler, assembler, and linker shell default options using the C_OPTION environment variable. If you do this, the shell uses the default options and/or input filenames that you name with C_OPTION every time you run the shell.

Setting up default options with the C_OPTION environment variable is useful when you want to run the shell consecutive times with the same set of options and/or input files. After the shell reads the command line and the input filenames, it reads the C_OPTION environment variable and processes it.

The set the C_OPTION environment variable, use this syntax:

- ❑ For C shells:

  setenv C_OPTION "*option*$_1$ [*option*$_2$ . . .]**"**

- ❑ For Bourne or Korn shells:

  C_OPTION="*option*$_1$ [*option*$_2$ . . .]**"**
  export C_OPTION

(Be sure to enclose the options within quotes.)

Environment variable options are specified in the same way and have the same meaning as they do on the command line. For example, if you want to always run quietly (the –q option), enable C source interlisting (the –s option), and link (the –z option), set up the C_OPTION environment variable as follows:

- ❑ For C shells:

  ```
  setenv C_OPTION ”–qs –z”
  ```

- ❑ For Bourne or Korn shells:

  ```
  C_OPTION=”–qs –z”
  export C_OPTION
  ```

In the following examples, each time you run the compiler shell, it runs the linker. Any options following –z on the command line or in C_OPTION are passed to the linker. This enables you to use the C_OPTION environment variable to specify default compiler and linker options and then specify additional compiler and linker options on the shell command line. If you have set –z in the environment variable and want to compile only, use the –c option of the shell. These additional examples assume C_OPTION is set as shown above:

```
cl470 *.c               ; compiles and links
cl470 –c *.c            ; only compiles
cl470 *.c –z lnk.cmd    ; compiles and links using a
                        ; command file
cl470 –c *.c –z lnk.cmd ; only compiles (–c overrides –z)
```

For more information about shell options, see the *TMS470R1x Optimizing C Compiler User’s Guide*. For more information on linker options, see the *TMS470R1x Assembly Language Tools User’s Guide.*

Specifying a temporary file directory (TMP)

The compiler shell program creates intermediate files as it processes your program. By default, the shell puts intermediate files in the current directory. However, you can name a specific directory for temporary files by using the TMP environment variable.

Using the TMP environment variable allows use of a RAM disk or other file systems. It also allows you to compile source files from a remote directory without writing any files into the directory where the source resides. This is useful for protected directories.

To set the TMP environment variable, use this syntax:

❑ For C shells:

setenv TMP ”*pathname***”**

❑ For Bourne or Korn shells:

TMP=”*pathname***”**
export TMP

(Be sure to enclose the directory name within quotes.)

For example, to set up a directory named temp for intermediate files, enter:

❑ For C shells:

```
setenv TMP ”/temp”
```

❑ For Bourne or Korn shells:

```
TMP=”/temp”
export TMP
```

Reinitializing your shell

When you modify your shell configuration file, you must ensure that the changes are made to your current session. Use one of the following commands to reread your system initialization file:

❑ For C shells:

```
source ~/.cshrc ↵
```

❑ For Bourne or Korn shells:

```
source ~/.profile ↵
```

Resetting defined environment variables

The environment variables that you define remain set until you reboot the system. If you want to clear an environment variable, use this command:

❑ For C shells:

unsetenv *variable name*

❑ For Bourne or Korn shells:

unset *variable name*

For example, to reset the A_DIR environment variable, enter one of these commands:

❑ For C shells:

```
unsetenv A_DIR
```

❑ For Bourne or Korn shells:

```
unset A_DIR
```

Verifying that the environment variables are set

To verify that the environment variables are set, enter:

```
set ↵
```

This command lists the path and environment variables and their current values.

4.4 Where to Go From Here

Your code generation tools are now installed. Now you should do each of the following tasks:

❑ Go to Chapter 5, *Getting Started With the Code Generation Tools*. This chapter provides you with an overview of how to invoke and use the assembler, linker, and compiler.

❑ Read Chapter 6, *Release Notes*. This chapter explains the new features included in release 1.20 of the code generation tools.

Chapter 5

Getting Started With the Code Generation Tools

This chapter helps you start using the assembler, linker, and compiler by providing a quick walkthrough of these tools. For more information about invoking and using these tools, see the *TMS470R1x Assembly Language Tools User's Guide* and the *TMS470R1x Optimizing C Compiler User's Guide*.

Topic **Page**

5.1 Getting Started With the Assembler and Linker

This section provides a quick walkthrough of the assembler and linker so that you can get started without reading the entire *TMS470R1x Assembly Language Tools User's Guide*. Example 5–1 through Example 5–6 show the most common methods for invoking the assembler and linker.

1) Create two short source files to use for the walkthrough; call them file1.asm and file2.asm. (See Example 5–1 and Example 5–2.)

Example 5–1. file1.asm

```
        .global inclw

start:  MOV     r6, #0
        MOV     r7, #0

loop:   BL      inclw
        BCC     loop

        .end
```

Example 5–2. file2.asm

```
        .global inclw

inclw:  ADDS    r7, r7, #1
        ADDCSS  r6, r6, #1
        MOV     pc, lr

        .end
```

2) Enter the following command to assemble file1.asm:

   `asm470 file1` ↵

   The **asm470** command invokes the assembler. The input source file is file1.asm. (If the input file extension is .asm, you do not have to specify the extension; the assembler uses .asm as the default.)

   This example creates an object file called file1.obj. The assembler creates an object file only if there are no errors in assembly. You can specify a name for the object file, but if you do not, the assembler uses the input filename with an extension of .obj.

3) Now enter the following command to assemble file2.asm:

   `asm470 file2.asm –l` ⏎

   This time, the assembler creates an object file called file2.obj. The –l (lowercase L) option tells the assembler to create a listing file; the listing file for this example is called file2.lst. It is not necessary to create a listing file, but it gives you information and assures you that the assembly has resulted in the desired object code. The listing file for this example is shown in Example 5–3.

Example 5–3. file2.lst, the Listing File Created by asm470 file2.asm –l

```
TMS470 COFF Assembler      Version 1.20     Sat Feb  8 15:22:13 1997
Copyright (c) 1995–1996    Texas Instruments Incorporated

file2.asm                                                     PAGE    1

       1
       2                             .global inclw
       3
       4 00000000 E2977001  inclw:  ADDS    r7, r7, #1
       5 00000004 22966001          ADDCSS  r6, r6, #1
       6 00000008 E1A0F00E          MOV     pc, lr
       7
       8                             .end

 No Errors, No Warnings
```

4) Now enter the following command to link file1.obj and file2.obj:

   `lnk470 file1 file2 –m lnker2.map –o prog.out` ⏎

   The **lnk470** command invokes the linker. The input object files are file1.obj and file2.obj. (If the input file extension is .obj, you do not have to specify the extension; the linker uses .obj as the default.) The linker combines file1.obj and file2.obj to create an executable object module called prog.out. The –o option supplies the name of the output module. Example 5–4 shows the map file resulting from this operation. (The map file is produced only if you use the –m option.)

Example 5–4. Output Map File, lnker2.map

```
*****************************************************
TMS470 COFF Linker          Version 1.20
*****************************************************
Sat Feb  8 15:24:43 1997

OUTPUT FILE NAME:   <prog.out>
ENTRY POINT SYMBOL: 0


SECTION ALLOCATION MAP

 output                                  attributes/
section   page    origin      length       input sections
--------  ----  ----------  ----------   ----------------
.text      0    00000000    0000001c
                  00000000    00000010     file1.obj (.text)
                  00000010    0000000c     file2.obj (.text)

.const     0    00000000    00000000     UNINITIALIZED

.data      0    00000000    00000000     UNINITIALIZED
                  00000000    00000000     file2.obj (.data)
                  00000000    00000000     file1.obj (.data)

.bss       0    00000000    00000000     UNINITIALIZED
                  00000000    00000000     file2.obj (.bss)
                  00000000    00000000     file1.obj (.bss)


GLOBAL SYMBOLS

address  name                             address  name
-------- ----                             -------- ----
00000000 .bss                             00000000 edata
00000000 .data                            00000000 .bss
00000000 .text                            00000000 end
00000000 edata                            00000000 .data
00000000 end                              00000000 .text
0000001c etext                            00000010 inclw
00000010 inclw                            0000001c etext

[7 symbols]
```

The two files, file1 and file2, can be linked together with or without a command file. However, using a command file allows you to configure your memory using the MEMORY and SECTIONS directives:

- ❑ The MEMORY directive lets you specify a model of target memory so that you can define the types of memory your system contains and the address ranges they occupy.
- ❑ The SECTIONS directive describes how input sections are combined into output sections and specifies where output sections are placed in memory.

You can include the linker options and filenames in the linker command file, or you can enter them on the command line. If you do not include a linker command file, the linker uses a default allocation algorithm. Refer to the *TMS470R1x Assembly Language Tools User's Guide* for more information about the linker command file and the default allocation algorithm.

Example 5–5. Sample Linker Command File, linker2.cmd

```
/* Specify the System Memory Map */

MEMORY
{
    D_MEM  : org = 0x00000000   len = 0x00001000  /* Data Memory    (RAM) */
    P_MEM  : org = 0x00001000   len = 0x00001000  /* Program Memory (ROM) */
}

/* Specify the Sections Allocation Into Memory */

SECTIONS
{
    .data    : {} > D_MEM  /* Initialized Data */
    .text    : {} > P_MEM  /* Code             */
}
```

Typing in the following command line using the linker command file shown in Example 5–5 results in the map file shown in Example 5–6.

```
lnk470 file1 file2 linker2.cmd –m linker2.map –o prog.out
```

Example 5–6. Linker Map File (linker2.map) Linked Using a Linker Command File

```
*****************************************************
TMS470 COFF Linker          Version 1.20
*****************************************************
Sat Feb  8 15:36:45 1997

OUTPUT FILE NAME:   <prog.out>
ENTRY POINT SYMBOL: 0


MEMORY CONFIGURATION

           name     origin    length      used    attributes    fill
         --------  --------  ---------  --------  ----------  --------
         D_MEM     00000000  000001000  00000000     RWIX
         P_MEM     00001000  000001000  0000001c     RWIX


SECTION ALLOCATION MAP

 output                                  attributes/
section   page    origin      length       input sections
--------  ----  ----------  ----------   ----------------
.data      0    00000000    00000000     UNINITIALIZED
                  00000000    00000000     file2.obj (.data)
                  00000000    00000000     file1.obj (.data)

.text      0    00001000    0000001c
                  00001000    00000010     file1.obj (.text)
                  00001010    0000000c     file2.obj (.text)

.bss       0    00000000    00000000     UNINITIALIZED
                  00000000    00000000     file2.obj (.bss)
                  00000000    00000000     file1.obj (.bss)


GLOBAL SYMBOLS

address  name                             address  name
-------- ----                             -------- ----
00000000 .bss                             00000000 edata
00000000 .data                            00000000 .bss
00001000 .text                            00000000 end
00000000 edata                            00000000 .data
00000000 end                              00001000 .text
0000101c etext                            00001010 inclw
00001010 inclw                            0000101c etext

[7 symbols]
```

5.2 Getting Started With the C Compiler

The TMS470R1x C compiler consists of many phases, including parsing, optimization, and code generation. The simplest way to compile is to use the shell program, which is included with the compiler. This section provides a quick walkthrough so that you can get started without reading the entire *TMS470R1x Optimizing C Compiler User's Guide*.

1) Create a sample file called function.c that contains the following code:

```
/**************************************/
/*           function.c               */
/* (Sample file for walkthrough)      */
/**************************************/
int main(int i)
{
   return(i < 0 ? –i : i );
}
```

2) To invoke the shell program to compile and assemble function.c, enter:

```
cl470 –o function ⏎
```

By default, the TMS470R1x shell program compiles and assembles 32-bit instructions. To compile 16-bit instructions, use the –mt option:

```
cl470 –o –mt funtion⏎
```

The shell program prints the following information as it compiles the program:

```
[function]
TMS470 ANSI C Compiler     Version 1.20
Copyright (c) 1995–1997    Texas Instruments Incorporated
   ”function.c”   ==> main
TMS470 ANSI C Optimizer    Version 1.20
Copyright (c) 1995–1997    Texas Instruments Incorporated
   ”function.c”   ==> main
TMS470 ANSI C Codegen      Version 1.20
Copyright (c) 1995–1997    Texas Instruments Incorporated
   ”function.c”:  ==> main
TMS470 COFF Assembler      Version 1.20
Copyright (c) 1995–1997    Texas Instruments Incorporated
 PASS 1
 PASS 2

 No Errors, No Warnings
```

By default, the shell deletes the assembly language file from the current directory after the file is assembled. If you want to inspect the assembly language output, use the –k option to retain the assembly language file:

```
cl470 –o –k function ⏎
```

3) Also by default, the shell creates a COFF object file as output; however, if you use the –z option, the output is an *executable* object module. The following examples show two ways of creating an executable object module:

   a) The example in step 2 creates an object file called function.obj. To create an executable object module, run the linker separately by invoking lnk470 as in the following example:

```
lnk470 –c function.obj lnk32.cmd –o function.out –l rts32.lib
```

   The –c linker option tells the linker to observe the C language linking conventions. The linker command file, lnk32.cmd, is shipped with the code generation tools. The –o option names the output module, function.out; if you don't use the –o option, the linker names the output module a.out. The –l option names the runtime-support library. You must have a runtime-support library before you can create an executable object module; the prebuilt runtime-support libraries rts32.lib and rts16.lib are included with the code generation tools.

   b) In this example, use the –z shell option, which tells the shell program to run the linker. The –z option is followed by linker options.

```
cl470 –o function.c –z lnk32.cmd –o function.out –l rts32.lib
```

   For more information on linker commands, see the *Linker Description* chapter of the *TMS470R1x Assembly Language Tools User's Guide.*

4) The TMS470R1x compiler package also includes an *interlist utility*. This program interlists the C source statements as comments in the assembly language compiler output, allowing you to inspect the assembly language generated for each line of C. To run the interlist utility, invoke the shell program with the –s option. For example:

```
cl470 –s function –z lnk32.cmd –o function.out
```

   The output of the interlist utility is written to the assembly language file created by the compiler. (The shell –s option implies –k; that is, when you use the interlist utility, the assembly file is automatically retained.)

Chapter 6

Release Notes

This chapter describes the media contents of the TMS470R1x tools kit. The tools are supported on SPARCstations, HP workstations, and PCs with DOS, Windows 3.1x, Windows 95, or Windows NT.

This chapter also contains documentation of tools and features that are new or have been changed since the last release.

Topic **Page**

6.1 Media Contents

The CD-ROM included in the TMS470R1x tools kit for SPARCstations and HP workstations contains the files listed in Table 6–1. The CD-ROM included in the TMS470R1x tools kit for PCs contains the files listed in Table 6–2.

Table 6–1. Media Contents for SPARCstations and HP Workstations

File	Description			
README.1st	Online release bulletin			
abs470	Absolute lister			
ac470	ANSI C parser			
ar470	Archiver			
asm470	Assembler			
cg470	Code generator			
cl470	Compiler shell program			
clist	C source interlist utility			
hex470	Hex conversion utility			
intvecs.asm	Sample interrupt vector setup file			
lnk470	COFF linker			
lnk16.cmd	Sample 16-bit linker command file			
lnk32.cmd	Sample 32-bit linker command file			
mk470	Library-build utility			
opt470	C optimizer			
rts16.lib	16-bit runtime-support library			
rts32.lib	32-bit rutime-support library			
rts.src	C runtime-support source library			
xref470	Cross-reference utility			
*.h	#include header files for RTS:			
	assert.h	limits.h	stdarg.h	stdlib.h
	ctype.h	math.h	stddef.h	string.h
	errno.h	setjmp.h	stdio.h	time.h
	float.h			

Table 6–2. Media Contents for PCs

File	Description			
readme.1st	Online release bulletin			
abs470.exe	Absolute lister			
ac470.exe	ANSI C parser			
ar470.exe	Archiver			
asm470.exe	Assembler			
cg470.exe	Code generator			
cl470.exe	Compiler shell program			
clist.exe	C source interlist utility			
hex470.exe	Hex conversion utility			
intvecs.asm	Sample interrupt vector setup file			
lnk470.exe	COFF linker			
lnk16.cmd	Sample 16-bit linker command file			
lnk32.cmd	Sample 32-bit linker command file			
mk470.exe	Library-build utility			
opt470.exe	C optimizer			
pminfo32.exe†	Utility to measure protected/real-mode switching			
rts16.lib	16-bit runtime-support library			
rts32.lib	32-bit runtime-support library			
rts.src	C runtime-support source library			
xref470.exe	Cross-reference utility			
*.h	#include header files for RTS:			
	assert.h	limits.h	stdarg.h	stdlib.h
	ctype.h	math.h	stddef.h	string.h
	errno.h	setjmp.h	stdio.h	time.h
	float.h			

† This file is included only in the tool kit for PCs running DOS.

6.2 Release Enhancements

Release 1.20 of the TMS470R1x code generation tools contains general enhancements as well as enhancements specific to the assembler and compiler. The following sections list these enhancements.

General enhancements

The TMS470R1x general enhancements include the following items:

❑ All known bugs have been removed.

❑ COFF version 2 is supported. Section names are no longer limited to eight characters, and filenames are no longer limited to 14 characters. Section name length is unrestricted, and filename length is restricted by the conventions of your operating system.

Note: Using Version 1 COFF

If you depend on a third-party COFF-dependent tool that does not support version 2 COFF, then use the –v1 linker option. The –v1 option instructs the linker to generate version 1 COFF. Ensure that your object modules do not have section names that are longer than eight characters or filenames that are longer than 14 characters. The –v1 option is a temporary option for this release and it will be removed in a future release.

Assembler enhancements

The TMS470R1x assembler enhancements are as follows:

❑ A feature of COFF version 2 is the use of subsections. You can create subsections of any section to give you tighter control of the memory map. Subsections are created using the .sect and .usect directives. For information on creating subsections, see the *Introduction to Common Object File Format* chapter of the *TMS470R1x Assembly Language Tools User's Guide*; for memory map examples using subsections, see the *Linker Description* chapter.

❑ The assembler supports conditionally linked sections. You select sections for conditional linking with the .clink assembler directive.

  The .clink directive marks the current section as a conditionally linked section. This section will be linked into the final output of the linker only if it is referenced by a linked section through a symbol reference. By default, all sections are linked.

The section in which the entry point of a C program is defined cannot be marked as a conditionally linked section. The compiler marks all function veneers as conditionally linked.

You must link with the –a linker option, which creates an absolute, executable output module, to enable conditional linking. The –j linker option disables conditional linking.

For information on the .clink directive, see the *Assembler Directives* chapter of the *TMS470R1x Assembly Language Tools User's Guide*; for information on the –j linker option, see the *Linker Description* chapter.

C compiler enhancements

The TMS470R1x optimizing C compiler enhancements are as follows. For more information, see the named chapter of the *TMS470R1x Optimizing C Compiler User's Guide*.

- ❑ You can specify command line options in a file. When you specify that file on the command line with the –@ shell option, the compiler reads the file and interprets it as if it contained part of the command line. For more information, see the *Compiler Description* chapter.
- ❑ The options that invoke the interlist utility with the shell have changed. The –s option interlists optimizer comments (if the optimizer is invoked) and assembly statements; otherwise it interlists C source and assembly statements. The –ss option interlists C source and assembly statements. A new option, the –os option, interlists optimizer comments with assembly statements. To interlist C source and assembly statements with optimizer comments, use the –os and –ss options when invoking the optimizer. For more information, see the *Compiler Description* and *Optimizing Your Code* chapters.
- ❑ Global register variables are now supported. For more information, see the *TMS470R1x C Language* chapter.
- ❑ Bit-field manipulation improvements are incorporated. For more information, see the *Runtime Environment* chapter.
- ❑ The alignment of structures has changed. In previous releases, all structures were word aligned. Now, a structure is aligned according to the alignment of the structure's most restrictive element. For more information, see the *Runtime Environment* chapter.
- ❑ Software interrupts are supported for C code. You define the software interrupt with the SWI_ALIAS pragma. For more information, see the *TMS470R1x C Language* and *Runtime Environment* chapters.

Appendix A

Troubleshooting DOS Systems

DOS/4GW™ is a memory manager that is embedded into the DOS version of the TMS470R1x code generation tools, so you may occasionally see DOS/4GW error messages while you are using the tools. The executable files for DOS/4GW are not shipped as such, nor is any documentation provided on this tool, except for the list of error messages.

Section A.2, *Kernel Error Messages*, and Section A.3, *DOS/4G Error Messages*, are excerpted from the *DOS/4GW User's Manual* (reproduced here with the permission of Tenberry Software, Inc). Included are lists of error messages with descriptions of the circumstances in which the error is most likely to occur and suggestions for remedying the problem. (Portions of the excerpt have been modified to provide you with specific information about using TI tools.)

Topic **Page**

A.1 Troubleshooting in the Protected-Mode Environment

Getting 32-bit programs to execute properly under DOS can be frustrating. Your computer's configuration and memory management can cause problems that may be difficult to find because many programs are interacting.

This list of error messages is reproduced in Section A.2 on page A-5 and Section A.3 on page A-9 because they may occur when you are executing any tools, since all of the tools have been assembled along with the DOS/4GW memory extender. When reading this material, consider these items:

- ❑ When an *Action* directs you to technical support, determine the configuration of your system by using the **PMINFO** program (on page A-3) before contacting technical support. See the *If You Need Assistance* section on page vi, for information about contacting technical support.
- ❑ Some error messages are not included in this appendix because they are rarely seen when using DOS/4GW with the TMS470R1x tools. Also, many of the messages that *are* documented here are seldom seen when using DOS/4GW with the TMS470R1x tools.

The PMINFO32.EXE program

Purpose: Run PMINFO.EXE to determine the performance of protected/real-mode switching and extended memory.

Notes: The time-based measurements made by PMINFO may vary slightly from run to run.

If this error message appears:

DOS/16M error: [17] system software does not follow VCPI or DPMI specifications

check for a statement in your CONFIG.SYS containing **NOEMS**. If such a statement exists, remove it and reboot your computer.

If the computer is not equipped with extended memory or if none is available for DOS/4GW, the extended-memory measurements do not display.

Other DOS/4GW error messages are in Section A.3, *DOS/4G Error Messages*.

Example: The following example shows the output of the PMINFO program on an 80486 AT™-compatible machine running at 33 MHz.

```
-================================= PMINFO =======================================-

           Protected Mode and Extended Memory Performance Measurement –– 4.45
                   Copyright (c) Tenberry Software, Inc. 1987 – 1996

      DOS memory   Extended memory   CPU performance equivalent to 33.0 MHz 80486
      ----------   ---------------
             640             17854   K bytes configured (according to BIOS).
             640             31744   K bytes physically present (SETUP).
             550             17585   K bytes available for DOS/16M programs.
      21.6 (0.0)        19.1 (0.5)   MB/sec word transfer rate (wait states).
      35.4 (0.5)        34.4 (0.5)   MB/sec 32–bit transfer rate (wait states).

      Overall cpu and memory performance (non–floating point) for typical
      DOS programs is 7.78 ± 0.62 times an 8MHz IBM PC/AT.

      Protected/Real switch rate = 18078/sec (55 µsec/switch, 33 up + 21 down),
      DOS/16M switch mode 11 (VCPI).
```

PMINFO provides the information shown in Table A–1.

Table A–1. PMINFO Fields

Measurement	**Purpose**
CPU performance	Shows the CPU processor equivalent and the speed of the CPU (in MHz)
According to BIOS	Shows the configured memory in DOS and extended memory as provided by the BIOS (interrupts 12h and 15h, function 88h)
SETUP	Shows the configuration obtained directly from the CMOS RAM as set by the computer's setup program. It is displayed only if the numbers are different from those in the BIOS line. They are different if the BIOS has reserved memory for itself or if another program has allocated memory and is intercepting the BIOS configuration requests to report less memory available than is physically configured.
DOS/16M programs	If displayed, shows the low and high addresses available to DOS/4GW in extended memory
Transfer rates	PMINFO tries to determine the memory architecture. Some architectures perform well under some circumstances and poorly under others; PMINFO shows the best and worst cases. The architectures detected are cache, interleaved, page-mode (or static column), and direct.
	Measurements are made by using 32-bit accesses and are reported as the number of megabytes per second that can be transferred. The number of wait states is reported in parentheses. The wait states can be a fractional number, like 0.5, if there is a wait state on writes but not on reads. Memory bandwidth (that is, how fast the CPU can access memory) accounts for 60% to 70% of the performance for typical programs (those that are not heavily dependent on floating-point math).
Overall CPU and memory performance	Shows a performance metric developed by Tenberry Software, Inc. (formerly known as Rational Systems, Inc.), indicating the expected throughput for the computer relative to a standard 8-MHz IBM PC/AT (disk accesses and floating-point operations are both excluded).
Protected/real switch rate	Shows the speed with which the computer can switch between real and protected modes, both as the maximum number of round-trip switches that can occur per second, and as the time for a single round-trip switch, broken into the real-to-protected (up) and protected-to-real (down) components.

A.2 Kernel Error Messages

This section describes error messages from the DOS/16M kernel embedded in the TMS470R1x code generation tools. Kernel error messages can occur because of severe resource shortages, corruption of the executable file, corruption of memory, operating system incompatibilities, or internal errors. All of these messages are quite rare.

DOS/16M protected mode available only with 386 or 486

Description	DOS4G did not detect the presence of a 386-, 486-, or Pentium-based processor.
Action	If you are running the tools on a 386 (or later) PC, rerun the program. If you are running the tools on a 286 PC, reinstall and run the tools on a 386 PC or later.

0: involuntary switch to real mode

Description	The computer was in protected mode but switched to real mode without going through DOS/16M. This error most often occurs because of an unrecoverable stack segment exception (stack overflow) but can also occur if the global descriptor table or interrupt descriptor table is corrupted.
Action	Restart your computer. If the problem persists, contact technical support.

2: not a DOS/16M executable <filename>

Description	DOS4G.EXE or a bound DOS/4G application has probably been corrupted in some way.
Action	Recopy the file from the source media.

6: not enough memory to load program

Description	There is not enough memory to load DOS/4G.
Action	Make more memory available and try again.

8: cannot open file <filename>

Description	The DOS/16M loader cannot load DOS/4G, probably because DOS has run out of file units.
Action	Set a larger FILES= entry in the CONFIG.SYS file, reboot, and try again.

9: cannot allocate tstack

Description	There is not enough memory to load DOS/4G.
Action	Make more memory available and try again.

10: cannot allocate memory for GDT

Description	There is not enough memory to load DOS/4G.
Action	Make more memory available and try again.

11: no passup stack selectors – GDT too small

Description	This error indicates an internal error in DOS/4G or an incompatibility with other software.
Action	Contact technical support.

12: no control program selectors – GDT too small

Description	This error indicates an internal error in DOS/4G or an incompatibility with other software.
Action	Contact technical support.

13: cannot allocate transfer buffer

Description	There is not enough memory to load DOS/4G.
Action	Make more memory available and try again.

14: premature EOF

Description	DOS4G.EXE or a bound DOS/4G application has probably been corrupted.
Action	Recopy the file from the source media.

15: protected mode available only with 386 or 486

Description	DOS/4G requires an 80386 (or later) CPU. It cannot run on an 80286 (or earlier) CPU.
Action	Reinstall and run the tools on a 386 (or later) PC.

17: system software does not follow VCPI or DPMI specifications

Description	Some memory-resident program has put your 386 or 486 CPU into virtual 8086 mode. This is done to provide special memory services to DOS programs, such as EMS simulation (EMS interface without EMS hardware) or high memory. In this mode, it is not possible to switch into protected mode unless the resident software follows a standard that DOS/16M supports (DPMI, VCPI, and XMS are the most common).
Action	Contact the vendor of your memory-management software.

22: cannot free memory

Description	Memory was probably corrupted during execution of your program.
Action	Make more memory available and try again.

23: no memory for VCPI page table

Description	There is not enough memory to load DOS/4G.
Action	Make more memory available and try again.

24: VCPI page table address incorrect

Description	There is an internal error.
Action	Contact technical support.

25: cannot initialize VCPI

Description	An incompatibility with other software was detected. DOS/16M has detected that VCPI is present, but VCPI returns an error when DOS/16M tries to initialize the interface.
Action	Find the other software that uses VCPI and disable it (stop its execution).

28: memory error, avail loop

Description	Memory was probably corrupted during execution of your program. Using an invalid or stale alias selector may cause this error. Incorrect manipulation of segment descriptors may also cause it.
Action	Rerun the program and/or restart your computer.

29: memory error, out of range

Description	Memory was probably corrupted during execution of your program. Writing through an invalid or stale alias selector may cause this error.
Action	Check your source code for references to variables that are not declared or are no longer in scope.

32: DPMI host error (possibly insufficient memory)
33: DPMI host error (need 64K XMS)
34: DPMI host error (cannot lock stack)

Description	Memory under DPMI is probably insufficient.
Action	Under Windows, make more physical memory available by eliminating or reducing any RAM drives or disk caches. You can also edit DEFAULT.PIF so that at least 64K bytes of XMS memory is available to non-Windows programs. Under OS/2, increase the DPMI_MEMORY_LIMIT in the DOS box settings.

35: general protection fault

Description	An internal error in DOS/4G was probably detected. Faults generated by your program should cause error 2001 instead.
Action	Contact technical support.

38: cannot use extended memory: HIMEM.SYS not version 2

Description	An incompatibility with an old version of HIMEM.SYS was detected.
Action	Upgrade to a more recent copy of DOS or upgrade your DOS memory extender.

40: not enough available extended memory (XMIN)

Description	An incompatibility with your memory manager or its configuration was detected.
Action	Try configuring the memory manager to provide more extended memory or change memory managers.

A.3 DOS/4G Error Messages

DOS/4G errors are more common than kernel errors when using DOS/4G or DOS/4GW with the TMS470R1x code generation tools. They are usually related to an unknown path name, corrupt files, or memory problems. Memory problems can include inadequate memory, poor configuration, or corrupted memory.

1000 ”can’t hook interrupts”

Description	A DPMI host has prevented DOS/4G from loading.
Action	Contact technical support.

1001 ”error in interrupt chain”

Description	A DOS/4G internal error was detected.
Action	Contact technical support.

1003 ”can’t lock extender kernel in memory”

Description	DOS/4G could not lock the kernel in physical memory, probably because of a memory shortage.
Action	Free some memory for the DOS/4G application.

1005 ”not enough memory for dispatcher data”

Description	There is not enough memory for DOS/4G to manage user-installed interrupt handlers properly.
Action	Free some memory for the DOS/4G application.

1007 ”can’t find file <program> to load”

Description	DOS/4G could not open the specified program. The file probably does not exist. It is possible that DOS ran out of file handles or that a network or similar utility has prohibited read access to the program.
Action	Make sure that the filename was spelled correctly.

1008 ”can’t load executable format for file <filename> [<error code>]”

Description	DOS/4G did not recognize the specified file as a valid executable file. DOS/4G can load linear executables (LE and LX) and EXPs (BW).
Action	Recopy the file from the source media.

3301 ”unhandled EMPTYFWD, GATE16, or unknown relocation”
3302 ”unhandled ALIAS16 reference to unaliased object”
3304 ”unhandled or unknown relocation”

Description	If your program was built for another platform that supports the LINEXE format, it may contain a construct that DOS/4G does not currently support, such as a call gate. One of these messages may also appear if your program has a problem mixing 16- and 32-bit code. A linker error is another likely cause.
Action	Check for viruses and reinstall the tools from the source media. If the problem persists, contact technical support.

Appendix B

Glossary

A

asm470: The name of the command that invokes the assembler for the TMS470R1x.

assembler: A software program that creates a machine-language program from a source file that contains assembly language instructions, directives, and macro definitions. The assembler substitutes absolute operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.

C

C compiler: A software program that translates C source statements into assembly language source statements.

code generator: A compiler tool that takes the file produced by the parser or the optimizer and produces an assembly language source file.

COFF: Common object file format. A binary object file format that promotes modular programming by supporting the concept of *sections*. All COFF sections are independently relocatable in memory space; you can place any section into any allocated block of target memory.

cl470: The name of the command that invokes the compiler shell program for the TMS470R1x. (The second character in the shell name is a lowercase L.)

D

DOS/4G: The base version for DOS/4GW. You may occasionally see this term in an error message. If so, see Appendix A, *Troubleshooting DOS Systems*, for the appropriate action.

DOS/4GW: A memory extender that is bound with the DOS version of the TMS470R1x code generation tools. *The executable DOS/4GW file is not shipped separately but is embedded within the other executables*. Error messages from DOS/4GW are included in Appendix A, *Troubleshooting DOS Systems*, to assist you in debugging. If you receive one of these error messages, contact technical support for assistance, and remember that the tools are shipped as object files with the memory extender embedded.

DOS/16M: The executable filename for a tool that is embedded in the TMS470R1x code generation tools. You may occasionally see this term in an error message. If so, see Appendix A, *Troubleshooting DOS Systems*, for the appropriate action.

E

environment variables: System symbols that you define and assign to a string. They are usually included in batch files (for example, .cshrc).

I

interlist utility: A compiler utility that inserts as comments your original C source statements into the assembly language output from the assembler. The C statements are inserted next to the equivalent assembly instructions.

L

linker: A software program that combines object files to form an object module that can be allocated into system memory and executed by the device.

lnk470: The name of the command that invokes the linker for the TMS470R1x.

O

optimizer: A software tool that improves the execution speed and reduces the size of C programs.

options: Command parameters that allow you to request additional or specific functions when you invoke a software tool.

P

pragma: Preprocessor directive that provides directions to the compiler about how to treat a particular statement.

protected-mode programs: 32-bit extended DOS programs. These programs require an extended memory manager and run on 80386, 80486, and Pentium based PCs only. Protected-mode programs can use all available RAM on a computer up to 64M bytes.

R

real-mode: 16-bit native DOS mode. This mode limits the available memory to 640K bytes. Calls to DOS may involve switching from protected to real mode. *DOS real-mode tools are no longer supported by the TMS470R1x code generation tools.*

S

shell program: A utility that lets you compile, assemble, and optionally link in one step. The shell runs one or more source modules through the compiler (including the parser, optimizer, and code generator), the assembler, and the linker.

structure: A collection of one or more variables grouped together under a single name.

swap file: The file where virtual memory (secondary memory) is allocated on the hard disk.

V

veneer: A sequence of instructions that serves as an alternate entry point into a routine if a state change is required.

virtual memory: The ability of a program to use more memory than a computer actually has available as RAM. This is accomplished by using a swap file on disk to augment RAM. When RAM is not sufficient, part of the program is swapped out to a disk file until it is needed again. The combination of the swap file and available RAM is the virtual memory. The TMS470R1x tools use the DOS/4GW memory extender to provide virtual-memory management (VMM). This memory extender is not provided as an executable file but is embedded in several of the tools shipped by TI. Contact technical support for more information.

Index