TEXAS INSTRUMENTS

TMS320C1x/C2x/C2xx/C5x Code Generation Tools

Release 6.60

Getting Started

1995 Microprocessor Development Systems

Printed in U.S.A., March 1995
D415001-9741 revision A

TMS320C1x/C2x/C2xx/C5x Code Generation Tools Getting Started

Release 6.60

TEXAS INSTRUMENTS

Printed on Recycled Paper

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales offices.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

IBM, PC, PC-DOS, and OS/2 are trademarks of International Business Machines Corp.

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

SPARC is a trademark of SPARC International, Inc.

SunOS, SunView, SunWindows, and Sun Workstation are trademarks of Sun Microsystems, Inc.

UNIX is a registered trademark of Unix System Laboratories, Inc.

Contents

Chapter 1

Getting Started With the Code Generation Tools

This package contains version 6.60 of the TMS320C1x/C2x/C2xx/C5x Code Generation Tools. The assembly language tools support the TMS320C1x/C2x/C2xx/C5x devices as defined in Section 1.1. The C compiler supports the TMS320C2x/C2xx/C5x devices as defined in Section 1.1.

These code generation tools can be installed on the following systems:

- 80386 PC or later with MS-DOS™ or OS/2™ v2.x and at least 4 Mbytes of RAM (16 Mbytes of RAM recommended to minimize performance impact)
- SPARC™ workstation with SunOS™ versions 4.1.x and 5.x (Solaris 2.x)

This document covers installation, system-specific notes, and release notes.

Topic **Page**

1.1 Introduction

The assembly language tools are composed of the following:

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

The C compiler tools include:

- Compiler shell program
- Code generator
- Interlist utility
- Library-build utility
- Optimizer
- Parser
- Miscellaneous libraries

The TMS320C1x/C2x/C2xx/C5x code generation tools can be installed on a SPARC workstation with SunOS versions 4.1x and 5.x (Solaris 2.x) or on a PC running DOS or OS/2, 80386 or later. They are not compatible with the 80286.

The code generation tools, when installed on a PC, require at least 4 Mbytes of memory, but you can expect some performance problems when using only 4 Mbytes (we recommend 16 Mbytes). If the tools run very slowly or hang, the problem may be due to insufficient memory.

Each machine configuration is unique. You may wish to free as much memory as possible before installing the tools, especially if you have less than 16 Mbytes.

Fixed-Point Devices Supported by the Assembly Language Tools

The assembly language tool set provides support for the upwardly compatible TMS320C1x/C2x/C2xx/C5x DSP processors, which are listed below:

TMS320C1x (or ’C1x) refers to the following devices*:*

TMS320C10			
TMS320C14	TMS320E14	TMS320P14	
TMS320C15	TMS320E15	TMS320P15	TMS320LC15
TMS320C16			
TMS320C17	TMS320E17	TMS320P17	TMS320LC17

TMS320C2x (or ’C2x) refers to the following devices:

TMS320C25	TMS320C26	TMS320E25

TMS320C2xx (or ’C2xx) refers to the following device:

TMS320C209

TMS320C5x (or ’C5x) refers to the following devices:

TMS320C50 TMS320C51 TMS320C52 TMS320C53
TMS320C56 TMS320C57

Fixed-Point Devices Supported by the C Compiler

The TMS320C2x/C2xx/C5x optimizing C compiler tool set provides support for the upwardly compatible ’C2x, ’C2xx, and ’C5x DSP processors, which are listed below:

TMS320C2x (or ’C2x) refers to the following devices:

TMS320C25 TMS320C26 TMS320E25

TMS320C2xx (or ’C2xx) refers to the following device:

TMS320C209

TMS320C5x (or ’C5x) refers to the following devices:

TMS320C50 TMS320C51 TMS320C52 TMS320C53
TMS320C56 TMS320C57

Terminology

***DOS/16M*:** executable filename for a tool that is embedded in the TMS320C1x/C2x/C2xx/C5x code generation tools. You may occasionally see this term in an error message. If so, refer to the listing of appropriate actions in Appendix A.

***DOS/4G*:** the base version for DOS/4GW. You may occasionally see this term in an error message. If so, refer to the listing of appropriate actions in Appendix A.

***DOS/4GW*:** a memory extender that is assembled along with the TMS320C1x/C2x/C2xx/C5x tools before shipment. Error messages from DOS/4GW are included in Appendix A of this manual to assist you in debugging. *The executable DOS/4GW file is not shipped separately but is embedded within the other executables.* When you look at the DOS/4GW error messages, remember to call TMS320 technical support if you need assistance. When an error message suggests rebuilding, reload the tools instead.

***environment variables*:** system symbols that you define and assign to a string. They are usually included in various batch files; for example, in AUTOEXEC.BAT. For more information, see Section 1.2.

***protected-mode programs*:** 32-bit extended MS-DOS programs. These programs require an extended memory manager and run only on 80386-, 80486-, and Pentium-based PCs. Protected-mode programs can utilize all available RAM on the computer.

***real mode*:** 16-bit native MS-DOS mode. This mode limits the available memory to 640K bytes. Calls to DOS may involve switching from protected to real mode. The real-mode tools are no longer supported in this release.

***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 TMS320C1x/C2x/C2xx/C5x tools use the DOS/4GW memory extender to provide virtual memory management. This memory extender is not provided as an executable file but is embedded in several object programs shipped by TI. Contact technical support for more information.

To obtain TMS320C1x/C2x/C2xx/C5x technical support, contact the DSP hotline:

DSP Hotline:	**(713) 274–2320**
FAX:	**(713) 274–2324**
Electronic Mail:	**4389750@mcimail.com.**

1.2 Environment Variables

When installing the tools on any appropriate machine, it is possible to define *environment variables* that set certain code generation tool parameters you will normally use. An environment variable is a system symbol that you define and assign to a string. When you use environment variables, default values are set, making each individual invocation of the tools simpler because these parameters need not be considered. Many of these defaults, when set using environment variables, may be overridden with an individual invocation of the tool.

The A_DIR Environment Variable

The assembler uses the environment variable **A_DIR** to name alternate directories that contain copy/include files or macro libraries. The command for assigning the environment variable is as follows:

Host	Enter
DOS or OS/2	`set  A_DIR=pathname;another pathname ...`
UNIX	`setenv A_DIR ″pathname;another pathname ... ″`

The *pathnames* are directories that contain copy/include files or macro libraries. You can separate the pathnames with a semicolon or a blank. In assembly source, you can use the .copy, .include, or .mlib directive without specifying any path information. If the assembler doesn't find the file in the directory that contains the current source file or in directories named by –i, it searches the paths named by the environment variable.

For example, assume that a file called source.asm contains these statements:

```
.copy ″copy1.asm″
.copy ″copy2.asm″
```

The files are stored in the directories shown below. The search path is set up with the commands shown in the table:

Host	Pathname	Enter
DOS or OS/2	`c:\320\files\copy1.asm` `c:\dsys\copy2.asm`	`set  A_DIR=c:\dsys` `dspa –ic:\320\files source.asm`
UNIX	`/320/files/copy1.asm` `/dsys/copy2.asm`	`setenv  A_DIR /dsys` `dspa  –i/320/files source.asm`

The assembler first searches for copy1.asm and copy2.asm in the current directory because source.asm is in the current directory. Then the assembler searches in the directory named with the –i option and finds copy1.asm. Finally, the assembler searches in the directory named with A_DIR and finds copy2.asm.

Note that the environment variable remains set until you reboot the system or reset the variable by entering one of these commands:

Host	Enter
DOS or OS/2	`set A_DIR=`
UNIX	`unsetenv A_DIR`

The C_DIR Environment Variable

The compiler uses the environment variable **C_DIR** to name alternate directories that contain #include files and libraries. For example, to specify a directory for #include files and libraries, set C_DIR with one of these commands:

Host	Enter
DOS or OS/2	`set C_DIR=c:\dsp\files`
UNIX	`setenv C_DIR "/dsp/files"`

Then you can include alt.h, found in the *files* directory, in this way:

```
#include "alt.h" or
#include <alt.h>
```

and invoke the compiler without the –i option:

```
dspcl source.c
```

This results in the compiler using the path in the environment variable to find the #include file.

The pathnames specified with C_DIR are directories that contain #include files. You can separate pathnames with a semicolon or with a blank. In C source, you can use the #include directive without specifying any path information; instead, you can specify the path information with C_DIR.

The environment variable remains set until you reboot the system or reset the variable by entering one of these commands:

Host	Enter
DOS or OS/2	`set C_DIR=`
UNIX	`unsetenv C_DIR`

The C_OPTION Environment Variable

You may find it useful to set the dspcl compiler, assembler, and linker shell default options using the C_OPTION environment variable; if you do so, these default options and/or input filenames are used every time you run the shell.

Setting up default options with the C_OPTION environment variable is especially 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 entire command line and the input filenames, it reads the C_OPTION environment variable and processes it.

Options specified with the environment variable 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, enable symbolic debugging, and link, set up the C_OPTION environment variable as follows:

Host	Enter
DOS or OS/2	`set C_OPTION=-qg -z`
UNIX	`setenv C_OPTION "-qg -z"`

Using the –z option enables linking. If you plan to link most of the time when using the shell, you can specify the –z option with C_OPTION. Later, if you need to invoke the shell without linking, you can use –c on the command line to override the –z specified with C_OPTION. These examples assume C_OPTION is set as shown previously:

```
dspcl *.c             ; compiles and links
dspcl -c *.c          ; only compiles
dspcl *.c -z c.cmd    ; compiles/links with command file
dspcl -c *.c -z c.cmd ; only compiles (-c overrides -z)
```

The DOS4GVM Environment Variable (MS-DOS Hosts Only)

Virtual memory management (VMM) allows protected-mode programs to use more RAM than your computer actually has. The DOS4GVM environment variable is used to control VMM. The DOS4GVM environment variable is set using the following format:

set DOS4GVM=[*option*[**#***value*]] [*option*[**#***value*]] ...

You must use # with options that take values; the DOS command shell will not accept an equal sign in place of #.

Setting **DOS4GVM** equal to 1 will cause the default values to be used for all options. For example:

```
set DOS4GVM=1
```

The **DOS4GVM** options, with their default values, are:

MINMEM The minimum amount of RAM managed by VMM. The default is 512K bytes.

MAXMEM The maximum amount of RAM managed by VMM. The default is 4 Mbytes.

SWAPMEM The minimum or initial size of the swap file. If this option is not used, the size defaults to **VIRTUALSIZE**, which defaults to 16 Mbytes.

SWAPINC The size by which the swap file grows. The default size is 64K bytes.

SWAPNAME The swap file name. The default name is DOS4GVM.SWP. By default, the file is in the root directory of the current drive. Specify the complete pathname if you want to keep the swap file somewhere else.

DELETESWAP Indicates whether the swap file is deleted when your program exits. Program start-up is quicker if the file is not deleted. Including DELETESWAP in the option string deletes the swap file. Omitting it, which is the default setting, does not delete the swap file.

VIRTUALSIZE The size of the virtual memory space. The default is 16 Mbytes.

You can initialize DOS4GVM in two ways:

1) Specify parameter values as arguments to the **DOS4GVM** environment variable, as shown in the example below.

   ```
   set DOS4GVM=deleteswap maxmem#8192 swapname#c:\swap.tmp
   ```

   This fixes the location of the swap file to a known location, sets the swap file size, and deletes the swap file when the application completes execution. With the default settings, a 16-Mbyte swap file is created on each logical device on which you compile files (C:, D:, etc.), and the swap file is *not* deleted when the compiler completes execution. For optimum performance (and safety), locate the swap file on your local hard drive, not on a network drive.

2) Create a configuration file with the file-type extension .VMC, and call it as an argument to the **DOS4GVM** environment variable, as shown below.

   ```
   set DOS4GVM=@NEW4G.VMC
   ```

   A .VMC file contains VMM parameters and settings as shown in the example below. Comments are permitted. Comments on lines by themselves are preceded by an exclamation point (!). Comments that follow option settings are preceded by white space. Do not insert blank lines; processing stops at the first blank line.

```
!Sample .VMC file
!This file shows the default parameter values
minmem = 512          At least 512 bytes of RAM required
maxmem = 4096         Uses no more than 4MB of RAM
virtualsize = 16384   Swap file + allocated memory is 16Mbytes
swapname = c:\swap.tmp
delete swap
```

See Appendix A for information on problems and solutions when using DOS/4GW.

The DOS4G Environment Variable (MS-DOS Hosts Only)

When you invoke a tool, you will see something like this:

```
DOS/4GW Professional Protected Mode Run-time  Version 1.96
Copyright (c) Rational Systems, Inc. 1990-1994
DSP Fixed Point COFF Assembler Version 6.60
Copyright (c) 1987-1994  Texas Instruments Incorporated
```

This DOS/4GW banner will appear each time that tool starts, because DOS/4GW is embedded in each tool. This banner can be suppressed by adding this line to the AUTOEXEC.BAT file:

```
SET DOS4G=quiet
```

The TMP Environment Variable

The shell program creates intermediate files as it processes your program. For example, the parser phase of the shell creates a temporary file used as input by the code generation phase. 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.

This feature allows the 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, enter one of these commands:

Host	Enter
DOS or OS/2	`set TMP=c:\temp`
UNIX	`setenv TMP "/temp"`

Setting Environment Variables in Your System Initialization File

You may want to set environment variables in your system initialization file. To do so, the same commands that would be entered on the command line should be entered as a line in the system initialization file. The name for the file for your operating system is in the table below.

Host	Filename
DOS	`AUTOEXEC.BAT`
OS/2	`CONFIG.SYS`
UNIX	`.profile`

1.3 Using the Code Generation Tools on SPARC Hosts

Version 6.60 of the TMS320C1x/C2x/C2xx/C5x code generation tools is compatible with SunOS versions 4.1.x and 5.x (Solaris 2.x) from Sun Microsystems, Inc. This release is dynamically linked to take advantage of shared libraries.

Note: SPARC Hosts

To use the code generation tools with Solaris 2.0, you must install the *Binary Compatibility Package (BCP)*.

Installation

The TMS320C1x/C2x/C2xx/C5x product tape is in TAR format. Follow these instructions to install the code generation tools package:

1) Mount the tape on your tape drive.

2) Make sure the current directory is the directory that you'll store the tools in, or change to that directory.

3) Enter the TAR command for your system; for example:

   `tar x`

   If you are working on a system that has the tape drive as the default, this command will copy the entire tape into the directory. Some systems do not default to the tape drive. For example, if you are using a Sun workstation, you would use the following command:

   `tar xvf /dev/rst8`

   This command extracts all files from device rst8 and copies them into your directory. The v option displays the name of each file as it is extracted.

4) You may wish to set environment variables in your shell startup file (for example, .cshrc, .login, or .profile) to make it easier to invoke the tools. See Section 1.2, page 1-5, for more information.

1.4 Using the Code Generation Tools on OS/2 Hosts

Version 6.60 of the TMS320C1x/C2x/C2xx/C5x code generation tools supports OS/2 v2.x releases. The code generation tools can be invoked from an OS/2 command session (cmd.exe).

Installation

The code generation tools package is shipped on 1.44-Mbyte disks. An OS2V2 directory contains the OS/2 executables on the product disks. To install the tools on an OS/2 system, perform the following steps:

1) Make backups of the product disks.

2) Start an OS/2 command session.

3) Create a directory to contain the code generation tools. Enter the following:

```
md c:\320tools
```

4) Copy the OS2V2 directory from each diskette onto your hard disk. Put each of the product disks in drive A and enter the following:

```
copy a:\os2v2\*.* c:\320tools
```

5) You may wish to set environment variables in your config.sys file. Refer to Section 1.2, page 1-5, for more information.

Note: About MS-DOS Tools

For OS/2 systems, do not copy the MS-DOS tools onto your hard disk. The MS-DOS tools are contained in the dos32 directories of the product disks. The copy command, as shown in step 3, will properly install only the OS/2 tools onto your hard disk.

The MS-DOS tools can be run with OS/2, but running them will require tuning DOS settings in OS/2 v2.x to enable larger amounts of DPMI (DOS protected memory interface) memory. The OS/2 default setting for DOS tools is 4 Mbytes, and some applications require a larger allocation of memory to compile successfully. It is recommended that you use 16 Mbytes of RAM to minimize performance impact.

The OS/2 tools do not require this tuning of memory; they automatically receive as much memory as they request, up to the limit of physical RAM plus available disk space on the volume where the swap file resides (SWAPPER.DAT).

1.5 Using the Code Generation Tools on MS-DOS Hosts

Version 6.60 of the TMS320C1x/C2x/C2xx/C5x code generation tools supports extended memory on MS-DOS. Extended memory lets you compile or assemble large files that could not be built previously under MS-DOS. Extended memory is enabled by the **DOS/4GW** memory extender from Tenberry Software, Inc. (formerly Rational Systems, Inc.), which is embedded in the TMS320C1x/C2x/C2xx/C5x code generation tools. You must use an 80386-, 80486-, or Pentium-based PC to take advantage of extended memory under MS-DOS.

PC Requirements

Version 6.60 of the TMS320C1x/C2x/C2xx/C5x code generation tools requires an 80386 or later PC.

Installation

The code generation tools package is shipped on 1.44-Mbyte disks. To install the tools on an MS-DOS system, perform the following steps:

1) Make backups of the product disks.

2) Create a directory to contain the code generation tools. Enter the following:

   ```
   md c:\320tools
   ```

3) To install the code generation tools, copy the files from each diskette to the hard disk. Put each product disk into drive A and enter the following:

   ```
   copy a:\dos32\*.* c:\320tools
   ```

Note: About OS/2 Tools

For MS-DOS systems, do not copy the OS/2 tools onto your hard disk. The OS/2 tools are contained in the os2v2 directories of the product disks. The copy command, as shown in step 3, will properly install only the MS-DOS tools onto your hard disk.

4) Add the path of the code generation tools package to your DOS path. Edit AUTOEXEC.BAT and find the line that includes PATH= . At the end of the line, type:

   ```
   C:\320TOOLS
   ```

   Save the file, exit the editor, and reboot your PC.

   For example, you may find a path statement that looks like this:

   ```
   PATH=C:\DOS;C:\WINDOWS
   ```

After editing the line, this example path statement looks like this:

```
PATH=C:\DOS;C:\WINDOWS;C:\DSPTOOLS
```

5) You may wish to set environment variables in your AUTOEXEC.BAT file. See Section 1.2, page 1-5 for more information.

Performance Considerations

Performance has been enhanced in version 6.60 of the TMS320C1x/C2x/C2xx/C5x code generation tools. Still, you may notice a speed degradation when you use the code generation tools. Much of this speed degradation is due to the switch rate from protected to real mode necessitated by DOS calls. Higher-speed processors and later-generation processors in the 80386, 80486, and Pentium series minimize the time needed for this switch.

Virtual memory management (VMM) may also degrade system performance. It is recommended that VMM be enabled only for programs that cannot be built with VMM disabled.

The PMINFO.EXE Program

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

Syntax: PMINFO.EXE

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 may be omitted.

Other DOS/4GW error messages are found in Appendix A.

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 -- 5.00
                  Copyright (c) Rational Systems, Inc. 1987 - 1993

      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 this information:

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 will be 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 will perform well under some circumstances and poorly under others; PMINFO will show both 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 (i.e., 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 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.

The RMINFO.EXE Program

Purpose: Supplies configuration information and the basis for real/protected-mode switching.

Syntax: RMINFO

Notes: RMINFO is a DOS/16M application, as is DOS/4GW. If neither your application nor PMINFO will run, try running RMINFO.

RMINFO starts DOS/4GW, stops your machine just short of switching from real mode to protected mode, and displays configuration information about your computer. The information shown by RMINFO can help you determine why DOS/4GW applications won't run on a particular machine.

Example: The following example shows the output of the RMINFO program on an 80486 AT-compatible machine.

```
–============================== RMINFO ======================================–


          DOS/16M Real Mode Information Program –– 5.00
          Copyright (c) Rational Systems, Inc. 1987 – 1993

          Machine and Environment:
                  Processor:              i486, coprocessor present
                  Machine Type:           10 (AT–compatible)
                  A20 now:                enabled
                  A20 switch rigor:       disabled
                  XMS host found
                  VCPI host found,
                   page table 0 at:       24000h

          Switching Functions
                  A20 switching:          AT–style KBC
                  To PM switch:           VCPI
                  To RM switch:           VCPI
                  Nominal switch mode:    11
                  Switch control flags:   0000

          Memory Interfaces:  (VCPI remapping in effect)
                  VCPI may provide:       17854 returnable
                  contiguous DOS memory   492K
```

RMINFO provides this information (depending on your system configuration, not all fields will appear for any one machine):

Measurement	***Purpose***
Machine and Environment:	
Processor	Tells whether the processor is a 286, 386, 486, or P5 and if a math coprocessor is present.
Machine Type	Tells whether the machine is an AT-compatible or a non-AT-compatible MS-DOS machine.
A20 now	Gives the current state of address line 20.
A20 switch rigor	Indicates whether DOS/16M rigorously controls enabling and disabling of address line 20 when switching modes.
DPMI host found	Indicates that your system has a DPMI host (it is not present if the system does not have a DPMI host).
XMS host found	Tells whether your system has any software using extended memory under the XMS discipline.
VCPI host found	Tells whether your system has any software using extended memory under the VCPI discipline.
Page table n at	Linear address of the page table that DOS/16M starts with.
Switching Functions:	
A20 switching	Tells whether DOS/16M uses PS/2-style or AT-style (using the keyboard controller) switching.
To PM switch	Shows the DOS/16M method for switching to protected mode.
To RM switch	Shows the DOS/16M method for switching to real mode.
Nominal switch mode	Shows the switch mode used for backward compatibility.
Switch control flags	Shows the switch control flags used for backward compatibility.
Memory Interfaces:	
VCPI remapping in effect	Indicates that memory from different sources can be remapped contiguously.

DPMI may provide	Shows the amount of memory that the DPMI host reports as available to DOS/16M and whether DOS/16M can give back the memory it gets from this source (it is not present if there is no DPMI host).
VCPI may provide	Shows the amount of memory that the VCPI host reports as available to DOS/16M and whether DOS/16M can give back the memory it gets from this source (it is not present if there is no VCPI host).
XMS may provide	Shows the amount of memory that the XMS manager reports as available to DOS/16M and whether DOS/16M can give back the memory it gets from this source (it is not present if there is no XMS host).
Top-down may provide	Shows the amount of memory that DOS/16M finds using interrupt 15h and function 88h and indicates whether DOS/16M can give back the memory it gets from this source (it is not present if there is no top-down host).
Other 16M may provide	Shows the amount of memory that DOS/16M finds from a resident DOS/16M kernel and whether DOS/16M can give back the memory it gets from this source (it is not present if there is no other 16M host).
Contiguous DOS memory	Shows the amount of DOS memory available to DOS/16M.

1.6 Assembler and Linker Walkthrough

Two tools you may use often are the assembler and the linker. This section provides a quick walkthrough so that you can get started without reading the entire *TMS320C1x/C2x/C2xx/C5x Assembly Language Tools User's Guide*. These examples show the most common methods for invoking the assembler and linker. There are also examples in the tools directory that can be run or printed to give you an idea of the necessary syntax and steps.

Create two short source files for the walkthrough; call them **filea.asm** and **fileb.asm**.

Example 1–1. filea.asm

```
                .file   "filea.asm"
                .global addvec
                .global __stack
                .global start

        __stack .usect  ".stack",0

                .data
        vector: .word   10,20,30,40

                .text
        start:
                LRLK    AR1,__stack
                LALK    vector
                LARP    AR1
                SACL    *+
                CALL    addvec
                MAR     *–
```

Example 1–2.fileb.asm

```
                .file   ”fileb.asm”
                .global addvec

                .text

        addvec:
                SAR     AR0,*+
                SAR     AR1,*
                LARK    AR0,1
                LAR     AR0,*0+,AR2

                LARK     AR2,–2
                MAR     *0+
                LAR     AR3,*,AR3

                ZAC
                RPTK    3
                ADD     *+

                LARP    AR1
                SBRK    2
                LAR     AR0,*
                RET
```

1) Enter the following command to assemble filea.asm:

   `dspa filea`

   The dspa command invokes the assembler. The input source file is filea.asm. (If the input file extension is .asm, you don’t have to specify the extension; the assembler uses .asm as the default.) This example creates an object file called filea.obj. The assembler creates an object file if it does not encounter any 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.

Note: The Assembler Creates ’C2x Code by Default

This version of the **dspa** command invokes the TMS320C1x/C2x/C2xx/C5x assembler. By default, the assembler generates code for the TMS320C2x. Use the –v assembler options to generate code for the ’C1x, ’C2xx, or ’C5x.

2) Now assemble fileb.asm; enter:

   `dspa fileb.asm –l`

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

Example 1–3.fileb.lst, the Listing File Created by dspa fileb.asm –l

```
DSP Fixed Point COFF Assembler      Version 6.60     Tue Feb  7 16:35:25 1994
Copyright (c) 1987–1994  Texas Instruments Incorporated

fileb.asm                                                              PAGE    1

       1                    .file   ”fileb.asm”
       2                    .global addvec
       3
       4 0000               .text
       5
       6 0000       addvec:
       7 0000 70a0          SAR     AR0,*+
       8 0001 7180          SAR     AR1,*
       9 0002 c001          LARK    AR0,1
      10 0003 30ea          LAR     AR0,*0+,AR2
      11
      12 0004 d200          LARK    AR2,–2
         0005 fffe
      13 0006 55e0          MAR     *0+
      14 0007 338b          LAR     AR3,*,AR3
      15
      16 0008 ca00          ZAC
      17 0009 cb03          RPTK    3
      18 000a 00a0          ADD     *+
      19
      20 000b 5589          LARP    AR1
      21 000c 7f02          SBRK    2
      22 000d 3080          LAR     AR0,*
      23 000e ce26          RET

 No Errors,  No Warnings
```

3) Link filea.obj and fileb.obj; enter:

   `dsplnk filea fileb –m lnkerb.map –o prog.out`

   The **dsplnk** command invokes the linker. The input object files are filea.obj and fileb.obj. (If the input file extension is .obj, you don’t have to specify the extension; the linker uses .obj as the default.) The linker combines filea.obj and fileb.obj to create an executable object module called prog.out. The –o option supplies the name of the output module. Example 1–4 shows the map file resulting from this operation (the map file is produced only if –m option used).

Example 1–4.Output Map File, lnkerb.map

```
**************************************************
DSP Fixed Point COFF Linker     Version 6.60
**************************************************
Tue Feb  7 16:43:55 1994

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

MEMORY CONFIGURATION

          name      origin     length     attributes     fill
        --------   --------   ---------   ----------   --------
PAGE 0: PROG       00001000   00000ef00      RWIX

PAGE 1: DATA       00000300   00000fd00      RW

SECTION ALLOCATION MAP

 output                                  attributes/
section   page    origin      length       input sections
--------  ----  ----------  ----------   ----------------
.text      0    00001000    00000018
                  00001000    00000009     filea.obj (.text)
                  00001009    0000000f     fileb.obj (.text)

.data      0    00001018    00000004
                  00001018    00000004     filea.obj (.data)
                  0000101c    00000000     fileb.obj (.data)

.bss       1    00000000    00000000     UNINITIALIZED
                  00000000    00000000     filea.obj (.bss)
                  00000000    00000000     fileb.obj (.bss)

.stack     1    00000300    00000400     UNINITIALIZED
                  00000300    00000000     filea.obj (.stack)

GLOBAL SYMBOLS

address  name                            address  name
-------- ----                            -------- ----
00000000 .bss                            00000000 .bss
00001018 .data                           00000000 end
00001000 .text                           00000300 __stack
00000400 __STACK_SIZE                    00000400 __STACK_SIZE
00000300 __stack                         00001000 .text
00001009 addvec                          00001000 start
0000101c edata                           00001009 addvec
00000000 end                             00001018 etext
00001018 etext                           00001018 .data
00001000 start                           0000101c edata

[10 symbols]
```

The two files, filea and fileb, 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 linker options and filenames can be contained in the linker command file, or they can be entered on the command line, and the linker command file then consists of part of the link information. If no linker command file is provided, the sections will be allocated at address 0x0 as in the previous map file.

Example 1–5. Linker Command File, linkerb.cmd

```
–stack 100

MEMORY
{
        PAGE 0 : PROG : origin = 20h,   length = 0FE0h
        PAGE 1 : DATA : origin = 1000h, length = 1000h
}

SECTIONS
{
        .text  > PROG PAGE 0
        .data  > DATA PAGE 1
        .stack > DATA PAGE 1
}
```

Typing in the following command line using the linker command file shown above results in the map file shown on the following page.

```
dsplnk filea fileb linkerb.cmd –m linkerb.map –o prog.out
```

Example 1–6.Linker Map File (linkerb.map) Linked Using Linker Command File

```
*************************************************
DSP Fixed Point COFF Linker     Version 6.60
*************************************************
Tue Feb  7 16:44:45 1994

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

MEMORY CONFIGURATION

           name      origin     length     attributes     fill
         --------   --------   ---------   ----------   --------
PAGE 0: PROG       00000020   000000fe0      RWIX

PAGE 1: DATA       00001000   000001000      RWIX

SECTION ALLOCATION MAP

 output                                  attributes/
section   page    origin      length       input sections
--------  ----  ----------  ----------   ----------------
.text      0    00000020    00000018
                  00000020    00000009     filea.obj (.text)
                  00000029    0000000f     fileb.obj (.text)

.data      1    00001000    00000004
                  00001000    00000004     filea.obj (.data)
                  00001004    00000000     fileb.obj (.data)

.stack     1    00001004    00000064     UNINITIALIZED
                  00001004    00000000     filea.obj (.stack)

.bss       1    00000000    00000000     UNINITIALIZED
                  00000000    00000000     filea.obj (.bss)
                  00000000    00000000     fileb.obj (.bss)

GLOBAL SYMBOLS

address  name                            address  name
-------- ----                            -------- ----
00000000 .bss                            00000000 .bss
00001000 .data                           00000000 end
00000020 .text                           00000020 .text
00000064 __STACK_SIZE                    00000020 start
00001004 __stack                         00000029 addvec
00000029 addvec                          00000038 etext
00001004 edata                           00000064 __STACK_SIZE
00000000 end                             00001000 .data
00000038 etext                           00001004 __stack
00000020 start                           00001004 edata

[10 symbols]
```

1.7 C Compiler Walkthrough

The TMS320C2x/C2xx/C5x C compiler consists of two passes: the first pass parses the code, and the second pass produces a single assembly language source file that must be assembled and linked. The simplest way to compile, assemble, and link a C program 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 *TMS320C2x/C2xx/C5x 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)       */
/*********************************************/
main ()
{
   int x = –3
   X = abs_func(X)
}
int abs_func(int i)
   int i;
   {
      int temp = i;
      if (temp < 0) temp *= –1;
      return (temp);
   }
```

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

```
dspcl  function
```

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

```
[function]
TMS320C2x/2xx/5x ANSI C Compiler      Version 6.60
Copyright (c) 1987–1995 Texas Instruments Incorporated
 ”function.c”: ==> main
TMS320C2x/2xx/5x ANSI C Codegen       Version 6.60
Copyright (c) 1987–1995 Texas Instruments Incorporated
 ”function.c”: ==> main
DSP Fixed Point COFF Assembler     Version 6.60
Copyright (c) 1987–1995 Texas Instruments Incorporated
 PASS 1
 PASS 2

No Errors, No Warnings
```

The shell program runs the two compiler passes and the assembler as follows:

dspac → C parser
dspcg → Code generator
dspa → Assembler

By default, the shell program deletes the assembly language file from the compiler after it is assembled. If you wish to inspect the assembly language output, use the **–k** option to retain the assembly language file:

```
dspcl function –k
```

3) Also by default, the shell program 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 the 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, link the object file with the runtime-support library rts25.lib:

      ```
      dsplnk –c function –o function.out –l rts25.lib
      ```

      This example uses the –c linker option because the code came from a C program. 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 tells the linker that the input file rts25.lib is an object library.

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

      ```
      dspcl function –z –o function.out –l rts25.lib
      ```

      This example runs the two compiler passes, the assembler, and the linker as follows:

      dspac → C parser
      dspcg → Code generator
      dspa → Assembler
      dsplnk → Linker

4) The TMS320C2x/C2xx/C5x 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:

   ```
   dspcl function –s –z –o function.out –l rts25.lib
   ```

   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 2

Release Notes

This chapter documents tools and features that are new or have been changed since the last release.

Topic **Page**

2.1 Media Contents

MS-DOS and OS/2 Systems

TMS320C1x/C2x/C2xx/C5x Assembly Language Tools

Diskette #1 — Assembly Language Tools

Directory	File	Description
dos32\		*Assembly Language Tools, 32-bit extended DOS version*
	dspa.exe	TMS320C1x/C2x/C2xx/C5x assembler
	dsplnk.exe	TMS320C1x/C2x/C2xx/C5x COFF linker
	dsphex.exe	TMS320C1x/C2x/C2xx/C5x hex conversion utility
	readme.1st	online release bulletin
os2v2\		*Assembly Language Tools, 32-bit OS/2 v2.x version*
	dspa.exe	TMS320C1x/C2x/C2xx/C5x assembler
	dsplnk.exe	TMS320C1x/C2x/C2xx/C5x COFF linker
	dsphex.exe	TMS320C1x/C2x/C2xx/C5x hex conversion utility
	readme.1st	online release bulletin

Diskette #2 — Assembly Language Tools

Directory	File	Description
dos32\		*Assembly Language Tools, 32-bit extended DOS version*
	dspabs.exe	TMS320C1x/C2x/C2xx/C5x absolute lister
	dspar.exe	TMS320C1x/C2x/C2xx/C5x archiver
	dspxref.exe	TMS320C1x/C2x/C2xx/C5x cross-reference lister
	pminfo.exe	utility to measure protected/real-mode switching
	rminfo.exe	utility to display machine configuration
os2v2\		*Assembly Language Tools, 32-bit OS/2 v2.x version*
	dspabs.exe	TMS320C1x/C2x/C2xx/C5x absolute lister
	dspar.exe	TMS320C1x/C2x/C2xx/C5x archiver
	dspxref.exe	TMS320C1x/C2x/C2xx/C5x cross-reference lister

TMS320C2x/C2xx/C5x Optimizing C Compiler

Diskette #1 — Optimizing C Compiler

Directory	File	Description
dos32\		*Optimizing C Compiler, 32-bit extended DOS version*
	dspac.exe	TMS320C2x/C2xx/C5x ANSI C parser
	dspcg.exe	TMS320C2x/C2xx/C5x code generator
	pminfo.exe	utility to measure protected/real-mode switching
	rminfo.exe	utility to display machine configuration
	*.h	#include header files for RTS:
		assert.h ctype.h errno.h float.h
		ioports.h limits.h math.h setjmp.h
		stdarg.h stddef.h stdlib.h string.h
		time.h
	README.1ST	online release bulletin
os2v2\		*Optimizing C Compiler, 32-bit OS/2 v2.x version*
	dspac.exe	TMS320C2x/C2xx/C5x ANSI C parser
	dspcg.exe	TMS320C2x/C2xx/C5x code generator
	*.h	#include header files for RTS:
		assert.h ctype.h errno.h float.h
		ioports.h limits.h math.h setjmp.h
		stdarg.h stddef.h stdlib.h string.h
		time.h
	README.1ST	online release bulletin

Diskette #2 — Optimizing C Compiler

Directory	File	Description
dos32\		*Optimizing C Compiler, 32-bit extended DOS version*
	dspcl.exe	TMS320C2x/C2xx/C5x compiler shell program
	dspopt.exe	TMS320C2x/C2xx/C5x C optimizer
os2v2\		*Optimizing C Compiler, 32-bit OS/2 v2.x version*
	dspcl.exe	TMS320C2x/C2xx/C5x compiler shell program
	dspopt.exe	TMS320C2x/C2xx/C5x C optimizer

Diskette #3 — Optimizing C Compiler

Directory	File	Description
dos32\		*Optimizing C Compiler, 32-bit extended DOS version*
	clist.exe	TMS320C2x/C2xx/C5x C source interlist utility
	dspmk.exe	TMS320C2x/C2xx/C5x library-build utility
	rts50.lib	ANSI standard runtime-support ’C5x object library
os2v2\		*Optimizing C Compiler, 32-bit OS/2 v2.x version*
	clist.exe	TMS320C2x/C2xx/C5x C source interlist utility
	dspmk.exe	TMS320C2x/C2xx/C5x library-build utility
	rts50.lib	ANSI standard runtime-support ’C5x object library

Diskette #4 — Optimizing C Compiler

Directory	File	Description
dos32\		*Optimizing C Compiler, 32-bit extended DOS version*
	rts.src	ANSI standard runtime-support source library
	rts25.lib	ANSI standard runtime-support ’C2x object library
	rts2xx.lib	ANSI standard runtime-support ’C2xx object library
os2v2\		*Optimizing C Compiler, 32-bit OS/2 v2.x version*
	rts.src	ANSI standard runtime-support source library
	rts25.lib	ANSI standard runtime-support ’C2x object library
	rts2xx.lib	ANSI standard runtime-support ’C2xx object library

SPARC Systems

TMS320C1x/C2x/C2xx/C5x Code Generation Tools

File	Description
README.1ST	online release bulletin
clist	TMS320C2x/C2xx/C5x C source interlist utility
dspa	TMS320C1x/C2x/C2xx/C5x assembler
dspabs	TMS320C1x/C2x/C2xx/C5x absolute lister
dspac	TMS320C2x/C2xx/C5x ANSI C parser
dspar	TMS320C1x/C2x/C2xx/C5x archiver
dspcg	TMS320C2x/C2xx/C5x code generator
dspcl	TMS320C2x/C2xx/C5x compiler shell program
dsphex	TMS320C1x/C2x/C2xx/C5x 8-bit hex conversion utility
dsplnk	TMS320C1x/C2x/C2xx/C5x 8-bit COFF linker
dspmk	TMS320C2x/C2xx/C5x library-build utility
dspopt	TMS320C2x/C2xx/C5x C optimizer
dspxref	TMS320C1x/C2x/C2xx/C5x cross-reference lister
lnk.cmd	TMS320C2x/C2xx/C5x sample linker control file
rts25.lib	TMS320C2x runtime-support library
rts2xx.lib	TMS320C2xx runtime-support library
rts50.lib	TMS320C5x runtime-support library
*.h	#include header files for RTS:

assert.h	ctype.h	errno.h	float.h
ioports.h	limits.h	math.h	setjmp.h
stdarg.h	stddef.h	stdlib.h	string.h
time.h			

2.2 Additional Information

'C5x Silicon Problem Workarounds

A compiler option (dspcl option –mx, dspcg option –x) allows you to avoid 'C5x silicon bugs. Use of this switch is necessary when preparing a program for use with 'C5x silicon device versions earlier than 2.0 if the program implements interrupts or is compiled with optimization.

There is one problem that this option does not work around. When you run the compiler with the OVLY and RAM status bits on, certain compiled code sequences will not execute correctly if both the code and the data reside in the 1K byte of on-chip RAM on the 'C51 or the same 2K-byte block of the 9K bytes of on-chip RAM on the 'C50. Use a linker command file to set the program and data spaces so that this conflict does not occur. See the latest silicon errata sheet for more information.

Silicon Bug Flags in RTS Assembly Source

In the source to the assembly-coded runtime-support functions used by the compiler, there is an assembly-time variable that enables workarounds to 'C5x silicon problems. The toolset is shipped with this switch turned on. If you are using 'C5x silicon version 2.0 or later, you should turn this switch off. The following steps turn off the silicon bug switch:

1) Unarchive all the source from the source RTS library:

```
dspar –x rts.src
```

2) Edit the following assembly files:

f_add.asm	f_cmp.asm	f_div.asm	f_ftoi.asm
f_ftol.asm	f_ftou.asm	f_itof.asm	f_ltof.asm
f_mul.asm	f_sub.asm	f_utof.asm	idiv.asm
ldiv.asm	saverest.asm	setjmp.asm	udiv.asm

Change the SBUGS variable to 0:

```
SBUGS  .set  0
```

3) Reassemble these files:

```
dspcl –v50 f_add.asm f_cmp.asm ...
```

4) Rearchive these files into the object RTS library:

```
dspar –r rts50.lib *.obj
```

Options Used to Build the Libraries

The libraries supplied with this release were built with the following command lines:

```
dspmk –v50 –o –mx rts.src –l rts50.lib
dspmk –o rts.src –l rts25.lib
dspmk –o rts.src –l rts2xx.lib
```

The ’C5x RTS library was built with full optimization and the silicon bug switch on. The ’C2x RTS and the ’C2xx RTS libraries were built with full optimization.

2.3 Release Enhancements

General Enhancements

The following enhancements are included in this release:

- ❑ All known bugs have been removed.
- ❑ Extended memory support is provided on PC versions using an 80386 or higher.
- ❑ SunOS 5.x (Solaris 2.x) platform is supported.
- ❑ The absolute lister utility has been added. See Chapter 9 of the *TMS320C1x/C2x/C2xx/C5x Assembly Language Tools User's Guide* for details.

Assembler Enhancements

The following enhancements are included with the TMS320C1x/C2x/C2xx/C5x DSP assembler. See the *TMS320C1x/C2x/C2xx/C5x Assembly Language Tools User's Guide* chapter reference given with each bullet for details.

The following assembler directives have been added.

- ❑ The .bfloat assembler directive places the floating-point representation of a single floating-point constant into a word in the current section. The .bfloat directive guarantees the object will not span a page boundary. See Chapter 4 for more information.
- ❑ The .blong assembler directive places one or more 32-bit values into consecutive words in the current section. The least significant word is stored first. The .blong directive guarantees the object will not span a page boundary. See Chapter 4 for more information.

The following assembler options have been added.

- ❑ The –p assembler option when used with the –v2xx option enables the porting of code to the TMS320C2xx. See Chapter 3 for more information.
- ❑ The –pp assembler option when used with the –v2xx assembler option defines the symbols .TMS32025 and .TMS3202xx. This enables you to use code written using the .TMS32025 and .TMS32050 symbols on the TMS320C2xx. See Chapter 3 for more information.
- ❑ The –v2xx assembler option tells the assembler to produce code for TMS3202xx devices. See Chapter 3 for more information.

Linker Enhancements

- The –b linker option disables the linker's default elimination of duplicate entries of symbolic debugging information. See Chapter 8 for more information.
- The –v0 linker option creates an old-style COFF file (version 0). This option supports compatibility with older debuggers that do not support version 1 COFF. See Chapter 8 for more information.
- The –w linker option enables a warning switch. If you use the –w option, when the linker detects that a section that is not explicitly specified in the SECTIONS directive has been created, a warning message is generated. See Chapter 8 for more information.
- The linker GROUP statement now allows you to group output sections at one run address but allocate the group members to separate load addresses. Previously, a group was allocated as a single output section and a separate load address for each member could not be defined. See Chapter 8 for details.

Assembly Language Tool Enhancements

- The COFF symbol table index of relocation entries has been expanded to four bytes. For compatibility with earlier COFF versions, there are now two versions for relocation entries. Version 0 COFF file relocation information entries use a 10-byte format. Version 1 COFF file relocation information entries use a12-byte format. See Appendix A for tables detailing the difference in the versions.
- A cross-reference lister utility has been added. It uses object files to produce a cross-reference listing showing symbols, their definitions, and their references in the linked source files. See Chapter 10 for details.

Note: Cross-Reference Utility and C Source Debuggers

Existing TMS320C2x/C2xx/C5x C source debuggers are not compatible with the new cross-reference utility. Files assembled with the –x option cannot be loaded by these debuggers.

- The hex conversion utility has been updated. See Chapter 11 for more information.

C Compiler Enhancements

The TMS320C2x/C2xx/C5x optimizing C compiler enhancements are listed below. See the *TMS320C2x/C2xx/C5x Optimizing C Compiler User's Guide* chapter reference given with each bullet for details.

- The floating-point math functions that were located in the floating-point library, flib.lib, have been placed in the runtime-support source library, rts.src.
- A special convention has been added to the register keyword to allow the allocation of global registers. When you use the allocation declaration at file level, the register will be permanently reserved from any other use by the optimizer and code generator for that file. See Chapter 4 for more information.

  The –r*register* command line option for the dspcl shell and the corresponding –g*register* option for the optimizer and code generator (if you are invoking the tools individually) allow you to prevent the compiler from using the named *register*. See Chapter 2 for more information.
- The –ms shell option tells the compiler to optimize for code space over speed. See Chapter 2 for details.
- The –v2xx shell option enables the use of TMS320C2xx instructions. See Chapter 2 for more information.
- The –pr shell option creates a parser error message file. See Chapter 2 for more information.
- The generic optimizer has been improved, including improved bit-field optimizations, and file-level optimizations have been added. See Chapter 2 for more information.
- The rts.src function ti_sprintf has been added. Time functions (asctime, etc.) now call ti_sprintf to format the time string. See Chapter 5 for more information.
- The keyword ioport supports access to the I/O port space of the TMS320C2x/C2xx/C5x devices. See Chapter 3 for more information.

Appendix A

DOS/4GW Error Messages

DOS/4GW is a memory manager that is embedded in the TMS320C1x/C2x/C2xx/C5x code generation tools, so you may occasionally see DOS/4GW error messages. The executable files are not shipped as such, nor is any documentation provided on this tool, except for the list of error messages.

This excerpt from the *DOS/4GW User's Manual* (reproduced here with the permission of Tenberry Software, Inc.) lists error messages with descriptions of the circumstances in which the error is most likely to occur and suggestions for remedying the problem.

Topic **Page**

A.1 Troubleshooting in the Protected-Mode Environment

Getting 32-bit programs to execute properly under MS-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 here because they may appear when executing any tools, since all the tools have been assembled along with the DOS/4GW memory extender. When reading this material, keep these considerations in mind:

- When an *Action* directs you to technical support, determine the configuration of your system by using the **pminfo** (page 1-15) and **rminfo** (page 1-17) programs before contacting technical support.

  To obtain TMS320C1x/C2x/C2xx/C5x technical support, contact the DSP hotline:

  DSP Hotline: **(713) 274–2320**

  FAX: **(713) 274–2324**

  Electronic Mail: **4389750@mcimail.com.**

- When you are directed to rebuild the product, simply reinstall the tools from the supplied disks. Our products are shipped as object files and cannot be rebuilt. Should reinstallation not remedy the situation, contact technical support.

- When no action is suggested for an error message and the action is not obvious, contact technical support.

- Some error messages are not included in this section because they are rarely seen when using DOS/4GW with the TMS320C1x/C2x/C2xx/C5x tools. Also, many of the messages that *are* documented here are seldom seen when using DOS/4GW with the TMS320C1x/C2x/C2xx/C5x tools. Nevertheless, you may find this text useful in debugging your programs.

Should you encounter any error message not listed here, or should problems persist when running the 32-bit protected-mode version of the code generation tools, contact technical support.

A.2 Kernel Error Messages

This section describes error messages from the DOS/16M kernel embedded in the TMS320C1x/C2x/C2xx/C5x 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.

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 Increase the stack size, recompile your program with stack overflow checking, or look into ways that the descriptor tables may have been overwritten.

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

Description DOS4G.EXE or a bound DOS/4G application has probably been corrupted.

Action Rebuild or recopy the file.

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 CONFIG.SYS, 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 Rebuild or recopy the file.

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.

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 This 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.

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.

Action Rerun the program and/or restart your computer.

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 MS-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 64KB 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 probably 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 TMS320C1x/C2x/C2xx/C5x code generation tools. They are usually related to an unknown pathname, 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 couldn’t lock the kernel in physical memory, probably because of a memory shortage.

Action Free some memory for the DOS/4G application.

1004 ”syntax is DOS4G <executable.xxx>”

Description Incorrect syntax was encountered in this command.

Action You must specify a program name.

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 doesn’t 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 Make sure the file is in the correct format (one of the acceptable formats).

2523 ”page fault on non-present mapped page”

Description Your program references memory that has been mapped to a nonexistent physical device using DPMI function 508h.

Action Make sure the device is present, or remove the reference.

2524 ”page fault on uncommitted page”

Description Your program references memory that was reserved with a call to DPMI function 504h but that was never committed (using a DPMI 507h or 508h call).

Action Commit the memory before you reference it.

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 Try to determine whether any of the above conditions exist. If you cannot locate the problem, contact technical support.

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.