TMS320 DSP
DESIGNER'S NOTEBOOK

Floating Point C Compiler: Tips and Tricks – Part I

APPLICATION BRIEF: SPRA229

Karen Baldwin
Digital Signal Processing Products
Semiconductor Group

Texas Instruments
June 1993

TEXAS INSTRUMENTS

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 application 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.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Examples

Floating Point C Compiler: Tips and Tricks – Part I

Abstract

There are some tricks for making the most out of the C Compiler. This document discusses the three ideas listed below, and includes a short code example for each.

- Solving the ’C40 Discontinuity Issue with Indirect Calls
- Making use of Relocatable C code
- Making a C Function Part of a Different Section

Design Problem

What are some of the tricks of the masters for making the most out of the C Compiler?

Solution

1. Solving the 'C40 Discontinuity Issue with Indirect Calls

The 'C40 has only relative jumps. Therefore PC discontinuities using direct-mode addressing are limited to a 24-bit range. This, coupled with the 'C40 memory map, makes it impossible to directly call a routine in on-chip memory because the compiler uses the direct form of CALL for calls to named functions. (The BR instruction is never used: all branches except returns must be within a single function, so the short conditional form is used.)

You can use indirect calls to call functions anywhere in the address space. You do this by declaring a pointer to a function and then calling via the pointer.

Example 1. Indirect Function Call

```
int f(); /* function that resides in internal mem */
int (*ptr_to_f)() = f; /* pointer to function f */

main()
{
   (*ptr_to_f)(); /* call function f indirectly */
}
```

2. Making use of Relocatable C code

You can specify different load addresses and run addresses for a section in the linker command file. But you have to write your own loader to move the section to the run address. If using assembly language you can use the `.label` statement to get the load address. Here is an example:

Example 2. Naming Load and Run Addresses

```
            .global sec_start   ; run address
            .global sec_end
            .global l_sec_start ; load address
            .global l_sec_end
sec_start:  .label l_sec_start  ; l_sec_start will
          ;contain the load address (sec_start = run
          ; address program code goes here
sec_end:    .label l_sec_end    ; same explanation
```

Your loader makes use of these labels to write from `l_sec_start` to `sec_start` and so on. The loader is a loop that copies the code from one location to the other. The C version is shown in listing 3.

Example 3. A Homemade Loader in C

```
/* also asm(“ with .global ..... */
asm(”sec_start .label l_sec_start") /* same as */
                                /* assembly version */
/* program code goes here */
void func(a,b,c)
<local variable declaration>
}
asm(“sec_end .label l_sec_end”)
```

3. Making a C Function Part of a Different Section

The C compiler does not directly specify a section name for the executable assembly code that it generates. It relies on the assembler defaulting the section name to `.text`. However, it is possible to relocate the executable code from a function into a user-defined section from within the C source.

This is accomplished by placing an `asm` statement that declares the new section before the actual function definition. Example 4 below uses a macro definition to provide a general solution to defining named sections.

Example 4. Macro Definition Solution To Defining Named Sections

```
#define sect(a) asm(“ .sect ”#a)
sect(“pp”); /* Creates .sect “pp” in */
                    /* asm code */
void func() { }
```

The section name will remain “pp” until changed. If other functions follow this function in the file and their code is not to be included within this section then reset the section name before the next function definition.