Using GCC Toolchain Options to Optimize Code Size

Document Number: MD00842
Revision 01.01
May 1, 2011

MIPS Technologies, Inc.
955 East Arques Avenue
Sunnyvale, CA 94085-4521

Copyright © 2011 MIPS Technologies Inc. All rights reserved.

Contents

1 Introduction

This application note explains how to use the MIPS GNU Compiler Collection (GCC) toolchain options to significantly reduce code size, and thus avoid the need for more complex optimizations, such as changing the style of code, redefining modules, or changing the system architecture.

The GNU Compiler Collection (GCC) is an integrated distribution of compilers for several major programming languages and architectures, and is available at *http://gcc.gnu.org*.

2 Software Environment

For optimal results, the software development environment should allow project files to be compiled separately, so as to enable the precise application of the appropriate code-reducing compiler options according to the requirements of each file. For example, enabling the `–mlong-call` option might be appropriate for code that required jumps to far distant functions, but enabling it for an entire software project would result in a larger code size and execution inefficiencies.

When the software environment cannot be modified to allow for per-file compilations, we suggest that code be compiled separately and put in a library that can be integrated into the system project.

3 MIPS-specific GCC Compiler Options

This section describes GCC's MIPS-specific compiler options that are especially useful for reducing code size. Refer to *http://gcc.gnu.org/onlinedocs/gcc-4.4.5/gcc/MIPS-Options.html#MIPS-Options* for a complete list of MIPS-specific GCC options.

3.1 -mno-long-calls

This option enables use of the jal instruction, which is more efficient for function calls but requires the caller and callee to be in the same 256-megabyte segment, which in turn requires the linker command file to locate the functions accordingly.

When code cannot be relocated, the following strategy can be used:

1. Use the `-mlong-calls` compiler option

2. Include the `long_call` attribute of the callee function in the caller's function declaration:

```
void __attribute__ ((long_call)) callee(void);
void caller(void)
{
    :
    callee();
:
}
```

Function calls to callee() from caller() will disable the jal instruction and instead utilize a lui/addiu/jalr/nop instruction to access 32-bit addresses.

Copyright © 2011 MIPS Technologies Inc. All rights reserved.

This option has no effect on abicalls code. The default is `-mno-long-calls`.

3.2 -mno-interlink-mips16

This option specifies that non-MIPS16 code is not required to be link-compatible with MIPS16 code. For code that uses microMIPS or MIPS16 instructions, use the `-minterlink-mips16` option to cause the compiler to add the mode-switch and alignment code required for the interlink from MIPS32 to the microMIPS or MIPS16 code.

Keep in mind that use of this option causes an increase in code size (the additional code to switch modes and realign 32-bit to 16-bit function calls) and a slight degradation in execution speed, so it should be used carefully and not for time-critical modules.

The prototypes of functions with mixed code must declared as follows:

- Function prototype for callee:

```
void __attribute__ (mips16e)) calleeMIPS16e(void); //MIPS16e callee function
void __attribute__ ((micromips)) calleeMicroMIPS(void); //MicroMIPS callee function

void caller(void)
{
    :
    calleeMIPS16e ();
    :
    calleeMicroMIPS();
    :
}
```

- Function prototype for caller:

```
void __attribute__ (nomips16e)) calleeMIPS32(void); //call by MIPS16e function

void __attribute__ ((nomicromips)) calleeMIPS32 (void); //call by MicroMIPS
function
```

3.3 -Gnum

This option directs the compiler to put definitions of externally-visible data in a small data section when that data is no bigger than `num` bytes. GCC can then use gp-relative addressing, which is a powerful tool for reducing code size and is a favorite among toolchain designers. Data that is stored within reach of the gp register can be accessed in a single instruction using a signed, 16-bit offset from the gp register ($28). Because the maximum addressing range is 64K bytes, the total size of the small data section (`.sdata`, `.sbss` , `.scommon`) should be less than 64K bytes.

The use of gp-relative addressing requires the cooperation of compiler, assembler, linker, and run-time initialization code in pooling all the "small" data items together into a single region, and then setting the gp register to point to the middle of that region. The gp register value is assigned by the linker and re-initialized when the system is booted, so check your linker and boot code to ensure correct initialization of the maximum useful size.

An example linker command file is that shown below.

```
.sdata :
    {
        _gp = . + 0x8000; // +0x8000 give a bias to allow gp register could fix
                        // into  -32768 to 32767 offset
```


Copyright © 2011 MIPS Technologies Inc. All rights reserved.

```
          *(.sdata)
          *(.sdata.*)
      } > ram
  //Don't insert any other section
  .sbss
      {
         *(.sbss)
          *(.sbss.*)
      } > ram
  //Don't insert any other section
  . scommon:
      {
          *(.scommon)
          *(.scommon.*)
      } > ram
```

In the above example, the gp register is set to 0x8000 at the start of `.sdata`, `.sbss` and `.scommon`, which allows the gp register to access an offset address of -32768 to 32767.

Make sure that all small data sections are declared as located inside the `.sdata`, `.sbss` or `.scommon` sections, and check the section names to make sure all small data will fit within these sections. And do not insert sections other than small data sections into the region between `.sdata`, `.sbss` and `.scommon`.

The system program designer may force some variables to be located at specific memory locations, for example, in internal scratchpad RAM or some special hardware driver region. Because the memory map is fixed, and there is usually a large gap between gp-relative locations and normal memory, those special memory locations should not be within range of the gp register in order to ensure that the 64K memory boundary is not exceeded. A common mistake is to fail to inform the compiler that it should not use gp-relative addressing for those memory locations.

Here is a simple example:

```
int   smallVar;
int   fixlocationVar __attribute__ ((section ("_iram")));
```

The compiler result is :

```
smallVar ? access by gp related
fixlocationVar ? non gp related access
```

Note: If `fixlocationVar` is exported to other C files, make sure the variable declaration is:

```
extern  int   fixlocationVar __attribute__ ((section ("_iram")));
```

A common mistake is to fail to declare the the variable's section type as `extern` in the `.c` or `.h` file. Without the `extern` section declaration and section attribute declared as `iram`, the compiler will incorrectly interpret `fixlocationVar` as accessible relative to the gp register.

3.4 -mno-split-addresses

This option disables use of the `%hi()` and `%lo()` assembler relocation operators. It has been replaced by `-mexplicit-relocs` (described below) but is remains available for backwards compatibility.

Copyright © 2011 MIPS Technologies Inc. All rights reserved.

3.5 -mno-explicit-relocs

This option disables the assembler's use of relocation operators for evaluating symbolic addresses. The assembler uses macros instead.

Use of this option with the `-mno-split-addresses` option creates more opportunities for linker relaxation (described in Section 5.1, "--relax"). For example, more "addiupc" instructions can be generated by linker relaxation to reduce code size. Note that individual object files may become larger with these two options, but that the final executable can be smaller with linker relaxation.

3.6 -membedded-data

This option directs the compiler to allocate variables to the read-only data section whenever possible, then to the small data section, and otherwise in data. Though this produces code that is slightly slower than the default, it reduces the amount of RAM required when executing, and thus may be preferred for some embedded systems.

4 Common GCC Compiler Options

This section describes GCC compiler options available for most microprocessor architectures that are especially useful for reducing code size.

4.1 -Os

This option directs the compiler to optimize for code size. It enables all -O2 optimizations that do not typically increase code size and performs further optimizations designed to reduce code size.

`-Os` disables the following optimization flags:

- -falign-functions
- -falign-jumps
- -falign-loops
- -falign-labels
- -freorder-blocks
- -freorder-blocks-and-partition
- -fprefetch-loop-arrays
- -ftree-vect-loop-version

The `-Os` option must be used to ensure optimally compact code. `-Os` enables all `-O2` optimizations that do not usually increase code size and performs additional special options that further reduce code size.

Copyright © 2011 MIPS Technologies Inc. All rights reserved.

4.2 -fshort-enums

This option directs the compiler to allocate to an `enum` type only as many bytes as required for the declared range of values. Specifically, the `enum` type will be equivalent to the smallest integer type that has enough room.

Note that code generated with the `-fshort-enums` option is not binary-compatible with code generated without that option. Use it to conform to a non-default application binary interface.

GCC does not enable this option by default.

4.3 -fsee

This option directs the compiler to eliminate redundant sign-extension instructions, and to move the non-redundant instructions to an optimal placement using lazy code motion (LCM).

4.4 -ffunction-sections -fdata-sections

This option directs the compiler to place each function or data item into its own section in the output file, if the target supports arbitrary sections. The section's name in the output file is determined by the name of the function or the name of the data item.

These options should be used whenever the linker is able to perform optimizations that improve locality of reference in the instruction space. In most cases, systems using files in ELF object format have these optimizations.

Note: The linker uses `--gc-sections` to remove unused sections.

4.5 -fomit-frame-pointer

This option directs the compiler not to keep the frame pointer in a register in cases where the function doesn’t use a frame pointer, thus avoiding the instructions required to save, set up, and restore frame pointers, and making an extra register available in many functions.

Note that use of this option makes debugging impossible on some machines.

4.6 -finline

This option enables the `inline` function attributes.

4.7 -fno-inline-small-functions

This option directs the compiler not to integrate functions into their callers when their body is smaller than the expected size of the function call code.

`-finline-small-functions` is enabled at level -O2.

4.8 -fno-inline-functions

This option directs the compiler not to integrate simple functions into their callers.

Copyright © 2011 MIPS Technologies Inc. All rights reserved.

`-finline-functions` is enabled at level -O3.

4.9 -finline-functions-called-once

This option directs the compiler to consider for inlining into their caller all static functions that are called once, even when they are not designated as `inline`. If a call to a given function is integrated, the function is not output as assembler code.

This option is enabled at levels -01, - 02, -03, and -0s.

5 GCC Linker Options

GCC linker options are described at *http://gcc.gnu.org/onlinedocs/gcc/Link-Options.html*.

Use the `-Wl,` option to pass linker options from GCC to LD. For example:

```
-Wl,--relax, -Wl,--gc-sections
```

5.1 --relax

This option instructs the linker to remove microMIPS instructions within `.text` sections.

Note that following relaxation, microMIPS functions will not be aligned to 4 bytes, so make sure that they are not called directly by jal instructions (though they can be called by MIPS32 jalr instructions). Also, do not use linker relaxation if there is data in the `.text` section that requires data alignment.

5.2 --gc-sections

This option removes unused sections.

6 References

1. MIPS Architecture for Programmers Volume I-B: Introduction to the microMIPS32 Architecture
   MIPS Document Number: MD00741

2. microMIPS™ GCC Toolchain Usage
   MIPS Document Number: MD00784

3. GCC, the GNU Compiler Collection
   http://gcc.gnu.org

4. GNU Binutils
   http://www.gnu.org/software/binutils

Copyright © 2011 MIPS Technologies Inc. All rights reserved.

7 Document Revision History

Revision	Date	Description
01.00	April 20, 2011	Initial release
01.01	May 1, 2011	Modify option definitions

Copyright © 2011 MIPS Technologies Inc. All rights reserved.

Copyright © 2011 MIPS Technologies, Inc. All rights reserved.

Unpublished rights (if any) reserved under the copyright laws of the United States of America and other countries.

This document contains information that is proprietary to MIPS Technologies, Inc. ("MIPS Technologies"). Any copying, reproducing, modifying or use of this information (in whole or in part) that is not expressly permitted in writing by MIPS Technologies or an authorized third party is strictly prohibited. At a minimum, this information is protected under unfair competition and copyright laws. Violations thereof may result in criminal penalties and fines.

Any document provided in source format (i.e., in a modifiable form such as in FrameMaker or Microsoft Word format) is subject to use and distribution restrictions that are independent of and supplemental to any and all confidentiality restrictions. UNDER NO CIRCUMSTANCES MAY A DOCUMENT PROVIDED IN SOURCE FORMAT BE DISTRIBUTED TO A THIRD PARTY IN SOURCE FORMAT WITHOUT THE EXPRESS WRITTEN PERMISSION OF MIPS TECHNOLOGIES, INC.

MIPS Technologies reserves the right to change the information contained in this document to improve function, design or otherwise. MIPS Technologies does not assume any liability arising out of the application or use of this information, or of any error or omission in such information. Any warranties, whether express, statutory, implied or otherwise, including but not limited to the implied warranties of merchantability or fitness for a particular purpose, are excluded. Except as expressly provided in any written license agreement from MIPS Technologies or an authorized third party, the furnishing of this document does not give recipient any license to any intellectual property rights, including any patent rights, that cover the information in this document.

The information contained in this document shall not be exported, reexported, transferred, or released, directly or indirectly, in violation of the law of any country or international law, regulation, treaty, Executive Order, statute, amendments or supplements thereto. Should a conflict arise regarding the export, reexport, transfer, or release of the information contained in this document, the laws of the United States of America shall be the governing law.

The information contained in this document constitutes one or more of the following: commercial computer software, commercial computer software documentation or other commercial items. If the user of this information, or any related documentation of any kind, including related technical data or manuals, is an agency, department, or other entity of the United States government ("Government"), the use, duplication, reproduction, release, modification, disclosure, or transfer of this information, or any related documentation of any kind, is restricted in accordance with Federal Acquisition Regulation 12.212 for civilian agencies and Defense Federal Acquisition Regulation Supplement 227.7202 for military agencies. The use of this information by the Government is further restricted in accordance with the terms of the license agreement(s) and/or applicable contract terms and conditions covering this information from MIPS Technologies or an authorized third party.

MIPS, MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MIPSr3, MIPS32, MIPS64, microMIPS32, mircoMIPS64, MIPS-3D, MIPS16, MIPS16e, MIPS-Based, MIPSsim, MIPSpro, MIPS Technologies logo, MIPS-VERIFIED, MIPS-VERIFIED logo, 4K, 4Kc, 4Km, 4Kp, 4KE, 4KEc, 4KEm, 4KEp, 4KS, 4KSc, 4KSd, M4K, M14K, 5K, 5Kc, 5Kf, 24K, 24Kc, 24Kf, 24KE, 24KEc, 24KEf, 34K, 34Kc, 34Kf, 74K, 74Kc, 74Kf, 1004K, 1004Kc, 1004Kf, R3000, R4000, R5000, ASMACRO, Atlas, "At the core of the user experience.", BusBridge, Bus Navigator, CLAM, CorExtend, CoreFPGA, CoreLV, EC, FPGA View, FS2, FS2 FIRST SILICON SOLUTIONS logo, FS2 NAVIGATOR, HyperDebug, HyperJTAG, IASim, JALGO, Logic Navigator, Malta, MDMX, MED, MGB, microMIPS, OCI, PDtrace, the Pipeline, Pro Series, SEAD, SEAD-2, SmartMIPS, SOC-it, System Navigator, and YAMON are trademarks or registered trademarks of MIPS Technologies, Inc. in the United States and other countries.

All other trademarks referred to herein are the property of their respective owners.

Template: nW1.03, Built with tags: 2B

Copyright © 2011 MIPS Technologies Inc. All rights reserved.