microMIPS™ ASE Usage

This paper explains how to use the microMIPS-specific features of the GNU Compiler Collection (GCC), GNU Assembler (AS), and GNU Linker (LD), and includes precautions for interlinking MIPS32 and microMIPS code and well-commented code examples. The purpose of this paper is to enable users to take full advantage of the substantial savings in code size provided by the microMIPS ASE

Document Number: MD00784
Revision 01.00
June 21, 2010

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

Contents

1 Introduction

The microMIPS™ ISA is a re-encoding of MIPS32 instructions to an optimized set of 16-bit and 32-bit instructions, with the most commonly used instructions re-encoded into 16-bit instructions, and with new instructions that result in additional reductions in code size. The performance of the microMIPS ISA is equivalent to the performance of MIPS32 and is assembly-language compatible with existing MIPS32 source code and at the MIPS32 ABI (Application Binary Interface) level, which defines the interface between applications and operating systems. The M14K™ [2] and M14Kc™ [3] are the first MIPS processor cores that support both the MIPS32 and the microMIPS ISAs.

This paper describes the microMIPS-specific options of the GNU Compiler Collection (GCC) [4], the GNU Assembler (AS) [5], and the GNU Linker (LD) [5] and explains their correct use. Code examples of each option are provided.

2 GCC for microMIPS

2.1 GCC Options

2.1.1 -mmicromips, -mno-micromips

The GCC options `-mmicromips` and `-mno-micromips` are used to specify that C files are to be compiled for the microMIPS ISA or MIPS32 ISA respectively. If neither option is specified, the MIPS32 ISA is used.

For example:

```
# cat add.c
int add (int a, int b)
{
  return a + b;
}

# mips-sde-elf-gcc -c -O2 -mmicromips add.c
# mips-sde-elf-objdump -dr add.o

add.o:     file format elf32-tradbigmips


Disassembly of section .text:

00000000 <add>:
   0:   459f            jr      ra
   2:   054a            addu    v0,a1,a0

# mips-sde-elf-gcc -c -O2 -mno-micromips add.c
# mips-sde-elf-objdump -dr add.o

add.o:     file format elf32-tradbigmips
```

```
Disassembly of section .text:

00000000 <add>:
   0:   03e00008        jr      ra
   4:   00a41021        addu    v0,a1,a0
```

2.1.2 -mjals, -mno-jals

The GCC options `-mjals` and`-mno-jals` enable or disable the generation of JALS instructions for microMIPS. JALS (Jump and Link, Short Delay Slot) is a microMIPS instruction that requires a 16-bit instruction in its delay slot, as opposed to the MIPS32 instruction JAL that requires a 32-bit instruction, so its use can produce substantial reductions in code size.

For example:

```
# cat call.c
void t2 ();

int t1 ()
{
  t2 ();
  return 2;
}

# mips-sde-elf-gcc -O2 -mmicromips -mjals -c call.c
# mips-sde-elf-objdump -dr call.o

call.o:     file format elf32-tradbigmips


Disassembly of section .text:

00000000 <t1>:
   0:   4ff5            addiu   sp,sp,-24
   2:   cbe5            sw      ra,20(sp)
   4:   7400 0000       jals    0 <t1>
                        4: R_MICROMIPS_26_S1    t2
   8:   0c00            nop
   a:   4be5            lw      ra,20(sp)
   c:   ed02            li      v0,2
   e:   4706            jraddiusp       24

# mips-sde-elf-gcc -O2 -mmicromips -mno-jals -c call.c
# mips-sde-elf-objdump -dr call.o

call.o:     file format elf32-tradbigmips


Disassembly of section .text:

00000000 <t1>:
   0:   4ff5            addiu   sp,sp,-24
   2:   cbe5            sw      ra,20(sp)
   4:   f400 0000       jal     0 <t1>
                        4: R_MICROMIPS_26_S1    t2
   8:   0000 0000       nop
   c:   4be5            lw      ra,20(sp)
   e:   ed02            li      v0,2
  10:   4706            jraddiusp       24
  12:   0c00            nop
```

For bare-metal systems (mips-sde-elf targets), `-mjals` is the default. For Linux systems (mips-linux-gnu targets), `-mno-jals` is the default. Note that if the target function of the JALS instruction is a MIPS32 function, the linker will issue an error, because it cannot transform a JALS to a JALX by expanding the delay slot instruction from 16 to 32 bits. Therefore, use `-mjals` only when all target functions of microMIPS calls are microMIPS functions. When there might be mode switches from microMIPS to MIPS32, use `-mno-jals`.

2.1.3 -minterlink-mips16, -mno-interlink-mips16

The GCC option `-minterlink-mips16` must be used when there are possible mode switches between micro-MIPS and MIPS32. Without `-minterlink-mips16` (or with `-mno-interlink-mips16)`, GCC performs leaf function optimization by converting a function call to a direct jump (J), but the linker is unable to transform J to JALX for the required mode switch. Therefore, both `-minterlink-mips16` and `-mno-jals` (Section 2.1.2 “-mjals, -mno-jals”) must be used for possible mode switches from microMIPS to MIPS32 code, and `-minterlink-mips16` must be used for possible mode switches from MIPS32 to microMIPS code.

For example:

```
# cat leaf.c
void s2();

void s1()
{
  s2();
}

# mips-sde-elf-gcc -c -O2 -mmicromips -mno-interlink-mips16 leaf.c
# mips-sde-elf-objdump -dr leaf.o

leaf.o:     file format elf32-tradbigmips


Disassembly of section .text:

00000000 <s1>:
   0:   d400 0000       j       0 <s1> # NOT  for mode switch
                        0: R_MICROMIPS_26_S1    s2
   4:   0c00            nop
   6:   0c00            nop

# mips-sde-elf-gcc -c -O2 -mmicromips -minterlink-mips16 leaf.c
# mips-sde-elf-objdump -dr leaf.o

leaf.o:     file format elf32-tradbigmips


Disassembly of section .text:

00000000 <s1>:
   0:   4ff5            addiu   sp,sp,-24
   2:   cbe5            sw      ra,20(sp)
   4:   7400 0000       jals    0 <s1> # NOT  for mode switch
```

```
                        4: R_MICROMIPS_26_S1    s2
   8:   0c00            nop
   a:   4be5            lw      ra,20(sp)
   c:   4706            jraddiusp       24
   e:   0c00            nop

# mips-sde-elf-gcc -c -O2 -mmicromips -minterlink-mips16 -mno-jals leaf.c
# mips-sde-elf-objdump -dr leaf.o

leaf.o:     file format elf32-tradbigmips


Disassembly of section .text:

00000000 <s1>:
   0:   4ff5            addiu   sp,sp,-24
   2:   cbe5            sw      ra,20(sp)
   4:   f400 0000       jal     0 <s1> #  for mode switch
                        4: R_MICROMIPS_26_S1    s2
   8:   0000 0000       nop
   c:   4be5            lw      ra,20(sp)
   e:   4706            jraddiusp       24
```

2.2 GCC Function Attributes

2.2.1 micromips, nomicromips

In C files, the function attributes `micromips` or `nomicromips` in the function declaration specifies which ISA to use for that function. These function attributes override the GCC options `-mmicromips` and `-mno-micromips` (Section 2.1 “GCC Options”). Using these function attributes is a powerful tool for reducing code size.

For example:

```
# cat addsub.c
// microMIPS ISA for add()
int __attribute__ ((micromips)) add (int a, int b)
{
  return a + b;
}

// MIPS32 ISA for sub()
int __attribute__ ((nomicromips)) sub (int a, int b)
{
  return a - b;
}

# mips-sde-elf-gcc -c -O2 addsub.c
# mips-sde-elf-objdump -dr addsub.o

addsub.o:     file format elf32-tradbigmips


Disassembly of section .text:
```

```
00000000 <add>:
   0:   459f            jr      ra
   2:   054a            addu    v0,a1,a0

00000004 <sub>:
   4:   03e00008        jr      ra
   8:   00851023        subu    v0,a0,a1
```

3 AS for microMIPS

3.1 AS Options

3.1.1 -mmicromips, -mno-micromips

The assembler options `-mmicromips` or `-mno-micromips` are used to select the assembler's ISA mode of operation. If neither option is specified, the default mode for generated code is the MIPS32 ISA.

For example:

```
# cat add.s
        .text
Add:
        addu    $2, $3, $4

# mips-sde-elf-as add.s -o add.o -mmicromips
# mips-sde-elf-objdump -dr add.o

add.o:     file format elf32-tradbigmips


Disassembly of section .text:

00000000 <Add>:
   0:   0546            addu    v0,v1,a0
   2:   0c00            nop

# mips-sde-elf-as add.s -o add.o -mno-micromips
# mips-sde-elf-objdump -dr add.o

add.o:     file format elf32-tradbigmips


Disassembly of section .text:

00000000 <Add>:
   0:   00641021        addu    v0,v1,a0
```

3.2 AS Directives

3.2.1 .set micromips, .set nomicromips

Within an assembly-language program, the `.set micromips` or `.set nomicromips` assembler directives can be used to specify the ISA to be used by the assembler. These directives override the AS options of `-mmicromips` and `-mno-micromips` (Section 3.1 “AS Options”). They are another powerful tool for reducing code size.

For example:

```
# cat addsub.s
        .text
        .set    micromips
Add:
        addu    $2, $3, $4

        .align  2
        .set    nomicromips
Sub:
        subu    $2, $3, $4

# mips-sde-elf-as addsub.s -o addsub.o
# mips-sde-elf-objdump -dr addsub.o

addsub.o:     file format elf32-tradbigmips


Disassembly of section .text:

00000000 <Add>:
   0:   0546            addu    v0,v1,a0
   2:   0c00            nop

00000004 <Sub>:
   4:   00641023        subu    v0,v1,a0
```

3.2.2 .insn

To enable processors to determine the current ISA (MIPS32 ISA or microMIPS ISA), the least-significant bit of an address (bit 0) is utilized as the ISA mode bit (0 = MIPS32 ISA, 1 = microMIPS ISA).. This mechanism enables calls to microMIPS or MIPS32 functions via the JALR instruction by setting a register value odd (for microMIPS) or even (for MIPS32) from the address.

Use of the `.insn` directive ensures the correct handling of the ISA mode bit by the assembler and the linker. Using the `.insn` directive following a label marks the label as a text symbol. A label can also be marked as a text symbol by following it with an instruction. The text symbol is important for the microMIPS ISA, because loading an address for a microMIPS text symbol to a register sets bit 0 of the register value to 1 (via the linker). Conversely, loading an address for a MIPS32 text symbol to a register sets bit 0 of the register value to 0 (via the linker).

For example:

```
# cat insn.s
.text
        .set    noreorder
        .ent    test
        .globl  test
        .align  2
test:
        la      $2, test2 # test2 is a text symbol
        jalr    $2
        nop
        la      $2, test3 # test3 is a text symbol
        jalr    $2
        nop
        .end test

        .align  2
        .ent    test2
test2:
        addu    $2, $3, $4
        .end test2

        .align  2
        .ent    test3
test3:
        .insn
        .word   0x00831150
        .end test3

# mips-sde-elf-as insn.s -o insn.o -mmicromips
# mips-sde-elf-ld insn.o -o insn -e test
# mips-sde-elf-objdump -dr insn

insn:     file format elf32-tradbigmips


Disassembly of section .text:

0040006c <test>:
  40006c:       41a2 0040       lui     v0,0x40
  400070:       3042 0089       addiu   v0,v0,137 # The bit 0 is 1
  400074:       45c2            jalr    v0
  400076:       0000 0000       nop
  40007a:       41a2 0040       lui     v0,0x40
  40007e:       3042 008d       addiu   v0,v0,141 # The bit 0 is 1
  400082:       45c2            jalr    v0
  400084:       0000 0000       nop

00400088 <test2>:
  400088:       0546            addu    v0,v1,a0
  40008a:       0c00            nop

0040008c <test3>:
  40008c:       0083 1150       addu    v0,v1,a0
```

If a symbol is not a text symbol (e.g., a symbol for a data label), loading this symbol to a register results in a value of 0 in bit 0 of the register for both the MIPS ISA and the microMIPS ISA (via the linker). So don't use `.insn` after a data label, and don't load an instruction label (e.g., an instruction is after a label) as a data label. Otherwise, there may be an address error when loading data from that address for microMIPS.

For example:

```
# cat data.s
        .text
        .set    noreorder
        .ent    test
        .globl  test
        .align  2
test:
        la      $2, mydata # mydata is a data symbol
        lw      $3, 0($2)
        .end test

        .align  2
        .data
mydata:
        .word   0x12345678

# mips-sde-elf-as data.s -o data.o -mmicromips
# mips-sde-elf-ld data.o -o datatest -e test
# mips-sde-elf-objdump -d --section=.text --section=.data datatest

datatest:     file format elf32-tradbigmips


Disassembly of section .text:

0040008c <test>:
  40008c:       41a2 0041       lui     v0,0x41
  400090:       3042 0098       addiu   v0,v0,152 # The bit 0 is 0
  400094:       69a0            lw      v1,0(v0)
  400096:       0c00            nop

Disassembly of section .data:

00410098 <_fdata>:
  410098:       12345678        beq     s1,s4,425a7c <_gp+0xd9ec>
```

3.3 Instruction Name Postfixes

3.3.1 16, 32

By default, the assembler uses 16-bit microMIPS instructions instead of 32-bit microMIPS instructions whenever possible. Attaching `16` or `32` to the end of an instruction name (but before a "`.`" if the name has a ".") can force the assembler to generate the 16-bit version or the 32-bit version of the microMIPS instruction.

For example:

```
# cat add3.s
        .text
Add:
        addu    $2, $3, $4
        addu16  $2, $3, $4
        addu32  $2, $3, $4

# mips-sde-elf-as add3.s -o add3.o -mmicromips
# mips-sde-elf-objdump -dr add3.o
```

```
add3.o:     file format elf32-tradbigmips


Disassembly of section .text:

00000000 <Add>:
   0:   0546            addu    v0,v1,a0
   2:   0546            addu    v0,v1,a0
   4:   0083 1150       addu    v0,v1,a0
```

Note that if there are no 16-bit or 32-bit versions of the microMIPS instruction, the assembler will issue an error.

3.4 Labels

All microMIPS functions must be preceded by a label, because microMIPS ISA information is stored in an ELF `st_other` field of the text symbol for the label. The label indicates to the assembler that the following function is in microMIPS mode. Without the label, microMIPS assembly code may be incorrectly disassembled by the utilities, as described in Section 5 "GNU Binutils http://www.gnu.org/software/binutils".

For example:

```
# cat add.s
        .text
        addu    $2, $3, $4
Add:
        addu    $2, $3, $4

# mips-sde-elf-as add.s -o add.o -mmicromips
# mips-sde-elf-objdump -dr add.o

add.o:     file format elf32-tradbigmips


Disassembly of section .text:

00000000 <Add-0x2>:
   0:   05460546        0x5460546 # Decoded in wrong ISA (MIPS32 ISA)

00000002 <Add>:
   2:   0546            addu    v0,v1,a0 # Decoded in microMIPS ISA
```

4 LD for microMIPS

4.1 LD Options

4.1.1 --relax

Using the `--relax` linker option enables the linker to perform *relaxation* for the purpose of implementing additional code-size reductions for microMIPS code.

When relaxation is enabled, the linker scans through all relocatable addresses in the object files, checking symbol values to remove unnecessary instructions or to convert 32-bit microMIPS instructions to 16-bit microMIPS instructions. Note that to pass this option from GCC to LD, you must use the `-Wl,--relax` compiler switch.

Following linker relaxation, microMIPS functions may be aligned to two bytes, which means they cannot be called from MIPS32 code (MIPS32 requires functions to be on a four-byte boundary). For this reason, `--relax` can only be used for microMIPS functions that are not called by MIPS32 code.

The following example shows that `--relax` yields smaller code after linking.

```
# cat relax.s
    .text
    .ent test
    .globl test
    .align 2
test:
    jal  test2
    nop
    .end test

test2:
    jr  $31
    nop

# mips-sde-elf-as relax.s -o relax.o -mmicromips
# mips-sde-elf-ld relax.o -o relax -e test
# mips-sde-elf-objdump -dr relax

relax:     file format elf32-tradbigmips


Disassembly of section .text:

0040006c <test>:
  40006c:       f420 003b       jal     400076 <test2>
  400070:       0000 0000       nop # This NOP is 32 bits
  400074:       0c00            nop

00400076 <test2>:
  400076:       459f            jr      ra
  400078:       0c00            nop
  40007a:       0c00            nop

# mips-sde-elf-ld relax.o -o relax -e test --relax
# mips-sde-elf-objdump -dr relax

relax:     file format elf32-tradbigmips


Disassembly of section .text:

0040006c <test>:
  40006c:       7420 003a       jals    400074 <test2>
  400070:       0c00            nop # This NOP is relaxed to 16 bit
  400072:       0c00            nop

00400074 <test2>:
  400074:       459f            jr      ra
  400076:       0c00            nop
  400078:       0c00            nop
```

4.2 microMIPS and MIPS32 Interlinking

The linker can interlink MIPS32 and microMIPS code automatically by converting JAL to JALX instructions whenever mode switches are required. However, linker errors may occur, such as in the following example.

```
# cat s2.c
void s2() {}

# cat leaf.c
void s2();

void s1()
{
  s2();
}

# mips-sde-elf-gcc -c -O2 s2.c
# mips-sde-elf-gcc -c -O2 leaf.c -mmicromips
# mips-sde-elf-ld s2.o leaf.o -o test -e s1
mips-sde-elf-ld: leaf.o: .text+0x0: jump to stub routine which is not jal
mips-sde-elf-ld: final link failed: Bad value
```

To avoid this error message ("`jump to stub routine which is not jal`"), recompile the microMIPS code with options "`-minterlink-mips16 -mno-jals`", and recompile MIPS32 code with "`-minterlink-mips16`", as discussed in Section 2.1.2 "-mjals, -mno-jals" and Section 2.1.3 "-minterlink-mips16, -mno-interlink-mips16". Also, in assembly files, make sure that no J (direct jump) or JALS instructions are used to call MIPS32 functions.

For example:

```
# mips-sde-elf-gcc -c -O2 leaf.c -mmicromips -minterlink-mips16 -mno-jals
# mips-sde-elf-ld s2.o leaf.o -o test -e s1
# mips-sde-elf-objdump -dr test

test:     file format elf32-tradbigmips


Disassembly of section .text:

0040006c <s2>:
  40006c:       03e00008        jr      ra
  400070:       00000000        nop

00400074 <s1>:
  400074:       4ff5            addiu   sp,sp,-24
  400076:       cbe5            sw      ra,20(sp)
  400078:       f010 001b       jalx    40006c <s2>
  40007c:       0000 0000       nop
  400080:       4be5            lw      ra,20(sp)
  400082:       4706            jraddiusp       24
```

5 Conclusion

As shown in this paper, designers of systems that incorporate the microMIPS ASE can make use of the microMIPS-specific options in GCC, AS, and LD to further improve code size. The reduction in code size has the additional advantages of reducing system memory requirements and power consumption.

For further information about the microMIPS ISA and GCC, the reader is referred to the documents listed below.

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

2. MIPS32® M14K™ Processor Core Family Datasheet
   MIPS Document: MD00666

3. MIPS32® M14Kc™ Processor Core Family Datasheet
   MIPS Document: MD00672

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

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