THE ELF-64™ TOOL CHAIN

APPLICATION NOTE AN-126

By Ketan Deshpande

INTRODUCTION

This application note describes the 64-bit C development tool chain available from Cygnus. The ELF-64™ tool chain is a 64-bit C-compiler tool chain that can be used to generate code for the R4600™ (Orion™) processor operating in an embedded application environment. It is based on the GNU tool chain available in the public domain. The executable created is an ELF (Executable and Linking Format) file.

TOOL CHAIN COMPONENTS

The ELF-64 tool chain consists of the following parts:

1. C-Compiler
2. Assembler
3. Linker
4. Source level Debugger
5. Librarian / Archiver
6. Binary Utilities

The C-compiler is ANSI C compliant and performs optimizations found in all state-of-the-art C-compilers. The compiler generates an intermediate assembly language file from a C file and calls the assembler to generate an ELF object file. The assembler supports the entire MIPS™ ISA (described in the book by Gerry Kane, “MIPS RISC Architecture”). The words “compiler” & “assembler” are used to refer to the cross-development environment too.

The linker links the object files created by the compiler, assembler and the librarian to create an ELF “executable”.

The debugger (gdb) provides remote source level debugging capability over a serial link; this is very useful when developing embedded applications.

The librarian/archiver allows the user to create archives of code sections that are frequently used, for linking with various applications.

The binary utilities are useful in extracting information about the ELF file created, generating a disassembled version of the executable, displaying section size information and converting to different file formats.

COMMAND LINE OPTIONS

The C-compiler and linker support a number of options. This application note mentions only a common subset of these options. For a complete listing and description of all options, the user should refer to the manual.

Compiler options

1. Options controlling the ISA level:

The ELF-64 compiler / assembler supports -mips1, -mips2 and -mips3 switches, to generate code for MIPS ISA I, II or III.

-mips1: Generates code for R30xx processors. This generates instructions that access 32-bit data.

-mips2: Generates code for R4x00 and R60xx processors. This generates instructions that access 32-bit data, and some R4x00 specific instructions.

-mips3: Generates code for R4x00 processors. This generates instructions that access 64-bit data, such as double word accesses.

For best utilization of the 64-bit Orion architecture, the mips3 switch should be used, which is also the default. When using the -mips3 switch, the compiler defaults to using 64-bit general purpose registers and 64-bit floating point registers. Integers and long words are 32-bits long, “long long” words are 64-bits. All addresses generated are 32-bits long. The compiler can be told to use non-default sizes for scalar data types, and to use specific processor pipelines for proper instruction scheduling.

2. Optimization & debugging options:

The most commonly used optimization options are -O and -O2, which perform a number of optimizations. The -O2 option performs all optimizations, except loop unrolling (which can be forced by using -funroll-loops) and omitting the frame pointer (-fomit-frame-pointer). Optimization can be switched off completely using -O0.

The -g option tells the compiler to insert debugging information in the object file. This is necessary when debugging with gdb. A debugging level can be specified (1, 2 or 3), depending on the amount of information the user wants to insert. The default is 2, which is typically sufficient to be able to debug using gdb.

For debugging purposes, using “-g -O0” or just “-g” is recommended. If optimization is also specified during debugging, some statements might get moved around, which could be confusing to the person doing the debugging.

3. Options changing the default data sizes:

The -mlong64 switch forces the compiler to generate code using 64 bit wide long words and addresses (pointers).

The -mint64 switch forces the compiler to generate code using 64 bit wide integers. The -mlong64 switch is assumed in this case.

The -mgp32 switch forces the compiler to generate code assuming the general purpose registers in the Orion are only 32 bits long.

The -mfp32 switch forces the compiler to generate code assuming the floating point registers in the Orion are only 32 bits long.

4. Options for proper scheduling:

Using the “-mcpu=” option tells the compiler to use a specific processor pipeline while scheduling instructions. -mcpu=Orion or -mcpu=r4600 tells the compiler to use the

Orion pipeline, and -mcpu=r4400 tells the compiler to use the R4400 pipeline. The compiler defaults to using -mcpu=Orion.

5. Floating point code generation:

The ELF-64 compiler defaults to generating hardware instructions for performing floating point operations. To force the compiler to use an emulation library, the -msoft-float option is specified, and the appropriate library used, at link time. Since the Orion has a Floating Point Accelerator, a user should never need to use this option, though the capability is available in the tool chain and may be used for future CPU products.

6. Other options:

-nostdinc: This option tells the compiler not to look in the standard include path for the include files. This is useful during embedded applications development, when the user needs to use non-standard libraries, which have their own include files.

-Wa or -Wl: This option allows the user to pass assembler and linker options on the C-compiler command line. e.g. -Wa,-alh instructs the compiler to invoke the assembler to list assembly and high-level source code to the display.

Linker options

1. Options controlling different sections in the executable:

The ELF-64 linker places the different sections in the ELF file at certain default addresses. These addresses can be changed using the -T option. To force the linker to place the .text section at a specific address, the option -Ttext <address> can be used. Similarly, use -Tdata and -Tbss to force the linker to locate the .data and .bss at specific addresses. In a case where all 3 section addresses are specified, it is the user's responsibility to see that the sections do not overlap. The linker uses a default script to place the different sections in the ELF file. Users can specify their own script files, thus finely controlling the appearance of the ELF executable, using the -T<scriptfilename> switch. A discussion of linker scripts is outside the scope of this application note; a sample linker script is shown below:

```
 OUTPUT_FORMAT("elf32-bigmips")   /* Output
file Format */
  OUTPUT_ARCH(mips)
  _DYNAMIC_LINK = 0;
  SECTIONS
  {
    /* Read-only sections, merged into text
segment: */
   /* .text section begins at address 0xbfc00000
*/
    .text 0xbfc00000  :
    {
      _ftext = . ;
      *(.text)
        CREATE_OBJECT_SYMBOLS   /* Create a
symbol for each input file */
      _etext = .;
    }
    .init    ALIGN(8):
      { *(.init)    } =0
    .fini    ALIGN(8)  :
      { *(.fini)    } =0
    .ctors   ALIGN(8)  :
      { *(.ctors)   }
    .dtors   ALIGN(8)  :
      { *(.dtors)   }
    /* Read only data section, aligned on 8-
byte boundary */
    .rodata  ALIGN(8)  :
      { *(.rodata)  }
    .rodata1 ALIGN(8)  :
      {
        *(.rodata1)
        . = ALIGN(8);
      }
    .reginfo . : { *(.reginfo) }
    .data  . :
    {
      _fdata = . ;
      *(.data)
      CONSTRUCTORS
    }
    .data1 ALIGN(8)  :
      { *(.data1) }
    _gp = . + 0x8000;
    .lit8 . : { *(.lit8) }
    .lit4 . : { *(.lit4) }

    /* Keep the small data sections together,
so single-instruction offsets can access them
all, and initialized data all before
uninitialized, so we can shorten the on-disk
segment size.  */

    .sdata   ALIGN(8)  : { *(.sdata) }
    _edata  =  .;
    __bss_start = 0xa0000200 ;
    .sbss   ALIGN(8) : { *(.sbss) *(.scommon)
}
    .bss     0xa0000200 :
    {
    _fbss = .;
     *(.bss)
       *(COMMON)   /*  All  uninitialized  &
unallocated data from all
                 input files */
     _end = . ;
     end = . ;
    }

    /* Debug sections.  These should never be
loadable, but they must have
     zero addresses for the debuggers to work
correctly.  */
    .line         0 :
      { *(.line)          }
    .debug        0 :
      { *(.debug)         }
```

```
    .debug_sfnames    0 :
      { *(.debug_sfnames)     }
    .debug_srcinfo    0 :
      { *(.debug_srcinfo)     }
    .debug_macinfo    0 :
      { *(.debug_macinfo)     }
    .debug_pubnames   0 :
      { *(.debug_pubnames)    }
    .debug_aranges    0 :
      { *(.debug_aranges)     }
}
```

The linker puts “small” data into the small bss (.sbss) and small data (.sdata) sections. “Small” data is data that is smaller than a certain size. This size can be changed from the default 8 bytes using -G <size>. If -G 0 is used, nothing will be placed in .sbss and .sdata. Elements placed in .sbss and .sdata can be accessed in a single instruction using _gp that is appropriately set, resulting in fast data access.

2. Other options:

-nostdlib: This option tells the compiler not to look in the standard library search path for the specified library files. This is useful during embedded applications development, when the user needs to use non-standard libraries.

New instructions

The ELF-64 compiler implements the “branch likely” instructions in the Orion, when -mips2 or -mips3 is specified. When faced with a choice, the compiler attempts to use the conventional branch instruction and fill the branch with a branch independent operation. However, if it cannot do that, it converts the instruction to a branch likely instruction, and copies the target instruction into the branch delay slot.

Another set of instructions implemented by the compiler are those instructions that can give access to unaligned data: LWL, LWR, SWL, SWR, LDL, LDR, SDL, SDR. Using __attribute__ ((packed)) to declare a variable inside a C structure causes the compiler to generate the above instructions whenever the packed data element is accessed.

Assembler Directives

1. **.set mips*n***

This directive allows the user to embed instructions from a higher level MIPS ISA, in a sequence of instructions that belong to another ISA. e.g. .set mips3 would allow the user to specifically enter ISA III instructions in ISA II or ISA I code. .set mips0 resets code generation to the default ISA.

When compiling an assembly file at a specific MIPS ISA, if instructions from a higher ISA are used, the assembler reports a warning, but assembles them anyway.

2. **.set noreorder / .set reorder**

Instructions in the block between the above directives are left as they are; no attempt is made to schedule them according to the pipeline requirements. It is the user’s responsibility to see that the delay slots are properly filled, and hazards are taken care of. The assembler defaults to .set reorder.

3. **.set noat**

This directive instructs the assembler not to use the "at" register, which is used by the assembler to expand certain synthetic instructions. The assembler can be instructed to use the at register using .set at. All the instructions between the .set noat and .set at should be native instructions, or if synthetic instructions are used, should not require the at register. The assembler defaults to .set at.

Binary Utilities

1. **nm**

This utility is used to display the symbol table from an ELF file. It lists the symbols from an ELF object file, along with the virtual address for each symbol. It also displays the section (text, data, bss etc.) in which this symbol was located.

e.g. nm matmult > mat.nm

The following is a part of mat.nm

```
80012000 T start
80012000 A _ftext
800123b0 T main
80012710 T Mult
80012770 T Add
  .....
80018010   B matrix1
8001a720   B matrix2
8001ce30   B matrix3
```

The symbols tagged with a T are text symbols, those with a B are uninitialized data that are placed in the .bss section, and those with a D are initialized data, and are placed in the .data section. The symbols tagged with an A are absolute addresses.

2. **objcopy**

This utility is used to convert the ELF executable to S-record format, suitable for downloading to a board like the IDT evaluation board. This utility can also be used to build S-records from which PROMs can be built (using the -p option). This can also be used to create S-records for byte-wide PROMs (using the -b option), with an interleaving factor, if necessary (using the -i option).

e.g. objcopy -O srec matmult matmult.sre

e.g. objcopy -O srec -p -b 0 -i 1 myprom myprom.sre creates an S-record file that can be used to build the zeroth byte-slice of an interleaved PROM.

3. **objdump**

This utility displays information about ELF object files. It can be used to generate symbol table information, similar to nm, (using the -t switch), generate a disassembly listing (using the -d switch) or section header information (using the -h option).

e.g. objdump -d matmult > matmult.dis

The following is a part of matmult.dis:

```
80012000 <eprol> lui $gp,32770
80012004 <start+4> addiu $gp,$gp,-352
80012008 <start+8> lui $v0,32769
8001200c <start+c> addiu $v0,$v0,32544
80012010 <start+10> lui $v1,32770
80012014 <start+14> addiu $v1,$v1,-2032
```

e.g. objdump -h matmult > matmult.hdr

The following is a part of matmult.hdr:

```
  SECTION 2 [.text]    : size 00004f50 vma
80012000 align 2**4
   ALLOC, LOAD, CODE
  SECTION 3 [.rdata]   : size 00000330 vma
80016f50 align 2**4
   ALLOC, LOAD, READONLY, DATA
  SECTION 4 [.data]    : size 00000c20 vma
80017280 align 2**4
   ALLOC, LOAD, DATA
  SECTION 7 [.sdata]   : size 00000080 vma
80017ea0 align 2**4
   ALLOC, LOAD, DATA
  SECTION 8 [.sbss]    : size 00000060 vma
80017f20 align 2**4
   ALLOC
  SECTION 9 [.bss]     : size 00007890 vma
80017f80 align 2**4
```

4. **size**

This utility is used to display the sizes of all sections in an ELF file, in decimal or hex format. It also displays the total size of all sections in the ELF file.

e.g. size matmult displays:

```
  text  data bss   dec    hex   filename
  20304 4048 30960 55312  d810  matmult
```

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.