GNU Toolchain

2019-09-27

1 About

The GNU toolchain is a collection of programming tools. A version for RISC-V is available at `https://github.com/riscv/riscv-gnu-toolchain`.

The objective of this document is to present the relevant tools for compiling and debugging software, and to offer quick guides and examples about using them.

2 Platforms

2.1 Target Triplet

A target triplet is used to identify systems. It allows build system to know if the code will run.

Some examples are :

- x86_64-linux-gnu
- x86_64-unknown-freebsd
- i686-pc-linux-gnu
- arm-linux-gnueabihf
- riscv32-unknown-elf

Target triplets have the following structure: **machine - vendor - operating system**

The **vendor** field may be left out. In that case, it is assumed to be the default (unknown). As such, x86_64-unknown-freebsd and x86_64-freebsd are the same.

The **operating system** field can be composed of two words. For example, x86_64-linux-gnu is actually:

- machine: x86_64
- vendor: unknown (left out)
- os: linux-gnu

2.2 Build, Host, Target

The **build platform** is the one on which the compilation tools are executed.

The **host platform** is the one on which the code will eventually run.

The **target platform** of a compiler is the one the compiler generates code for.

The GNU tools state the platforms they are configured for. The target triplet can be included in the filename of the binaries:

- aarch64-linux-gnu-gcc
- riscv32-unknown-elf-gdb
- riscv64-unknown-elf-objdump

You may also find it with the -v option:

```
$ gcc -v
[...]
Target: x86_64-linux-gnu
Configured with: [...] --build=x86_64-linux-gnu --host=x86_64-linux-gnu --
    target=x86_64-linux-gnu
[...]
```

Or the -dumpmachine option:

```
$ gcc -dumpmachine
x86_64-linux-gnu
```

2.3 RISC-V extensions

The RISC-V instruction set architecture can use extensions. There is the core ISA, standard extensions, and custom extensions.

Each extension brings its own instructions. Consequently, the toolchain must be configured with the correct architecture, or you might encounter errors. This is especially true with custom non-standard extensions.

You can find information about the target architecture:

```
$ riscv32-unknown-elf-gcc -v
[...]
Target: riscv32-unknown-elf
Configured with: [...] --with-arch=rv32i [...]
[...]
```

3 GCC

GCC is the GNU Compiler Collection. It is a compiler system which supports many languages, including C and C++.

3.1 Basic usage

If *hello.c* is a C source code file, you can compile it into an executable file *hello.exe* with

```
$ gcc -o hello.exe hello.c
```

3.2 Options

GCC has **many** options. A full list is available at `https://gcc.gnu.org/onlinedocs/gcc/Option-Summary.html`

3.2.1 Debugging

Regarding debugging, important options are:

- **-g** : produce debugging information that GDB can use
- **-Og** : optimization level of choice for debugging

3.2.2 Linker

To produce executable files, GCC includes a linker. You may need to provide your own linker script for some target platforms, such as bare-board targets without an operating system.

- **-T script** : use *script* as the linker script

4 GDB

GDB is the GNU debugger. It can be used to:

- load a program to a remote system
- read the target's registers
- step through a program
- debug a program

A full guide of GDB is available at `https://sourceware.org/gdb/download/onlinedocs/gdb/index.html`

4.1 Using GDB for a remote RISC-V target

This section presents a way of debugging a program for a bare-metal application running on a RISC-V target. Assuming you have a *hello.c* file and compiled it into a *hello.exe* executable file:

- launch gdb with the program

```
$ riscv32-unknown-elf-gdb hello.exe
```

- Connect to a GDB translator such as OpenOCD

```
(gdb) target remote [host]:[port]
```

- load the program

```
(gdb) load
```

4.2 Breakpoints, Continuing, Stepping

Setting a breakpoint:

```
(gdb) b location
```

location can be a function name, a line number, or an address of an instruction

```
(gdb) b main
(gdb) b 20
(gdb) b *0x12345678
```

Resume program execution, i.e. continuing:

```
(gdb) c
```

Step through a single source line:

```
(gdb) s
```

Step through a single instruction:

```
(gdb) si
```

4.3 Layouts

Layouts transform the GDB interface and provide more information about the program.

```
(gdb) layout layout_name
```

Layout names are:

- src : displays source
- asm : displays disassembly
- split : displays source and disassembly
- regs : displays registers

4.4 Other

Show all current breakpoints:

```
(gdb) info breakpoints
```

Show all registers and their contents:

```
(gdb) info all-registers
```

Show an expression's value:

```
(gdb) print expression
```

Jump to a *location*:

```
(gdb) j location
```

5 objdump

objdump is a program that can be used to disassemble an executable file.

The program has many options that you can check here: `https://sourceware.org/binutils/docs/binutils/objdump.html`

5.1 Example

The following example uses the options **-D** (disassemble the contents of all sections) and **-S** (display source code intermixed with disassembly).

```
$ riscv32-unknown-elf-objdump -D -S hello.exe
```

This command prints the assembly to stdout. You can pipe it to save it in a file.

```
$ riscv32-unknown-elf-objdump -D -S hello.exe > hello.D
```