Building gcc as a Cross-Compiler

Neil Klingensmith
Department of Electrical and Computer Engineering
University of Wisconsin-Madison, USA
naklingensmi@wisc.edu

September 8, 2011

1 Introduction

This document describes the process for building `gcc` as a cross-compiler on UNIX-based systems. Cross-compiling versions of `gcc` are availible in binary format from commercial organizations such as Code Sourcery [5] and Microcross [7]. It is usually faster to download and install a free binary distribution than to build one from scratch.

The instructions in this document are known to work in Linux-based operating systems as well as in Mac OS X versions 10.5 and 10.6. Please refer to section 1.3 for more details on software prerequisites on different systems.

Please note that this document is a work in progress. Suggestions on how to improve or clarify its content are greatly appreciated.

1.1 Target Architectures

This guide provides explicit instructions for building `gcc` to target the ColdFire (m68k) architecture processors manufactured by Freescale. However, the instructions in this guide have been confirmed to work for ARM targets as well. In order to build for a different target architecture, modify the `--target=` field in the configure commands to suit your needs. For example, to compile for ARM, specify

```
--target=arm-eabi
```

on the configure command line instead of `--target=m68k-elf`.

1.2 Disclaimer

The author of this document, the University of Wisconsin, and the Engineering Department, assume no responsibility for damage caused to property during attempts to follow the instructions in the guide. Instructions in this guide are provided without warranty of any kind, explicit or implied.

1.3 Requirements

This section explains prerequisites for different operating systems. In general, the procedure should work on any system with a working native version of `gcc`.

Please note that the **compilation and installation process should be run as the root user to avoid problems with file access permissions**.

In the past, some users reported problems compiling a working version of `gcc` and related utilities using certain combinations of `gcc` and binutils[1][6].

1.3.1 Linux

The procedure outlined in this guide has been tested and is known to work with Gentoo Linux[2]. No modifications to the procedure or additional software needs to be installed.

Other distributions of the GNU/Linux operating system may require the user to explicitly install `gcc`.

1.3.2 Mac OS X

This guide is known to be compatible with Mac OS versions 10.5 and 10.6. It has not been tested on versions 10.7 or higher. Reports of success or failure would be appreciated.

Since Mac OS X is not distributed with a native version of `gcc`, **the Apple developer tools (XCode) must be installed before proceeding with the instructions in this guide.**

Additionally, Mac OS users need to install a GNU version of the mpfr library. MPFR can be installed with the Macports package manager (http://www.macports.org). Once Macports is present, use the command

```
port install mpfr
```

to install mpfr.

2 Procedural Overview

The build process will proceed as follows:

1. Compile binutils for the embedded target.
2. Build a bootstrap version of `gcc`.
3. Build newlib (an embedded C library) using the bootstrap version of `gcc`.
4. Rebuild `gcc` against newlib.

The compilation process is complicated by a circular dependence between `gcc` and newlib. The compiler must be present on the system to compile newlib for the embedded target. However, `gcc` refers to symbols in the C library when compiling user programs, so `gcc` must be recompiled against newlib.

3 Procedure

The source code for gcc [1] and binutils [3] is available from the GNU website, and newlib [4] is available from Red Hat.

3.1 Binutils

Extract binutils to a working directory:

```
tar xvjf binutils-2.19.1.tar.bz2
```

Use this procedure to extract the gcc and newlib packages as well.

Create a new directory under binutils called build. This is the directory from which the compilation commands will be issued. Also, make a directory for the binaries called m68k-elftools.

```
mkdir binutils-2.19.1/build

mkdir /opt/local/m68k-elftools

cd binutils-2.19.1/build
```

[1] GNU object file manipulation suite, required to build gcc

Note that the directory paths for the above command lines will vary depending on the version of binutils.

Configure binutils.

```
../configure --target=m68k-elf
--prefix=/opt/local/m68k-elftools
--enable-interwork --enable-multilib
--with-gnu-as --with-gnu-ld --disable-nls
```

Build and install binutils.

```
make -j3
make install
```

3.2 Bootstrap gcc

Change to the gcc/build directory and configure the bootstrap version of gcc.

```
cd ~/gcc-x.x.x
mkdir build
cd build
```

Build and install the bootstrap version of gcc.

```
../configure
--with-libiconv-prefix=/opt/local
--target=m68k-elf
--prefix=/opt/local/m68k-elftools/
--enable-interwork --enable-multilib
--enable-languages="c" --with-newlib
--without-headers --disable-shared
--with-gnu-as --with-gnu-ld
--with-gmp=/opt/local
--with-mpfr=/opt/local --disable-libssp
```

```
make -j3[2]
make install
```

3.3 Newlib

Change to the newlib directory and configure newlib using the bootstrap version of gcc.

```
cd ~/newlib-x.x.x
mkdir build
cd build
```

Build and install newlib.

```
../configure --target=m68k-elf
--prefix=/opt/local/m68k-elftools/
--enable-interwork --enable-multilib
--with-gnu-as --with-gnu-ld --disable-nls
```

```
make -j3
make install
```

3.4 Final gcc

Change back to the gcc directory and rebuild gcc against newlib.

```
cd ~/gcc-x.x.x/build
rm -rf *
```

[2]The -jN option specifies the number of concurrent jobs run by make. A commonly accepted number of jobs is 1 + (number of cores). For example, on a dual-core machine, the command to begin the build process would be make -j3.

```
../configure
--with-libiconv-prefix=/opt/local
--target=m68k-elf
--prefix=/opt/local/m68k-elftools/
--enable-interwork --enable-multilib
--enable-languages="c" --with-newlib
--disable-shared --with-gnu-as
--with-gnu-ld --with-gmp=/opt/local
--with-mpfr=/opt/local --disable-libssp

               make -j3

             make install
```

Add the binary directory to the PATH environment variable.

```
export
PATH=/opt/local/m68k-elftools/bin:$PATH
```

This will cause the terminal to automatically recognize commands like `m68k-elf-gcc`. The PATH variable is reset each time a users logs in, so the above command should be added to each user's bash profile script. The location of the profile script varies depending on the version of bash, but is usually located in the user's home directory and named `.profile` or similar.

References

[1] Gcc, the gnu compiler collection. http://gcc.gnu.org/.

[2] Gentoo linux. http://www.gentoo.org.

[3] Gnu binutils. http://www.gnu.org/software/binutils/.

[4] Newlib. http://sourceware.org/newlib/.

[5] CodeSourcery. http://www.codesourcery.com.

[6] Dan Kegel. Crosstool build results. http://kegel.com/crosstool/crosstool-0.43/buildlogs/.

[7] Microcross. http://microcross.com.