Cross Compiler HOWTO

How to Build a C Cross Compiler for a Target Processor

Building C Cross Compilers for the PowerPC (PPC403GC), MC68000 family (MC68000, etc.), and MIPS targets:

The cross compiler described in this note is created by adapting the GCC compiler developed by the Free Software Foundation (http://www.fsf.org/software/software.html). This compiler may be used to compile programs written in C, C++, or Objective C. The GCC Home Page is located at http://www.fsf.org/software/gcc/gcc.html. This page contains links to locations that can supply the source code for GCC. In order to provide assembly, linking, and other services, we will also need the current "binutils" package and its documentation. The binutils Home Page is located at http://www.fsf.org/software/binutils/binutils.html and its documentation is located at http://www.fsf.org/manual/binutils/index.html. The GNU Debugging Program may also be used with a target computer. For more information also refer to Jeff Webb's Cross-Compiler page at http://bode.hogtown.ufl.edu/~mustang/gnu/gcc/cross/gcc-cross.html

These instructions refer to using a Linux host system, i.e., the compilers actually execute on the host system, they generate code for the target CPU. You may have to help the configure command figure out which host system you are running. The instructions are essentially the same when using a Win9X host. The package to get is call dev-src.tar.bz2 (~24 Mb) which is available from ftp://ftp.sunsite.utk.edu/pub/cygwin/cygwin-b20/. In order to build a Win9X hosted development system, you will also need to install the file "full.exe" from the same site. This tool is the main component of the Cygwin projects development system. Its web page is at http://sourceware.cygnus.com/cygwin.

To build a cross compiler for a specific processor one compiles the programs in the files specified below using a specific "target selection" option on the command line. The target options are "--target=powerpc-ibm-eabi" for the PPC403GC, "--target=m68k-coff" for the MC68000 family processors (including the Coldfire), and "--target=mips-elf" for the MIPS processor.

Get (ftp) the following files from a source specified by the documentation given above. Here we used ftp://ftp.gnu.org/

```
/pub/gnu/binutils/binutils-2.9.1.tar.gz         (~5.56 MB)
/pub/gnu/gcc/gcc-2.95.2.tar.gz                  (~12.5 MB)
/pub/gnu/gdb/gdb-4.18.tar.gz                    (~11.38 MB)
```

These files are also on recent CDROM distributions of the Linux system. Look for the GNU source file directory.

Place the tar.gz files in a temporary directory, e.g., ~/cross_compiler_src. The gdb file is needed only if you want to use gdb as a program debugger for your target system.

Unpack the first two files into a temporary directory for the specific cross compiler in your home directory. I used "ppc_cross_compiler" as a temporary directory in which to build the PowerPC cross utilities, "m68k_cross_compiler" for the 68000 version, and "mips_cross_compiler" for the MIPS version.

The following steps accomplished this task:

```
> cd ~
> mkdir ppc_cross_compiler
> cd ppc_cross_compiler
> tar -xvzf ~/cross_compiler_src/binutils-2.9.1.tar.gz
> tar -xvzf ~/cross_compiler_src/gcc-2.95.2.tar.gz
```

These steps create a ppc_cross_compiler directory as a subdirectory off of your home directory. The "tar" commands directly uncompress, create suitable subdirectories, and place the files where they belong.

Also, a similar series of steps should be performed to create the MIPS cross compiler:

```
> cd ~
> mkdir mips_cross_compiler
> cd mips_cross_compiler
> tar -xvzf ~/cross_compiler_src/binutils-2.9.1.tar.gz
> tar -xvzf ~/cross_compiler_src/gcc-2.95.2.tar.gz
```

Binutils:

The next step is to build the assembler, linker, and other tools (as, ld, objcopy, etc.). These are in the binutils package.

For the PowerPC version:

```
> cd ppc_cross_compiler/binutils-2.9.1
> configure --target=powerpc-ibm-eabi --program-prefix=ppc-
```

If configure has trouble identifying your host system add the command line parameter --host=i486-unknown-linux. The "linux" word may have the form "linuxold" or "linuxaout". Later Linux systems are using the "ELF" form and this is the default for "linux". The problem shows up when configure finds out that you have a Pentium (586) or Pentium Pro (686). At the moment, systems are not identified that way.

A similar line is used for configuring to build the m68k binutils.

```
> cd m68k_cross_compiler/binutils-2.9.1
> configure --target=m68k-coff --program-prefix=m68k-
```

Here configure prepares for the linker producing the "COFF" output file format. We will rely on objcopy (part of the binutils) to do file conversions for our target systems.

A similar line is used for configuring to build the mips binutils.

```
> cd mips_cross_compiler/binutils-2.9.1
> configure --target=mips-elf --program-prefix=mips-
```

Here configure prepares for the linker producing the "ELF" output file format. We will rely on objcopy (part of the binutils) to do file conversions for our target systems.

The next step is to "make" the binary utilities. We now change to the appropriate binutils-2.7 directory and do the "make":

PowerPC:

```
> cd ppc_cross_compiler/binutils-2.2.9.1
> make
```

M68K

```
> cd m68k_cross_compiler/binutils-2.2.9.1
> make
```

MIPS

```
> cd mips_cross_compiler/binutils-2.2.9.1
> make
```

"Making" of the utilities takes a while, especially on a slow host machien. If the process gets into an infinite loop, check your system clock (date) to see if the date is set more or less correctly. We've had this problem on several machines. Make gets into a loop remaking itself.

After the "make" operation completes, change to root by using "su" and then type (you are still in the appropriate binutils directory):

```
# make install
```

The install step places the files in the place specified in the configure step, see appendix 1. After this step completes, check in the "bin" directory specified for installation (default is /usr/local/bin) to see if several programs with the "ppc-" or "m68k-" prefix were installed.

GCC:

Now we can compile gcc. Note that the gcc directory was created when the "tar-file" was "untarred" above. Move to the gcc directory in the appropriate temporary directory created above.

```
> cd ppc_cross_compiler/gcc-2.95.2
 or
> cd m68k_cross_compiler/gcc-2.95.2
 or
> cd mips_cross_compiler/gcc-2.95.2
```

In either case edit the file "libgcc2.c" and add the following line before any "#include ..":

```
#define inhibit_libc
```

This line inhibits the creation of the stdio part of libc. We would normally prepare our own versions of the character I/O routines.

Now type the following to build the Makefile that builds gcc:

```
> configure --target=powerpc-ibm-eabi --prefix=/usr/local --program-prefix=ppc- --program-suffix=
        or
> configure --target=m68k-coff --prefix=/usr/local --program-prefix=m68k- --program-suffix=
        or
> configure --target=mips-elf --prefix=/usr/local --program-prefix=mips- --program-suffix=
```

The command line for configure is very long. Do not type "Return" until after the "...suffix=". Use the "--host=" parameter above if configure has trouble identifying your system. The result of the configuration is to create a Makefile for a compiler called ppc-gcc (m68k-gcc). When run, make will compile ppc-gcc (m68k-gcc) place it in /usr/local/bin.

Once configure completes its work, "make" the compiler by typing:

```
> make LANGUAGES=c
```

Much later, if everything has gone alright, change to root (su) and type (you should still be in the appropriate gcc directory):

```
# make install LANGUAGES="c c++"
```

When this step completes, the entire tool set will have been compiled and installed in /usr/local/bin. "Include" and "lib" files will have been placed in corresponding directories in /usr/local (include and lib). Suitable man pages will be placed in the "man" system so that "man ppc-gcc" will show the man page for the new compiler. It looks very much like the one for gcc.

Using the tools is similar to doing normal "C" compilation on your Linux system. As mentioned above, the man pages work. Here are some simple command lines to compile, assemble, and link various types of programs.

```
> ppc-gcc -S sourcefile.c -o sourcefile.s
> ppc-gcc sourcefile.c -o sourcefile
> m68k-gcc sourcefile.c -o sourcefile
> mips-gcc sourcefile.c -o sourcefile
```

GDB: (Preliminary - m68k example)

```
> cd m68k_cross_compiler
> tar -xvzf ~/cross_compiler_src/gdb-4.18.tar.gz
> ls
binutils-2.9.1/  gcc-2.95.2/   gdb-4.18/

> cd gdb-4.18
> configure --target=m68k-coff --program-prefix=m68k-
Configuring for a i486-unknown-linux host.
Created "Makefile" in /home/mal/m68k_cross_compiler/gdb-4.18 using
"config/mh-linux"

> make
```

Appendices:

Appendix 1: Command line arguments to "configure" - Matt Chidester
The --target= is obvious. The "eabi" part refers to the embedded flavor of PowerPC. For a list of supported targets, check near the middle of the INSTALL file in the gcc directory. There are several flavors of 68k targets. The m68k-coff, eabi, and mips-elf do not require a resident operating system on the target.

The --program-prefix= (and a corresponding --program-suffix=) are strings which will be attached to the program names so they don't get confused with the native assembler/linker/etc. I have it set up to name them as ppc-*, m68k-*, and mips-*. By default, they'd be called powerpc-ibm-eabi-*, but I'm waaaay too lazy to type all that every time.

Another useful command line argument is --prefix=. This is the directory where the outputs will be places. The default is --prefix=/usr/local which works rather well. The binaries are put in /usr/local/bin, the libraries in /usr/local/lib, etc.

Appendix 2: Some notes on the place of gdb in the toolset - MAL
Gdb is a program debugger from the Free Software Foundation (GNU) that runs on the host computer. It can greatly assist in the debugging of assembler, C, C++, and ObjC programs. You have possibly encountered it when debugging programs compiled for your host system. Gdb can also help debug programs on a target system connected to the host's serial port or network link. Certain functions must be embedded in the target system to assist with communications with the host. These are relatively simple programs and some versions are supplied with the distribution.

Look in the directory /usr/doc/gdb on your favorite Linux system. Read the files README and NEWS. There is

more information than that given here. Also the gdb manual in postscript is available on the ftp site (for EEL5745).

Here is a note from the gdb README file:

```
The files m68k-stub.c, i386-stub.c, and sparc-stub.c are examples of remote
stubs to be used with remote.c.  They are designed to run standalone on an
m68k, i386, or SPARC cpu and communicate properly with the remote.c stub
over a serial line.

The file rem-multi.shar contains a general stub that can probably run on
various different flavors of unix to allow debugging over a serial line
from one machine to another.
```

At the moment, the tool combination of gdb and some routines in your target system (possibly as part of your monitor) gives you a debugging system that is reminiscent of the PCBUG/68HC11EVBU combination used in EEL4744 some semesters.

Here is another note from the gdb README file:

```
X Windows versus GDB
====================

There is an "xxgdb", which seems to work for simple operations,
which was posted to comp.sources.x.

For those interested in auto display of source and the availability of
an editor while debugging I suggest trying gdb-mode in GNU Emacs
(Try typing M-x gdb RETURN).  Comments on this mode are welcome.

Those interested in experimenting with a new kind of gdb-mode
should load gdb/gdba.el into GNU Emacs 19.25 or later.  Comments
on this mode are also welcome.
```

The X Window front end for gdb is called xxgdb (there are several versions of this). A little from the xxgdb man page will give you a peek at its capabilities:

```
Xxgdb is a graphical user interface to  the  gdb  debugger
under  the  X  Window System.  It provides visual feedback
and mouse input for the user to control program  execution
through  breakpoints, to examine and traverse the function
call stack, to display values of variables and data struc-
tures, and to browse source files and functions.
```

Another selection is also instructive:

```
Xxgdb consists of the following subwindows:

File Window         Display  the full pathname of the file
                    displayed in the  source  window,  and
                    the line number of the caret.

Source Window       Display the contents of a source file.
```

Message Window	Display the execution status and error messages of xxgdb .
Command Window	Provide a list of the common gdb commands which are invoked by simply clicking the LEFT mouse button.
Dialogue Window	Provide a typing interface to gdb.
Display Window	Provide a window for displaying variables each time execution stops.
Popup Windows	Provide windows for displaying variables (see "Displaying C Data Structures" below).

The relative sizes of the source window, command window, and the dialogue window can be adjusted by dragging the grip (a small square near the right edge of a horizontal border) with the LEFT mouse button down.

Appendix 3: m68k-as

For EEL5745, use --m68008 or --m5200 (Coldfire) as a command line argument. Also the command line argument "--register-prefix-optional" can be used to eliminate requirement for "%" prefix on register names.

Some points of difference between the GNU Assembler m68k-as and A68k.

	A68K	**m68k-as**
1) Registers:	D0,..,D7,A0,..,A7	%D0,..,%D7,%A0,..,%A7 can be shut off using --register-prefix-optional on command line
2) Hexadecimal:	$FF00E	0xFF00E
3) Equate:	UCR EQU $14	.EQU UCR,0x14