Cross Compilation

Suresh Purini, IIIT-H

I copied substantial part of this document verbatim from a section of the article "Build a GCC-based cross compiler for Linux". So please do not circulate this hand-out on the web and it is strictly for the course pupose. Also some places there is an use of "I", it is not "me", it is the author of the orignal document.

1 What is Cross Compilation?

There are times when the platform you're developing on and the computer you're developing for don't match. For example, you might want to build a PowerPC/Linux application from your x86/Linux laptop. Whenever the development and deployment platforms are different, we call it as *cross compilation.*

2 The Need for a Cross Compiler

It isn't always possible to write and build an application on the same platform. For many embedded environments, for example, the reduced memory space – often less than 256 MB, maybe even less than 64 MB, both for RAM and for storage – just isn't practical. A reasonable C compiler, the associated tools, and the C Library required won't fit into such a small space much less run.

Actually developing in such an environment is obviously even more difficult. Niceties like a full-blown editor (for example, emacs) or a full-blown development environment are unavailable, assuming that you can even access and use the system with a keyboard and display. Many embedded solutions don't even have the benefit of network access.

Cross-compilers enable you to develop on one platform (the host) while actually building for an alternative system (the target). The target computer doesn't need to be available: All you need is a compiler that knows how to write machine code for your target platform. Cross-compilers can be useful in other situations, too. I once had to work on a computer with no C compiler installed, and I had no easy way of obtaining a precompiled binary. I did, however, have the necessary source code for the GNU Compiler Collection (GCC), a C library (newlib), and the binary utilities on a computer that did have a C compiler. With these tools, I was able to build first a cross-compiler and then a native compiler for the target computer that I could copy across and use directly.

Cross-compilers can also be handy when you have a slow machine and a much faster one and want to build in minutes rather than hours or days. I've used this method to perform an upgrade to a new software version on a computer on which it would have taken 2-3 days to rebuild all the components – all while the machine was performing its already-significant server duties.

3 Example

This section is not copy guys. I spent lot of time to draw these pictures :)

Figures 1 and 2 shows examples of native and cross compilation. Figure 3 shows a compiler

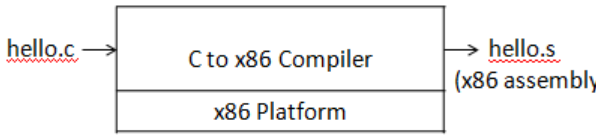

Figure 1: Native Compilation

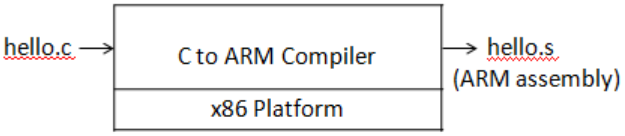

Figure 2: Cross Compilation

running x86 platform translation the C-to-ARM compiler `gcc-c2arm.c` in to x86 assembly code `gcc-c2arm.s`. This program can run on any x86 machine and translates any C program to ARM assembly. So if we supply the C-to-ARM compiler `gcc-c2arm.c` to this program (refer Figure 4, it generates ARM assembly code `gcc-c2arm.s`, which can only run on any ARM processor based system.

Figure 3:

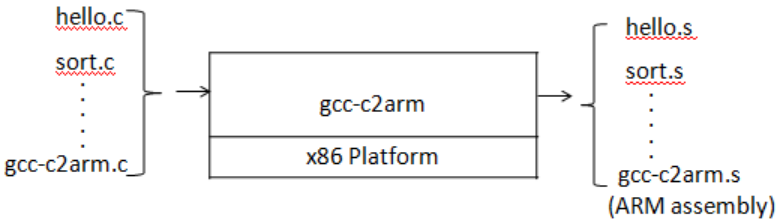

Figure 4: