

## Lecture 3: Toolchains

Instructor: Prabal Dutta

Date: October 2, 2012

*Note: Unless otherwise specified, these notes assume: (i) an ARM Cortex-M3 processor operating in little endian mode; (ii) the ARM EABI application binary interface; and (iii) the GNU GCC toolchain.*

**Toolchains**

A complete software toolchain includes programs to convert source code into binary machine code, link together separately assembled/compiled code modules, disassemble the binaries, and convert their formats.

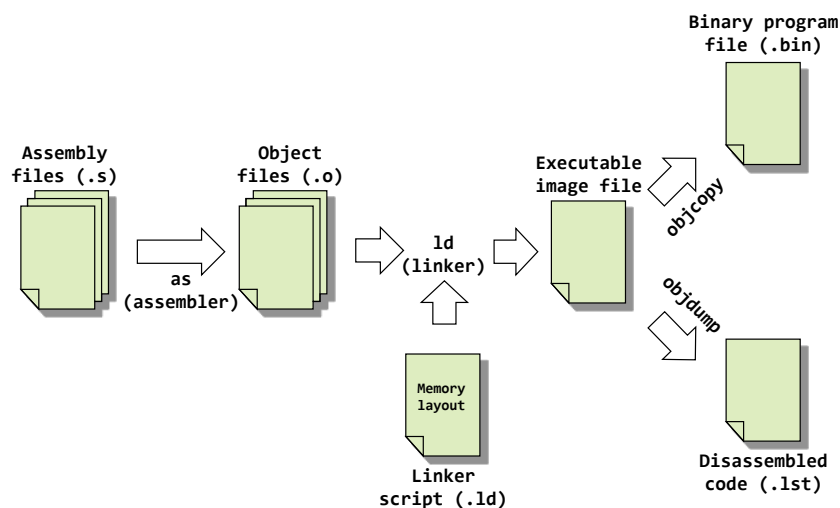


Figure 0.1: Assembler Toolchain.

A typical GNU (GNU's Not Unix) assembler toolchain includes several programs that interact as shown in Figure 0.1 and perform the following functions:

- **as** is the *assembler* and it converts human-readable assembly language programs into binary machine language code. It typically takes as input `.s` assembly files and outputs `.o` object files.
- **ld** is the *linker* and it is used to combine multiple object files by resolving their external symbol references and relocating their data sections, and outputting a single executable file. It typically takes as input `.o` object files and `.ld` linker scripts and outputs `.out` executable files.
- **objcopy** is a *translation* utility that copies and converts the contents of an object file from one format (e.g. `.out`) another (e.g. `.bin`).
- **objdump** is a *disassembler* but it can also display various other information about object files. It is often used to disassemble binary files (e.g. `.out`) into a canonical assembly language listing (e.g. `.lst`).

If you plan to write C, C++, or mixed assembly/C code, then you will also need a compiler. The GNU Compiler Collection (GCC) includes all of the tools described previously as well as a C/C++ compiler. The

C/C++ compiler (`gcc/g++`) can take as input a mix of C and C++ source files, pre-assembled object files, and pre-compiled library files, and from these inputs it can generate object files, call the linker, and output executable image files, as Figure 0.2 shows.

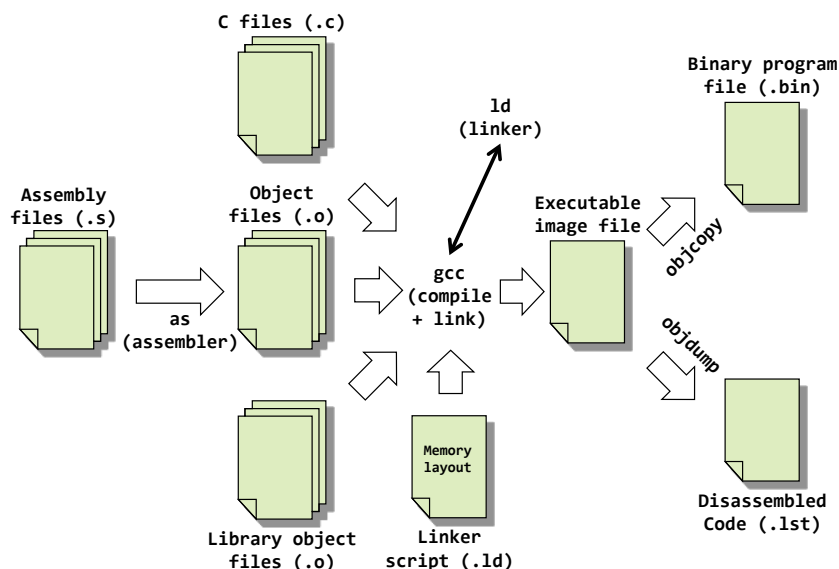


Figure 0.2: Compiler/Assembler Toolchain.

There are a number of other binary tools, in addition to the ones shown above, that are available in the GNU `binutils` collection. They include:

- `ar` is a utility for creating, modifying and extracting from archives.
- `nlmconv` converts object code into an NLM.
- `nm` lists symbols from object files.
- `ranlib` generates an index to the contents of an archive.
- `readelf` displays information from ELF-format object file.
- `size` displays the sections of an object or archive, and their sizes.
- `strip` Discards symbols embedded in object files.

## Installation

Installing a software toolchain is the first step in developing software for a new hardware platform. This involves identifying and installing the assembler, linker, compiler, software libraries, simulators, programmers, and other supporting tools for the target architecture. In this case, our target is the ARM Cortex-M3 processor which is based on the ARMv7 architecture. Our preferred toolchain, for reasons of cost and accessibility, is GNU GCC compiler toolchain. GCC is a popular, open source toolchain that can generate code for a wide range of architectures including Intel's x86, ARM v4/5/6/7, TI's MSP, Atmel's AVR, and many others. And, since most GCC tools are free for download, they can be installed by anyone! Note: when the *host* and *target* architectures are different, the toolchain is called a *cross compiler* (e.g. if you develop code

on a Linux machine based on the x86 architecture, but you're compiling for an ARM target, then you need Linux-based ARM-targeting cross compiler).

Unix cross compiler naming conventions can seem mystifying. If you search for an ARM compiler, you might stumble across the following toolchains: `arm-none-linux-gnueabi`, `arm-none-eabi`, `arm-eabi`, and `arm-softfloat-linux-gnu`, among others. This might leave you wondering about the method to the naming madness. Unix cross compilers are *loosely* named using a convention of the form `arch[-vendor][-os]-abi`. The `arch` refers to the target architecture, which in our case is ARM. The `vendor` nominally refers to the toolchain supplier. The `os` refers to the target operating system, if any, and is used to decide which libraries (e.g. newlib, glibc, crt0, etc.) to link and which syscall conventions to employ. The `abi` specifies which application binary interface convention is being employed, which ensures that binaries generated by different tools can interoperate.

Some examples will illustrate the rough naming convention:

- `arm-none-eabi` is the toolchain we use in this class. This toolchain targets the ARM architecture, has no vendor, does not target an operating system (i.e. targets a “bare metal” system), and complies with the ARM EABI.
- `i686-apple-darwin10-gcc-4.2.1` is the version of GCC on my MacBook Pro. This toolchain targets the Intel i686 architecture, the vendor is Apple, and the target OS is Darwin version 10.
- `arm-none-linux-gnueabi` is the toolchain that can be installed in Debian-based systems using a package manager like `apt` (the package is called `gcc-arm-linux-gnueabi`). This toolchain targets the ARM architecture, has no vendor, creates binaries that run on the Linux operating system, and uses the GNU EABI. In other words, it is used to target ARM-based Linux systems.
- `arm-eabi` (Android ARM compiler)

**Linux without Admin Rights (e.g. CAEN).** If you *do not* have admin access to a Linux machine (or virtual machine), *or* you want to install a suite with built-in simulator, you could install the ARM GCC toolchain by downloading pre-built Sourcery CodeBench Lite binaries from Mentor Graphics (note: these are command-line only tools).

Use your own Linux machine or login to a CAEN Linux machine, e.g.

```
$ ssh -XY uniqueness@loginlinux.engin.umich.edu
```

Open a web browser and go to the following web site:

```
https://sourcery.mentor.com/sgpp/lite/arm/portal/release1802
```

Click the **Download Lite Edition** button, which will take you a registration page. Enter your First Name, Last Name, Email, Country, and ZIP code. Note that you need to use a *valid email address* since a download URL is emailed to the address that you provide.

Click the **Get Lite!** button, which will then take you to a another page that states, “Check your E-mail for Download Link.” Check your email for the download URL.

Open the download link that was emailed to you. This should take you to the “Sourcery Tools Support Portal” and land you on the “Sourcery G++ Lite 2011.03-42 for ARM EABI” page. You’ll see several *Packages* that you can *Download*.

Click the **IA32 GNU/Linux TAR** link under the **Advanced Packages** option and save the 35 MB file that you download (`arm-2011.03-42-arm-none-eabi-i686-pc-linux-gnu.tar.bz2`) in your home directory.

Download the “Getting Started Guide (PDF)” (`getting-started.pdf`) that is linked from the **Documentation** section.

Read §2.4.3 of the “Getting Started Guide (PDF)” to familiarize yourself with installing Sourcery G++ Lite from a compressed archive. Once you’ve downloaded the `.bz2` file to your home directory, you can unzip the downloaded file and extract the tar into a new directory (e.g. `$HOME/arm-tools`, in my case), using commands like:

```
% cd ~
% ls -la arm*
-rw-r--r--. 1 prabal users 37272394 Sep 23 17:44 arm-2011.03-42-arm-none-eabi-i686-pc...
% bunzip2 arm-2011.03-42-arm-none-eabi-i686-pc-linux-gnu.tar.bz2
% mkdir -p $HOME/arm-tools
% cd arm-tools/
% tar xf ../arm-2011.03-42-arm-none-eabi-i686-pc-linux-gnu.tar
```

Read §2.6.2 to familiarize yourself with setting up the environment on GNU/Linux hosts.

If you use the C Shell (`csh` or `tcsh`), then the following command will place the toolchains binaries on the system’s executable search path.

```
% setenv PATH $HOME/arm-tools/arm-2011.03/bin:$PATH
```

If Bourne Shell (`sh`), Bourne-again Shell (`bash`), Korn Shell (`ksh`), or another shell, use:

```
% PATH=$HOME/arm-tools/arm-2011.03/bin:$PATH
% export PATH
```

And if you don’t want to re-enter these commands every time you login or open a new shell, then you will want to add these lines into your shell’s startup configuration file, e.g. `.cshrc`, `.bashrc`, and/or `.bash.profile`.

At this point, you should have the toolchain installed and accessible. Typing `arm-none-eabi-gcc -v` should print a bunch of compiler details and then print as the last line the following, assuming everything installed as expected:

```
gcc version 4.5.2 (Sourcery G++ Lite 2011.03-42)
```

It turns out that many of the utilities that we learned about, like the assembler, are not found in toolchain’s `bin` directory.

```
% arm-none-eabi-as
arm-none-eabi-as: Command not found.
```

Typing `ls` in the toolchain’s `bin` directory shows why: `arm-none-eabi-as` isn’t there!

```
% ls -la
-rwxr-xr-x. 1 prabal users 569376 Apr 20 2011 arm-none-eabi-addr2line
```

```

-rwxr-xr-x. 2 prabal users 223944 Apr 20 2011 arm-none-eabi-c++
-rwxr-xr-x. 1 prabal users 567936 Apr 20 2011 arm-none-eabi-c++filt
-rwxr-xr-x. 1 prabal users 222600 Apr 20 2011 arm-none-eabi-cpp
-rwxr-xr-x. 1 prabal users 18612 Apr 20 2011 arm-none-eabi-elfedit
-rwxr-xr-x. 2 prabal users 223944 Apr 20 2011 arm-none-eabi-g++
-rwxr-xr-x. 2 prabal users 221768 Apr 20 2011 arm-none-eabi-gcc
-rwxr-xr-x. 2 prabal users 221768 Apr 20 2011 arm-none-eabi-gcc-4.5.2
-rwxr-xr-x. 1 prabal users 26812 Apr 20 2011 arm-none-eabi-gcov
-rwxr-xr-x. 1 prabal users 3378472 Apr 20 2011 arm-none-eabi-gdb
-rwxr-xr-x. 1 prabal users 3378472 Apr 20 2011 arm-none-eabi-gdbtui
-rwxr-xr-x. 1 prabal users 630916 Apr 20 2011 arm-none-eabi-gprof
-rwxr-xr-x. 1 prabal users 340780 Apr 20 2011 arm-none-eabi-readelf
-rwxr-xr-x. 1 prabal users 771476 Apr 20 2011 arm-none-eabi-run
-rwxr-xr-x. 1 prabal users 572292 Apr 20 2011 arm-none-eabi-size
-rwxr-xr-x. 1 prabal users 369120 Apr 20 2011 arm-none-eabi-sprite
-rwxr-xr-x. 1 prabal users 572224 Apr 20 2011 arm-none-eabi-strings

```

A number of other programs, including the assembler (as), linker/loader (ld), objcopy (objcopy), and objdump (objdump) appear to be missing. In reality, they exist in the arm-2011.3/arm-none-eabi/bin directory, but the readme in that directory states:

```

The executables in this directory are for internal use by the compiler
and may not operate correctly when used directly. This directory
should not be placed on your PATH. Instead, you should use the
executables in ../../bin/ and place that directory on your PATH.

```

Sounds like an ominous warning! But, we'll ignore this warning for pedagogical reasons and create soft links to the executables in this directory from the toolchain's standard bin directory as follows:

```

% cd $HOME/arm-tools/arm-2011.03/bin
% ln -s ../arm-none-eabi/bin/ar arm-none-eabi-ar
% ln -s ../arm-none-eabi/bin/as arm-none-eabi-as
% ln -s ../arm-none-eabi/bin/ld arm-none-eabi-ld
% ln -s ../arm-none-eabi/bin/nm arm-none-eabi-nm
% ln -s ../arm-none-eabi/bin/objcopy arm-none-eabi-objcopy
% ln -s ../arm-none-eabi/bin/objdump arm-none-eabi-objdump
% ln -s ../arm-none-eabi/bin/ranlib arm-none-eabi-ranlib
% ln -s ../arm-none-eabi/bin/strip arm-none-eabi-strip

```

At this point, you should have a working installation. Try typing in the following into a file called main.c:

```

#include <stdio.h>
int factorial(int n) {
    if (n == 0)
        return 1;
    return n * factorial (n - 1);
}
int main () {
    int i;
    int n;

```

```

    for (i = 0; i < 10; ++i) {
        n = factorial (i);
        printf ("factorial(%d) = %d\n", i, n);
    }
    return 0;
}

```

Assuming you entered the program correctly, the following line should compile and link this program:

```
% arm-none-eabi-gcc -o factorial main.c -T generic-hosted.ld
```

And at this point, you should be able to use the simulator to run the executable:

```

% arm-none-eabi-run factorial
factorial(0) = 1
factorial(1) = 1
factorial(2) = 2
factorial(3) = 6
factorial(4) = 24
factorial(5) = 120
factorial(6) = 720
factorial(7) = 5040
factorial(8) = 40320
factorial(9) = 362880

```

You can now try out various examples provided in lecture notes for Class #3 – ISA, Assembly, Toolchains.