Guide: GCC and Bare Metal Programming

Written by Pat Hanrahan and Julie Zelenski

This guide gives a brief overview of what is unique about compiling C programs to execute in a bare-metal environment.

Hosted versus non-hosted (standalone) environments

A typical program is compiled for a *hosted* system where it has access to the standard libraries and facilities provided by the operating system layer. In hosted mode, the program runs at the pleasure of the host operating system. In contrast, a bare metal program is non-hosted; it does not stand on top of an operating system or library; it runs entirely standalone. The program has the freedom to do whatever it wants, without any pesky interference from a overbearing OS, but cannot count on any facilities other than what it provides for itself.

By default, `gcc` compiles assuming a hosted environment, since this is the common case. To properly compile a bare metal program, we need to set the appropriate compiler and linker options to ensure the program is configured to run standalone.

Compiler option `-ffreestanding`

This `gcc` option directs the compiler to limit this program to only those features available in the freestanding environment.

```
$ arm-none-eabi-gcc -ffreestanding -c blink.c
```

In freestanding mode, the only available standard header files are: `<float.h>`, `<iso646.h>`, `<limits.h>`, `<stdarg.h>`, `<stdbool.h>`, `<stddef.h>`, and `<stdint.h>` (C99 standard 4.6). These headers define the types appropriate for the machine being used, as well as useful constants such as the minimum and maximum values for different types. The other standard header files (`<stdio.h>`, `<string.h>` and so on) are not to be used.

In hosted mode, the `main` function must adhere to a rigid specification. Execution begins at the function named `main` and its signature must typically match:

```
int main(int argv, char *argv[], char *env[])   // main in hosted env
```

The compiler will issue warnings if you define `main` differently for a hosted program.

Freestanding mode removes the special semantics for the `main` function. In the standalone world, `main` can have any type signature and it is configurable whether it is `main` or some other function that starts the program. A typical main signature for a freestanding program is simply:

```
void main(void)                                 // main in bare metal env
```

The `-ffreestanding` option also directs the compiler to not assume that standard functions have their usual definitions. This will prevent the compiler from making optimizations based on assumptions about the behaviors of the standard libraries. For example, in a hosted environment, `gcc` is assured that the available library meets the specification of the language standard. It can transform `printf("hi\n")` into `puts("hi")` because it *knows* from the definition of the standard IO library that these two functions behave equivalently in this case. In freestanding mode, you could define your own `puts` function and your version of `puts` could act completely differently than the standard `puts` function, making such a substitution invalid. Thus when `-ffreestanding` is used, `gcc` does not assume a standard library environment and will not make such optimizations.

It maybe a bit surprising to learn that even when compiling in freestanding mode, gcc can emit a call to `memcpy` or `memset`. It uses these routines to block-copy a large-ish chunk of data, such as when initializing an array or struct or passing a struct in or out of a function. In some situations, you can rearrange your code to avoid the need for block memory transfer, e.g. assign struct fields individually rather copy the entire struct. Where unavoidable, you must supply your own implementation of `memcpy`.

Linker options for default libraries and start files

The linker option `-nostdlib` is used to link a program intended to run standalone. `-nostdlib` implies the individual options `-nodefaultlibs` and `-nostartfiles`. Below we discuss the two options separately, but the most typical use is just `nostdlib` for one-stop shopping.

When linking a hosted program, standard system libraries such as `libc` are linked by default, giving the program access to all standard functions (`printf`, `strlen` and friends). The linker option `-nodefaultlibs` disables linking with those default libraries; the only libraries linked are exactly those that you explicitly name to the linker using the `-l` flag.

`libgcc.a` is a standard library (linked by default, excluded by `-nodefaultlibs`) that provides internal subroutines to overcome shortcomings of particular machines. For example, the ARM processor does not include a division instruction. The ARM version of `libgcc.a` includes a division function and the compiler emits calls to that function where needed. A program that attempts to use division and is linked `-nodefaultlibs` will fail to link. The linker error will be something akin to

```
arm-none-eabi-ld: main.o: in function `main':
main.c:11: undefined reference to `__aeabi_idiv'
```

You can resolve this reference by linking with `libgcc.a` (`-lgcc`).

Note that `libgcc` does **not** supply `memcpy` and related functions. Buried deep in https://gcc.gnu.org/onlinedocs/gcc/Standards.html, there is a small callout that notes this:

“Most of the compiler support routines used by GCC are present in libgcc, but there are a few exceptions. GCC requires the freestanding environment provide `memcpy`, `memmove`, `memset` and `memcmp`.”

If your program requires one of these routines, you will need to supply it yourself.

Normally, when a program begins to run, the standard start function is called. This function sets up the machine to run the program. A common task performed by start is to initialize default values for any variables in your program and call the `main` function.

The option `-nostartfiles` instructs the linker to not use the standard system startup functions nor link the code containing those functions.

If you don't link to a start function, program variables may not be properly initialized. You may need to provide your own start function when running in standalone mode.