Installing The GCC-MIPS Compiler

Yehowshua Immanuel

January 17, 2020

Background

In the early part of this class, we'll be designing a MIPS processor. To test, our processor, we need it to execute instructions. While it is possible to assemble instructions from opcodes by hand, and type them up into a memory array in Verilog manually, there are already tools available to do this.

What Compilers Do

A compiler will take a source language such as C and convert it into some intermediate language. This intermediate language is usually one step above assembly. The compiler, in another pass will then take the intermediate language and lower it into assembly internally. After this, the compiler will **assemble** or **encode** the instructions into opcodes. Lastly, the compiler will often emit one or more object files.

These different stages are usually referred to as compiler passes. Examples of compilers that behave like this are GNU C Compiler and Clang+LLVM compiler toolchain.

Usually, the passes look like this:

1. Compiler starts by lexing source
2. Parser generates an abstract syntax tree
3. Compiler runs AST optimizations
4. Compiler emits intermediate
5. Compiler runs more optimization passes
6. Compiler generates assembly files with symbols

   You can usually pass flags to a compiler to stop here. For example, to get assembly out of gcc, you can pass the `-S` flag.

   ```
   gcc -S foo.c
   ```

   Also, **symbols** are references to names of functions, or links to locations of data that will be translated into addresses once the linker resolves symbols from object files

7. Translate the assembly to object files. Object files have mostly assembled opcodes, but still have unresolved symbols and possibly addresses. Often times, people will have the compiler stop here using the `-o` flag. For example:

   ```
   gcc foo.c -o foo.o
   gcc bar.c -o bar.o
   ld foo.o bar.o -o foobar
   ```

8. The linker is invoked to link the object files together.

The Linker

The linker uses a linker script that tells it how to place certain parts of the generated executable into memory. For example, the instructions at the start of the program may need to be placed at address 0x4000.

Usually, the linker uses a default linker script that is specific to the operating system.

Also, many processors use interrupts. For an embedded processor, the interrupt routines have to be placed at a specific address in memory. In a general computer processor with an operating system, usually the kernel places interrupt routines at the right spots in memory during boot-up.

For this class

In this class, since we won't be designing a processor that is completely capable of supporting C library routines from `libstdc`, we will be mainly working with assembly files.

For example, printf won't work on our processor - we're only focusing on 15 out of 72 instructions.

Thus, we don't need a full compiler toolchain, we just need the assembler and linker.

GCC Binutils provides a MIPs linker and assembler.

Install Binutils

First make sure that you have install brew

Then do:

```
brew install bracketmaster/rtl/binutils-mips-elf
```

If you can do:

```
mips-elf-as --help

Usage: mips-elf-as [option...] [asmfile...]
Options:
  -a[sub-option...]   turn on listings
                         Sub-options [default hls]:
                         c      omit false conditionals
...
```

Then you're good to go.