
Due Date: Thursday, May 31, 6.00 PM

Introduction

This project is intended to develop your familiarity with ARM assembly language and the ARM development tool suite. You will demonstrate this familiarity by studying the assembly representation of a C function.

Diversion: The Newlib C Library

The standard C library that is included with the CodeSourcery G++ Lite tools is called NewLib. The homepage for this project, which includes download and documentation ("Docs") link, is:

```
http://sourceware.org/newlib
```

You will need to download both the source code for the library and the documentation files. Note that both are also available from CodeSourcery (e.g., see the course webpage for a link to the Newlib documentation).

Library Files

The standard C library is created by compiling several individual source files (mostly written in C) then linking them together using the 'ar' tool (i.e., the arm-none-eabi-ar program in the CodeSourcery G++ Lite distribution). The 'ar' tool (short for ***ARchiver***) creates libraries (archives) with the extension **.a** (again, for archive). The standard C library is called ***libc.a***.

Remember from Laboratory #1 that you can get GCC to tell you where the ***libc.a*** file is for a particular target architecture:

```
CC -print-file-name=libc.a
```

in which CC is defined as in Laboratory #1 for targeting the LM3S6965 device:

```
alias CC='arm-none-eabi-gcc -Wall -O3 -march=armv7-m -mcpu=cortex-m3 -mthumb -mfix-cortex-m3-ldrd'
```

Now copy libc.a file from the path obtained in the above step into temp folder. Then switch to that folder.

```
cd /cygdrive/c/temp  # The temporary directory you made in Lab #1
```

You can list the contents of the library as follows:

```
arm-none-eabi-ar t libc.a
```

This will provide a listing of all the object files (with the `.o` extension) that were placed in the library. Where did each object file come from? By compiling the file with the same name but with the `.c` extension from the Newlib source code.

A library (or archive), then, is created by compiling several related C source files then using the '**ar**' tool to simply package them together into a single file.

Understanding Functions

Object files can be extracted from libraries using the '`x`' command to the '`ar`' program. Try the following:

```
arm-none-eabi-ar x libc.a lib_a-abs.o     # Extract a .o file from the library
```

This will extract the ***lib_a-abs.o*** object file from the library as the new file ***lib_a-abs.o*** in the current directory (the ***lib_a-abs.o*** object file will still remain in the library too).

- Read about the ***abs()*** standard C library function in the Newlib documentation.
- Find the ***abs.c*** file in the source tree of the Newlib source code by clicking on ***newlib ftp directory*** link under ***Downloads*** page. Read the C source code behind this function.
- Finally, let's look at this function's object code from the ***abs.o*** file we just extracted:

```
arm-none-eabi-objdump -S lib_a-abs.o
```

This "object dump" program takes the raw binary codes in the object file and interprets them as ARM instructions, nicely formatted to your screen. How convenient!

You now have 3 views of a C function:

- the function documentation
- the function source code
- the function assembly code

Can you explain how the assembly code implements the C source code for the ***abs()*** function?

Your Project

1. Your assigned C function will be e-mailed to you.

2. Save the C code to a file named ***function.c***. Compile the C function to assembly code using the GCC compiler as follows:

   ```
   CC -DPREFER_SIZE_OVER_SPEED -Os -S function.c
   ```

   That's a capital letter 'O' in the `-Os` option above to indicate "*optimization for minimum size*". Don't forget this else your function will be MUCH longer and harder to analyze. Also, you should compile your function with the '`-O3`' option too, which tells the compiler to optimize for speed instead of size.

Use whichever option leads to smaller code – less work for you! **<u>Be sure to indicate in your report which option you used</u>**.

Some of the Newlib C library functions have alternative implementations that use "tricks" to try to speed up their operation. These tricks GREATLY COMPLICATE the function and make your analysis work much harder. This is why it is important to specify the "`-DPREFER_SIZE_OVER_SPEED`" option to the compiler, so that the simpler implementation is compiled.

3. Read the documentation, the source code, and the assembly code for your function. Understand all three of the above thoroughly.

4. Present your C code and assembly code in your report. Include line numbers on both for easy reference.

5. For each line of assembly code, neatly present a 1 page summary of the instruction with the following items:
   a) The instruction
   b) The state of all relevant registers before and after the instruction executes
   c) The state of all condition codes (N, Z, V, C) after the instruction executes
   d) What the instruction does from a "***big picture***" point of view (i.e., how does it help to implement the C function)

   If your C function translates to 20 assembly language instructions, there should be 20 pages of output i.e., 1 page for each instruction.

   <u>NOTE</u>: Assembler directives (like `.file`, `.thumb`, etc.) do not count as instructions, and neither do assembler labels (like `.L3`).

6. Perform a timing analysis on the assembly code for some "***typical***" input. Do not choose "***end cases***" that are unusual or unlikely, although a thorough analysis will consider such cases. Clearly identify what you are using for input data.

   The timing analysis will report how many CPU cycles the function takes to execute given your choice of input.

   A "***parameterized***" timing analysis is much more useful than a number. For example, if your input has N bytes in it, it would be very nice if your timing analysis revealed an equation that was a function of N.

   How much time does each instruction take? Instruction timings are shown in Section 3.3 of the ARM Cortex-M3 Technical Reference Manual (Revision r2p0, document DDI 0337H).

<u>Deliverables</u>

- Listing of the original C function, with line numbers.
- Assembly-code listing of the original C function as generated by the compiler, with line numbers
- Line-by-line analysis of the assembly code, as described above, 1 page per instruction, neatly presented and organized.

- Cycle timing analysis of the original C function, as described above. If your function takes a variable amount of time to execute depending upon the input, an analysis of function behavior for various input cases is expected.
- "***Big picture***" comments on how the assembly code relates to the C code: anything that is unexpected, anything you could have done better than the compiler, etc.

You must demonstrate a thorough understanding of function behavior, assembly language, the connection between C code and assembly language, and an understanding of assembly language cycle timings. These will be graded subjectively so you will need to put in more than a minimal effort in your textual explanations.

Submit all of the above electronically to your laboratory instructor.

Group / Individual Work

Undergraduate students are required to work in groups of 2 on this.

Graduate students are required to work on this assignment by themselves.

Grading

Your laboratory instructor will discuss grading with you in the laboratory.