Arm simulator 2

By: Daniel Recker

Bob Jones University

Cps 310

# Table of Contents

[Table of Contents 1](#_Toc404547887)

[Introduction 3](#_Toc404547888)

[Features 3](#_Toc404547889)

[C-Level: 3](#_Toc404547890)

[Extra Credit: 3](#_Toc404547891)

[Soft-Ware Prerequisites 4](#_Toc404547892)

[Build and Test 4](#_Toc404547893)

[Build 4](#_Toc404547894)

[Test 5](#_Toc404547895)

[Configuration 5](#_Toc404547896)

[Log 5](#_Toc404547897)

[Trace 5](#_Toc404547898)

[User Guide 5](#_Toc404547899)

[Command Line Options 5](#_Toc404547900)

[Load 6](#_Toc404547901)

[Execute 6](#_Toc404547902)

[Test 6](#_Toc404547903)

[Debug 6](#_Toc404547904)

[Memory Resize 6](#_Toc404547905)

[Create Executables 7](#_Toc404547906)

[Common Combinations 7](#_Toc404547907)

[Soft-Ware Architecture 8](#_Toc404547908)

[UML 8](#_Toc404547909)

[Discussion 9](#_Toc404547910)

[Bug Report 10](#_Toc404547911)

[Test File Info 10](#_Toc404547912)

[Grade Level Omissions 10](#_Toc404547913)

[Appendices 11](#_Toc404547914)

[Journal 11](#_Toc404547915)

[Git Log 14](#_Toc404547916)

[Source Code 15](file:///C:\Users\Daniel\OneDrive\Junior%201st\Micro\sim2Report.docx#_Toc404547917)

# Introduction

This report documents the simulated ARM processor with GDB connection program and its features. This report contains the instructions to use the program, and the types of programs it can run. Also contained in this report is the source code, and description of the structure behind the code. The ARM computer will be able to load, and execute ELF file executables in a simulated environment with fifteen registers and RAM. The program can be attached to a GDB client to inspect values as they happen and to step through the program. With this program a user can run ARM machine code.

# Features

A list of special features sectioned by grade level.

## C-Level:

The arm simulator loads and executes ELF file type ARM machine code.

A list of usable arm commands are:

MOV, MVN, ADD, SUB, RSB, MUL, AND, ORR, EOR, BIC, LDR, STR, STM, LDM, B, BX, BL, CMP

Many addressing modes are supported. Including:

Immediate offset, Register with Immediate Shift offset, Post-Index offset, Pre-Index offset, and offset addressing

The simulator can take input for programs like mersenne.exe, and outputs text and answers to the screen.

## Extra Credit:

The program simulator is able to respond to commands from a GDB client for debugging purposes. The GDB client can control the ARM Simulator when it is started with the --debug flag.

The GDB can load files, step the program, continue the program to the end, insert breakpoints in the program, toggle the trace log, and view register values. To load a file into ARM use the load command in GDB. After the file has been loaded you can then step, run, and inspect the executable file. To step one command at a time use the GDB “stepi” command. To run the entire program use the “continue” command. To insert a breakpoint it is recommended to follow this suggested order:

1. B \* 0xaddress
2. Target remote :8080
3. Load file
4. Continue

This is the tested and working way to insert breakpoints at memory address locations in GDB.

The GDB client takes the place of a GUI.

\*WARNING The GDB is not able to interact with the program regarding I/O. The user must input the text via the command line terminal the program is running in.

It also has the Register shifted by Register offset addressing mode functionality that is demonstrated in sieve.exe.

# Soft-Ware Prerequisites

The program was created and run on a Windows 8.1 using visual Studio 2013. The submitted executable was compiled using the windows csc.exe program located in the “C:\Windows\Microsoft.NET\Framework\v4.0.30319” folder. The tested and verified OS is Windows 8.1.

# Build and Test

**Instructions on building and testing the ARM simulator from source code.**

## **Build**

**The project can be built and tested as follows. To build the project first go to the “\src” folder. Run the terminal command:**

C:\>C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe \*.cs

## Test

This will create an executable named Program.exe. In order for tests to be run the files “test1.exe”, “test2.exe”, and “test3.exe” must be located into the same working directory. Copies of those files are located in the “\install” folder.

Once the test files have been added to the working directory run the program:

C:\>Program.exe --test

# Configuration

Information about log files and trace information.

## Log

The log.txt file is permanently set as the output file for user readable information. It contains detailed information about the running program.

## Trace

The trace.log file contains information about the registers, address location, step count, and flags. To turn tracing off the ARM simulator must be connected to the GDB client. The GDB command to turn off logging is:

GDB>monitor t

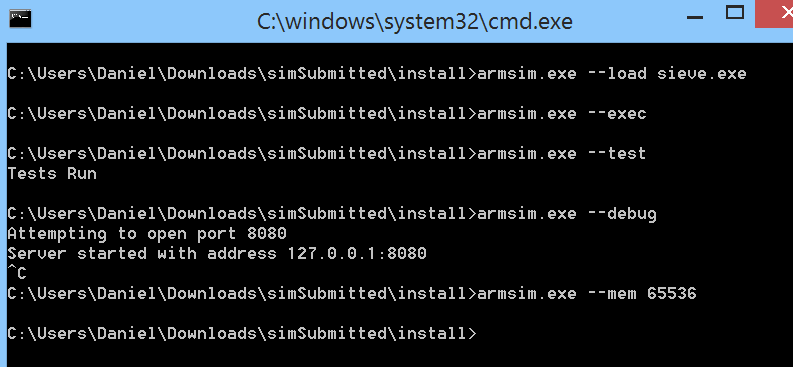
This will toggle the trace off. The same command is used to toggle the trace on. If the trace is toggled on, then the trace.log file will be reset.

# User Guide

A guide on how to use and operate the Arm Simulator.

## Command Line Options

The Arm simulator has a number of command line options.



### Load

The --load command will load a file into the simulated RAM. To load a file type --load *FileName*. This command will only load the file and will then quit.

### Execute

The --exec command will execute a loaded file. If no file has been loaded this command has no effect.

### Test

The --test command will run the Simulator’s comprehensive Unit Test and display the text “Tests Run”. **WARNING**: The three required test files must be present. See Build and Test Section.

### Debug

The --debug command will have the Simulator begin listening on Port :8080 for a connection from a GDB client. A GDB client can then connect to the Simulator by using the command

target remote :8080

The Simulator supports commands from GDB. These commands are discussed in Features under Extra Credit.

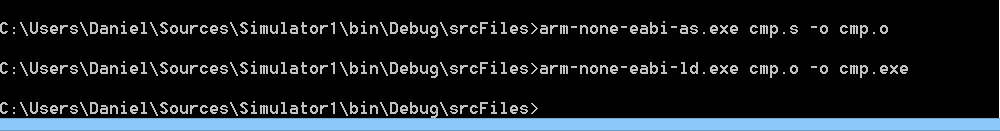
### Memory Resize

The --mem command allows you to change the size of simulated RAM.

--mem *sizeOfMemory*

Some files may be too large to load into the default size of memory and this command will need to be used.

## Create Executables



To create executables for the Arm Simulator it is recommended to install the Sourcery Tool chain from <http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/evaluations/arm-eabi/>

Once installed this will give you two important programs:

arm-none-eabi-as.exe

arm-none-eabi-ld.exe

When you create your Arm Program be sure to include an \_start: tag at the beginning of the program.

After you create an ARM program, run the two programs as so:

C:/>arm-non-eabi-as.exe *myProg.s* -o *myProg.*o

C:/>arm-none-eabi-ld.exe *myProg.*o-o *myProg.*exe

That file should then be able to be loaded into the Arm Simulator.

**WARNING**: When this was tested the generated file was too large for the default RAM size so it is recommended to use the --mem option with a larger RAM size.

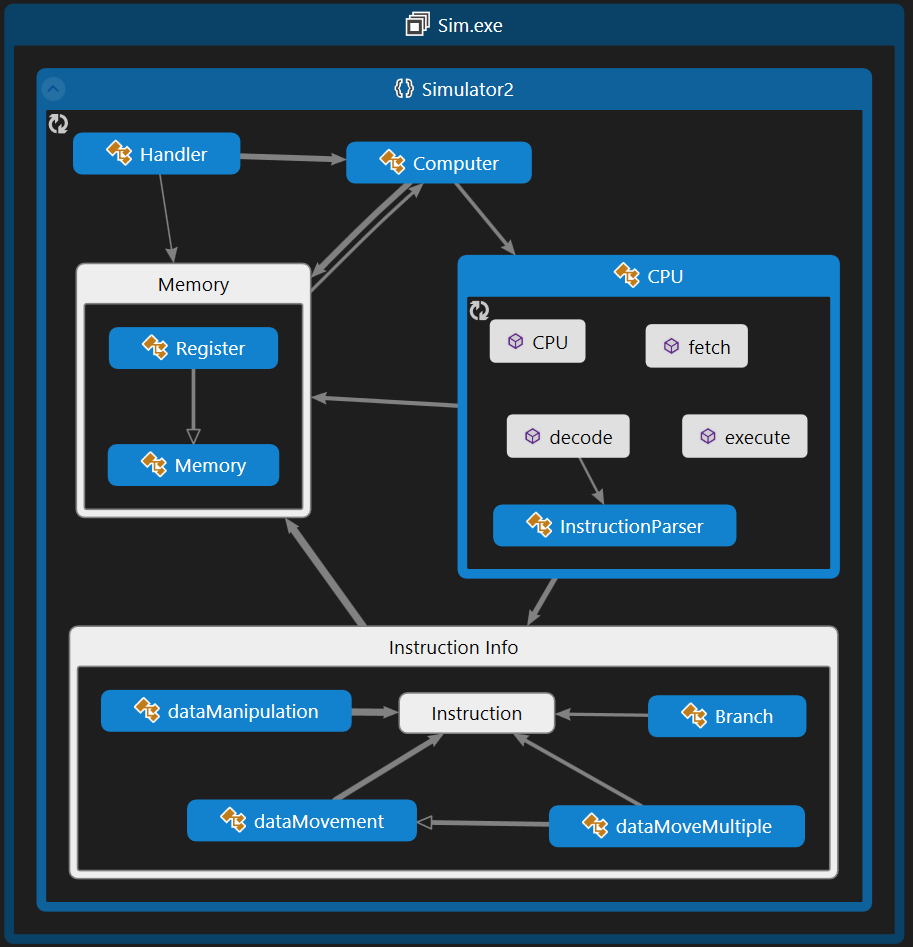
## Common Combinations

* The --load option and --exec option are commonly used together.
* The GDB client can load files to the Simulator, but a file can be preloaded with the --load option.

# Soft-Ware Architecture

Overview of the organization of the project.

## UML

Visualization of the main classes in the project and how they interact with each other.

## Discussion

The code is divided into multiple sections. The computer class is the base of the program. It holds instances of the Registers and RAM.

The Computer class is a singleton class that reads the executable file and loads it into the simulated RAM. The Computer also holds an instance of the CPU. When the Computer steps it runs the go function once. When the Computer is run or continued the go function is run on a separate thread.

During go the Computer class does two things. First it runs a separate thread that listens for IO at the input and output addresses. Second it runs the CPU’s fetch, decode, and execute functions. When the program reaches the end the RAM, Registers, and Condition Flags are reset to the state before the program was run.

The CPU holds references to the Computer’s instances of RAM and Registers. This class runs the fetch, decode, and execute logic. It fetches the instruction from RAM according to Register 15, decodes the instruction, and executes the instruction. The instruction parser decodes the instruction setting the instruction to the proper type of instruction. The instruction is then executed.

The Instruction Class is the parent class of the instruction types. After the instruction has been parsed it is one of the subset types. Each instruction type has its own self-parsing methods to set appropriate variables, and self-execution methods that update the Registers and RAM.

The handler is started if the --debug option is specified. This Class listens to port :8080 for a connection from a GDB client. Once a connection is made the Class receives and parses commands from the client.

# Bug Report

This section contains the results from tests and known bugs.

## Test File Info

|  |  |  |
| --- | --- | --- |
| File Name | Result | Discussion |
| ctest.exe | Pass |  |
| btest.exe | Pass |  |
| ldmstm.exe | Pass |  |
| pcaddr.exe | Pass |  |
| branch.exe | Pass |  |
| cmp.exe | Pass |  |
| sieve.exe | Pass |  |
| pointers.exe | Pass |  |
| locals.exe | Pass |  |
| quicksort.exe | Expected Result | Too large for a diff checker to check. |
| mersenne\_no\_input.exe | Expected Result | Too large for a diff checker to check. Takes a long time to run. |
| mersenne.exe | Expected Result | Too large for a diff checker to check. Takes a long time to run. |

## Grade Level Omissions

* + Multithreading with use of GDB to stop a program during execution is not implemented.

# Appendices

The journal, Git Log, and source code for the project.

## Journal

Total Hours 59

|  |  |  |  |
| --- | --- | --- | --- |
| Sim 2 Dates | Info | Total Time | 10.2 |
| Sat Nov 15 | Working with Ben, and Curtis to get an early submission in. Bouncing ideas off of eachother and figured out what needed to be done. Project is finished. Just some code clean up! |  | 5 |
| Thu Nov 13 | Quick fix to the monitor command |  | 0.2 |
| Sat Nov 8 | Fixed some of my compare logic. My C flag logic was wrong. I understand it now. Also had issues in my checkCond function dyslexia mixed up the Z and V flags. Renamed V to F. |  | 3 |
| Thu Nov 6 | First Try at implementing Bx. Some trouble with the PC register being 8 off. |  | 2 |
|  |  |  |  |
| Sim 1 Dates | Info | Total Time | 15 |
| Wed Oct 15 2014 -0400 | Pretty sure I finished a B version… not that bad. Don't know what everyone else is complaining about. Think I'll chill for a few days now. I have what? 15 days? |  | 2 |
| Mon Oct 13 2014 -0400 | refactored my code to more object oriented. Instructions now have a run(ref reg[], ref RAM) command that they inherit from the overall instruction class.Inheritance so that each command will call its own run method |  | 1 |
| Sun Oct 12 2014 -0400 | Wrote so many unit tests…. Every thing is ship shape |  | 2 |
| Sat Oct 11 2014 | Lots accomplished!!! Got sub, bic and all the other opcody commands working. Mostly a copy and paste job. Also got st ldr and others going. A little tricky with all of the bits. But ldr and str are in the same function just check a flag. Hahaha. Branching also works which is cool. Load and store multiple put the registers lower reg # in lower address space all of the time. This is how the documentation made it sound it worked so it's what I attempted to do. |  | 4 |
| Sat Oct 11 2014 -0400 | Mov command now can be shifted by a register that was shifted by another register. So that logic is in place for all of the other commands!!1 |  | 2 |
| Fri Oct 9 2014 -0400 | More testing. Mov command now works. Stubbed out the ability to process all of the other commands. The loops and if statements are there just need to be implemented |  | 3 |
| Wed Oct 8 2014 -0400 | unit tested FDE logic. |  | 1 |
|  |  |  |  |
| GDB Start Dates | Info | Total Time | 28.8 |
| Tue Oct 7 2014 | Buffed out test cases even more to match the fact of our singleton classes. |  | 1 |
| Sun Oct 5 2014 | Hammering out a few kinks before turning in on Wednesday |  | 2 |
| Sat Oct 4 2014 | Met with ben at star bucks and busted out code for 5 hours!!! Fixed our IPAddress issue because mine only does ipv6. Fully linked the GBD front end to the computer back end. Implemented breakpoints. |  | 5 |
|  | program now clears everything when it finishes running. |  |  |
|  | changed statval to a string to make all of our lives easier |  |  |
|  | Added status struct |  |  |
|  | get thread Status |  |  |
|  | step is no longer multi threaded since it does not need to be stopped. |  |  |
|  | Work on step command |  |  |
|  | log has time stamps |  |  |
|  | Memory fix / beginning of write register |  |  |
|  | write Reg takes a byte array |  |  |
|  | IPAddress resolution |  |  |
| Fri Oct 3 2014 | Cleaneed up some console writes. loading a file is no longer required. cleaned up trace opening and closing logic. Made sure the trace is always closed during testing. That was giving some issues. |  | 2 |
|  | Changed my test flag logic to be inteligent and not written by a three year old… |  |  |
| Tue Sep 30 2014 | Continued work with reading the memory |  | 3 |
|  | fixed the writeRAM loop |  |  |
|  | fixed private bug |  |  |
|  | fixed logging error. |  |  |
|  | Back end breakpoint logic |  |  |
|  | Commit before removal of code. This commit is a fail safe in case something goes absolutely horribly wrong and we have to rebuild |  |  |
|  | added comments of which functions to call for which gdb requests |  |  |
| Sun Sep 28 2014 | hooked up the computer to the handler, to respond to tests. loads up the ELF file first so it has something to play with. created a byteArrayToString method that takes a byte array and the size of it and spits it out as string. |  | 0.5 |
| Sat Sep 27 2014 | dump methods now return byte[] instead of 2D arrays. Trace works now. |  | 1 |
| Fri Sep 26 10:14:17 2014 | Continued work with outlining framework for commands. Went through GDB documentation and created stubs for commands I thought were needed. |  | 0.3 |
| Wed Sep 24 2014 | Always check the status of the trace file before writing to it. I also decided that a logger class that is accesible to all members would be nice. |  | 3 |
| Tue Sep 23 2014 | Created more computer functionality for preparation for GDB stub. Dump registers, get at ram address etc. |  | 1 |
| Fri Sep 19 2014 | Set up the FDE cycle to grab commands and pass them along the work flow. CPU is set up to reference the RAM and registers. Fleshed out test cases. |  | 3 |
| Thu Sep 18 2014 | Got a basic command line interface to work with. Incorporated the ability to view the RAM. Worked on logging a lot. Some issues with the trace.log being open and trying to open it again… |  | 3 |
| Wed Sep 17 2014 | Stubbed out a register class that inherits from Memory. Stubbed out basic classes. Injected loader logic into the computer class. |  | 3 |
| Tue Sep 16 2014 | First commit with Ben. |  | 1 |

## Git Log

Sat Nov 15 12:33:30 2014 -0500 Submitted. at12:20 PM on nov:15:14 in the year of our Lord And saviour Jesus Christ. Done.

Sat Nov 15 11:58:21 2014 -0500 finished IO C version is done. Next is code cleanup

Thu Nov 13 12:26:05 2014 -0500 made the monitor t command only toggle trace instead of any monitor command

Sat Nov 8 23:15:37 2014 -0500 got the branch.log working

Sat Nov 8 22:51:03 2014 -0500 got the cmp test working.

Thu Nov 6 20:59:33 2014 -0500 implemented bx, made condition codes print, 2 hours

Fri Oct 31 08:15:35 2014 -0400 submitted sim1

Fri Oct 31 08:15:14 2014 -0400 bug fixes

Thu Oct 30 16:54:49 2014 -0400 Getting ready for submission.

Wed Oct 29 10:47:29 2014 -0400 quits if file not found

Mon Oct 20 10:48:26 2014 -0400 multiply test

Sun Oct 19 22:00:52 2014 -0400 here we go

Sun Oct 19 21:57:07 2014 -0400 more tests

Wed Oct 15 19:15:38 2014 -0400 here

Wed Oct 15 19:08:57 2014 -0400 push from vs

Wed Oct 15 19:06:17 2014 -0400 first solo commit



## Source Code

