Low Power Server

University of Rochester ECE Senior Design

Simulator Report

Sakhile Mathunjwa

Contents

1 Overview of the Simulator

Fusion Core is a processor architecture that is still under development. The aim of the project is to optimize the final product for power consumption, from the hardware down to the instruction set architecture. Even for a grand project like this one, it is of utmost importance that the ISA does what it ought to do correctly. That is exactly where the Fusion-Reactor Simulator comes in. The simulator is a software program that has been designed to mimic the Fusion Core processor architecture instruction execution. The simulator is meant to aid the architecture designers validate their design. Developers may also find the simulator useful for debugging purposes.

1.1 Use Case

The downloaded file will contain makefiles and scripts that will be used for building the source code. Once compiled, an executable will be generated and will reside in the same folder as the the source files. To run the simulator, the user will have to have in the same folder or another, a program written in fusion-core assembly code. The executable being simulated should have been built using the the fusion-elf binutils toolchain. Refer to sections below for further instructions

1.2 Design Choices and Functionality

On top of the back end that implements the actual simulator, the designers chose to add a simple user interface to make following a running program as easy for users. The simulator has the following capabilities:

- Displays all essential information about the processor's current state. This information is needed for debugging purposes.
- Has different modes, the most important being the ability to allow the user to step through a program, one instruction at a time.
- Currently handles a few exceptions. More exception handling capabilities will be added later.

2 How to use the Simulator

This sections provides directions on how to install the simulator on your own system and how to run it.

2.1 How to Compilation

The user will need the follow the steps below:

- To familiarise yourself with the fusion-core instruction set, you will need to clone this (https://github.com/bit0fun/Fusion-Core.git) git repository. It contains documents that provide a full list of the architecture's instruction set and other specifications.
- To download the fusion-elf tool-chain, clone this (https://github.com/bit0fun/fusion-core-binutils.git) git repository and run bulid_script.sh. You may need to install any libraries you may be missing that this program depends on.
- Finally clone this(https://github.com/bit0fun/fusion-reactor.git) repository. Run make in the the terminal inside the fusion_reactor directory.

2.2 Running the Simulator

To run your program, run the following command in the directory that contains the compiled program:

```
./fusion-react <elf filename>
```

The program will display a blank screen. The user can choose from any of the following options going forward:

1. Press r: The program will run to completion without any further input from the user
2. Press s: The program will run one instruction per key press.
3. Press q: This option will cleanly end the program.

Regardless of the option the user chooses above, the last option has to be eventually chosen to end the program.

2.3 Troubleshooting

If the correct libraries are not installed, the compilation error should be used as a guide to fix the problem. Feel free to consult online resources to figure out what these errors mean.
Even when the ncurses library installed, the user may come across the error:

```
libncursesw.so.6: cannot open shared object file: No such file or directory
```

To solve this problem on an linux machine, run the following command

```
ln -s libncursesw.so.5 /lib/x86_64-linux-gnu/libncursesw.so.6
```

2.4 Known Bugs

3 Simulator Internals Documentation

3.1 ELF Handling

For the purpose of extracting both machine code, and data from the the elf file, the developers used a C library. The library provides an API that parses the elf file and places the information in the relevent user provided arrays. On top of that it provides a structure elements that hold important information like the address of the beginning of the text segment, its size etc which are useful for tracking the progress of the program.

3.2 CPU Simulation

To simulate the CPU, the developers used macros to mimic the control unit, and functions for each instructions.

3.3 Memory Simulation

A combination of arrays and multidimensional pointers were used to hold various kinds of information. For memory elements that have predefined sizes like registers, arrays were the obvious choice. On the other hand, data that depends on user inputs is held in pointers whose sizes can be varied according to need. This eliminates the inefficient alternative of hoarding memory that does not actually get used.
The text document was store as a double pointer to make it easy to access different lines of text from the disassembled file. This was important for the purpose of highlighting the line on the code text currently being executed as the program runs. Also important to mention is the choice of using the disassembly file over the assembly language text file. The disassembly file has the advantage that it juxtaposes human readable instructions with their actual address in the instruction memory, which makes it easy for the user to follow the program.

3.4 User Interface

The program has been written in the C language. The nature of the language lends itself well to the task. As C is a low level language, bit manipulation was made very easy. It also gave the developers the ability to use memory sparingly through the use of the dynamic memory allocation tools it provides. The extensive use of pointers also meant that C was going to be the developers language of choice. Lastly, C provided the developers with very easy to use functions to represent numbers in different number systems.
The designers opted to use the ncurses library for creating the user interface for the reasons that:

1. ncurses user interfaces are commonly written in C. There were therefore a lot of resources out there available to the developers to learn from and fewer issues to run into.
2. It is a very simple user interface that displays on the terminal. The developers could therefore keep the design as simple as possible but also presentable.