Project Report

Alejandro Carrillo

Gerardo Hechavarria

**Purpose**

The purpose of this research project is to convert make a disassembler that reads from a binary file, and outputs the corresponding assembly instructions.

A disassembler performs human readability, making it suitable to use it as a reverse- engineering tool, which can reveal the design or architecture of a program, meaning it can help for a better program comprehension.

This program is of great importance to us, since we consider it is going to gives us a better idea of how computer understand program using the machine language. Also, the ability to manually decode the binary code given by any program based on the MIPS architecture.

This project will allow to gain knowledge and experience with MIPS architecture, and how assembly programs are generated and interpreted. As an outcome, our team will gain experience with MIPS assembly language

**Mission**

Our team seeks to understand how assembly code get translated to machine or binary code, focusing only in the RISC (reduced instruction set computer) architecture, and in one particular assembly language which for our purpose we are going to lean on MIPSx32. Thus, mainly we will need to study how MIPSx32 assembly language works, how it does its coding and decoding, and how these are implemented. After we deeply read and understand how MIPS assembly language generate its binary code, we will then have the task of writing a program that takes that binary generated code by the assembler as an input, and with it generates the corresponding MIPS assembler code.

**Building the program**

To start building this program we first looked at MIPS assembly language, and how it generates its binary code. Some of the tools we used to for our better understanding of this assembly language was the MIPS simulator Mars. We found out that Mars offered us some very helpful tools like a Memory Reference Visualization, BHT Simulator, Floating Point representation, and Instruction Statistic tools. Nevertheless, the most helpful of all the Mars’s tools was its execute windows, which gives you detailed information of each instruction being assembled. It gives you location in memory of the instruction, and the corresponding hexadecimal or binary, to the user choice, of each instruction. Which was very helpful to practice our MIPS encoding and decoding skills, and to verify if we were going in the right track and if our program was working properly.

We also leaned a lot on MIPS green-sheet, which gives you a general but very helpful information of the most important and used instructions. From MIPS green-sheet we were able to understand the different types of MIPS instructions [R, I, J]. How each of them are implemented, the number of fields in the instruction’s 32 bits block each of the type have, the number of bits that each of their fields occupy, and the type of binary that each field holds, sing or unsing.

In other way that we used the MIPS simulator MARS was to generate our binary files. To do so once you have the assembly code that you want to get the binary file from opened on the simulator, one the clicks on *File >> Dump* *Memory To File,* and under Dump Format select *Binary Text.* Our program then uses this generated binary file as input, and with it generates back the assembly instructions.

Our program cover most of the most used and important instructions. Instructions that couldn’t be covered were the FR and FI instructions. Instructions that were not covered our program print them as unimplemented.

**Running the program**

To run this program, it is necessary to add the command *-std=c++11* in compiler configuration of the ide. This is necessary for the program to run so that it can access that library. After the program is ran it will ask the user for name of the binary file that the user want to disassemble; note that such binary file needs to be in the same folder as the program executable. After the user has input the name of the binary file, the program should then generate the corresponding MIPS instruction for such binaries.

**Conclusion**

Although there was maybe more sophisticated ways to go about this project, like we tried to generate an object file from a c++ program, which would contain the encrypted binaries of such programs, but even though, we tried to find the way to do such thing, none of the ones we tried really worked as we wanted to. The other option was to use a toll chain cross compilation, but we also felt that with all other commitments we had, the most simple and efficient way was to use the Dump Memory tool that the MIPS simulator Mars offered us. We learned a lot about the relationship between the compiler, assembler and linker, and of course MIPS assembly language. We now understand much better, how high-level programs got translated through a compiler, then an assembler to generate the machine code, giving us a better perception, what good and efficient programming is about. In the process of working in this project we sure encountered a lot of problems and discovered how useful a strong open source community is. In the end we are some how proud of being able to get our goal accomplished and our program running without errors. Although we know that with a little more time and availability, we could have done a much complete and better work.