University of Colorado
Colorado Springs

CS 4200, Spring 2022: Computer Architecture

Instructor: Mong Sim

Project 1

Keaton Raymond

2022/02/13

Issues

The first issue that I encountered was figuring out what exactly is the function of a disassembler. I was able to figure this out through doing research and reading about disassemblers where I learned that a disassembler essentially converts binary machine level instructions to human readable assembly code. However, knowing what a disassembler does is the very first step and doesn't help me too much when it comes to actually making and implementing one. This was by far the most difficult issue to solve; for this issue, I talked to a couple classmates, did some research online, and talked to the professor before finally deciding to make my own solution which will be further discussed in the section "How It Works". The next issue that I ran into was with the virtual machine on my MacBook. For one reason or another, despite reinstalling Visual Studio several times, I would always get an error that I was missing several dll files anytime I would try and run the program. This issue took two or three days of troubleshooting before I came up with the idea to just use an old windows laptop, and luckily that worked. For now, the core issue remains to be solved, but at least I was able to find a way around it. The final issue that I ran into was with my output. The only way that I was able to get the soft processor to enter the disassembler was to have the argument in the file path for Dhrystone be a one rather than a zero, which leaves me with an infinite loop of three or four instructions. After a few hours of inspection, I was unable to find a solution to this problem and I believe it lies in my implementation method. However, I believe that when my code is examined it shows my understanding and knowledge of the functionality of a disassembler and it outputs the same thing. Given more time, I would have liked to learn how to properly write and implement a disassembler so that it worked perfectly and this output issue was nonexistent; however, given the time restraints with this project as well as other responsibilities, this was not possible.

How It Works

The way that I decided to write a disassembler was not using the actual bits of the instruction but using a binary string version of the instruction along with a custom parse. First, the instruction is converted to binary and placed into an int array, which is then converted to a string. Through the process of converting the instruction to binary, the instruction is placed into the int array backwards, so it is then flipped so that it is in the proper order. Now, most of the instructions are less than 32 bits long so the instruction is then extended to be 32 bits. Now that there is a workable binary string, the opcode is parsed out and used to determine the instruction type. Then, for each of the types, the corresponding registers and function segments are parsed out. The registers are then converted to their ABI format, and finally using

func3, funct7, and for some of them the immediate register bits, the specific operation is determined, and a line of assembly code is printed out using the format:

Binary instruction operation rs1, rs2, rd

The last 3 registers are determined by the registers used for that operation, whether they are the rs1 and rs2 registers, imm registers, or the PC.

While I do understand that this implementation does not demonstrate a knowledge of bit manipulation or memory management with C, it does demonstrate a knowledge of how a disassembler works, though the implementation itself may be crude.

Flow Chart