EE-357 Final Project Report

I worked on a team with Douglass Chen, Matthew Pohlmann, and Angelica Tran to create the EE-357 final project, a virtual processor written in assembly which takes in a set of commands in custom machine code. Some details about how to use our processor are as follows:

* The processor works with long-word sized instructions, that is, 32 bits. At the beginning of the program, space for the instructions is allocated. In the test case specified in the assignment, 11 instructions are supplied so our program allocates 11 long-words at the beginning. The program is also set to always run these 11 given instructions. In order to change the instructions for our virtual processor to run, all the user has to do is copy and paste the line “move.l #%00000000000000000000000000000000, (a0)+”, replacing the zeroes with the appropriate machine-code instruction, in the format specified in the project assignment.
* Each of the eight virtual registers is of size 32 bits, and it is accessible using a three-bit register address. Register 0 is reset to zero before the execution of each virtual-processor instruction.

In order to process the instructions, we used a separate branch statement for each one. First we logically shift the whole instruction so that the first six bits are the only ones that remain the same, and then we decode these six bits to see which instruction to branch to. Once we branch to the appropriate instruction, the code at that branch statement will execute the instruction then return back to the end of the main processing loop, which then increments the program counter and loops back to the beginning of the main loop. If the instruction has to do with setting the program counter, then we do not want to increment the program counter at the end of the main loop, and there is another branch statement at the end of the main loop that the instruction can branch to which does not increment the program counter.

We have several subroutines which make it easier to execute each instruction, since multiple instructions often do common things as part of their execution. The GET\_REG\_D2 subroutine is an easy way to get the value in one of the virtual registers. The SAVE\_D2\_TO\_REG\_D3 subroutine is the opposite; it is an easy way to save a value into one of the virtual registers. We also have a WAIT subroutine, which can be called whenever we want the code to run slow enough for humans to perceive its progression. The main loop’s call to this subroutine is commented out by default because it makes the program take a long time to execute, but it can be easily uncommented if the user wants to be able to follow the commands being performed by the virtual processor.

Our most puzzling difficulty had to do with the bit sizes and signs of the immediate values in the virtual processor instructions. The problem was that the immediate values supplied were always 20 bits, whereas the virtual registers (and the real registers) are 32 bits. The 20-bit immediate value could not just be simply passed into a 32-bit virtual or real register because if the immediate value is negative, it will not be sign-extended in the 32-bit register. So, to fix this, we ended up logical-shifting the 20-bit number to the left 12 bits so that the most significant bit of the 20-bit number was in the most significant bit of the 32-bit register, and then we arithmetic-shifted the number to the right 12 bits, which preserves the sign. This results in all the bits from the 21st to 32nd bit being filled with whatever the original 20th bit was—that is, sign extending.

The parts that I contributed to the project include the underlying infrastructure, the ADD command, the LOAD command, and the GET\_REG\_D2 and SAVE\_D2\_TO\_REG\_D3 subroutines. I also participated in helping my teammates work on their contributions.

This was overall a very interesting project because it was a tough logical problem which had to be solved using the relatively barebones commands in ColdFire assembly. It would have been really easy to complete the project with a high-level language like C++, but using assembly allowed me to learn a new language and make a robust and fast project.