Tony Ngo – Project A2 Task 4

A close up of text on a black background

Description automatically generated Picture 1: Compiling, assembly, execution, and creating the debugger of “second” program

A close up of text on a black background

Description automatically generated Picture 2: Debugging of “second” program

The “second” program is formatted very similar like the “first” program, except for the fact that we are storing memory values instead of loading values into just registers. To write the program itself, I used the nano text editor within the terminal. To assemble the program, I used “***as -o second.o second.s***”, which creates an objective file named “second.o”. To link the program, I used “***ld -o second second.o***”, which creates an executable called “second”, however you do not see output when “***./second***” is ran because all the values are loaded into the registers, which we will need to use the GDB debugger to see the values. To assemble a program that can be used by GDB, you will need to add the “-g” flag, which makes the assembly of the program now “***as -g -o second.o second.s***”. The linking will be the same, but the executable can now be ran into GDB as “***gdb second***”. In the debugger, I first typed “list” to show all of the code that I wrote, then I added a flag at 15, by entering “***b 15***”. To look into the actual memory at the breakpoint, I typed “***x/3xw 0x1008c***”, which is the memory value where my breakpoint is located. “***x/3xw***” indicates that it is going to display three words in hexadecimal, “***0x1008c***” is the location where my breakpoint is located. These values served as a reference so we knew where our code was stored to.

While writing the program, I did not run into any complications in terms of errors since I did most of my syntax-based errors during Project 1. However, I forgot to do “#9” instead of “9” because of the syntax in x86 vs ARM and it gave me compiler errors.

A close up of text on a black background

Description automatically generated Picture 1: Compiling, assembly, and execution of Arithmetic2

\*insert pic of debugging\*

A close up of text on a black background

Description automatically generatedPicture 2: Debugging of Arithmetic2

In this program, I stored values into “val1”, “val2”, and “val3” and accessed them by loading them into registers r1, r2, and r3 and computed the values using the registers and stored the final value into r4. I wrote the program using nano in my terminal. To assemble this program, I used the command “***as -o arithmetic2.o arithmetic2.s***”, which creates an objective file called “arithmetic2.o”. I linked my program using the command “***ld -o arithmetic2 arithmetic2.o***”, which creates an executable called “arithmetic2”, but to be able to debug it using the GDB debugger, we need to reassemble using “***as -g -o arithmetic2.o arithmetic2.s***”, and we have to relink it using “***ld -o arithmetic2 arithmetic2.o***”. In the debugger, I used “list” to show all the code that I wrote. To look into the actual memory at the breakpoint, I typed “***x/3xw 0x1008c***”, which is the memory value where my breakpoint is located. “***x/3xw***” indicates that it is going to display three words in hexadecimal, “***0x1008c***” is the location where my breakpoint is located. These values served as a reference so we knew where our code was stored to. I added a flag at 26 using “b 26”, this was to check the registers after my code executed to see if I did my arithmetic right, which I did because r4 has 30 loaded into it and the arithmetic equation would be “11 + 9 + 16 – 6”, which is equal to 30.

While writing the program, I did not run into any complications in terms of errors since I did most of my syntax-based errors during Project 1.