CSC3210 Project 4 Task 4 – Tony Ngo

Part 1

A screenshot of a cell phone

Description automatically generated

This is the code for the “fourth” program, it was written in the Nano GNU.

A close up of text on a black background

Description automatically generated

When I originally compiled it using “as -g -o fourth.o fourth.s”, I came into an error on line 20 where the immediate value 1 did not have a # behind it. So I put that in and was able to compile, then I linked the code using “ld -o fourth fourth.o”. I then used “gdb fourth” to debug the code.

A screenshot of text

Description automatically generated

This is the beginning of the gdb debugger which I opened using “gdb fourth”. I used “list” to show all of the code written, then used “b 7” to set a breakpoint at line 7, and finally ran the program using “run”.

A screenshot of a computer

Description automatically generated

I used “stepi” to keep stepping through the code and used “info registers” to check at each step if the registers changed, and I got to line 17 and was able to find that the register for the zero flag was up, and that’s because in the line before we set the zero flag, and afterward jumps to line 20 because it was true that z==1.

Part 2

A screenshot of a cell phone

Description automatically generatedThis is the updated fourth.s from Part 2, I used the Nano GNU to write the program. In the project instructions it tells us to add “bnq”, but the function is actually “bne”.

A screenshot of a cell phone

Description automatically generated

I compiled the code using “as -g -o fourth.o fourth.s”, I then linked the code using “ld -o fourth fourth.o”, and showed the gdb debugger was working after I used the function “gdb fourth” and the debugger opened.

A screenshot of a cell phone

Description automatically generated

I used “list” to show my code, and then set “b 7” to be my breakpoint and used “run” to start the execution.

A screenshot of a cell phone

Description automatically generated

I used “stepi” to step through the code and “info registers” to check the value of the registers at each step increase. In “cpsr”, the zero flag was still up, and the code just went to “thatpart” because it did not need to jump since b was equal.

Part 3

A screenshot of a cell phone

Description automatically generated

This is my code for the ControlStructure1 program; I used the nano GNU to write the code. I used bgt instead of ble because when creating code in assembly, you have to have it move in logical order, and if you use “ble” the code will not be in sequential order, which creates inefficiency in the code.

A close up of a logo

Description automatically generated

I used as -g -o ControlStructure1.o ControlStructure1.s to assemble the program, and then used “ld -o ControlStructure1 ControlStructure1.o” to link the program, and then used “gdb ControlStructure1” to open up the GDB debugger and confirmed that it was working.

A close up of text on a black background

Description automatically generated

I used “list” to show my code, then I used “b 7” to create a breakpoint at line 7, then I ran the code using “run”.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

I ran my code and this was the result. It is correct because I set my value of “x” as 0, and since 0 > 3 is not true, it just went to the next step and subtracted 1 from 0 (which is correct because 0xfffffff is -1) and jumped over “thenpart”.