Name: Brock Stamper

Class: CS2160 – Assembly

Time spent: ~10 hours

Final version file name: bstampe2-project1.s

How to Run:

To run the code, take one of the .s files, open it on this website: https://comparch.edu.cvut.cz/qtrvsim/app/https://comparch.edu.cvut.cz/qtrvsim/app/, compile, and hit run. You will likely have to hit run many times because the program will halt on each ecall (and some iterations have A LOT of ecalls)

Brief Description:

In this assignment I was tasked with taking the project1.s file and editing it to include new functions and slightly different functionality. In part 1 I simple had to run the file. In part 2, I had to make a read and write function that replaced the bulk of the code in the main function. In part 3, I had to have the program read and write each char 1 by 1 in and out of the terminal. This introduced 2 new functions and had me rework the read and write function I had previously written.

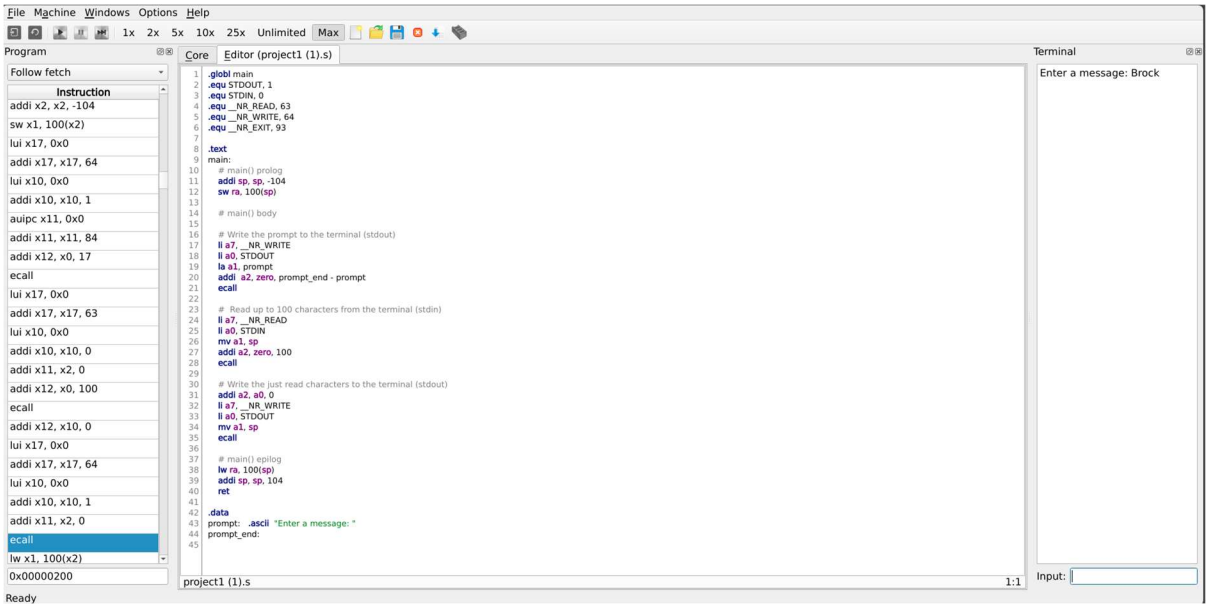
Resources: Google, StackOverflow, Reddit, Official RISC-V documentation, past assignments

Test Cases:

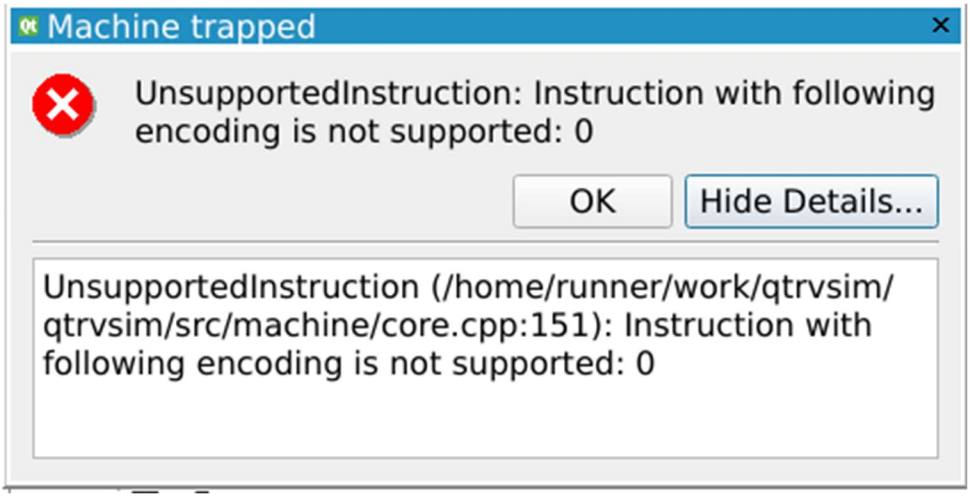
1. The prompt prints: Tests whether or not the default hardcoded text prints
    a. Input: None
    b. Expected output: “Enter a message: ”
    c. Actual output: “Enter a message: ”
    d. How to run: Run the program from the start. It should be the first thing that prints
2. My input message is taken: Tests whether or not the program is taking in the input correctly
    a. Input: “Brock”
    b. Expected output: “Brock”
    c. Actual output: “Brock”
    d. How to run: After running the above case, type “Brock” into the input box. If it disappears, then it most likely worked. Run the program to completion and if it printed back out “Brock” into the terminal, test passed.

Difficulties

1.1 First run attempt



1.2 Unsupported instruction error



1.3 Buttons

1.3.1 I could not find the right buttons intuitively and it did take me a minute to figure out how to compile the code. This is because the button names that I assume should be showing up if you hover over a button were not showing up. This is the button to compile

btw:

2.1 Implementing ‘write_string’ function

The only major issue I ran into was when I called the second jal on line 38. This was the code at the time.

```
addi a1, a0, 0
mv a0, sp
# Call write function
jal write_string
```

```
write_string:
    mv a2, a0
    li a7, __NR_WRITE
    li a0, STDOUT
    ecall
    ret
```

It would just stall on the ecall line and not move forward. I did lose a lot of progress the first time this happened so before running from then on, I would save a copy to my download as a temp location. If it worked, I moved it to the main git repo and uploaded it.

Eventually, after I watched the registers and moved the program line by line, I noticed that these lines were backwards:

```
addi a1, a0, 0
mv a0, sp
```

I needed to save a0 into a0 (aka do nothing with a0) and save sp into a1. This being the case, addi a0, a0, 0 is redundant so I removed the line and changed the second line to mv a1, sp. This fixed the issue and allowed me to finish up v2.0.

2.2 Implementing the ‘read_string’ function

This was also fairly straightforward. After thinking through which register I needed to save the sp and 100 I got it to work quickly. I didn’t run into any major issues with this one.

3.1 Implementing the ‘putchar’ and ‘puts’ functions

The biggest issue I ran into with this was these few lines here:

```
#body
WHILE:
    mv a0, s1
    lb t0, 0(a0)
    beqz t0, DONE
    addi s1, s1, 1
    jal putchar
    lb t0, 0(a0)
    blt t0, zero, ERROR
    j WHILE
```

The logic was sound but I had real trouble saving the address, moving the address, then comparing the value of the address. I figured out eventually that mv s1 (the address of prompt) to a0 for the putchar function was good, then I just had to load the byte at that address into a temp value to make sure we were still good. I then load the byte again after putchar just to make sure that t0 is set to the proper value.

3.2 Implementing the 'getchar' and 'gets' functions

This part was a little easier because of my work on the previous part. However, this code, yet again, gave me a bit of a headache. I found that for whatever reason, the first letter was not getting saved correctly so it would print the input without the first letter. Looking at the registers, it seemed that a0 was not incrementing so the next letter we read replaced the previous one but then did not do that going forward… I found out I needed to add a mv t0, a0 before looping to fix it.

```
WHILE:
    jal getchar
    lb t1, 0(a0)
    blt t1, zero, ERROR
    sb t1, 0(t0)
    addi t0, t0, 1
    li t2, 10
    beq t1, t2, DONE
    mv a0, t0
    j WHILE
```