x86 Assembly Writeup

Name: Joshua Abraham

Date: 9 November 2017

Current Module: x86 Assembly

Project Name: "Fibonacci"

Project Goals:

The relay project aims to create a program written in x86 assembly that calculates the n^{th} Fibonacci number, where $0 \leq n \leq 500$.

Considerations:

- The program must take one command line parameter, n.
- The program should produce an error message if no parameter is passed.
- The program must be able to calculate Fibonacci numbers from $0 \leq n \leq 500$.

Initial Design:

The project is composed of the following files:

· *Makefile*: The main makefile for the project.

· *fibonacci.s*: The source code for fibonacci.

· *test.sh*: The test runner for the project.

Data Flow:

My fibonacci program begins execution by performing command-line argument checks and string to integer conversion. The first check is on the number of arguments. The second check occurs during the string conversion routine. If the user input begins with a '-', the program exits and prints a usage statement. Next, it finishes converting the input into an integer. The program then enters the Fibonacci algorithm shown in Figure 1:

$$f(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ f(n-1) + f(n-2) & \text{otherwise} \end{cases} \quad (1)$$

If the input is zero or one, the program handles these "Edge cases" in Edge_fib and exits cleanly. Otherwise the program continues into a loop where it uses three six-QWORD chunk numbers to calculate a desired Fibonacci number.

```
long num, one = 0, two = 1, fib_num;
for (int i = 1; i < num; i++) {
            fib_num = one + two;
            one = two;
            two = fib_num;
}
```

Listing 1: Fibonacci in C

Code listing 1. shows the Fibonacci algorithm the program uses to calculate $f(n)$ where $2 \leq n \leq 500$. The program uses three 6 QWORD chunk variables to calculate each successive Fibonacci number. Using the traditional rcx register to count until it reaches the user input, the program follows the structure of the C code in Listing 1. Once the number is calculated, the program prints the number using a call to the C function *printf*, then returns 0 and exits.

Communications Protocol:

No communication occurs besides from user to program (command-line) and program to user (STDOUT).

Potential Pitfalls:

- Calculating extremely large numbers (well over 100's of septendecillions)
- Keeping assembly code well organized
- Flow control of program without high-level constructs such as while/for loops.

Test Plan:

User Tests:

· Performed test specified in the project supplement manual.

· Gave malformed input (strings, more than one argument, negative numbers, numbers larger than 500).

· Gave no input.

Test Cases:

· Used test.sh to automate testing.

Conclusion: