Name: Stephen Paradis
Date: May 20, 2016
Current Module: Intel Assembly using C
Project Name: Fibonacci

Project Goals:

The goal of this project was to create a fibonacci program with assembly. The project should take a number and print out the corresponding nth fibonacci number where n was that given number.

Considerations:

- The program should be made solely in assembly
- The program should be able to be made from the top-level directory by invoking 'make'
- The program can be modified to take up to the 300th fibonacci number
- The value can be printed in hex, or in decimal as a challenge
- The program should not break from any given input

Initial Design:

At first my program was built to use scanf to get user input for the nth fibonacci number. However that was changed to accept the number on the command line instead. The number is read in and strtol is used to convert the number into a long. I use 4 registers and create my number using add, adc(add with carry), and xchg to exchange the values in the registers. Lastly all values are passed into the proper registers and used as arguments for printf which prints the complete hex value.

Data Flow:

The program starts by getting the second argument on the command line and calling strtol to convert that to a number. It then performs error checking to see if the value is not between 0-300 and jumps to the fail label and prints out the error message. Otherwise it continues to clear the registers the program is using. It continues into the fibonacci label and proceeds to calculate the number. When it gets to the nth fibonacci number it jumps out of the loop and prints the hex value of the number.

Communication Protocol:

None.

Potential Pitfalls:

One possible pitfall is overwriting registers when printf is called which would cancel values the program should keep. Allowing for negative number or numbers that are too high for the registers used. Not clearing out the registers

the program is using. Not catching the correct return for failure from strtol function call.

Test Plan:

User Test:

1. Ran the program with no input
2. Ran the program with a negative number on the command line
3. Ran the program with 0 as the command line argument
4. Ran the program with a random valid number from 0-300
5. Ran the program with a number higher than 300
6. Ran the program with something other than a number

Automated Test:

None.

Test Cases:

1. Ran the program as ./fibonacci to check if it would raise a usage error
2. Ran the program as ./fibonacci -1 to check if it would run or raise an error usage
3. Ran the program as ./fibonacci 0 to check if it would print out 0 as it is supposed to or if it would raise an unwanted error
4. Ran the program as ./fibonacci ‘n’, where n wa a random valid value between 0-300
5. Ran the program as ./fibonacci 301 to see if it would raise the correct error
6. Rand the program as ./fibonacci str to see if it would raise the error instead of trying to execute

Expected Result:

For each of these tests I wanted to see if it would raise the error for not being between 0-300 or not being a number. Each of those tests should have ran the program only if the number given on the command line was between 0-300. Otherwise it should have raise a usage error and quit out.

Conclusion:

At the beginning I tried using scanf however it needed the stack pointer to be aligned correctly and it would give me a segmentation fault if I did not put the stack pointer back to where it started from. Because of this error I ended up going with command line input instead which I had to use strtol to convert the string to a correct number. From there I had to use 4 registers to completely store up to the 300th fibonacci number. Once I used ‘adc’(add with carry) I was able to do the fibonacci sequence. One possible way I could think of to optimize it would be to allow for more registers to be used for bigger numbers of fibonacci. Another way to optimize this program would be to switch to static memory to allow myself more possible scaling.