Assignment 1

Introduction to Assembly Language

Computer Structure

Alfonso Pineda (100472157)

Course 2023/2024

Degree in Computer Engineering

TABLE OF CONTENTS

Exercise 1

E(): decided to implement this function first because it's simpler than the others (It only takes 1 external function, the factorial function, to successfully implement E()).

The first step I took was to implement the factorial function which takes as argument an integer and returns a float with single precision. The following pseudocode is the one corresponding to factorial.

```
factorial (arguments: int i) return: float{
    t0 = 0
    t1 = 1
    ft0 = ConvertToFloat(t1)
    while( t0 != i ){
        t0 += 1
        ft1 = ConvertToFloat(t0)
        ft0 *= ft1
    }
    return ft0
}
```

I decided to use floats to store and process the factorial because I tried to use integers and it raised an error of overflow (The error let me compile and run the entire program but it returned wrong results).

The next step was to implement the E() function per se. The result starts with a 1 because the factorial of 0 is 1, then I use s0 as a counter, in terms of float registers I used fs0 to store the 1 (needed for the division), fs2 to store the result of the division between 1 and the result of the factorial and fs3 for the final result. I needed to store the ra into the stack because I used factorial, this means that the E function (as the cos, sin and tg) are non-terminal functions, then in order to follow the standard I used the s registers. The following pseudocode corresponds to E().

```
E (arguments: none) return: float{
    s0 = 1
    fs0 = ConvertToFloat(s0)
    fs3 = fs0
    s0 = 0
    s1 = 6
    s2 = 1
    while (s0 != s1){
        s0 += 1
        fa0 = factorial(s0)
        fs2 = fs0 / fa0
        fs3 += fs2
    }
    return fs3
}
```

Data entered	Description of test	Result expected	Result obtained
	Test if function works	2.7182	2,7180

cos(x): The next function I implemented was the cosine function because it's the second most simple function since I only need to multiply the argument by 2 in the exponent and the factorial. The following steps were to implement a function for the sign of cosine and a function to calculate the power of some number. The arguments of the power function are floats because the argument of the cosine can be a decimal number with decimals.

```
exponent_sign(arguments: int n) returns: int{
    n %= 2
    if (n != 0){
        return -1
    }
    return 1
}
power_of(arguments: float x, int n) returns: float{
    t0 = 1
    ft0 =x
    while (t0 != a0){
        x *= ft0
        t += 1
    }
    return x
}
```

In terms of the design of the cosine function I needed to store the return address and the argument of the cosine function in the stack, In order to always have an error of less than 0.001 I put an arbitrary number in the top of the loop that will always have the first three decimals correct, The following pseudo code is for the cosine.

```
cos(arguments: float x) returns floar{
    s0 = 1
    s1 = 12
    fs0 = ConvertToFloat(s0)
    while (s0 != s1){
        fs1 = ConvertToFloat(Sign_power(s0))
        fs1 *= power_of(x, 2 * s0)
        fs2 = factorial(2 * s0)
        fs1 /= fs2
        fs0 += fs1
        s0++
    }
}
```

Data entered	Description of test	Result expected	Result obtained
-6.2	more or less -2pi	0.996	0.996

-3.14	more or less pi	-0.999	-0.999
-1	random number	0.540	0.540
0	0	1	1
2	random number	-0.416	-0.416
4.8	random number	0.087	0.087

sin(x): I literally copied and pasted the cosine function and added one in all the (2 * s0)

Data entered	Description of test	Result expected	Result obtained
-6.2	more or less -2pi	0.083	0.083
-3.14	more or less pi	-0.001	-0.001
-1	random number	-0.841	-0.841
0	0	0	0
2	random number	0.909	0.909
4.8	random number	-0.996	-0.996

tg(x): I only calculated the sine and the cosine in their respective functions and then divided them. I know that this can be improved by making a custom function of all the previous ones where I mix the loops in order to calculate half the loops, but I am short of time.

I don’t have time to include a battery of tests of the tg function, but It’s fine except the 0 case

Exercise 2

SinMatrix(iAddress, oAddress, rows, columns): For this exercise I only needed to loop through the matrix applying the sine function, I don't see necessary the making of a pseudo code for this (it doesn't even represent correctly what's happening)

The test cases are the same as the sin function but in an "array" (sequentially organized in memory)

Extra testing

I also tested using different functions in the same run, to test the robustness of the functions and I made the functions for it to work.

Conclusion

I think that this lab was too long, too difficult for the level of this course, I would like to elaborate more but I don't have time.