Universidad
Carlos III de Madrid

Computer Structure

DEGREE IN COMPUTER ENGINEERING

Practice 1. Introduction to Assembly Language

Course 2023/2024

Salvador Ayala Iglesias | 100495832 | 100495832@alumnos.uc3m.es | g89
Inés Fuai Guillén Peña | 100495752 | 100495752@alumnos.uc3m.es | g89

Table of Contents

Exercise 1

For sin and cos functions we have created two auxiliary functions: factorial and exponential. Factorial auxiliary function is also used inside the E function. tg function uses sin and cos functions, so it also uses both auxiliary functions factorial and exponential.

The factorial function gets an integer value in a0 and returns (a0)! by iterating and multiplying by the consecutive terms from 1 to a0's value. Before the loop we check if the value of a0 is 0, since 0! = 1, and we skip the loop if so, returning 1.

The exponential function receives two parameters, fa0(base) and a0(exponent) and calculates fa0^a0 by iterating in a loop. Before the loop we check if the value of a0 is 0, since x^0 = 1 for any x (different from 0), and we skip the loop if so, returning 1.

All pseudocodes were written in python.

- Function sin(x):

We must create a function that meets the following summation: $\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$

BEHAVIOUR:

The function sin receives one parameter (x) in fa0.

First, we check if the given value is 0. If this happens, we jump to the end without running the function, since we know sin(0) = 0 If the condition is not met, we enter a loop after loading the needed values in registers.

Inside the loop, we call two auxiliary functions (exponential and factorial) to calculate the value of x^(2n+1) and (2n+1)! respectively. Then, the values are divided to get the general term of the sum (for the actual value of n), excepting the sign, (-1)^n, which is calculated later.

Once we have the value of the general term, we get the absolute value of it and compare it with the error (0.001). If the value of the term is smaller than the error, we can conclude that the addition of it to the whole sum of terms is not relevant to achieve the expected value. Therefore, we jump to the end of the function without adding it.

If the value is not smaller than the error, we multiply it by the sign and add it to the total. Before iterating again, we increase the value of n by 1 and change the sign of the "sign" register.

After all this process, we move the value of the sum to the return register fa0 and return.

PSEUDOCODE:

```
def sin(x):
    if x == 0:
        return 0

    summing = 0
    error = 0.001
    sign = 1
    count = 0
    while True:
        aux = 2 * count
        aux += 1
        term = exp(x, aux) / factorial(aux)
        if abs(term) < error:
            break
        term *= sign
        summing += term
        sign *= -1
        count += 1

    return summing
```

BATTERY TEST:

Data to enter	Description of the test	Expected result (within the range of error)	Obtained result
X = 0	We test if the value when the input 0 is correct	0	0 (OK)
X = 2π	We test if the value when the input 2π is correct	0	0 (OK)
X = -2π	We test if the value when the input -2π is correct	0	0 (OK)
X = 3π/2	We test if the value when the input 3π/2 is correct	-1	-1 (OK)
X = -3π/2	We test if the value when the input -3π/2 is correct	1	1 (OK)
X = π/2	We test if the value when the input π/2 is correct	1	1 (OK)
X = -π/2	We test if the value when the input -π/2 is correct	-1	-1 (OK)
X = 2	We test if the value when the input 2 is correct	0,909	0,909 (OK)
X = 4	We test if the value when the input 4 is correct	-0,757	-0,757 (OK)

- **Function cos(x):**

We must create a function that meets the following summation: $\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n}$

BEHAVIOUR:

The function cos receives one parameter (x) in fa0.

Due to its similarity with the sin function, the behaviour is nearly the same except two main changes.

cos(0) = 1, so we check the value given in fa0 and return a 1 in case the given value

is 0.

Since the given formula for cos is the same as the given one for sin excepting the change of 2n+1 to 2n, the code used for the calculation of them is the same, but implementing these changes. Therefore, the behaviour described for sin is the same as the one for cos, but changing this value.

PSEUDOCODE:

```
def cos(x):
    if x == 0:
        return 1

    summing = 0
    error = 0.001
    sign = 1
    count = 0
    while True:
        aux = 2 * count
        term = exp(x, aux) / factorial(aux)
        if abs(term) < error:
            break
        term *= sign
        summing += term
        sign *= -1
        count += 1

    return summing
```

BATTERY TEST:

Data to enter	Description of the test	Expected result (within the range of error)	Obtained result
X = 0	We test if the value when the input 0 is correct	1	1 (OK)
X = 2π	We test if the value when the input 2π is correct	1	1 (OK)
X = -2π	We test if the value when the input -2π is correct	1	1 (OK)
X = 3π/2	We test if the value when the input 3π/2 is correct	0	0 (OK)
X = -3π/2	We test if the value when the input -3π/2 is correct	0	0 (OK)
X = π/2	We test if the value when the input π/2 is correct	0	0 (OK)
X = -π/2	We test if the value when the input -π/2 is correct	0	0 (OK)
X = 2	We test if the value when the input 2 is correct	-0,416	-0.416 (OK)
X = 4	We test if the value when the input 4 is correct	-0,654	-0,654 (OK)

- Function tg(x):

BEHAVIOUR:

We know tg(x) = sin(x)/cos(x).

We first check if the denominator (cos) is 0, in this case we have to return +inf.

Within the given range of the test (-2π, +2π), the only values of x that make cos(x) = 0 are +-π/2 and +-3π/2. Therefore, we will check if the given value coincides with any of these values to avoid any unnecessary calculations.

In order to achieve this, we do π/2 - x and take the absolute value. If this value is lower than the error (0.001), we jump to the special condition and return +Inf. If not, we also check if π/2 - x (in absolute value) is smaller than the error and jump to the special condition if so.

The comparison is done this way to avoid floating point approximation errors, considering any difference smaller than the given error (0.001) negligible. By doing this calculation, we avoid calculating both sin and cos.

If the given values do not coincide with any of the special values, we calculate sin(x) and cos(x) by using the previous functions for both of them.We divide the obtained values and return the result, which will be the value of tg(x).

PSEUDOCODE:

```
def tg(x):

    if x == math.pi / 2 or x == 3 * math.pi / 2:
        return math.inf
    sin_x = sin(x)
    cos_x = cos(x)
    return sin_x / cos_x
```

BATTERY TEST:

Data to enter	Description of the test	Expected result (within the range of error)	Obtained result
X = 0	We test if the value when the input 0 is correct	0	0 (OK)
X = 2π	We test if the value when the input 2π is correct	0	0 (OK)
X = -2π	We test if the value when the input -2π is correct	0	0 (OK)
X = 3π/2	We test if the value when the input 3π/2 is correct	+inf	+inf (OK)
X = -3π/2	We test if the value when the input -3π/2 is correct	+inf	+inf (OK)
X = π/2	We test if the value when the input π/2 is correct	+inf	+inf (OK)
X = -π/2	We test if the value when the input -π/2 is correct	+inf	+inf (OK)

X = 2	We test if the value when the input 2 is correct	-2,185	-2,185 (OK)
X = 4	We test if the value when the input 4 is correct	1,158	1,158 (OK)

- **Function E():**

We must create a function that meets the following summation: $$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

BEHAVIOUR:

In this function, we use a loop where we first call the factorial function and then we divide 1 by the result obtained. We obtain the value of the summation term.

As in the previous functions, we check if the term is smaller than the error: end the loop if so, add it and repeat if not.

PSEUDOCODE:

```
def E():

    summing = 0
    error = 0.001
    count = 0
    while True:
        term = 1 / factorial(count)
        if abs(term) < error:
            break
        summing += term
        count += 1

    return summing
```

BATTERY TEST:

Data to enter	Description of the test	Expected result	Obtained result
No input	We must check whether the obtained value is the correct one.	2,718	2.718 (OK)

Exercise 2

In this exercise, we have to create a function which receives a matrix A and has to obtain a new matrix B, whose values are the sin of the values of the first matrix (A). The function does not return anything.

BEHAVIOUR:

The function receives four parameters: address of the start of the matrix A in a0, address of the matrix B in a1, number of rows of the matrices in a2 and number of columns of the matrices in a2.

The function takes the number of rows and columns and multiplies them to get the total number of terms in the matrix, that will also be the length (in words) of the memory space used for both matrices. This value will indicate when to stop iterating.

Starting from the first term, we save the word stored in the address of A, pass it to fa0 and calculate the sin of it. We store the value (store word) in the address of B. We iterate by adding 4 to both addresses, being the address value stored in both registers the address of the next terms to be calculated and stored, respectively. Also, we decrease the "number of terms" by 1. This way, we can stop the loop when the number of terms reaches 0.

PSEUDOCODE:

```
def SinMatrix(A,B,r,c):

    size = r * c
    count = 0 # needed for python
    while size > 0:
        B[count] = sin(A[count])
        size -= 1
        count += 1
```

BATTERY TEST:

Due to the fact that the function does not return anything, this test is done using a 2x2 matrix and checking the terms of the result function (B) after the function is executed. The values must be equal to the sin(x), being x the value of each term in the matrix.

Data to enter	Description of the test	Expected result (within the range of error)	Obtained result
X = 0,5	We test if the value when the input 0,5 is correct	0,009	0.009 (OK)
X = 7,25	We test if the value when the input 2π is correct	0,126	0,126 (OK)
X = -3,6	We test if the value when the input -2π is correct	-0,063	-0.063 (OK)
X = -26,9	We test if the value when the input -2π is correct	-0,452	-0.452 (OK)

Problems encountered

The main problem found in exercise 1 was the calculation of the factorial function. The calculation of x! for values larger than 7 caused errors in the calculation of sine and cosine, since the value exceeded the integer maximum and it returned a 0 instead of the actual result (overflow). This was fixed by calculating and returning the value directly in floating point format. The range of floating point numbers is larger, which allows the calculations without any issue.

There were no major problems found in exercise 2.

Time spent

The development of this practice has taken us around 26 hours, over 1 week. The usage of CREATOR to edit, compile and debug the code was key to achieve the objectives in such a short time.

Conclusions

The development of this lab has helped us to better understand the functioning of the registers and the stack. It has also secured our understanding of RISC-V language and the conventions (parameter passing) around it.

Along the development of the lab, we came up with some ideas that helped optimise the code. We decided to implement some of them, only the ones that allowed a better and faster functioning without changing the structure of the given formulas (exercise 1). A further improvement can be made, but it implies reducing the readability of the code and significantly changing the approach with respect to the given formulas.

We can conclude that, after finishing the lab, we have a better understanding of both the programming language and the computer and register's structure. We also found it entertaining and useful for our career.