System Programming Project

Students

- Jonas Balsfulland (MatrNr. 6660582)
- Felix Stegmaier (MatrNr. 6079153)

Submitted Files

Filename	Description
project.java	High-Level implementation of the algorithm
project.s	MIPS assembly implementation
documentation.md	This documentation in markdown format
documentation.pdf	This documentation in PDF format
ProjectTINF15AIA.pdf	The project assignment

Assignment

Write a MIPS assembly program to compute *sin(x)*, *cos(x)* and *tan(x)*. The concrete project assignment was handed out and can be found attached.

Algorithm and Optimization

Taylor approximation

The taylor approximation for *sin(x)* can be found in the project assignment. However is is more efficient to approximate *cos(x)* using taylor around the center point 0 with the following formula:

```
cos(x)  = sum[(-1)^i * x^(2*i) / ((2*i)!)]
          from i=0 to infinity
        = 1 - x^2/2! + x^4/4! - x^6/6! + x^8/8! - ...
```

We chose to implement *cos(x)* over *sin(x)* because:

- *cos(x)* is axially symmetric to the y=0 axis, so *cos(x) = cos(-x)*
- the taylor approximation of *cos(x)* uses smaller values inside the sum since *2i* is used

instead of $2i+1$, therefore the error will be smaller.

- *(2i)!* is smaller than *(2i+1)!* by a factor of $2i+1$ and can better fit into an 32-bit integer.

The calculation of each term can be simplified by defining the following series:

```
k_(i+1) = (-1)^i * k_i * x^2 / (2*i * (2*i-1)) for k >= 0
with k_0 = 1
```

So summing over *k_i* from 0 to infinity is equal to the formula given above. Using this iterative approach the result of the calculation of the previous term can be used to calculate the next one.

Mapping x to the right quadrant

Since this formula is centered around $x=0$ and the sum can only be performed for a finite amount of terms, the result of the calculation will have an error of $R(x) \sim x^m+1$ where m is the last index of the sum.

But because *cos(x)* is symmetric, periodic with a period of *2*Pi* and because it is symmetric within the period it is possible to reduce the range that is needed to be calculated correctly to the interval *[0, Pi/2)*.

The mapping to calculate *cos(x)* can be performed as following:

1. *cos(x) = cos(abs(x))*
2. *cos(x) = cos(x mod (2*Pi))*
3. - *0 <= x < Pi/2 ==> cos(x) = cos(x)*
   - *Pi/2 <= x < Pi ==> cos(x) = -cos(Pi - x)*
   - *Pi <= x < 2Pi ==> cos(x) = cos(x - Pi)*

Calculating sin(x) and tan(x)

sin(x) and *tan(x)* can be computed from *cos(x)* using the following conversions:

- *sin(x) = cos(x - Pi/2)*
- *tan(x) = sin(x) / cos(x)*

Output table

As input the user should provide three numbers, *n*, *x_min* and *x_max*. As output the program should generate a table containing *n* equidistant values in the interval of *[x_min, x_max]*. Therefore the distance between each value needs to be *(x_max - x_min) / (n-1)* given that *x_max > x_min* and *n > 1*. If *n=1* is given, only the calculations for *x_min* will be performed.

For the table itself the following considerations were taken:
SPIM outputs between 1 and around 20 characters when printing a double using systemcall 3. However finding out how many characters were actually printed is tedious, so a really 'pretty' table layout is hard to achieve. A solution to this would be implementing `printf`, which is out

of scope for this project.

```
| x | sin(x) | cos(x) | tan(x) |
| --- | --- | --- | --- |
| 0.0 | 0.0 | 1.0 | 0.0 |
| ... | ... | ... | ... |
```

One solution is to use a more flexible table layout, like the one above. This syntax is compatible to markdown table syntax, so it can be easily rendered into a proper table.

This Example renders into the following table.

x	sin(x)	cos(x)	tan(x)
0.0	0.0	1.0	0.0
...	...	...	...