**CMSC 411 Final Project – Presentation Outline/General Things to Discuss**

Highlighted = someone add to this

**Approach**

The project calls for an implementation of the CORDIC (Coordinate Rotation Digital) algorithm using hyperbolic sin and cos functions as well as the exponential function (ex). In order to implement the CORDIC algorithm we first had to learn how it works.

From the Documentation, we know that...

> The CORDIC algorithm involves a series of rotations of increasingly smaller pre-determined angles, in order to approximate a target angle.

AND

> In the hyperbolic implementation, those angles are arctanh(0.5i), where i is the rotation number which goes from 1 to the maximum number of rotations.

SO

1) For each rotation, new x and y values are calculated using the formulas:

x’ = x + d \* y/2i

y’ = y + d \* x/2i

Note: Explain what each variable means

2) For the hyperbolic implementation of the CORDIC algorithm:

y starts at 0

x starts at 1.207534

Note: Make sure to explain why x starts at 1.207534

3) When all of the rotations are done...

x is approximately the value of cosh(t)

y is approximately the value of sinh(t)

We can get et by adding cosh(t) and sinh(t)

4) With the above things in mind, we began the implementation.

Should we tell Chi that we started the implementation in C and then translated to assembly?

**Code**

Explain the most important steps as to how the code works:

[Mention general logic/idea for each step?]

1. Move initial variables into registers (K, x, y, and desired angle) and initialize counter
2. Start loop and iterated 92 times (precision float is 23 bits multiplied by 4 for each word so 92), and check whether angle is positive or negative
3. For each iteration in loop
   1. Shift x and y by i+1 and store in register
   2. Add shifted x to y and store in register (and vice versa)
   3. Substract the desired angle by arctantable[i] and increment counters
4. Once i = 92, load x and y into floating point registers and divide by 216 to get sinh and cosh. And then add x and together to get ex and load in floating point register

When discussing the code, explain magic numbers (what is the purpose of those weird numbers in the code with no immediate context) and how the numbers are stored and modified during the calculations and why they are stored modified the way that they are.

Examples:

> **Show where we enter the input and what the input represents**

> Why does x start at 1.207534? For the regular CORDIC, K is 1.646760 which makes the scaling factor (1/K) equal to .607. For the hyperbolic CORDIC algorithm, K is equal to half of the original K or .828159 which makes the scaling factor 1.207534

> Explain how we avoid multiplying during the CORDIC calculations? See above. Using shifts, adds and predetermined rotations.

> Explain what the arctanh table is used for (fixed predetermined angles)

> Maybe mention that we used Floating-Point Instructions to store/manipulate numbers for sinh, cosh, and ex. Values for sinh, cosh, and ex are in VFP registers s3, s4, and s5 as 32 bit floats.

> All numbers are multiplied by 216 before doing the calculations, and divided by 216 after calculations, because scaling this much in the beginning always use to represent an evenly distributed subset of real numbers roughly from -32768 to 32768.

**Issues**

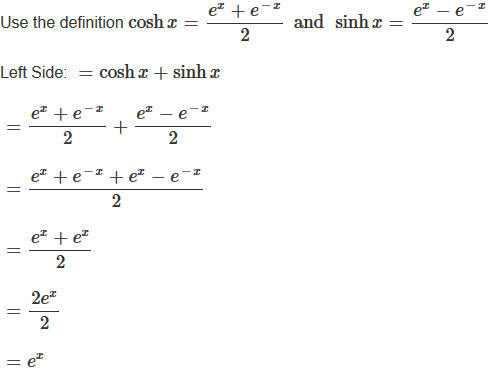
Code Related Issues (From Documentation):

“The hardest part about this project was completing the Assembly code because the algorithm was only documented well for sin and cos online. There was very little information about how to change the Cordic algorithm to compute cosh, sinh, and ex .”

Solution – How did we find out how to implement the hyperbolic sin and cos function?

Once we have hyperbolic sin and cos, we can get ex because ex = sinh(x) + cosh(x).

Proof:



Non-code Related Issues:

Understanding how to calculate CPI – within the group, several members were confused about how CPI was calculated.

**Results**

Present clock speed results:

The most important thing is to state how we calculated each stat...

CPI = Clock Cycles per Second / Average Instructions per Second

Total Computer Cycles = CPI \* Total Instructions

Total Processing Time = Total Computer Cycles / Clock Cycles per Second

|  |  |  |  |
| --- | --- | --- | --- |
| **Statistic** | **32kHz clock** | **1MHz clock** | **1GHz clock** |
| CPI |  |  |  |
| Total computer cycles |  |  |  |
| Total processing time |  |  |  |

Present sample test data:

**Note: Chi may ignore our sample numbers and ask to input his numbers**

Our numbers...

|  |  |  |  |
| --- | --- | --- | --- |
| **x** | **cosh(x)** | **sinh(x)** | **ex** |
| 0 | 1.003 | 0.04332 | 1.046 |
| 0.88 | 1.416 | 1 | 2.416 |
| π/4 | 1.327 | 0.870 | 2.198 |
| - π/4 | 1.327 | -0.8704 | 0.4568 |
| -1 | 1.546 | -1.178 | 0.3688 |