COIS 2300H Lab 7

This is to familiarize yourself with the basics of outputting a program to assembly code (sadly, x86 assembly makes MIPS look easy). Visual studio up to 2008 will output Mips but they removed that for whatever reason. Gcc will let do this as well (and it can output MIPS directly) and you can use that instead of visual studio, but I’ll leave that to you to figure out on your own if it’s your thing.

The theory side of this lecture is a lead in to other concepts in the course, notably pipelines and optimization.

Mac users: Either boot into windows, use GCC or use a lab computer. I don’t think visual studio for mac even supports C++, and I’m not inclined to figure out the best solution to that problem.

Step one making a C/C++ program.

Visual studio - > new project

C++ console project

No precompiled header or SDL checks

I called mine CpptoASM but you can call it whatever, make sure you know where you actually created the project though.

We’re going to make a really useless C++ program. We’re then going to look at the actual x86 assembly generated for it.

Sorry in advance that you need some bizarre headers and we’re going to use cout to write to and from the console but you’ll get the idea hopefully (cout is functionally very similar to System.Out in C# but it’s simpler for all we want to do).

Write your code to say this

int a = 1;

int b = 2;

float fA = 1.0;

float fB = 3.0;

int main()

{

int c = a + b;

float fC = fA / fB;

return 0;

}

You should be able to figure out what that does. Build and run (a black screen will flash at you and then… nothing, there isn’t any output or anything).

You can normally see assembly code while debugging C++ in visual studio (and eclipse too). For this in Visual Studio put a breakpoint on code in question and when debugger hits it right click (on the source code) and find "Go To Assembly" ( or press CTRL+ALT+D ) (could also be called disassembly, or similar), that’s not necessarily going to get you much on a program this simple but it should be possible anyway.

Now we’re going to muck with some settings in visual studio. Project menu -> Projectname Settings - > C/C++ -> Output files

Under Assembler output change that box to “Assembly With Source Code (/FAs)

I think you have to rebuild your program for it to actually do that.

Open up the CpptoASM\CpptoASM\Debug directory and find the CpptoASM.asm file, open it (with Notepad++ if you’re smart, visual studio will open it but does no text colouring/highlighting or anything helpful to make it readable).

Now change your program to the following:   
  
#include "stdafx.h"

#include <iostream>

using namespace std;

int a = 1;

int b = 2;

float fA = 1.0;

float fB = 3.0;

int main()

{

int c = a + b;

cout << c << endl;

float fC = fA / fB;

cout << fC;

cin.get();

return 0;

}

The function cout writes to the console (black screen with text on it), cin asks for user input, so pressing enter causes the program to close. Endl; is just end line.

Run the program.

No problem, right? Good. (You can still look at the assembly but don’t bother quite yet).

Change the main function of your program to

float fC = fA / fB;

cout.precision(9);

cout << fC;

cin.get();

return 0;

(So you just removed the integer stuff, and are now displaying only the floating point result to 9 significant figures).

Run the program

0.333333343

Uh… that’s not right.

Change the precision to 25 and try again.

0.3333333432674407958984375

That’s still not right. That’s not helpful at all.

If you change the precision to larger than 25 it gives the same result.

Ok so that was simple.

But we need another calculation to illustrate a fairly important concept.

Try

int main() {

float inc = 0.000001, sum = 0.0;

for (int i = 1; i <= 1000000; ++i) sum += inc;

cout<< sum;

}

So we add 1/1000 000 a million times we should get 1 right?

We actually get 1.00904

Well that’s not right. That’s a problem with floating point precision, 0.000001 (base 10) can’t be neatly represented in binary so each time we add it, we keep adding our errors. This is a problem in floating point error compounding.

Ok now lets look at optimization

Convert your above program to the following

int main()

{

cout.precision(10);

float inc = 0.000001, sum = 0.0;

for (float i = 1.0; i <= 1000000.0;)

{

sum += (1.0 / i);

i += 1.0;

}

cout<< sum;

cin.get();

return 0;

}

Build, run

Note that the result is different if you try doubles versus floats (more floating point precision problems). Older versions of visual studio or different CPU’s may produce different results when you run that calculation.

I get

14.35735798

Let me know in class if you get a different result when running it (with floats).

Regardless. Open up your debug folder and make a copy of the .asm file you just outputted.

Now go back to visual studio, project properties etc. -> Optimization under C/C++. Change optimization to none and to full (or one of the other options, they are labelled /O1 /O2, /O3 etc.) .

Build, run make copies of the ASM files.

Go to <https://www.diffchecker.com/diff>

And paste in the no optimization on one side and full optimization on the other.

**Show us the comparison of the assembly codes**

Importantly, look at around lines 50 – 70 where the two different implementations of basic math operations (sum += (1.0/i) and i += 1.0 are shown.

While I don’t expect you to understand how those lines of code are different yet, it should be obvious enough that they are very very different.

(Optimized code can run anywhere from slower to 5x faster than non-optimized, that’s part of what we’re learning in the theory side of class, but the core concept we’re going to look at is what does it mean to optimize code? Key concepts: Order of operations, real versus floating point, pipelining, and the role of different CPU architectures).