VIRTUAL CPU

Course No: CENG-606

Lab #: 10 (Final)

Name: Seenan Bunni

Student ID: 821-622-107

Submission Date: April 27, 2015

**Description**

This project is essentially the making of a virtual CPU (central processing unit) emulator through the utilization of the C language. It acts as an interactive shell interface for users that allow them to load files in, display the file’s memory/data, trace or run the file (if the file is a file that can be run such as a binary file), zero out the registry, or even write bytes to the file.

This VPC, known by its C name as CPUI.c (for CPU Interface) was mostly developed on a VMware server that utilized the ‘VIM’ editor to compile and debug its workings…

**Final State Of The Program**

I would say that the program runs ok. There’s also no warnings or errors when compiling, which is always a good sign. As for run time issues, I managed to finally solve the issue with my ‘trace’ function, which was the only problem I was having for a few solid weeks now.

One thing I’ve noticed so far is the conflict between uintX\_t values and unsigned char/int/long/short values. They don’t always equal the same thing like uint8\_t is to unsigned char, but on some websites, they would think uint64\_t is what equals to long, but yet uint32\_t is what a long would be. I also saw one that said uint8\_t could be unsigned int…That really messed with my head. Maybe they’re referring to higher bit machines? I don’t know.

It should work on another compiler such as Visual Studio, because I followed the C standard of ‘all declarations should be on-top of any statement/function.’, which originally gave me a ton of errors…And I used uintX\_t types so byte-distinctions across systems shouldn’t happen.

**Summary Of Testing**

How I tested it was pretty simple…

Due to the trace command proving useful in debugging, all I needed was to create or obtain a working assembly program (such as test\_immediate.bin)…I needed to also analyze the data (hand-assembly to check the actual data such as what’s seen when you display the memory…what’s in test\_immediate.doc, basically). Same idea for addarray.bin or others.

Then, whenever I traced through it (with, hopefully, a working trace/fetch/execute command), it should’ve became apparent with what came into the registers if my program was messed up or not, due to me having the actual data to check on.

Like I proclaimed in the ‘Final State’ portion of this report, I had a problem lasting for at least 2 weeks, before and after most of my functions got deleted somehow (which I had to re-write)…Which was just solved recently through some small fixes needed, pointed out by a professor.

**Conclusions**

Honestly, aside from the mess it has been trying to craft this thing from bits of C code as well as debugging the issues I had, it’s pretty crazy all said and done. What was made here could pass to be an actual CPU (when done properly)…Things like these would’ve been sold for like $50000 back then in the 1900s or something. I’m probably exaggerating. But you can just think Intel with their i7 has something like this, but a thousand times more complicated…with actual hardware to go with it (aside from our actual computer).

I did have an issue with one aspect of the project, which was indicated to me by the add\_array.doc…When the program is run (add\_array.bin), the data at memory location 0120 doesn’t seem to change. I guess that translates to other programs as well. I don’t know why. All other aspects of the CPU seem to work fine.

**Sample Output**

This is some output of the simple test\_immediate.bin file (loading in the file, and tracing 4 times). The second picture has the last two traces of the test\_immediate.bin file and some memory modify workings. (The last two traces show the stop flag and SCZ portion of the register display working as well.). The third picture is an output of addarray.bin, near the end (specifically, 4 traces from the end). Shows a CCR flag being set, along with the stopflag when it hits the E000 instruction. Also has the memory displayed. Fourth picture shows the data after the program is run at 0120, while the data before the program is run is at 0100 and 0110. (It should change but it doesn’t.)

Fourth picture, you might notice I also threw in a ‘z’…That’s just to show you it works (with the following ‘r’ input). It had no impact on the data in offset 100, length 30, because I’ve tested the program before.

(And sorry if you can’t really read it, Microsoft Word sucks).

