Translating Loops and Observing Address Space Addresses

In this week, we observe the assembly code `gcc` generates under different optimization targets and new compilation options to generate assembly code that faithfluly follows the lecture slides.

Specifically, we use special `gcc` options to turn off position-independent code and execution. This way `gcc` generates assembly code that more closely resembles the code shown in the lecture slides. Use Google to try and get a sense of what these options do on your own.

We provide the Makefile with the source code to make it easy for you to compile the demo code. You should change `-Og` to `-O1` and back manually (read below).

Repeat the following activities using the source code we provide on the lecture webpage.

- Use the provided C source code files to translate for and while loops into x86-64 assembly
- Notice the for/while translation into a `jump to middle` pattern when the `-Og` optimization option of gcc is used
- Notice the for/while translation into a `first test` pattern when the `-O1` optimization option of gcc is used
- Notice how the switch statement is translated into a `jump table-based implementation` when the `-Og` optimization is used
- Compile and run the locate binaries with and without the PIC/PIE-related flags and observe where each part of the executable goes in memory
- Follow the instructions in the demo to print out the addresses in the jump table. (Recall the examine command `x/8xg` *starting_address*)

We also provide a `swap.c`. Learn to observe register updates in `gdb` by stepping through the swap code and using `info registers` command of gdb.

Optional Activity

Read about the `gcc` optimization flags we have used in lecture demos

`https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html`