When you are working with a new programming language, you should get in the habit if compiling and testing often.

1. Once your program is working and compiles without generating any error messages or warnings, compile your program again using these flags

gcc -ansi -pedantic -Wall cards.ce

Look up what these flags do. Did you get additional warnings? See if you can get your program to compile cleanly with these flags.

- ansi: tells the compiler to implement in ansi language, which disable features incomtabile with ansi.

- pedantic : used with –ansi, tells the compiler to adhere strictly to ansi standard.

- Wall : enables all warnings.

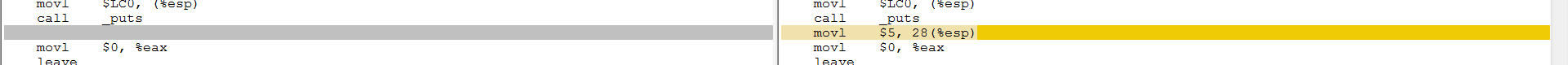
1. Check your modified program into the repo.
2. Add this line to hello.c

int x = 5;

Compile and look at the assembly language output (see [this section of *Think OS*](http://greenteapress.com/thinkos/html/thinkos002.html#toc8). Can you find the code that corresponds to this line?

“movl $5, 28(%esp) ”

1. What happens if you turn on optimization using the flag -O2?

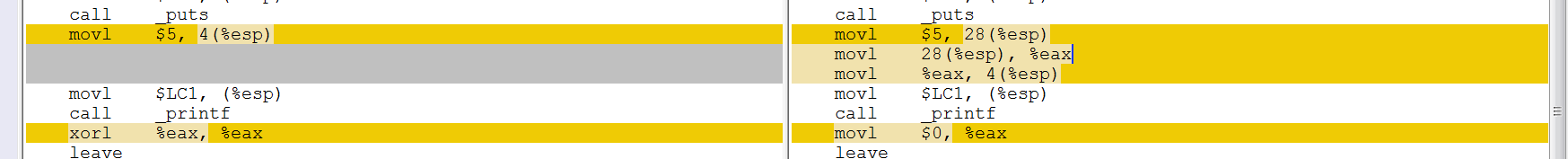


After optimization vs before optimization

Line disappears, does not save 5 anymore, (since it is not used)

Allocates less space in stack pointer

1. Modify the printf statement to print x, then compile it with and without optimization. What effect does it have when you print x?



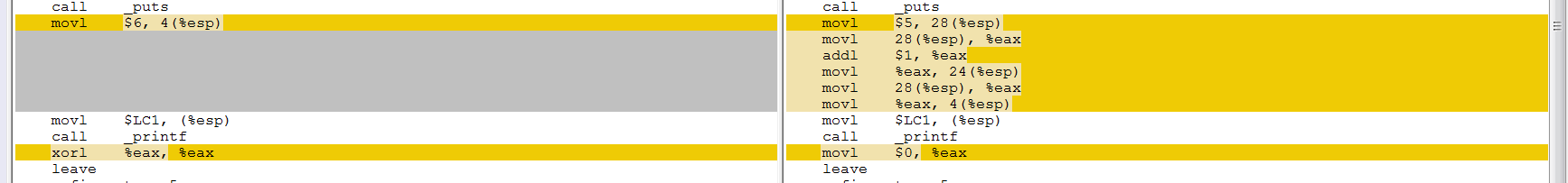
After optimization vs before optimization

Less movement of data, but x = 5 is declared in both cases.

1. Add

int y = x + 1;

And then print the value of y (but not x). Compile with and without optimization. What happens to x and y?



After optimization vs before optimization

Optimized: 6 is directly printed instead of performing addition in assembly code

Non-optimized: performs movement and addition of data in assembly code.

What conclusions can you draw about how optimization works?

1. Add comments to the code to explain these experiments and the results, then check in the final version.

Optimization minimizes the number of steps that are executed in assembly code and hence runs faster and uses less storage space to store unnecessary steps. However, due to the optimization, there might be bugs that will surface due to the reduction of steps in assembly (which is processed during the compilation).