Derek Johnson, dej3tc, 11/18/19
Optimized Code:

In order to show the changes at the assembly level that will occur with code optimization I created three simple methods in C++. The first is an adder method that takes in an int t and outputs t += 1. The second method, called point, takes in and integer and outputs a pointer to that integer. The third and final method, main, takes no arguments and performs several operations on a counter variable. The operations are shown in the appendix below.

It is immediately obvious that the -O2 flag has significantly altered the assembly code in 4 main ways.

1) The first and most apparent is the reduction in operations. The unoptimized code is 43 lines long while the optimized code is only 9. While it is not always true that less lines mean faster runtimes, in general this will hold.
2) In the main method, loop unrolling is used to speed up the process that occurs in the for loop. Instead of adding two to the counter 6 times as will happen in the unoptimized assembly, the final value after all of the operations will be loaded into the return register. The compiler can do this because this value will remain constant for all runs of the program.
3) For the adder method the main difference between the optimized and unoptimized code is the command used. In the unoptimized code, the parameter was loaded into a spot on the stack and then the add command was used to add 1 to it. In the optimized code, the lea command was used to perform addition and movement into the return register in one command.
4) The point function is a fairly illogical method and because of this the optimized assembly does nothing except xor the return value. The unoptimized assembly will make commands for every action you write, no matter how illogical.

APPendix:

C++ Code

```cpp
int adder(int c){
    return c+=1;
}

int * point(int p){
    int * t =  &p;
    return t;
}

int main() {
   const int two = 2;
   int counter = 0;
   for(int i=0; i<6;i++) {
     counter += two;
   }
   counter = adder(counter);
   int * p = point(counter);
   counter = counter^two;
   return counter;
}
```

Optimized Compilation

```asm
adder(int):
        lea     eax, [rdi+1]
        ret
point(int):
        xor     eax, eax
        ret
main:
        mov     eax, 15
        ret
```

Unoptimized Compilation

```asm
adder(int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], edi
        add     DWORD PTR [rbp-4], 1
        mov     eax, DWORD PTR [rbp-4]
        pop     rbp
        ret
point(int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-20], edi
        lea     rax, [rbp-20]
        mov     QWORD PTR [rbp-8], rax
        mov     rax, QWORD PTR [rbp-8]
        pop     rbp
        ret
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     DWORD PTR [rbp-12], 2
        mov     DWORD PTR [rbp-4], 0
        mov     DWORD PTR [rbp-8], 0
.L7:
        cmp     DWORD PTR [rbp-8], 5
        jg      .L6
        add     DWORD PTR [rbp-4], 2
        add     DWORD PTR [rbp-8], 1
        jmp     .L7
.L6:
        mov     eax, DWORD PTR [rbp-4]
        mov     edi, eax
        call    adder(int)
        mov     DWORD PTR [rbp-4], eax
        mov     eax, DWORD PTR [rbp-4]
        mov     edi, eax
        call    point(int)
        mov     QWORD PTR [rbp-24], rax
        xor     DWORD PTR [rbp-4], 2
        mov     eax, DWORD PTR [rbp-4]
        leave
        ret
```