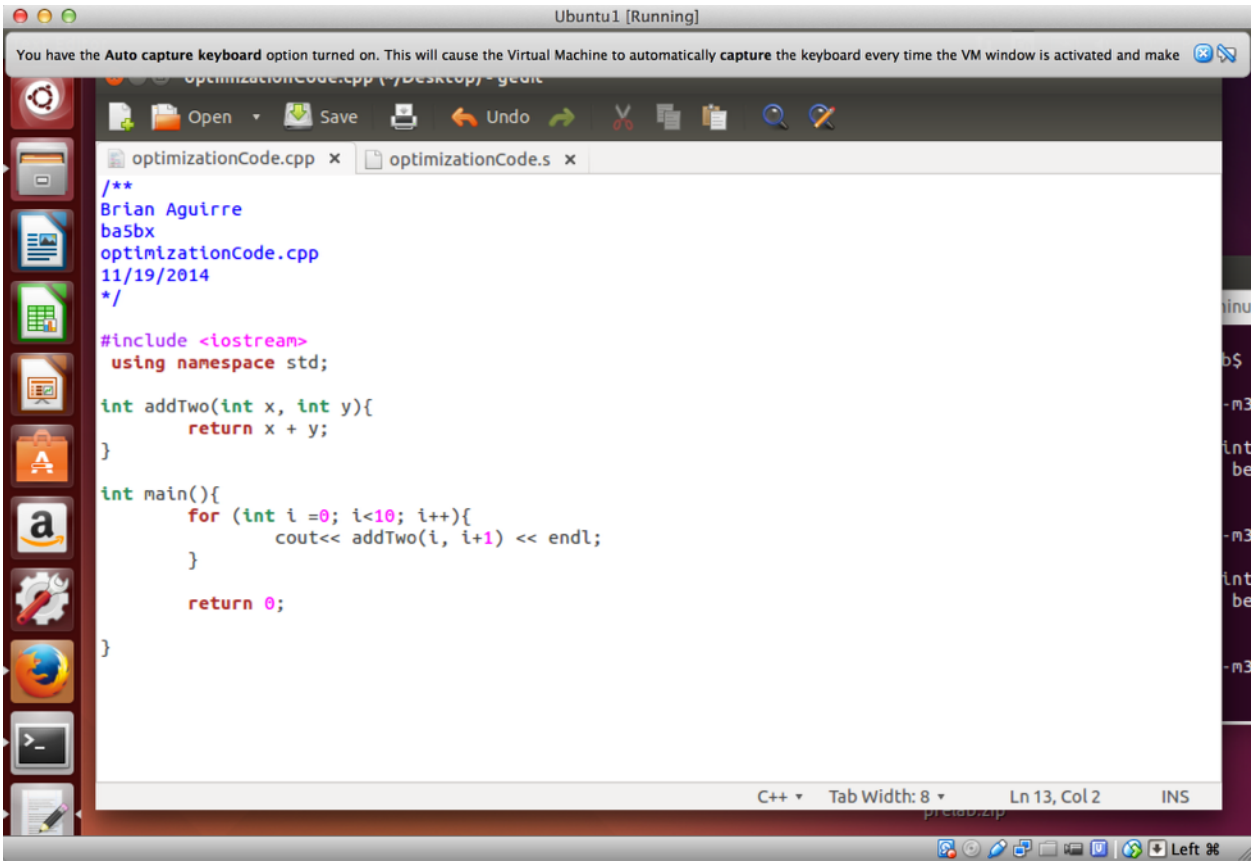
Brian Aguirre
ba5bx
inlab9.pdf
11/19/2014

Optimization:

For the in-lab portion of lab 9, I chose to do question 3, pertaining to optimization within loops and function calls. The following screen capture is the code that was then converted into x86 using the -S tag:



Notice how within the code, a method “addTwo” is created, which takes in two int types, x and y, and their sum is returned. This method is then called in the main method, where there is a for loop feeding it numbers from 0-10 (technically 0-11 as y is x + 1).

When using the -S tag for the .s file and then the -O2 tag for the optimized version of the .s file, the result of each is the following:

Normal Output:

```
    .comm   _ZSt8__ioinit,1,1
    .text
    .globl  _Z6addTwoii
    .type   _Z6addTwoii, @function
_Z6addTwoii:
.LFB971:
    .cfi_startproc
    push    ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    mov ebp, esp
    .cfi_def_cfa_register 5
    mov eax, DWORD PTR [ebp+12]
    mov edx, DWORD PTR [ebp+8]
    add eax, edx
    pop ebp
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
.LFE971:
    .size   _Z6addTwoii, .-_Z6addTwoii
    .globl  main
    .type   main, @function
main:
.LFB972:
    .cfi_startproc
    push    ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    mov ebp, esp
```

Line 1, Column 1 | Tab Size: 4 | R

optimizationCode.s — UNREGISTERED

optimizationCode.cpp × | optimizationCode.s × | optimizationCode-O2.s ×

```
    mov ebp, esp
    .cfi_def_cfa_register 5
    and esp, -16
    sub esp, 32
    mov DWORD PTR [esp+28], 0
    jmp .L4
.L5:
    mov eax, DWORD PTR [esp+28]
    add eax, 1
    mov DWORD PTR [esp+4], eax
    mov eax, DWORD PTR [esp+28]
    mov DWORD PTR [esp], eax
    call    _Z6addTwoii
    mov DWORD PTR [esp+4], eax
    mov DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
    call    _ZNSolsEi
    mov DWORD PTR [esp+4], OFFSET FLAT:_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostream
    mov DWORD PTR [esp], eax
    call    _ZNSolsEPFRSoS_E
    add DWORD PTR [esp+28], 1
.L4:
    cmp DWORD PTR [esp+28], 9
    jle .L5
    mov eax, 0
    leave
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
.LFE972:
    .size   main, .-main
    .type   _Z41__static_initialization_and_destruction_0ii, @function
_Z41__static_initialization_and_destruction_0ii:
.LFB978:
```

Line 1, Column 1 | Tab Size: 4 | R

optimizationCode.s — UNREGISTERED

optimizationCode.cpp | optimizationCode.s | optimizationCode-O2.s

```
66  _Z41__static_initialization_and_destruction_0ii:
67  .LFB978:
68      .cfi_startproc
69      push    ebp
70      .cfi_def_cfa_offset 8
71      .cfi_offset 5, -8
72      mov ebp, esp
73      .cfi_def_cfa_register 5
74      sub esp, 24
75      cmp DWORD PTR [ebp+8], 1
76      jne .L7
77      cmp DWORD PTR [ebp+12], 65535
78      jne .L7
79      mov DWORD PTR [esp], OFFSET FLAT:_ZStL8__ioinit
80      call    _ZNSt8ios_base4InitC1Ev
81      mov DWORD PTR [esp+8], OFFSET FLAT:__dso_handle
82      mov DWORD PTR [esp+4], OFFSET FLAT:_ZStL8__ioinit
83      mov DWORD PTR [esp], OFFSET FLAT:_ZNSt8ios_base4InitD1Ev
84      call    __cxa_atexit
85  .L7:
86      leave
87      .cfi_restore 5
88      .cfi_def_cfa 4, 4
89      ret
90      .cfi_endproc
91  .LFE978:
92      .size   _Z41__static_initialization_and_destruction_0ii, .-_Z41__static_initializat
93      .type   _GLOBAL__sub_I__Z6addTwoii, @function
94  _GLOBAL__sub_I__Z6addTwoii:
95  .LFB979:
96      .cfi_startproc
97      push    ebp
98      .cfi_def_cfa_offset 8
```

Line 1, Column 1 | Tab Size: 4 | R

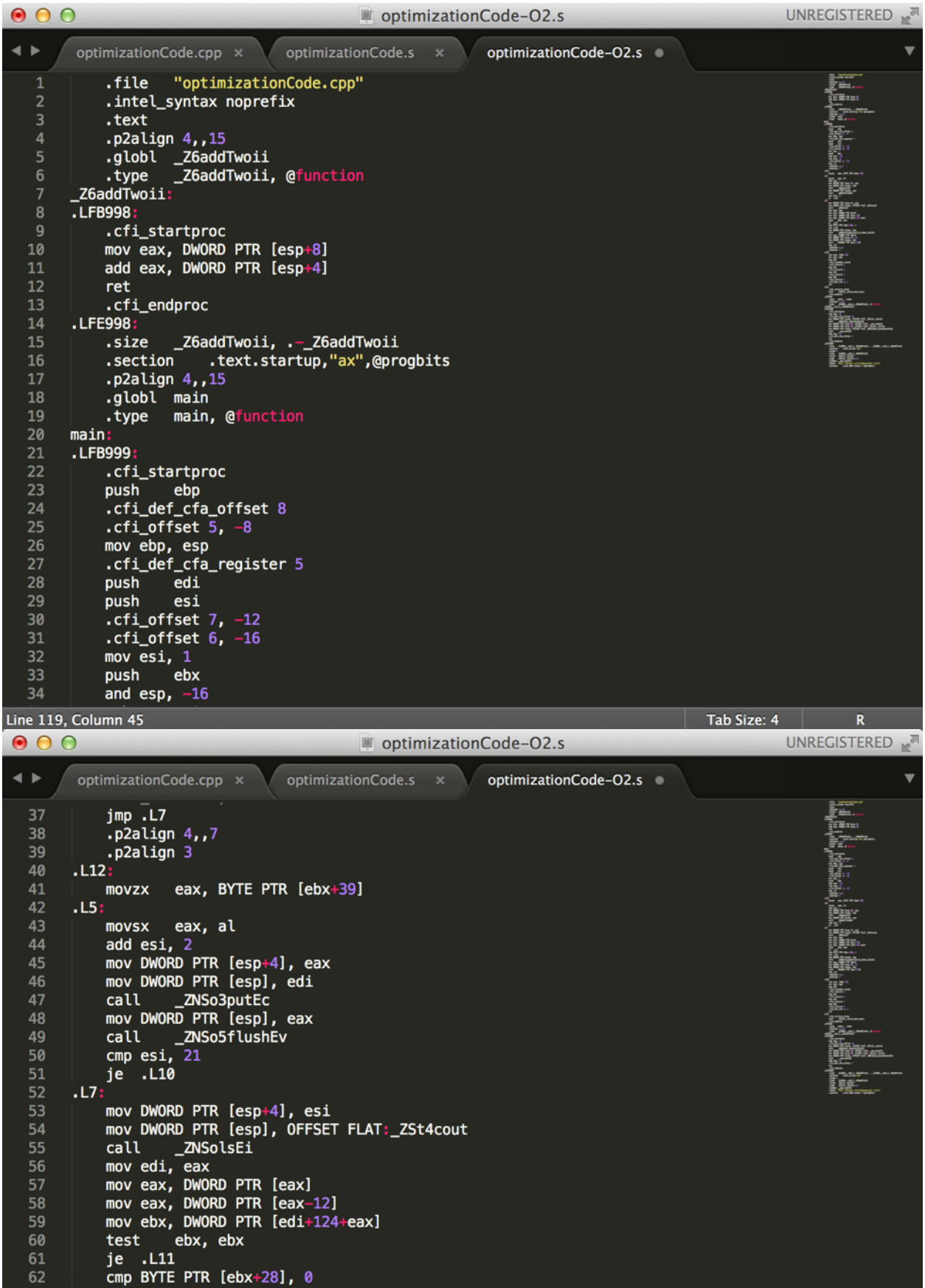
optimizationCode.s — UNREGISTERED

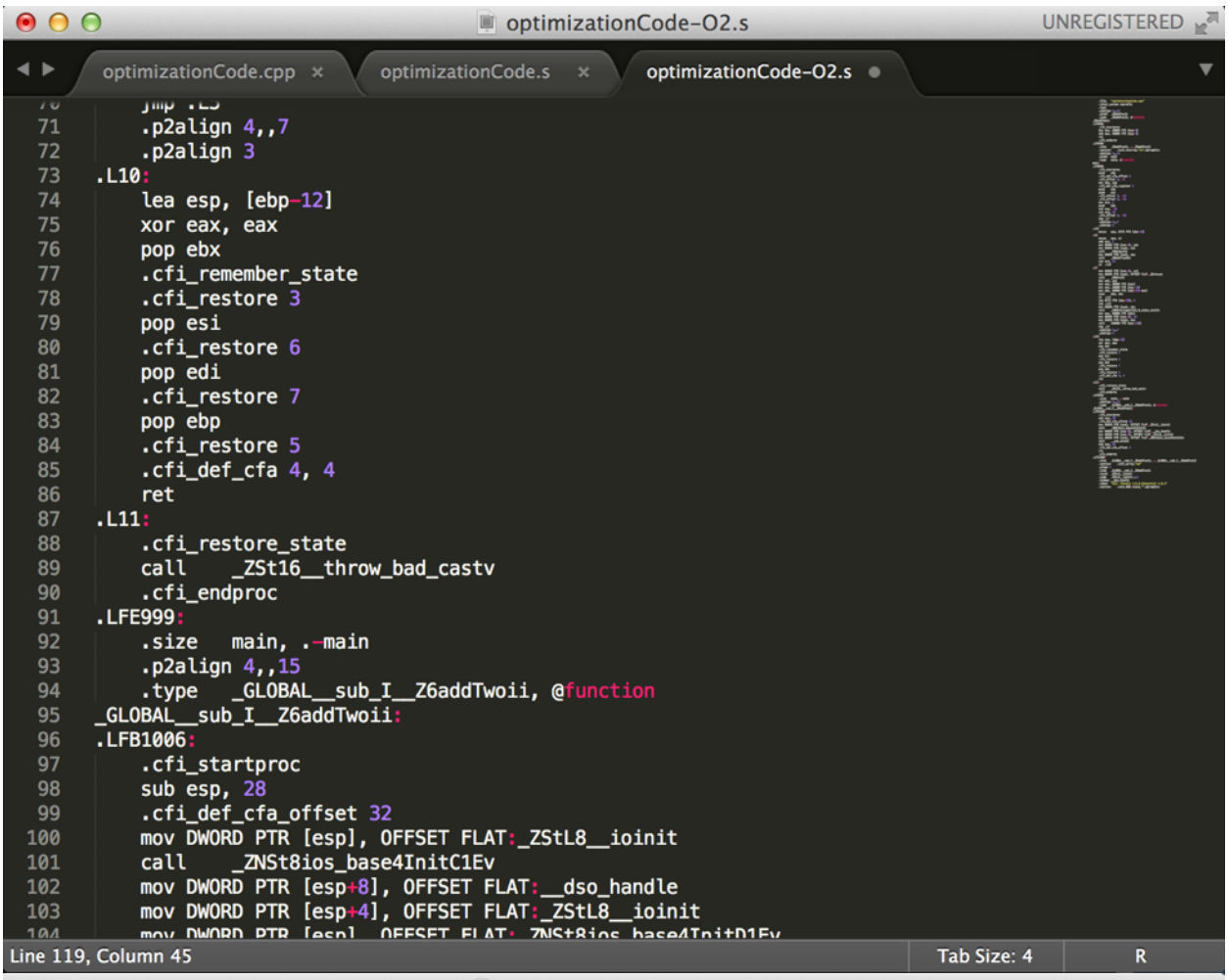
optimizationCode.cpp | optimizationCode.s | optimizationCode-O2.s

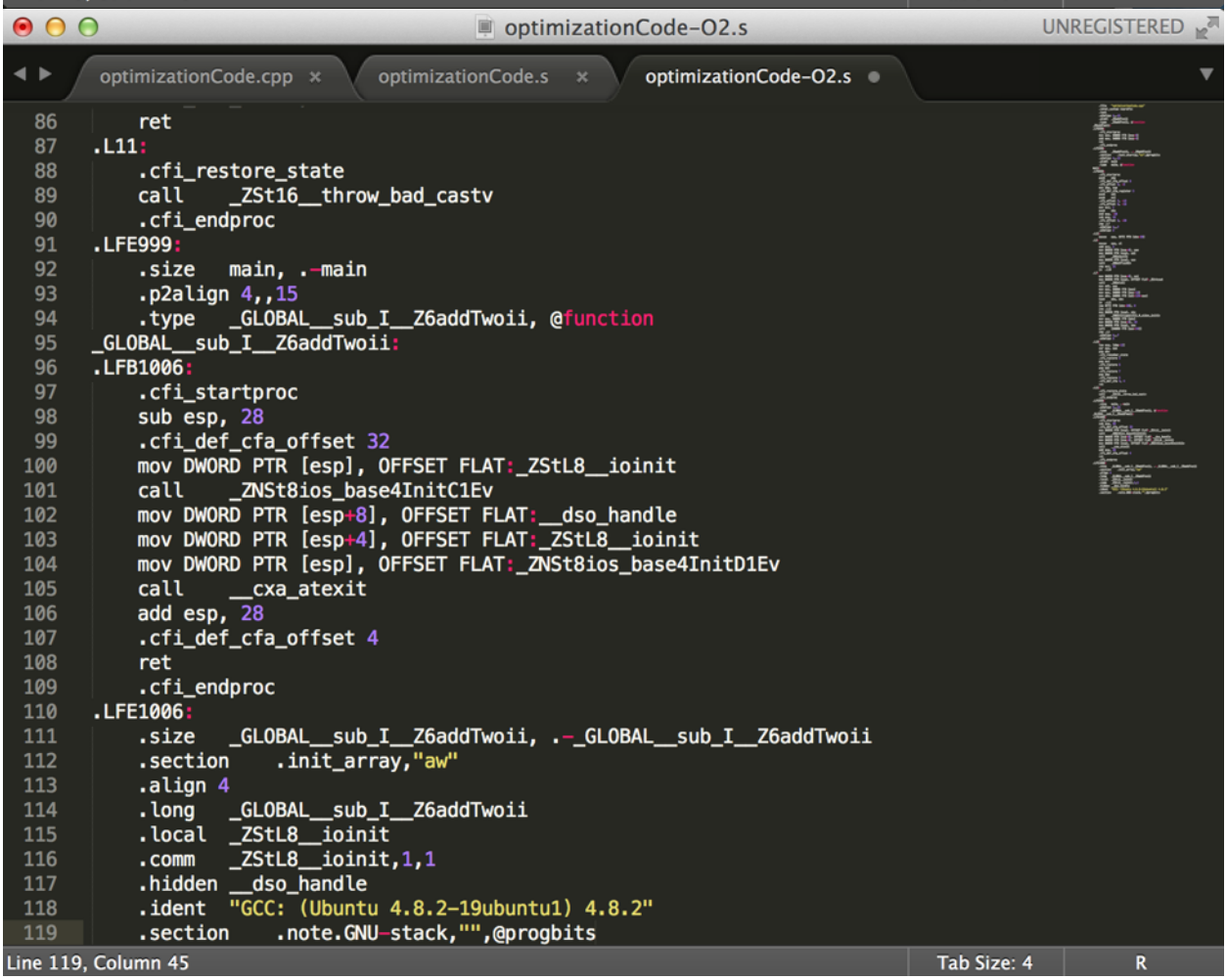
```
86      leave
87      .cfi_restore 5
88      .cfi_def_cfa 4, 4
89      ret
90      .cfi_endproc
91  .LFE978:
92      .size   _Z41__static_initialization_and_destruction_0ii, .-_Z41__static_initializat
93      .type   _GLOBAL__sub_I__Z6addTwoii, @function
94  _GLOBAL__sub_I__Z6addTwoii:
95  .LFB979:
96      .cfi_startproc
97      push    ebp
98      .cfi_def_cfa_offset 8
99      .cfi_offset 5, -8
100     mov ebp, esp
101     .cfi_def_cfa_register 5
102     sub esp, 24
103     mov DWORD PTR [esp+4], 65535
104     mov DWORD PTR [esp], 1
105     call    _Z41__static_initialization_and_destruction_0ii
106     leave
107     .cfi_restore 5
108     .cfi_def_cfa 4, 4
109     ret
110     .cfi_endproc
111 .LFE979:
112     .size   _GLOBAL__sub_I__Z6addTwoii, .-_GLOBAL__sub_I__Z6addTwoii
113     .section    .init_array,"aw"
114     .align 4
115     .long   _GLOBAL__sub_I__Z6addTwoii
116     .hidden __dso_handle
117     .ident  "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
118     .section    .note.GNU-stack,"",@progbits
119
```

Line 1, Column 1 | Tab Size: 4 | R

Optimized Version




optimizationCode-O2.s
UNREGISTERED
optimizationCode.cpp
optimizationCode.s
optimizationCode-O2.s
.p2align 4,,7
.p2align 3
.L10:
lea esp, [ebp-12]
xor eax, eax
pop ebx
.cfi_remember_state
.cfi_restore 3
pop esi
.cfi_restore 6
pop edi
.cfi_restore 7
pop ebp
.cfi_restore 5
.cfi_def_cfa 4, 4
ret
.L11:
.cfi_restore_state
call _ZSt16__throw_bad_castv
.cfi_endproc
.LFE999:
.size main, .-main
.p2align 4,,15
.type _GLOBAL__sub_I__Z6addTwoii, @function
_GLOBAL__sub_I__Z6addTwoii:
.LFB1006:
.cfi_startproc
sub esp, 28
.cfi_def_cfa_offset 32
mov DWORD PTR [esp], OFFSET FLAT:_ZStL8__ioinit
call _ZNSt8ios_base4InitC1Ev
mov DWORD PTR [esp+8], OFFSET FLAT:__dso_handle
mov DWORD PTR [esp+4], OFFSET FLAT:_ZStL8__ioinit
Line 119, Column 45
Tab Size: 4
R


optimizationCode-O2.s
UNREGISTERED
optimizationCode.cpp
optimizationCode.s
optimizationCode-O2.s
ret
.L11:
.cfi_restore_state
call _ZSt16__throw_bad_castv
.cfi_endproc
.LFE999:
.size main, .-main
.p2align 4,,15
.type _GLOBAL__sub_I__Z6addTwoii, @function
_GLOBAL__sub_I__Z6addTwoii:
.LFB1006:
.cfi_startproc
sub esp, 28
.cfi_def_cfa_offset 32
mov DWORD PTR [esp], OFFSET FLAT:_ZStL8__ioinit
call _ZNSt8ios_base4InitC1Ev
mov DWORD PTR [esp+8], OFFSET FLAT:__dso_handle
mov DWORD PTR [esp+4], OFFSET FLAT:_ZStL8__ioinit
mov DWORD PTR [esp], OFFSET FLAT:_ZNSt8ios_base4InitD1Ev
call __cxa_atexit
add esp, 28
.cfi_def_cfa_offset 4
ret
.cfi_endproc
.LFE1006:
.size _GLOBAL__sub_I__Z6addTwoii, .-_GLOBAL__sub_I__Z6addTwoii
.section .init_array,"aw"
.align 4
.long _GLOBAL__sub_I__Z6addTwoii
.local _ZStL8__ioinit
.comm _ZStL8__ioinit,1,1
.hidden __dso_handle
.ident "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
.section .note.GNU-stack,"",@progbits
Line 119, Column 45
Tab Size: 4
R

Something I immediately notices was that the optimized version was much harder to follow. Although there are still things I cannot understand about the un-optimized version, the normal output, I can generally understand and see what it is doing. While in the optimized version, there are small little snippets and overall the structure is different. After reading a bit more about why the optimized version looks considerably different, I found two explanations. The first stated that there are less offsets within the optimized version and more use of registers. This can be seen in the comparison of .L5 in the non-optimized version of the .s file and the .L10 in the optimized version. In .L5 there are several lines in which offsets of ESP are being moved or called in order to do the calculations. In comparison, the optimized code in the .L10 section, it shows that there were at least 3 registers used, and throughout the rest of the code, it has less offsets since it instead uses the registers. Which means that probably using these registers to store values is much after than going to the stack and fetching the data based on their locations off of offsets.

Another reason the structure is different is because as I was looking at the two, the un-optimized version has a call for the _Z6addTwoii which is the method I defined in order to add the two numbers but the optimized version does not have a call line for it. As I read more, I found out that the optimized version does a different version of the loop which is called loop unwinding or loop unrolling. According to a UMD professor (included within the citations), this method allows for the program to be faster as it neglects to be referencing back for some kind of pointer or out bounds for the loop and rather just counts. This method is indeed faster but it also does not do much in terms of the size of the code as it is clear that the optimized version is literally as long (in terms of lines) as the un-optimized version.

Something else to note about the un-optimized version vs. the optimized version is the manner in which things are done. In the un-optimized version the line mov eax, 0 is done with xor eax, eax. These two do the same but perhaps the mov command is more taxing (albeit by a very small fraction of a second) than xor, but lines like these are found throughout the two, which definitely add up. Something else is the fact that there are less call lines in the optimized version than the un-optimized version. This is probably because despite both having defined all of the code I wrote in c++, the optimized version probably considered certain things to be unnecessary and found it's own way to work around it (I don't completely understand it now, but I can definitely tell it performs things differently). But I guess that's something to also note, that even though the optimized version is faster and although in this case it takes the same amount of lines, it is also much more confusing, since the computer has essentially done things it's own way and users are left to try to figure out which way that is, and the unoptimized version, although slower, it's much more what I would write/have written in x86.

Citations:

http://www.cs.virginia.edu/~evans/cs216/guides/x86.html
http://www.eecg.toronto.edu/~amza/www.mindsec.com/files/x86regs.html
http://www.cs.umd.edu/~meesh/cmsc411/website/proj01/proja/loop.html