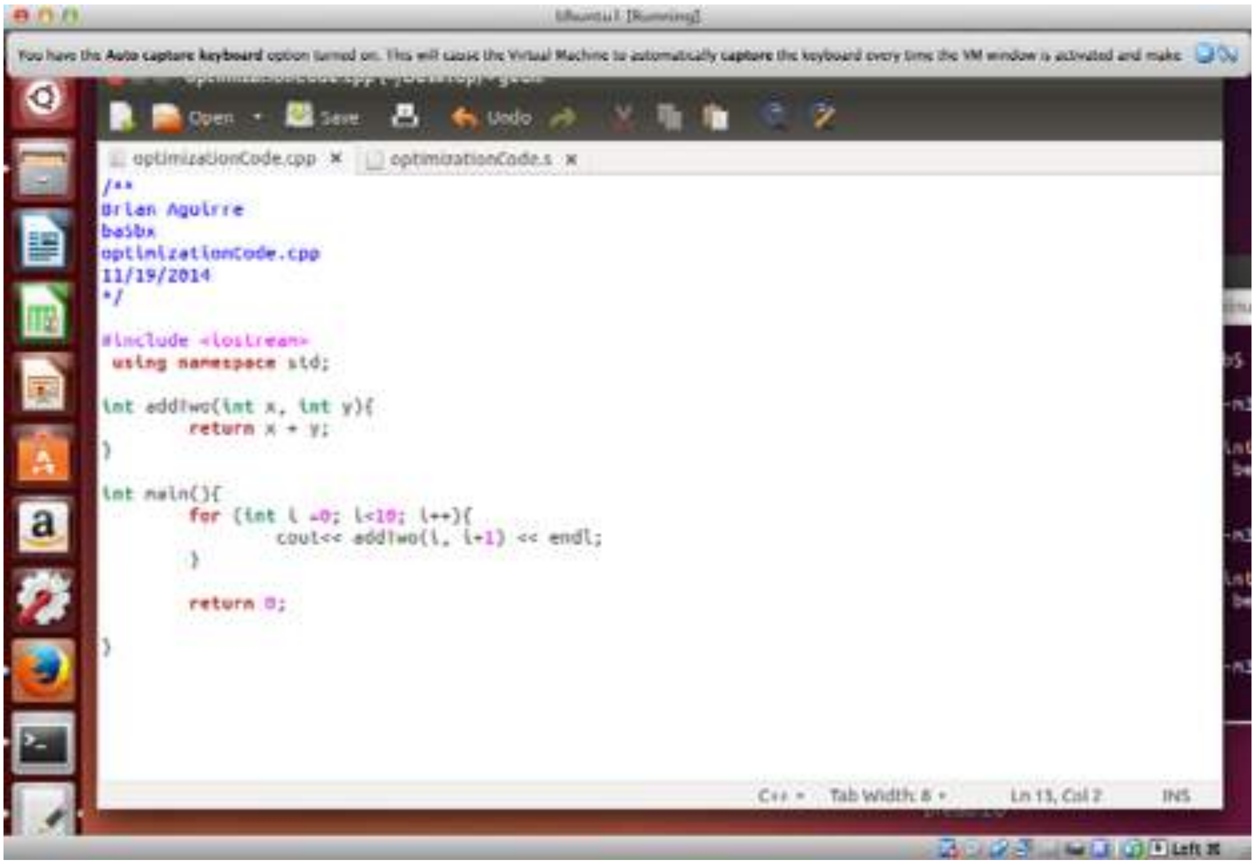
Brian Aguirre
ba5bx
postlab.pdf
11/21/2014

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:

optimizationCode.s — UNREGISTERED

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

```
	.file	"optimizationCode.cpp"
	.intel_syntax noprefix
	.local	_ZStL8__ioinit
	.comm	_ZStL8__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
```

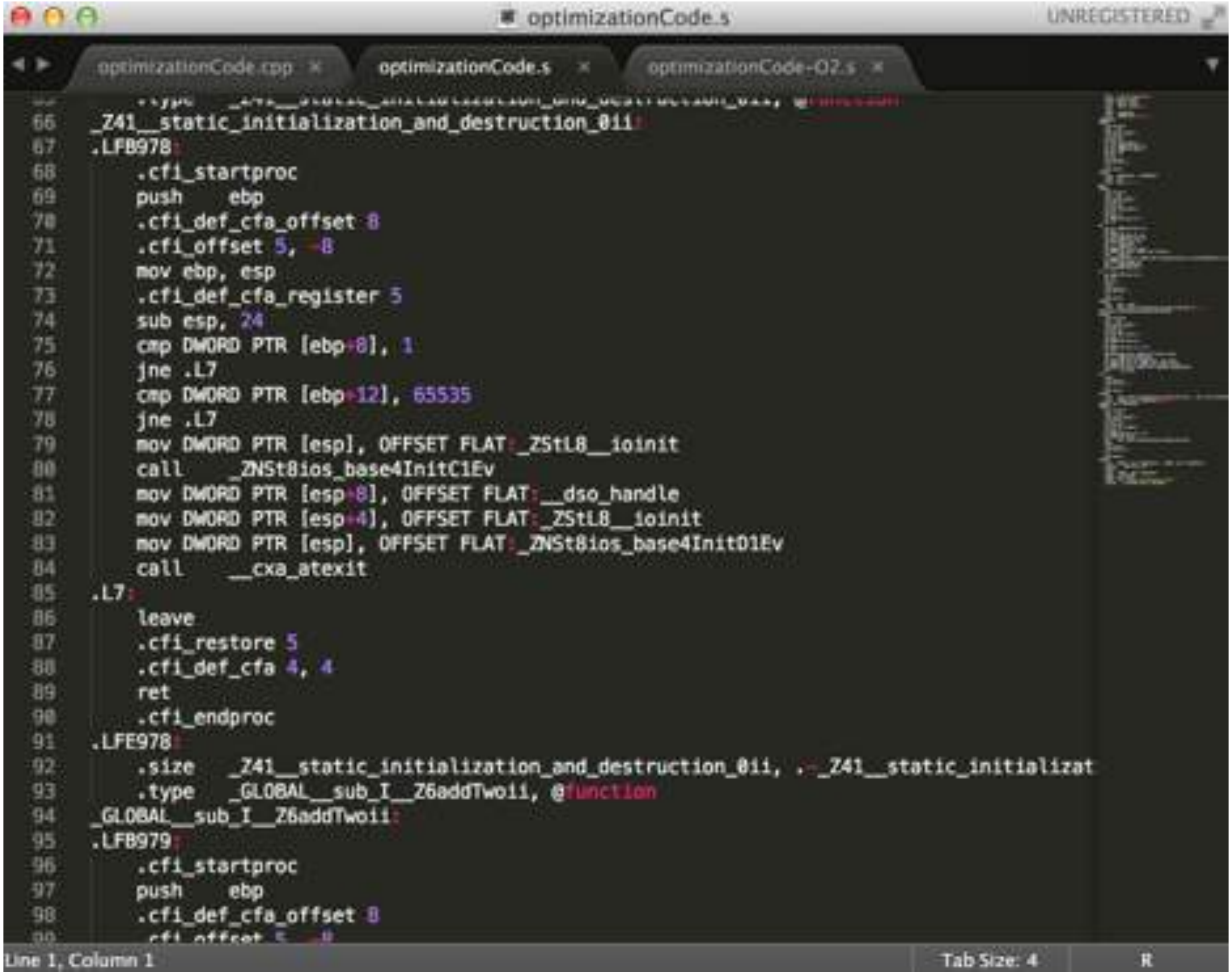
Line 1, Column 1 | Tab Size: 4

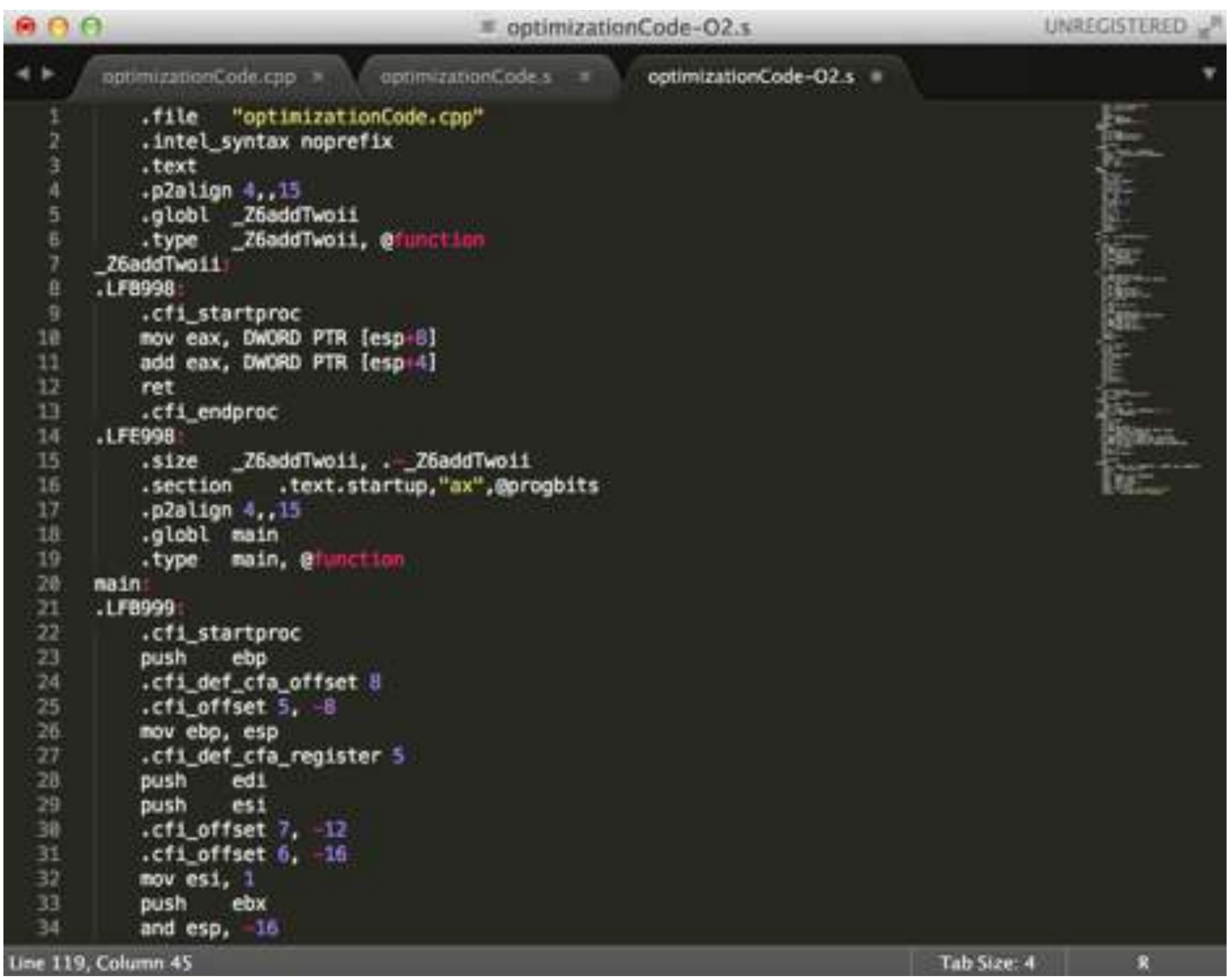
optimizationCode.s — UNREGISTERED

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


optimizationCode.s
UNREGISTERED
optimizationCode.cpp
optimizationCode.s
optimizationCode-O2.s
_Z41__static_initialization_and_destruction_0ii:
.LFB978:
.cfi_startproc
push ebp
.cfi_def_cfa_offset 8
.cfi_offset 5, -8
mov ebp, esp
.cfi_def_cfa_register 5
sub esp, 24
cmp DWORD PTR [ebp+8], 1
jne .L7
cmp DWORD PTR [ebp+12], 65535
jne .L7
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
.L7:
leave
.cfi_restore 5
.cfi_def_cfa 4, 4
ret
.cfi_endproc
.LFE978:
.size _Z41__static_initialization_and_destruction_0ii, .-_Z41__static_initializat
.type _GLOBAL__sub_I__Z6addTwoii, @function
_GLOBAL__sub_I__Z6addTwoii:
.LFB979:
.cfi_startproc
push ebp
.cfi_def_cfa_offset 8
Line 1, Column 1
Tab Size: 4
R


optimizationCode-O2.s
UNREGISTERED
optimizationCode.cpp
optimizationCode.s
optimizationCode-O2.s
.file "optimizationCode.cpp"
.intel_syntax noprefix
.text
.p2align 4,,15
.globl _Z6addTwoii
.type _Z6addTwoii, @function
_Z6addTwoii:
.LFB998:
.cfi_startproc
mov eax, DWORD PTR [esp+8]
add eax, DWORD PTR [esp+4]
ret
.cfi_endproc
.LFE998:
.size _Z6addTwoii, .-_Z6addTwoii
.section .text.startup,"ax",@progbits
.p2align 4,,15
.globl main
.type main, @function
main:
.LFB999:
.cfi_startproc
push ebp
.cfi_def_cfa_offset 8
.cfi_offset 5, -8
mov ebp, esp
.cfi_def_cfa_register 5
push edi
push esi
.cfi_offset 7, -12
.cfi_offset 6, -16
mov esi, 1
push ebx
and esp, -16
Line 119, Column 45
Tab Size: 4
R

```
	leave
	.cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
.LFE978:
	.size	_Z41__static_initialization_and_destruction_0ii, .-_Z41__static_initializat
	.type	_GLOBAL__sub_I__Z6addTwoii, @function
_GLOBAL__sub_I__Z6addTwoii:
.LFB979:
	.cfi_startproc
	push	ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	mov ebp, esp
	.cfi_def_cfa_register 5
	sub esp, 24
	mov DWORD PTR [esp+4], 65535
	mov DWORD PTR [esp], 1
	call	_Z41__static_initialization_and_destruction_0ii
	leave
	.cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
.LFE979:
	.size	_GLOBAL__sub_I__Z6addTwoii, .-_GLOBAL__sub_I__Z6addTwoii
	.section	.init_array,"aw"
	.align 4
	.long	_GLOBAL__sub_I__Z6addTwoii
	.hidden __dso_handle
	.ident	"GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
	.section	.note.GNU-stack,"",@progbits
```

```
	.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
```

optimizationCode-O2.s

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

optimizationCode-O2.s

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

```
	jmp .L7
	.p2align 4,,7
	.p2align 3
.L12:
	movzx	eax, BYTE PTR [ebx+39]
.L5:
	movsx	eax, al
	add esi, 2
	mov DWORD PTR [esp+4], eax
	mov DWORD PTR [esp], edi
	call	_ZNSo3putEc
	mov DWORD PTR [esp], eax
	call	_ZNSo5flushEv
	cmp esi, 21
	je	.L10
.L7:
	mov DWORD PTR [esp+4], esi
	mov DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
	call	_ZNSolsEi
	mov edi, eax
	mov eax, DWORD PTR [eax]
	mov eax, DWORD PTR [eax-12]
	mov ebx, DWORD PTR [edi+124+eax]
	test	ebx, ebx
	je	.L11
	cmp BYTE PTR [ebx+28], 0
	jne .L12
	mov DWORD PTR [esp], ebx
	call	_ZNKSt5ctypeIcE13_M_widen_initEv
	mov eax, DWORD PTR [ebx]
	mov DWORD PTR [esp+4], 10
	mov DWORD PTR [esp], ebx
	call	[DWORD PTR [eax+24]]
	jmp .L5
```

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; lines like lea and other quicker ways to perform additions or moving of data. 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:

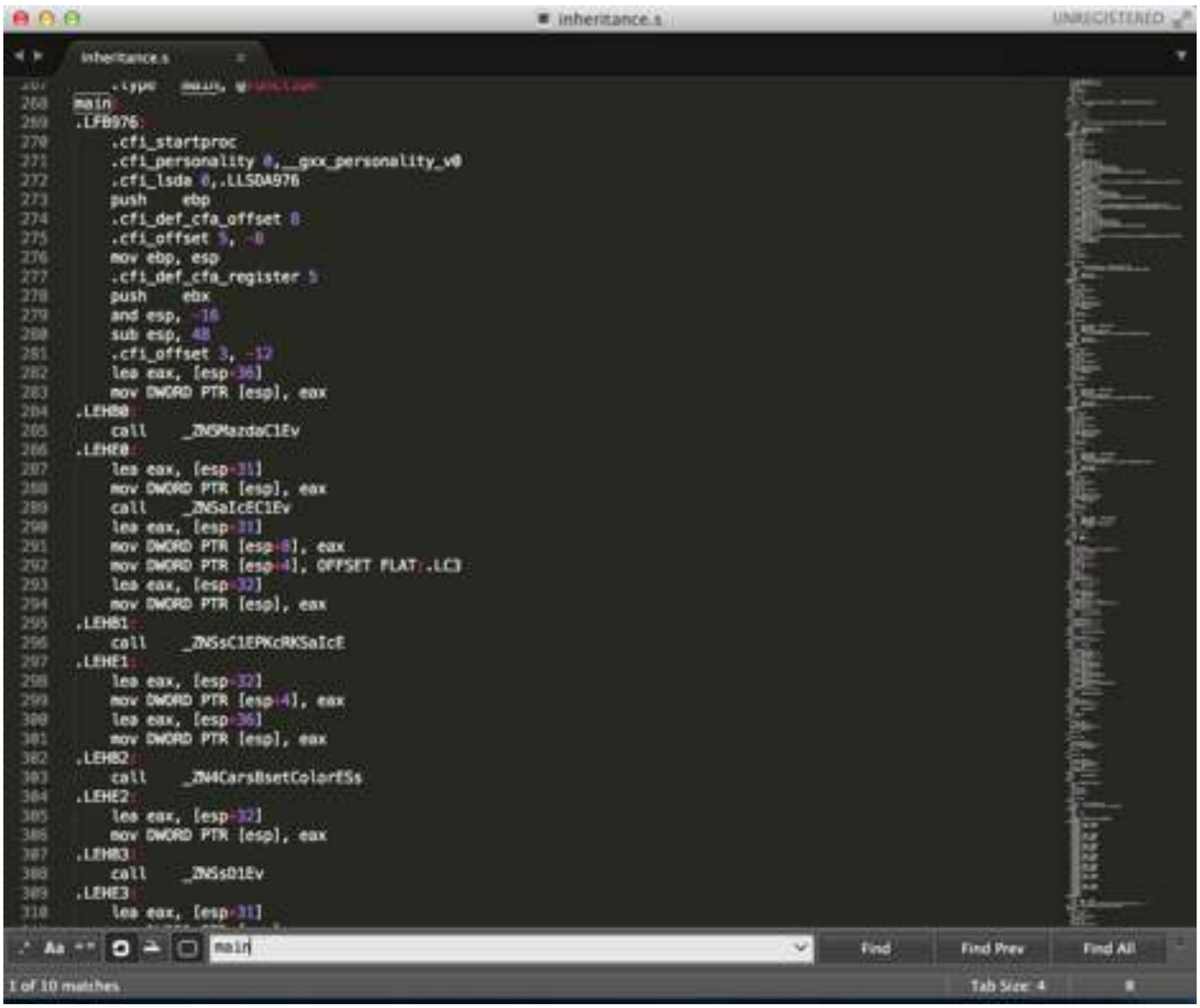
http://www.cs.virginia.edu/~evans/cs216/guides/x86.html
http://www.eecg.toronto.edu/~amza/www.mindsec.com/files/x86regs.html
http://www.cs.umd.edu/~meesh/cmsc411/website/proj01/proja/loop.html

Inheritance:

For the second portion of the report, I have chosen to do inheritance within C++. In order to test and view what the assembly equivalent for an inherited class and object declaration looked like, I wrote the following code:

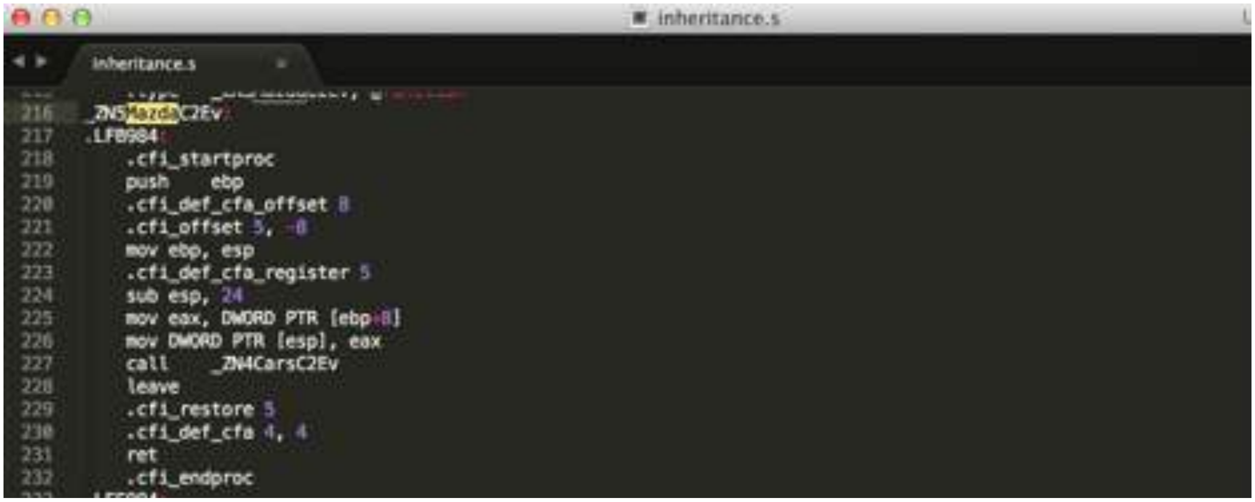
```cpp
/**
Brian Aguirre
ba5bx
inheritance.cpp
11/21/2014
*/

#include <iostream>
#include <string>

using namespace std;

class Cars{

protected:
        string color;
        int year;
        int milage;

public:
        void setColor(string c){
        color = c;
        }

        void setYear(int y){
        year = y;
        }

        void setMilage(int m){
        milage = m;
        }

};


class Mazda: public Cars{

public:
    void driveOneThousand(){
    milage+=1000;
}

    void carInfo(){
    cout << "Year:" << year << endl;
    cout << "Color:" << color << endl;
    cout << "Milage:" << milage << endl;
}
};

int main(){
    Mazda BriansCar;
    BriansCar.setColor("black");
    BriansCar.setYear(2014);
    BriansCar.setMilage(0);
    BriansCar.driveOneThousand();

    BriansCar.carInfo();

    return 0;

}
```

The code above contains two classes, Cars and Mazda. Mazda is a child child of Cars, as the name would imply, since all Mazda vehicles are cars of some sort. This lab has introduced me on how to declare inherited classes. The key word to do so is class inherited_class: public parentClass{ }. Although the keyword public does not have to be necessarily public, but that's a little beyond of what we're trying to achieve here. The code above allows the Class Mazda to utilize the public methods within Cars and the variables in order to set the attributes and data for each specific Mazda car. What I have done is I have written up a Mazda object called BriansCar. Then I set the attributes and and then print them out. The .s file, which contains the x86 assembly for the code above ends up being several lines long; 480 to be exact. For the sake of keeping it brief, only key screen shots will be portrayed.



Within the main: portion of the assembly code, we see that data is being offset to make sure it accessed the right information. Once that has been done, it calls the Mazda portion of the x86 since it is setting all the values for the Mazda object. Then when looking at the Mazda bit (shown in the following screen shot) we can see that the registers are being prepared to store data, but the most important part is that the Cars portion of the assembly is then called in line

227. That then allows for the Mazda to access the Cars methods, which then set the values for the space allotted for the variables for Car objects and anything that inherits the class.




```
216 _ZN5MazdaC2Ev:
217 .LFB984:
218     .cfi_startproc
219     push    ebp
220     .cfi_def_cfa_offset 8
221     .cfi_offset 5, -8
222     mov ebp, esp
223     .cfi_def_cfa_register 5
224     sub esp, 24
225     mov eax, DWORD PTR [ebp+8]
226     mov DWORD PTR [esp], eax
227     call    _ZN4CarsC2Ev
228     leave
229     .cfi_restore 5
230     .cfi_def_cfa 4, 4
231     ret
232     .cfi_endproc
```

Following the call then takes you to the following snippet:

```
166 _ZN4CarsC2Ev:
167 .LFB979:
168     .cfi_startproc
169     push    ebp
170     .cfi_def_cfa_offset 8
171     .cfi_offset 5, -8
172     mov ebp, esp
173     .cfi_def_cfa_register 5
174     sub esp, 24
175     mov eax, DWORD PTR [ebp+8]
176     mov DWORD PTR [esp], eax
177     call    _ZNSsC1Ev
178     leave
179     .cfi_restore 5
180     .cfi_def_cfa 4, 4
181     ret
182     .cfi_endproc
```

It is interesting to point out that the methods are defined before the class comes up. But the methods make references to the Class, and since Mazda makes reference to the Cars bit above, it is able to then find the memory space where the methods and variables are defined. Something interesting to note os that the methods I defined are shown first and then the class name is then shown; meaning setYear, setColor, setMilage, driveOneThousand, etc, are defined first within assembly and the the snippet above is shown. Which is perhaps similar to how ebp works; it stacks things up backwards, showing the methods first and then the class references because that way it can keep track of how things are aligned and it knows where memory is since it is placed linearly on the stack.

Looking back, there are also 2 portions of the Cars class, the one above and the one shown on the next page. I cannot follow it perfectly well, but could one serve as a constructor of the class and the other the actual memory reference for the whole class.

```
    .type   _ZN4CarsD2Ev, @function
_ZN4CarsD2Ev:
.LFB982:
    .cfi_startproc
    push    ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    mov ebp, esp
    .cfi_def_cfa_register 5
    sub esp, 24
    mov eax, DWORD PTR [ebp+8]
    mov DWORD PTR [esp], eax
    call    _ZNSsD1Ev
    leave
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
.LFE982:
    .size   _ZN4CarsD2Ev, .-_ZN4CarsD2Ev
    .weak   _ZN4CarsD1Ev
    .set    _ZN4CarsD1Ev,_ZN4CarsD2Ev
    .section    .text._ZN5MazdaC2Ev,"axG",@progbits,_ZN5MazdaC5Ev,comdat
    .align 2
    .weak   _ZN5MazdaC2Ev
    .type   _ZN5MazdaC2Ev, @function
```

So that’s how x86 is placing the methods and makes calls to each within the code, but something important to point out as well is that the variables, since they are just static ones and they are being set within the subroutine for setter methods, they are treated like global variables as shown below:

```
    .section    .rodata
.LC3:
    .string "black"
    .text
    .globl  main
    .type   main, @function
```

“Black” here is a .string and it is also being set within the main portion of the code.

Information From the following sources:

http://en.wikipedia.org/wiki/Call_stack#FRAME-POINTER
http://en.wikipedia.org/wiki/X86
http://nongnu.askapache.com//pgubook/ProgrammingGroundUp-1-0-booksize.pdf