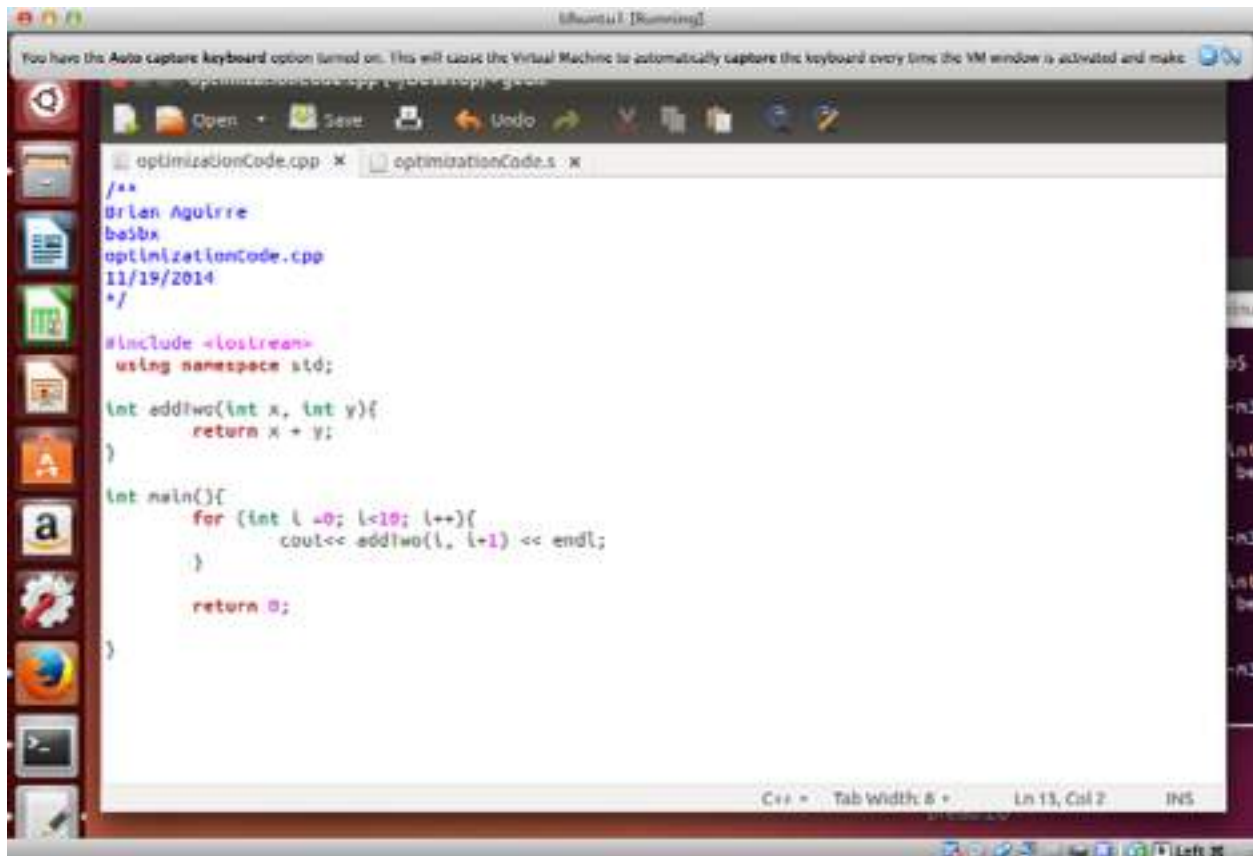


Brian Aguirre  
ba5bx  
postlab.pdf  
11/21/2014

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:



The screenshot shows a code editor window titled 'optimizationCode.cpp' with the following C++ code:

```
/**
 * Brian Aguirre
 * ba5bx
 * optimizationCode.cpp
 * 11/19/2014
 */

#include <iostream>
using namespace std;

int addTwo(int x, int y){
    return x + y;
}

int main(){
    for (int i = 0; i < 10; i++){
        cout << addTwo(i, i+1) << endl;
    }

    return 0;
}
```

The editor interface includes a toolbar with 'Open', 'Save', 'Undo', and 'Redo' buttons, and a status bar at the bottom indicating 'C++', 'Tab width: 8', 'Ln 13, Col 2', and 'INS'.

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
1  .file "optimizationCode.cpp"
2  .intel_syntax noprefix
3  .local _ZStL8_oinit
4  .comm _ZStL8_oinit,1,1
5  .text
6  .globl _Z6addTwoi1
7  .type _Z6addTwoi1, @function
8  _Z6addTwoi1:
9  .LFB971:
10  .cfi_startproc
11  push    ebp
12  .cfi_def_cfa_offset 8
13  .cfi_offset 5, -8
14  mov     ebp, esp
15  .cfi_def_cfa_register 5
16  mov     eax, DWORD PTR [ebp+12]
17  mov     edx, DWORD PTR [ebp+8]
18  add     eax, edx
19  pop     ebp
20  .cfi_restore 5
21  .cfi_def_cfa 4, 4
22  ret
23  .cfi_endproc
24  .LFE971:
25  .size _Z6addTwoi1, .-_Z6addTwoi1
26  .globl main
27  .type main, @function
28  main:
29  .LFB972:
30  .cfi_startproc
31  push    ebp
32  .cfi_def_cfa_offset 8
33  .cfi_offset 5, -8
34  mov     ebp, esp
```

Line 1, Column 1      Tab Size: 4      R

```
optimizationCode.s  UNREGISTERED
optimizationCode.cpp  optimizationCode.s  optimizationCode-O2.s
34  mov     ebp, esp
35  .cfi_def_cfa_register 5
36  and     esp, -16
37  sub     esp, 32
38  mov     DWORD PTR [esp+28], 0
39  jmp     .L4
40  .L5:
41  mov     eax, DWORD PTR [esp+28]
42  add     eax, 1
43  mov     DWORD PTR [esp+4], eax
44  mov     eax, DWORD PTR [esp+28]
45  mov     DWORD PTR [esp], eax
46  call    _Z6addTwoi1
47  mov     DWORD PTR [esp+4], eax
48  mov     DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
49  call    _ZNSolsEi
50  mov     DWORD PTR [esp+4], OFFSET FLAT:_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostream
51  mov     DWORD PTR [esp], eax
52  call    _ZNSolsEPFRSoS_E
53  add     DWORD PTR [esp+28], 1
54  .L4:
55  cmp     DWORD PTR [esp+28], 9
56  jle     .L5
57  mov     eax, 0
58  leave
59  .cfi_restore 5
60  .cfi_def_cfa 4, 4
61  ret
62  .cfi_endproc
63  .LFE972:
64  .size main, .-main
65  .type _Z41_static_initialization_and_destruction_0i1, @function
66  _Z41_static_initialization_and_destruction_0i1:
67  .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_01i:
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_01i, .-_Z41_static_initializat
93  .type   _GLOBAL__sub_I_Z6addTwoi, @function
94  _GLOBAL__sub_I_Z6addTwoi:
95  .LFB979:
96  .cfi_startproc
97  push    ebp
98  .cfi_def_cfa_offset 8
99  .cfi_offset 5, -8
```

Line 1, Column 1      Tab Size: 4      R

```
optimizationCode-O2.s  UNREGISTERED
optimizationCode.cpp  optimizationCode.s  optimizationCode-O2.s
1  .file "optimizationCode.cpp"
2  .intel_syntax noprefix
3  .text
4  .p2align 4,,15
5  .globl _Z6addTwoi
6  .type   _Z6addTwoi, @function
7  _Z6addTwoi:
8  .LFB998:
9  .cfi_startproc
10 mov     eax, DWORD PTR [esp+8]
11 add     eax, DWORD PTR [esp+4]
12 ret
13 .cfi_endproc
14 .LFE998:
15 .size   _Z6addTwoi, .-_Z6addTwoi
16 .section .text.startup,"ax",@progbits
17 .p2align 4,,15
18 .globl main
19 .type   main, @function
20 main:
21 .LFB999:
22 .cfi_startproc
23 push    ebp
24 .cfi_def_cfa_offset 8
25 .cfi_offset 5, -8
26 mov     ebp, esp
27 .cfi_def_cfa_register 5
28 push    edi
29 push    esi
30 .cfi_offset 7, -12
31 .cfi_offset 6, -16
32 mov     esi, 1
33 push    ebx
34 and     esp, -16
```

Line 119, Column 45      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

optimizationCode-O2.s UNREGISTERED

optimizationCode.cpp optimizationCode.s optimizationCode-O2.s

```

71     .p2align 4,,7
72     .p2align 3
73 .L10:
74     lea     esp, [ebp-12]
75     xor     eax, eax
76     pop     ebx
77     .cfi_remember_state
78     .cfi_restore 3
79     pop     esi
80     .cfi_restore 6
81     pop     edi
82     .cfi_restore 7
83     pop     ebp
84     .cfi_restore 5
85     .cfi_def_cfa 4, 4
86     ret
87 .L11:
88     .cfi_restore_state
89     call    _ZSt16__throw_bad_castv
90     .cfi_endproc
91 .LFE999:
92     .size main, .- main
93     .p2align 4,,15
94     .type _GLOBAL__sub_I_Z6addTwoii, @function
95 _GLOBAL__sub_I_Z6addTwoii:
96 .LFB1006:
97     .cfi_startproc
98     sub     esp, 28
99     .cfi_def_cfa_offset 32
100    mov     DWORD PTR [esp], OFFSET FLAT:_ZStL8__ioinit
101    call    _ZNSt8ios_base4InitC1Ev
102    mov     DWORD PTR [esp+8], OFFSET FLAT:__dso_handle
103    mov     DWORD PTR [esp+4], OFFSET FLAT:_ZStL8__ioinit
104    mov     DWORD PTR [esp], OFFSET FLAT:_ZNSt8ios_base4InitC1Ev

```

Line 119, Column 45 Tab Size: 4 R



```
optimizationCode-O2.s  UNREGISTERED
optimizationCode.cpp  optimizationCode.s  optimizationCode-O2.s
86     ret
87 .L11:
88     .cfi_restore_state
89     call __ZSt16__throw_bad_castv
90     .cfi_endproc
91 .LFE999:
92     .size main, . - main
93     .p2align 4,,15
94     .type __GLOBAL_sub_I__Z6addTwoi1, @function
95 __GLOBAL_sub_I__Z6addTwoi1:
96 .LFB1006:
97     .cfi_startproc
98     sub esp, 28
99     .cfi_def_cfa_offset 32
100    mov DWORD PTR [esp], OFFSET FLAT: __ZStL8__ioinit
101    call __ZNSt8ios_base4InitC1Ev
102    mov DWORD PTR [esp+8], OFFSET FLAT: __dso_handle
103    mov DWORD PTR [esp+4], OFFSET FLAT: __ZStL8__ioinit
104    mov DWORD PTR [esp], OFFSET FLAT: __ZNSt8ios_base4InitD1Ev
105    call __cxa_atexit
106    add esp, 28
107    .cfi_def_cfa_offset 4
108    ret
109    .cfi_endproc
110 .LFE1006:
111     .size __GLOBAL_sub_I__Z6addTwoi1, . - __GLOBAL_sub_I__Z6addTwoi1
112     .section .init_array,"aw"
113     .align 4
114     .long __GLOBAL_sub_I__Z6addTwoi1
115     .local __ZStL8__ioinit
116     .comm __ZStL8__ioinit,1,1
117     .hidden __dso_handle
118     .ident "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
119     .section .note.GNU-stack,"",@progbits
```

Line 119, Column 45      Tab Size: 4      R

```
optimizationCode-O2.s  UNREGISTERED
optimizationCode.cpp  optimizationCode.s  optimizationCode-O2.s
37     jmp .L7
38     .p2align 4,,7
39     .p2align 3
40 .L12:
41     movzx eax, BYTE PTR [ebx+39]
42 .L5:
43     movsx eax, al
44     add esi, 2
45     mov DWORD PTR [esp+4], eax
46     mov DWORD PTR [esp], edi
47     call __ZN5o3putEc
48     mov DWORD PTR [esp], eax
49     call __ZN5o5flushEv
50     cmp esi, 21
51     je .L10
52 .L7:
53     mov DWORD PTR [esp+4], esi
54     mov DWORD PTR [esp], OFFSET FLAT: __ZSt4cout
55     call __ZN5o5sEi
56     mov edi, eax
57     mov eax, DWORD PTR [eax]
58     mov eax, DWORD PTR [eax-12]
59     mov ebx, DWORD PTR [edi+124+eax]
60     test ebx, ebx
61     je .L11
62     cmp BYTE PTR [ebx+28], 0
63     jne .L12
64     mov DWORD PTR [esp], ebx
65     call __ZNKSt5ctypeIcE13_M_widen_initEv
66     mov eax, DWORD PTR [ebx]
67     mov DWORD PTR [esp+4], 10
68     mov DWORD PTR [esp], ebx
69     call [DWORD PTR [eax+24]]
70     jmp .L5
```

Line 119, Column 45      Tab Size: 4      R

Something I immediately noticed 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 faster 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 its 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 its 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/cpsc411/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:

A screenshot of a C++ code editor window titled 'inheritance.cpp'. The code defines a base class 'Cars' and a derived class 'Mazda' which inherits from 'Cars'. The 'Cars' class has protected attributes 'color', 'year', and 'milage', and public methods 'setColor', 'setYear', and 'setMilage'. The 'Mazda' class inherits from 'Cars' and has a public method 'driveOneThousand' that increments 'milage' by 1000, and a 'carInfo' method that prints the car's details. The 'main' function creates a 'Mazda' object, sets its color to 'black', year to 2014, and milage to 0, then calls 'driveOneThousand' and 'carInfo' to demonstrate the inheritance and object behavior.

```
1  /*
2  Brian Aguirre
3  ba5bx
4  inheritance.cpp
5  11/21/2014
6  */
7
8  #include <iostream>
9  #include <string>
10
11 using namespace std;
12
13 class Cars{
14
15 protected:
16     string color;
17     int year;
18     int milage;
19
20 public:
21     void setColor(string c){
22         color = c;
23     }
24
25     void setYear(int y){
26         year = y;
27     }
28
29     void setMilage(int m){
30         milage = m;
31     }
32
33 },:
34
35
36 class Mazda: public Cars{
37
38 public:
39     void driveOneThousand(){
40         milage+=1000;
41     }
42
43     void carInfo(){
44         cout << "Year:" << year << endl;
45         cout << "Color:" << color << endl;
46         cout << "Milage:" << milage << endl;
47     }
48 };
49
50 int main(){
51     Mazda BriansCar;
52     BriansCar.setColor("black");
53     BriansCar.setYear(2014);
54     BriansCar.setMilage(0);
55     BriansCar.driveOneThousand();
56
57     BriansCar.carInfo();
58
59     return 0;
60 }
61
62
```

Line 1, Column 1 Tab Size: 4

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.

```

267  .type    main, @function
268  main:
269  .LFB976:
270  .cfi_startproc
271  .cfi_personality 0, __gxx_personality_v0
272  .cfi_lsda 0, .LLS0A976
273  push    ebp
274  .cfi_def_cfa_offset 8
275  .cfi_offset 5, -8
276  mov     ebp, esp
277  .cfi_def_cfa_register 5
278  push    ebx
279  and     esp, -16
280  sub     esp, 48
281  .cfi_offset 3, -12
282  lea     eax, [esp+36]
283  mov     DWORD PTR [esp], eax
284  .LEH00:
285  call    _ZN9MazdaC1Ev
286  .LEH01:
287  lea     eax, [esp+31]
288  mov     DWORD PTR [esp], eax
289  call    _ZN5a1cEC1Ev
290  lea     eax, [esp+31]
291  mov     DWORD PTR [esp+8], eax
292  mov     DWORD PTR [esp+4], OFFSET FLAT:_LC3
293  lea     eax, [esp+32]
294  mov     DWORD PTR [esp], eax
295  .LEH01:
296  call    _ZN5sC1EPKcRK5a1cE
297  .LEH01:
298  lea     eax, [esp+32]
299  mov     DWORD PTR [esp+4], eax
300  lea     eax, [esp+36]
301  mov     DWORD PTR [esp], eax
302  .LEH02:
303  call    _ZN4CarsBsetColorE5s
304  .LEH02:
305  lea     eax, [esp+32]
306  mov     DWORD PTR [esp], eax
307  .LEH03:
308  call    _ZN5sD1Ev
309  .LEH03:
310  lea     eax, [esp+31]

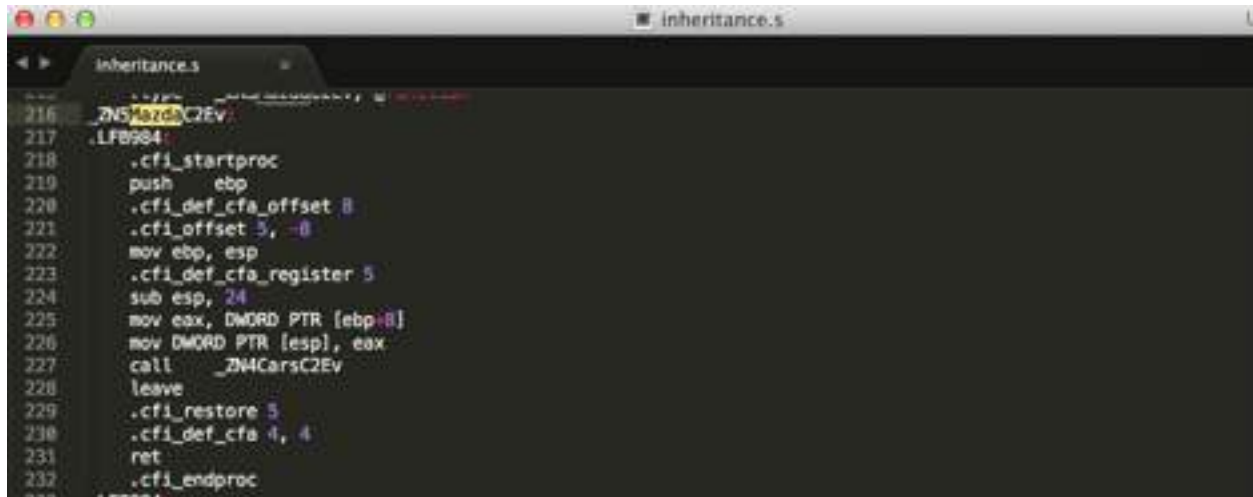
```

1 of 10 matches

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.

A screenshot of a debugger window titled 'inheritance.s'. The assembly code is displayed with line numbers 216 through 232. Line 216 is highlighted, showing the instruction 'call \_ZNS4CarsC2Ev'. The code includes standard x86-64 assembly instructions for stack frame setup and cleanup, such as 'push ebp', 'mov ebp, esp', and 'ret'.

```
216  _ZNS4CarsC2Ev:
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    _ZNS4CarsC2Ev
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:

A screenshot of a debugger window showing assembly code for the function '\_ZNS4CarsC2Ev'. The code is displayed with line numbers 166 through 182. Line 177 is highlighted, showing the instruction 'call \_ZNS4CarsC2Ev'. The code includes standard x86-64 assembly instructions for stack frame setup and cleanup, such as 'push ebp', 'mov ebp, esp', and 'ret'.

```
166  _ZNS4CarsC2Ev:
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    _ZNS4CarsC2Ev
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 is 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    _ZN5sD1Ev
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/Call_stack#FRAME-POINTER)

<http://en.wikipedia.org/wiki/X86>

<http://nongnu.askapache.com/pgubook/ProgrammingGroundUp-1-0-booksize.pdf>