In Lab Report

Dynamic Dispatch

As we have discussed in class, the nature of dynamic dispatch makes the complier does not know which member function to invoke until it generates the code. Before runtime, all the virtual function's address are stored inside a virtual method table waiting to be invoked, and the object, which contains the pointer to the virtual method table shall call the pointer to finish the implementation.

In the snippet I created, I built two objects a, and b which both contains two virtual functions ret() and ret1(). However, these two have different behaviors. In the main function, I created a "a1" object with type a and "a2" object with type b. Later I invoked the two member functions (ret() and ret1()) inside two objects.

From the assembly code complied, I found that after the assembly has invoked the function to create two objects respectively, it then move the DWORD PTR [esp+24] to the register eax. As mentioned before, the address toward the virtual method table. Hence moving the pointer at esp+24 shall bring us to the virtual method table. Then it takes the pointer address of eax which is the address in the virtual

method table that pointing toward the actual member function. In the end, it will take the address of the member function "mov eax, DWORD PTR [eax]" and call the function "call eax". When invoking the second function inside the object, it will do the same operation as before but when reaching the virtual method table, it will add "4" to the register eax since all member functions inside the virtual method table are located in order by 4 byte difference. When it tried to call a different object, it will then do the same process again with a virtual method table address stored in the object

```
#include <iostream>

#include <iostream>

using namespace std;

class a{
  public:
    virtual void ret() const {cout<<"I am a"<<endl;}
    virtual ~a(){}

;

class b:public a{
  public:
    virtual void ret() const {cout<<"I am b"<<endl;}

virtual ~a(){}

};

class b:public a{
  public:
    virtual void ret() const{cout<<"I am b"<<endl;}

virtual void ret() const {cout<<"I am b"<<endl;}

;

int main(){
    a *a1 = new a;
    a *a2 = new b;

a1->ret();
    a1->ret();
    a2->ret1();
    return 0;
}
```

```
mov DWORD PTR [esp], ebx
          call
                   ZN1aC1Ev
          mov DWORD PTR [esp+24], ebx
248
          mov DWORD PTR [esp], 4
249
                   _Znwj
          call
          mov ebx, eax
          mov DWORD PTR [esp], ebx
          call
                   _ZN1bC1Ev
          mov DWORD PTR [esp+28], ebx
          mov eax, DWORD PTR [esp+24]
          mov eax, DWORD PTR
                              [eax]
          mov eax, DWORD PTR
                              [eax]
          mov edx, DWORD PTR [esp+24]
259
          mov DWORD PTR [esp], edx
          call
          mov eax, DWORD PTR [esp+24]
          mov eax, DWORD PTR [eax]
263
          add eax, 4
          mov eax, DWORD PTR [eax]
264
          mov edx, DWORD PTR [esp+24]
          mov DWORD PTR [esp], edx
67
          call
          mov eax, DWORD PTR [esp+28]
          mov eax, DWORD PTR [eax]
70
          mov eax, DWORD PTR [eax]
          mov edx, DWORD PTR [esp+28]
          mov DWORD PTR [esp], edx
          call
                   eax
74
          mov eax, DWORD PTR [esp+28]
          mov eax, DWORD PTR [eax]
276
          add eax, 4
              eax, DWORD PTR [eax]
78
          mov edx, DWORD PTR [esp+28]
          mov DWORD PTR [esp], edx
79
```

(DWORD PTR [esp+24] for object a and DWORD PTR [esp+28] for object b]). As we have mentioned before, the calling process for dynamic dispatch is different from the previous method calling techniques since the method's name is never shown but represented by a pointer instead. Since the complier will not know which function to invoker, it is important to use the virtual method table to point toward the actual method and connect with the actual object or it will be impossible for the code to know which method to invoke in the end.

Optimization

Compiling with -O2 flag shall give us an optimized version of the assembly code. In the second snippet I created, I complied and generates its assembly with and without -O2 flag and then compare the result to see the specific optimizations it used.

The first optimizations I saw by comparison is the usage of registers instead of pointers. In the main functions, the unoptimized code on the left, it uses a lot of DWORD PTR of esp in order to perform operations. On the optimized code on the right; however, it pushes a new register ebx at the beginning of the code and uses it to throughout the assembly code to access variables and calculate. Similar case applied in the triple function where the

optimized code used register esi at the beginning while the unoptimized code did not. Maximizing the register usage will help to accelerate the runtime of the program.

Secondly, the code uses a lot of complex instructions to simplify the amount of instructions to perform one certain operation. In the triple function section line18 to 20, the original code was asked to do the additions with three registers. In the optimized code, it was simplified into one command "lea [eax+eax*2]" Merging three commands into one line will help the process of optimization.

As mentioned before, using "lea" allows "multiple math operations all in one

```
#include <iostream>
#
```

```
.cfi_startproc
                                                                                                                     .cfi_startproc
             push ebp
.cfi_def_cfa_offset 8
                                                                                                                     push ebp
.cfi_def_cfa_offset 8
                                                                                                        72
73
74
75
76
77
78
80
81
82
83
84
85
86
87
89
90
91
             .cfi_offset 5, -8
                                                                                                                     .cfi_offset 5, -8
             mov ebp, esp
.cfi_def_cfa_register 5
                                                                                                                     mov ebp, esp
.cfi_def_cfa_register 5
                                                                                                                     push ebx
.cfi_offset 3, -12
             and esp, -16

sub esp, 32

mov DWORD PTR [esp+28], 0
                                                                                                                     and esp, -16
sub esp, 16
             jmp .L3
            mov eax, DWORD PTR [esp+28]
mov DWORD PTR [esp], eax
call _Z6triplet
add DWORD PTR [esp+28], 1
                                                                                                                     .p2align 4,,7 .p2align 3
                                                                                                                     mov DWORD PTR [esp], ebx
                                                                                                                     add ebx, 1
call _Z6triplei
cmp ebx, 10
jne .L10
             cmp DWORD PTR [esp+28], 9
             jle .L4
             mov eax, 0
       leave
                                                                                                                     xor eax, eax
                                                                                                                      mov ebx, DWORD PTR [ebp-4]
             .cfi_def_cfa 4, 4
                                                                                                                     .cfi_restore 5
             .cfi_endproc
                                                                                                                     .cfi_restore 3
.cfi_def_cfa 4, 4
                                                                                                                     ret _____.cfi_endproc
             .type _Z41__static_initialization_and_destruction_0ii, @fun
       98
99
100
68
69
70
71
                                                                                                                     .p2align 4,,15
                                                                                                                .type _GLOBAL_sub_I_Z6triplei, @function
_GLOBAL_sub_I_Z6triplei:
             .cfi_startproc
             push ebp
.cfi_def_cfa_offset 8
```

Illustration 1: Main Function

instruction and it does not affect the flags register so you can put in in between one register being modified and a flags comparison jump on the next line."

In the triplefunction section, the optimized code also uses a command called "movzx". After

On the other hand, it also uses a lot of "xor" commands in the main function to make the register zero. It is much quicker comparing to "and" operation and it is also a good way to clear the dependencies on registers.

```
.globl _Z6triplei
_type _Z6triplei, @function
_Z6triplei:
                                                                                                                                                                                                             push esi
.cfi_def_cfa_offset 8
.cfi_offset 6, -8
              push ebp
.cfi_def_cfa_offset 8
                                                                                                                                                                                                               push ebx
.cfi_def_cfa_offset 12
.cfi_offset 3, -12
              mov ebp, esp .
.cfi_def_cfa_register 5
.cfi_def_cfa_register 5
sub esp, 24
mov edx, DMORD PTR [ebp+8]
mov eax, edx
add eax, eax
add eax, edx
mov DWORD PTR [ebp+8], eax
mov DWORD PTR [ebp+8]
mov DWORD PTR [esp+4], eax
mov DWORD PTR [esp+4], eax
mov DWORD PTR [esp], oFFSET FLAT:_ZSt4cout
call _ZMSolsEi
mov DWORD PTR [esp+4], OFFSET FLAT:_ZSt4endlicSt11char_traits>
siceErost13basic_ostreamIT_TO_ES6_
mov DWORD PTR [esp], eax
call _ZNSolsEPFRSoS_E
leave
                                                                                                                                                                                                             mov eax, DWORD PTR [esp+32]
mov DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
lea eax, [eax+eax*2]
mov DWORD PTR [esp+4], eax
                                                                                                                                                                                                            call __ZNSolsEi
mov est, eax
mov eax, DWORD PTR [eax]
mov eax, DWORD PTR [eax-12]
mov ebx, DWORD PTR [esi+124+eax]
test ebx ebx
                                                                                                                                                                                                              test ebx, ebx
je .L7
cmp BYTE PTR [ebx+28], 0
                                                                                                                                                                                                              je .L3
movzx eax, BYTE PTR [ebx+39]
              leave
.cfi_restore 5
.cfi_def_cfa 4, 4
               ret ____
.cfi_endproc
                                                                                                                                                                                                              mov DWORD PTR [esp], esi
mov DWORD PTR [esp+4], eax
                                                                                                                                                                                                               .971:
.size _Z6triplei, .-_Z6triplei
.globl main
.type main, @function
               .cfi_startproc
                                                                                                                                                                                                                         _restore 3
```

Illustration 2: Triple Function

On optimization that I did expect but did not show up was the unrolling loops. In general, the code optimization shall make unroll the loop in order to accelerate. However, it is possible that the code is too small so the optimization did not touch on that area.

Reference:

1. Assembly Optimization Tips, Mark Larson, 04/14/16 http://mark.masmcode.com/