// Dion Niazi dn3gy 11 04 2017 inlab9.pdf
2) Optimized Code

The C++ program I created was intended to show the optimized assembly code vs the non-optimized assembly code. The C++ program is called optimized.cpp and it has a for loop and calls the abs() function, to see the difference between the optimized and non-optimized code. The below code is my C++ code.

```
#include <iostream>
#include <stdlib.h>
using namespace std;
int main()
{
  int a = 1;
  for(int i =0;i<10;i++)
    {
      a-=a*i;
      a+=abs(a);
    }
  cout<< a<<endl;
  return 0;
}
```

Below is my non-optimized code.

```
        .text
        .intel_syntax noprefix
        .file   "optimized.cpp"
        .section        .text.startup,"ax",@progbits
        .p2align        4, 0x90
        .type   __cxx_global_var_init,@function
__cxx_global_var_init:                  # @__cxx_global_var_init
        .cfi_startproc
# BB#0:
        push    rbp
.Ltmp0:
    .cfi_def_cfa_offset 16
.Ltmp1:
        .cfi_offset rbp, -16
        mov     rbp, rsp
.Ltmp2:
        .cfi_def_cfa_register rbp
        sub     rsp, 16
        movabs  rdi, _ZStL8__ioinit
        call    _ZNSt8ios_base4InitC1Ev
        movabs  rdi, _ZNSt8ios_base4InitD1Ev
        movabs  rsi, _ZStL8__ioinit
        movabs  rdx, __dso_handle
```

```
        call    __cxa_atexit
        mov     dword ptr [rbp - 4], eax # 4-byte Spill
        add     rsp, 16
        pop     rbp
        ret
.Lfunc_end0:
        .size   __cxx_global_var_init, .Lfunc_end0-
__cxx_global_var_init
        .cfi_endproc
.text
        .globl  main
        .p2align        4, 0x90
        .type   main,@function
main:                                   # @main
        .cfi_startproc
# BB#0:
        push    rbp
.Ltmp3:
        .cfi_def_cfa_offset 16
.Ltmp4:
.cfi_offset rbp, -16
        mov     rbp, rsp
.Ltmp5:
        .cfi_def_cfa_register rbp
        sub     rsp, 32
        mov     dword ptr [rbp - 4], 0
        mov     dword ptr [rbp - 8], 1
        mov     dword ptr [rbp - 12], 0
.LBB1_1:                                # =>This Inner Loop
Header: Depth=1
        cmp     dword ptr [rbp - 12], 10
        jge     .LBB1_4
# BB#2:                                 #   in Loop: Header=BB1_1
Depth=1
        mov     eax, dword ptr [rbp - 8]
        imul    eax, dword ptr [rbp - 12]
        mov     ecx, dword ptr [rbp - 8]
        sub     ecx, eax
        mov     dword ptr [rbp - 8], ecx
        mov     edi, dword ptr [rbp - 8]
        call    abs
        add     eax, dword ptr [rbp - 8]
        mov     dword ptr [rbp - 8], eax
# BB#3:                                 #   in Loop: Header=BB1_1
Depth=1
mov     eax, dword ptr [rbp - 12]
        add     eax, 1
        mov     dword ptr [rbp - 12], eax
        jmp     .LBB1_1
.LBB1_4:
        movabs  rdi, _ZSt4cout
        mov     esi, dword ptr [rbp - 8]
```

```
        call    _ZNSolsEi
        movabs  rsi,
_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
        mov     rdi, rax
        call    _ZNSolsEPFRSoS_E
        xor     ecx, ecx
        mov     qword ptr [rbp - 24], rax # 8-byte Spill
        mov     eax, ecx
        add     rsp, 32
        pop     rbp
        ret
.Lfunc_end1:
        .size   main, .Lfunc_end1-main
        .cfi_endproc

        .section        .text.startup,"ax",@progbits
        .p2align        4, 0x90
        .type   _GLOBAL__sub_I_optimized.cpp,@function
_GLOBAL__sub_I_optimized.cpp:           #
@_GLOBAL__sub_I_optimized.cpp
        .cfi_startproc
# BB#0:
        push    rbp
.Ltmp6:
        .cfi_def_cfa_offset 16
.Ltmp7:
        .cfi_offset rbp, -16
        mov     rbp, rsp
.Ltmp8:
        .cfi_def_cfa_register rbp
        call    __cxx_global_var_init
        pop     rbp
        ret
.Lfunc_end2:
        .size   _GLOBAL__sub_I_optimized.cpp, .Lfunc_end2-
_GLOBAL__sub_I_optimi\
zed.cpp
        .cfi_endproc

        .type   _ZStL8__ioinit,@object  # @_ZStL8__ioinit
        .local  _ZStL8__ioinit
        .comm   _ZStL8__ioinit,1,1
        .section        .init_array,"aw",@init_array
        .p2align        3
        .quad   _GLOBAL__sub_I_optimized.cpp

        .ident  "clang version 3.9.1-svn288847-1~exp1
(branches/release_39)"
        .section        ".note.GNU-stack","",@progbits
```

The code below is the optimized assembly code using the –O2 flag. I also changed the color of the code to blue to make it easier to look at.

```
.text
        .intel_syntax noprefix
        .file   "optimized.cpp"
        .globl  main
        .p2align        4, 0x90
        .type   main,@function
main:                                   # @main
        .cfi_startproc
# BB#0:
        push    r14
.Ltmp0:
        .cfi_def_cfa_offset 16
        push    rbx
.Ltmp1:
        .cfi_def_cfa_offset 24
        push    rax
.Ltmp2:
        .cfi_def_cfa_offset 32
.Ltmp3:
        .cfi_offset rbx, -24
.Ltmp4:
        .cfi_offset r14, -16
        mov     edi, _ZSt4cout
        xor     esi, esi
        call    _ZNSolsEi
        mov     r14, rax
        mov     rax, qword ptr [r14]
        mov     rax, qword ptr [rax - 24]
        mov     rbx, qword ptr [r14 + rax + 240]
        test    rbx, rbx
        je      .LBB0_5
# BB#1:
        cmp     byte ptr [rbx + 56], 0
        je      .LBB0_3
# BB#2:
        mov     al, byte ptr [rbx + 67]
        jmp     .LBB0_4
.LBB0_3:
        mov     rdi, rbx
        call    _ZNKSt5ctypeIcE13_M_widen_initEv
        mov     rax, qword ptr [rbx]
        mov     esi, 10
        mov     rdi, rbx
        call    qword ptr [rax + 48]
```

```
.LBB0_4:
        movsx   esi, al
        mov     rdi, r14
        call    _ZNSo3putEc
        mov     rdi, rax
        call    _ZNSo5flushEv
        xor     eax, eax
        add     rsp, 8
        pop     rbx
        pop     r14
        ret
.LBB0_5:
        call    _ZSt16__throw_bad_castv
.Lfunc_end0:
        .size   main, .Lfunc_end0-main
        .cfi_endproc

        .section        .text.startup,"ax",@progbits
        .p2align        4, 0x90
        .type   _GLOBAL__sub_I_optimized.cpp,@function
_GLOBAL__sub_I_optimized.cpp:           #
@_GLOBAL__sub_I_optimized.cpp
        .cfi_startproc
# BB#0:
        push    rax
.Ltmp5:
        .cfi_def_cfa_offset 16
        mov     edi, _ZStL8__ioinit
        call    _ZNSt8ios_base4InitC1Ev
        mov     edi, _ZNSt8ios_base4InitD1Ev
        mov     esi, _ZStL8__ioinit
        mov     edx, __dso_handle
        pop     rax
        jmp     __cxa_atexit            # TAILCALL
.Lfunc_end1:
        .size   _GLOBAL__sub_I_optimized.cpp, .Lfunc_end1-
_GLOBAL__sub_I_optimi\
zed.cpp
        .cfi_endproc

        .type   _ZStL8__ioinit,@object  # @_ZStL8__ioinit
        .local  _ZStL8__ioinit
        .comm   _ZStL8__ioinit,1,1
        .section        .init_array,"aw",@init_array
        .p2align        3
        .quad   _GLOBAL__sub_I_optimized.cpp
        .ident  "clang version 3.9.1-svn288847-1~exp1
(branches/release_39)"
```

```
        .section        ".note.GNU-stack","",@progbits
```

First thing I noticed was that the optimized code had less lines of code than the non-optimized one (90 lines to 116 lines). I'm guessing because the code is being optimized it has to have less lines of code to reduce the time being spent on looking at more instructions, while at the same time do the same instructions as the non-optimized code.

Second thing I noticed was that the optimized code used more of registers like r14 than using rsp and rbp and offsets. Using more registers seem to be the realistic approach to increasing speed overall because of faster access.

The third thing that seemed different was the call of the loop and the function in both optimized and non-optimized code. The optimized code doesn't seem to call the loop anywhere if I'm not mistaken and I definitely don't see a call to the abs() function. I guess the loop may be called, but any code that was in the non-optimized code that involved the loops and was not really required was not included in the optimized code and the abs() function wasn't called because I believe it really wasn't affecting my program at all because all the numbers I passed in were positive, so it got rid of it.

The fourth thing that seemed different between the optimized and non-optimized code is that the optimized code used only qword and the non-optimized used both qword and dword. I don't know why using qword instead of dword helps optimize code because qword uses more bits than dword and that may affect memory, so I really don't understand why qword only shows up in the optimized code.

Dynamic Dispatch

Dynamic Dispatch is the decision on which member function to invoke during runtime of an object. In my C++ file I included both classes I had created and the main subroutine so that I could look more into dynamic dispatch. I really don't know a lot about assembly and dynamic dispatch is still a fuzzy subject for me, but I will try to explain it the best I can. The code I got, which was an example of dynamic dispatch, is shown below.

```
class A {
public:
   virtual void f();
   ...
};
```

```
class B: public A {
public:
   virtual void f();
...
};
```

```
int main()
```

```
{
   A *pa = new B();

   pa->f();
   ...
}
```
*Source 1 used for the code

So the question is, how does the complier know where the f() function is positioned at and I believe it is by using a virtual table as a map to the member function to the memory offset. I believe when in the assembly code, one line that calls the function, mem_set, is where the memory offset is occurring. The virtual keyword implemented in C++makes dynamic dispatch occur, without it the compiler wouldn't know what to do. There is a lot of things going on in assembly that I really don't understand is going on, but seems to mov a lot of registers around and call a bunch of functions with random names. Also it's noted about reloading a register which maybe about loading the memory offsetted in the virtual table.[2] The assembly code is shown below.

```
        .section
__TEXT,__text,regular,pure_instructions
        .macosx_version_min 10, 12
        .intel_syntax noprefix
        .globl  __ZN1A1fEv
        .align  4, 0x90
__ZN1A1fEv:                             ## @_ZN1A1fEv
        .cfi_startproc
## BB#0:
        push    rbp
Ltmp0:
        .cfi_def_cfa_offset 16
Ltmp1:
    .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp2:
        .cfi_def_cfa_register rbp
        mov     qword ptr [rbp - 8], rdi
        pop     rbp
        ret
        .cfi_endproc
.globl  __ZN1B1fEv
        .align  4, 0x90
__ZN1B1fEv:                             ## @_ZN1B1fEv
        .cfi_startproc
```

```
## BB#0:
        push    rbp
Ltmp3:
        .cfi_def_cfa_offset 16
Ltmp4:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp5:
        .cfi_def_cfa_register rbp
        mov     qword ptr [rbp - 8], rdi
        pop     rbp
        ret
        .cfi_endproc

        .globl  _main
        .align  4, 0x90
_main:                                  ## @main
  .cfi_startproc
## BB#0:
        push    rbp
Ltmp6:
        .cfi_def_cfa_offset 16
Ltmp7:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp8:
        .cfi_def_cfa_register rbp
        sub     rsp, 32
        mov     eax, 8
        mov     edi, eax
        mov     dword ptr [rbp - 4], 0
        call    __Znwm
        xor     esi, esi
        mov     ecx, 8
        mov     edx, ecx
        mov     rdi, rax
        mov     qword ptr [rbp - 24], rax ## 8-byte Spill
        call    _memset
   mov     rdi, qword ptr [rbp - 24] ## 8-byte Reload
        call    __ZN1BC1Ev
        mov     rax, qword ptr [rbp - 24] ## 8-byte Reload
        mov     qword ptr [rbp - 16], rax
        mov     rax, qword ptr [rbp - 16]
        mov     rdx, qword ptr [rax]
        mov     rdi, rax
        call    qword ptr [rdx]
        xor     eax, eax
        add     rsp, 32
```

```
        pop     rbp
        ret
        .cfi_endproc

        .globl  __ZN1BC1Ev
        .weak_def_can_be_hidden __ZN1BC1Ev
        .align  4, 0x90
__ZN1BC1Ev:                             ## @_ZN1BC1Ev
        .cfi_startproc
## BB#0:
        push    rbp
       .cfi_def_cfa_offset 16
Ltmp10:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp11:
        .cfi_def_cfa_register rbp
        sub     rsp, 16
        mov     qword ptr [rbp - 8], rdi
        mov     rdi, qword ptr [rbp - 8]
        call    __ZN1BC2Ev
        add     rsp, 16
        pop     rbp
        ret
        .cfi_endproc

        .globl  __ZN1BC2Ev
        .weak_def_can_be_hidden __ZN1BC2Ev
        .align  4, 0x90
__ZN1BC2Ev:                             ## @_ZN1BC2Ev
        .cfi_startproc
## BB#0:
        push    rbp
Ltmp12:
        .cfi_def_cfa_offset 16
Ltmp13:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp14:
        .cfi_def_cfa_register rbp
        sub     rsp, 16
        mov     qword ptr [rbp - 8], rdi
        mov     rdi, qword ptr [rbp - 8]
        mov     rax, rdi
        mov     qword ptr [rbp - 16], rdi ## 8-byte Spill
        mov     rdi, rax
        call    __ZN1AC2Ev
        lea     rax, [rip + __ZTV1B]
```

```
        add     rax, 16
        mov     rdi, qword ptr [rbp - 16] ## 8-byte Reload
        mov     qword ptr [rdi], rax
        add     rsp, 16
        pop     rbp
        ret
        .cfi_endproc
    .globl  __ZN1AC2Ev
        .weak_def_can_be_hidden __ZN1AC2Ev
        .align  4, 0x90
__ZN1AC2Ev:                             ## @_ZN1AC2Ev
        .cfi_startproc
## BB#0:
        push    rbp
Ltmp15:
        .cfi_def_cfa_offset 16
Ltmp16:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp17:
        .cfi_def_cfa_register rbp
        lea     rax, [rip + __ZTV1A]
        add     rax, 16
        mov     qword ptr [rbp - 8], rdi
        mov     rdi, qword ptr [rbp - 8]
        mov     qword ptr [rdi], rax
        pop     rbp
        ret
```

Sources

[1]http://condor.depaul.edu/ichu/csc447/notes/wk10/Dynamic2.htm

[2]http://m-hewedy.blogspot.com/2011/09/dynamic-binding-dispatch-by-example.html