Fr2md Farid Rajabi Nia 11am

Optimized code: Compare code generated normally to optimized code. To create optimized code, you will need to use the -O2 compiler flag. Can you make any guesses as to why the optimized code looks as it does? What is being optimized? Be sure to show your original sample code as well as the optimized version. Try loops and function calls to see what "optimizing" does. Be aware that if instructions are "not necessary" to the final output of the program then they may be optimized away completely! This does not lead to very interesting comparisons. Describe at least four (non-trivial) differences you see between 'normal' code and optimized code.

I wrote a simple loop that adds the value of I to a local variable and then print the result on the screen. I am trying to see the differences in a loop and cout function. The c++ code is as follows:

```
int total=0;
for (int i=0; i<5;i++){
 total=total+i;}
cout<<total<<endl;
```

Obviously, the optimized one was shorter. The actual assembly file was 113 lines but the optimized one was 91. There are less labels defined in the optimized version. I have noticed that there are less moving the data around in the unoptimized one. For example, the whole chunk of storing into stacks is deleted in the optimized one.

```
sub     rsp, 32
mov     dword ptr [rbp - 4], 0
mov     dword ptr [rbp - 8], 0
mov     dword ptr [rbp - 12], 0
```

I assume that the optimized one is trying to use storing in the stacks and playing with the available registers to manipulate with the values. Also in the unoptimized one, I can clearly the for loop in the assembly code but in the other version, it is kind of hard to pinpoint exactly when it starts and jumps arounds until the end of the loop. The loop in the unoptimized one is as follows:

```
.LBB1_1:
        cmp     dword ptr [rbp - 12], 5
        jge     .LBB1_4
# BB#2:
        mov     eax, dword ptr [rbp - 8]
        add     eax, dword ptr [rbp - 12]
        mov     dword ptr [rbp - 8], eax
# BB#3:
        mov     eax, dword ptr [rbp - 12]
        add     eax, 1
        mov     dword ptr [rbp - 12], eax
        jmp     .LBB1_1
```

I was trying to follow the process of cout in both codes and I did not see a major difference. The only thing that I noticed is that there is one or two lines that are missing in the optimized one are those when the data is being pushed back into stacked.

I changed my c++ code to a simple function call in the main to see how the optimization changes the assembly code when dealing with functions. I noticed the same patterns of reducing the number of pushing into stacks and number of labels.

```
void ftn (int x){
   cout<<x+1<<endl;}
int main(){
            int x=5;
           ftn(x);
           return 0;}
```

I think the optimized assembly code was easier to follow and keep track of. It started with defining the function as in the c++ model and went to the main function and the call of the function. However, in the unoptimized one, there are some lines of moving some data around and some pushing data into stacks before defining the function. Also, I have noticed that the optimized code is using "Lea" to get the memory address and "movsx", but I don't see the same thing in the unoptimized code. "**movsx** and movzx are special versions of mov which are designed to be used between signed (**movsx**) and unsigned (movzx) registers of different sizes. **movsx**means move with sign extension." (https://wiki.skullsecurity.org/index.php?title=Simple_Instructions)

The unoptimized code:

```
# 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
        .align  16, 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], 0
        mov     dword ptr [rbp - 12], 0
.LBB1_1:                        # =>This Inner Loop Header: Depth=1
        cmp     dword ptr [rbp - 12], 5
        jge     .LBB1_4
# BB#2:                         #   in Loop: Header=BB1_1 Depth=1
        mov     eax, dword ptr [rbp - 8]
        add     eax, dword ptr [rbp - 12]
        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
        .align  16, 0x90
        .type   _GLOBAL__sub_I_t.cpp,@function
_GLOBAL__sub_I_t.cpp:                   # @_GLOBAL__sub_I_t.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
```

And the optimized code:

```
# 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
        mov     esi, 10
        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:                         # %_ZSt13__check_facetISt5ctypeIcEERKT_PS3_.exit
        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:                        # %_ZNKSt5ctypeIcE5widenEc.exit
        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
        .align  16, 0x90
        .type   _GLOBAL__sub_I_t.cpp,@function
_GLOBAL__sub_I_t.cpp:           # @_GLOBAL__sub_I_t.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
```

Inheritance:
Following the example in the slides, I have created a dummy class Person and I converted it to .s file and reviewed the assembly code. Then I added another class student who inherited from Person. In the main, I have created an object of the student class and turned into the .s file to observe the turn of the events.
Here is my c++ code:

```
class Person{
public:
  Person(string n);
  ~Person();
  string name; };
Person::Person(string example){
name=nippet; }
Person::~Person

class Student:
public Person{
public:
Student(string example  , int number);
~Student();
int id; };
Student::Student(string n, int o):Person(n){
id=o;}
Student::~Student()

int main(){
Student stud("Farid", 6996);
return 0;
}
```

The assembly file that I was looking at had more than 500 lines so I am not copy pasting all of the lines here. When I was following the code in assembly I noticed that first the class Person is defined in the code. Assembly dedicated 64 bits for the class and then move the data around the registers and the stacks to make the transition for the next stage of program, Sudent.

```
        .text
        .globl  _ZN6PersonC2ENSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE
        .align  16, 0x90
        .type   _ZN6PersonC2ENSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE,@function
_ZN6PersonC2ENSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE: #
@_ZN6PersonC2ENSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE
```

```
.Lfunc_begin0:
        .cfi_startproc
        .cfi_personality 3, __gxx_personality_v0
        .cfi_lsda 3, .Lexception0
# BB#0:
        push    rbp
.Ltmp9:
        .cfi_def_cfa_offset 16
.Ltmp10:
        .cfi_offset rbp, -16
        mov     rbp, rsp
.Ltmp11:
        .cfi_def_cfa_register rbp
        sub     rsp, 64
        mov     qword ptr [rbp - 8], rdi
        mov     rax, rdi
        mov     qword ptr [rbp - 32], rdi # 8-byte Spill
        mov     qword ptr [rbp - 40], rax # 8-byte Spill
        mov     qword ptr [rbp - 48], rsi # 8-byte Spill
        call    _ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEC1Ev
.Ltmp3:
        mov     rdi, qword ptr [rbp - 32] # 8-byte Reload
        mov     rsi, qword ptr [rbp - 48] # 8-byte Reload
        call    _ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEaSERKS4_
.Ltmp4:
        mov     qword ptr [rbp - 56], rax # 8-byte Spill
        jmp     .LBB1_1
.LBB1_1:
        add     rsp, 64
        pop     rbp
        ret
.LBB1_2:
.Ltmp5:
        mov     ecx, edx
        mov     qword ptr [rbp - 16], rax
        mov     dword ptr [rbp - 20], ecx
.Ltmp6:
        mov     rdi, qword ptr [rbp - 40] # 8-byte Reload
        call    _ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEED1Ev
.Ltmp7:
        jmp     .LBB1_3
.LBB1_3:
        jmp     .LBB1_4
.LBB1_4:
        mov     rdi, qword ptr [rbp - 16]
        call    _Unwind_Resume
```

It made 3 calls inside the class. I have noticed the word “size” in the code several times but I could not figure out what it does. “Unwind” is a function that is being called several time and I have no Idea what it does. I tried googling it but it did not get anything useful.

In line 244, student class is introduced. The memory allocated for student is sub rsp, 96. The same moving around the registers and the stacks is happening in the this class however, we have an extra call from the parent class in here.   The reason is the child class " contains pointers to the inherited methods. This is due to the fact that polymorphism requires the address of a method or inner object to be figured out at runtime."
(https://en.wikibooks.org/wiki/X86_Disassembly/Objects_and_Classes)

```
	.text
	.globl	_ZN7StudentC2ENSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEEi
	.align	16, 0x90
	.type	_ZN7StudentC2ENSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEEi,@function
_ZN7StudentC2ENSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEEi: #
@_ZN7StudentC2ENSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEEi
.Lfunc_begin2:
	.cfi_startproc
	.cfi_personality 3, __gxx_personality_v0
	.cfi_lsda 3, .Lexception2
# BB#0:
	push	rbp
.Ltmp29:
	.cfi_def_cfa_offset 16
.Ltmp30:
	.cfi_offset rbp, -16
	mov	rbp, rsp
.Ltmp31:
	.cfi_def_cfa_register rbp
	sub	rsp, 96
	mov	qword ptr [rbp - 8], rdi
	mov	dword ptr [rbp - 12], edx
	mov	rdi, qword ptr [rbp - 8]
	lea	rax, [rbp - 48]
	mov	qword ptr [rbp - 72], rdi # 8-byte Spill
	mov	rdi, rax
	mov	qword ptr [rbp - 80], rax # 8-byte Spill
	call	_ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEC1ERKS4_
.Ltmp23:
	mov	rdi, qword ptr [rbp - 72] # 8-byte Reload
	mov	rsi, qword ptr [rbp - 80] # 8-byte Reload
	call	_ZN6PersonC2ENSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE
.Ltmp24:
	jmp	.LBB4_1
.LBB4_1:
	lea	rdi, [rbp - 48]
	call	_ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEED1Ev
	mov	eax, dword ptr [rbp - 12]
	mov	rdi, qword ptr [rbp - 72] # 8-byte Reload
	mov	dword ptr [rdi + 32], eax
	add	rsp, 96
	pop	rbp
	ret
.LBB4_2:
.Ltmp25:
        mov     ecx, edx
        mov     qword ptr [rbp - 56], rax
        mov     dword ptr [rbp - 60], ecx
.Ltmp26:
        lea     rdi, [rbp - 48]
        call    _ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEED1Ev
.Ltmp27:
        jmp     .LBB4_3
.LBB4_3:
        jmp     .LBB4_4
.LBB4_4:
        mov     rdi, qword ptr [rbp - 56]
        call    _Unwind_Resume
```

It was interesting to me that assembly treated the main function the same as classes. Allocating memory and calling student and person class to create the object.
I have tried to figure out how the destructor works but I could not find its trace in the assembly code. My assumption is that destructors are the last things to be executed by the program and I was focusing on the last lines of the code after main is executed. But no luck. These are the last lines of the code and I am assuming that destruction is taking place in Lfunc_end7 at the end of the file. The destruction process should start in the student class and then go to the parent class. Meaning that student destructor must be called before Person destructor.

```
        .section  .text.startup,"ax",@progbits
        .align    16, 0x90
        .type     _GLOBAL__sub_I_t.cpp,@function
_GLOBAL__sub_I_t.cpp:               # @_GLOBAL__sub_I_t.cpp
        .cfi_startproc
# BB#0:
        push      rbp
.Ltmp57:
        .cfi_def_cfa_offset 16
.Ltmp58:
        .cfi_offset rbp, -16
        mov       rbp, rsp
.Ltmp59:
        .cfi_def_cfa_register rbp
        call      __cxx_global_var_init
        pop       rbp
        ret
.Lfunc_end7:
        .size     _GLOBAL__sub_I_t.cpp, .Lfunc_end7-_GLOBAL__sub_I_t.cpp
        .cfi_endproc

        .type     _ZStL8__ioinit,@object  # @_ZStL8__ioinit
        .local    _ZStL8__ioinit
        .comm     _ZStL8__ioinit,1,1
        .type     .L.str,@object        # @.str
        .section  .rodata.str1.1,"aMS",@progbits,1
.L.str:
        .asciz
        .size     .L.str, 27

        .type     .L.str.1,@object      # @.str.1
.L.str.1:
	.asciz
	.size	.L.str.1, 21

	.type	.L.str.2,@object	# @.str.2
.L.str.2:
	.asciz	"Farid"
	.size	.L.str.2, 6

	.section	.init_array,"aw",@init_array
	.align	8
	.quad	_GLOBAL__sub_I_t.cpp

	.globl	_ZN6PersonC1ENSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE
	.type	_ZN6PersonC1ENSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE,@function
_ZN6PersonC1ENSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE =
_ZN6PersonC2ENSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE
	.globl	_ZN6PersonD1Ev
	.type	_ZN6PersonD1Ev,@function
_ZN6PersonD1Ev = _ZN6PersonD2Ev
	.globl	_ZN7StudentC1ENSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEEi
	.type	_ZN7StudentC1ENSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEEi,@function
_ZN7StudentC1ENSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEEi =
_ZN7StudentC2ENSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEEi
	.globl	_ZN7StudentD1Ev
	.type	_ZN7StudentD1Ev,@function
_ZN7StudentD1Ev = _ZN7StudentD2Ev
	.ident	"clang version 3.8.0-2ubuntu4 (tags/RELEASE_380/final)"
	.section	".note.GNU-stack","",@progbits
```