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
```