Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Санкт-Петербургский политехнический университет Петра Великого»

—

Институт компьютерных наук и кибербезопасности

**Высшая школа кибербезопасности**

**ЛАБОРАТОРНАЯ РАБОТА №1**

**ВВЕДЕНИЕ. ОПТИМИЗИРУЮЩИЕ КОМПИЛЯТОРЫ**

по дисциплине «Языки программирования»

Выполнили

студенты гр. 5131001/30002 Мишенев Н. С.

Квашеникова В. М.

<*подпись*>

Преподаватель

программист Малышев Е. В.

<*подпись*>

Санкт-Петербург

2024г.

СОДЕРЖАНИЕ

[ЦЕЛЬ РАБОТЫ 3](#__RefHeading___1)

[ХОД РАБОТЫ 4](#__RefHeading___2)

[1. Выбор компилятора и оптимизаций для компиляции. 4](#__RefHeading___3)

[2. Оптимизации и их описание. 4](#__RefHeading___4)

[3. Оптимизация файла по размеру, сравнение полученных бинарных файлов. 12](#__RefHeading___5)

[ВЫВОД 13](#__RefHeading___6)

[ПРИЛОЖЕНИЕ А 14](#__RefHeading___7)

ЦЕЛЬ РАБОТЫ

Цель работы – изучить принципы работы оптимизирующих компиляторов.

ХОД РАБОТЫ

## **Выбор компилятора и оптимизаций для компиляции.**

Для компиляции программы был выбран компилятор ***GCC*** и соответствующие флаги оптимизации для обоих случаев.

Для оптимизации по скорости, при компиляции файла была использована следующая команда:

gcc -Ofast OPTBENCH.c -o optimized.exe

Для проведения оптимизации по скорости использовался только один флаг ***-Ofast,*** это самая сильная оптимизация по скорости. Она игнорирует строгое соответствие стандартам компиляции. ***-Ofast*** использует все функции оптимизации ***-O3***. Она также позволяет выполнять оптимизации, которые недопустимы для всех программ, совместимых со стандартами.

Для оптимизации по размеру была использована следующая команда:

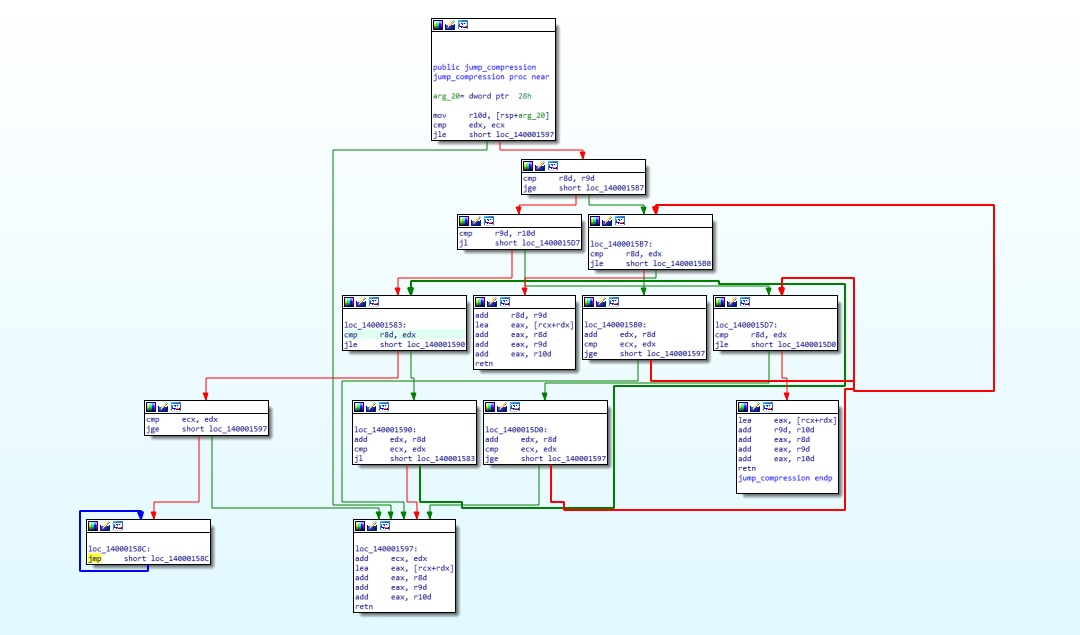
gcc -m64 -Oz -flto OPTBENCH.c -o size-optimized.exe

Флаг ***-m64*** включает режим 64 битного окружения. Флаг ***-Oz*** включает агрессивную оптимизацию по размеру. Он эквивалентен флагу ***-Os*** но использует еще несколько оптимизаций для уменьшения размера. Флаг ***-flto*** включает оптимизации линковщика.

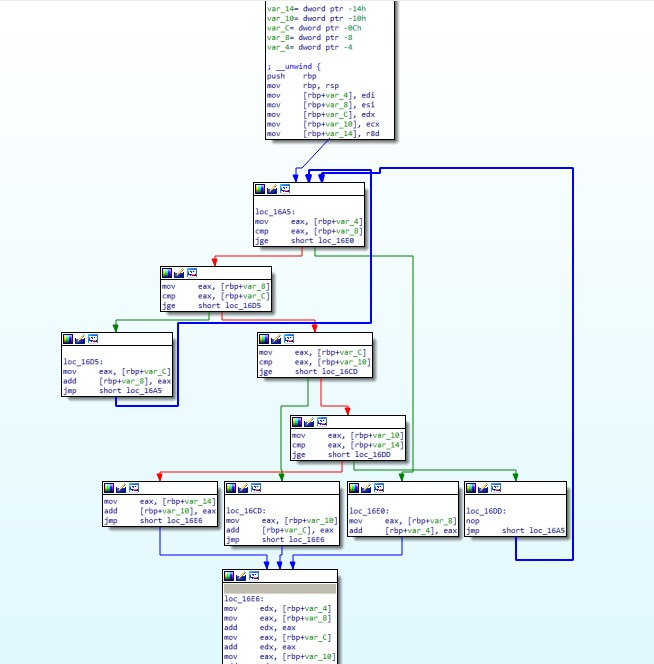
## **Оптимизации и их описание.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Оптимизация | Код на С | Неоптимизир. asm | Оптимиирован. asm | Коммент. |
| Отказ от циклов | for (i = 0; i < 3; i++)  ivector[i] = 1; | mov DWORD PTR [rip+0x2ede],0x0 # 4040 <i>  mov DWORD PTR [rip+0x2ede],0x0 # 4040 <i>  jmp 1191 <main+0x48>  mov eax,DWORD PTR [rip+0x2ed6] # 4040 <i>  cdqe  lea rdx,[rax\*4+0x0]  lea rax,[rip+0x2f3d] # 40b8 <ivector>  mov DWORD PTR [rdx+rax\*1],0x1  mov eax,DWORD PTR [rip+0x2eb8] # 4040 <i>  add eax,0x1  mov DWORD PTR [rip+0x2eaf],eax # 4040 <i>  mov eax,DWORD PTR [rip+0x2ea9] # 4040 <i>  cmp eax,0x2  jle 1164 <main+0x1b> | mov DWORD PTR [rip+0x5465],0x1 <ivector+0x8>  mov DWORD PTR [rip+0x54d7],0x3 <i> | Цикл не был сгенерирован, вместо этого в переменную `i` заносится 3, как будто цикл полностью прошел. И в массив `vector` добавляется значение 1, потому что присваивание постоянно |
| Переприсваивание | i2 = 5;  j4 = 6;  i2 = j4; | mov DWORD PTR [rip+0x2eae],0x5 # 4054 <i2>  mov DWORD PTR [rip+0x2ec8],0x6 # 4078 <j4>  mov eax,DWORD PTR [rip+0x2ec2] # 4078 <j4>  mov DWORD PTR [rip+0x2e98],eax # 4054 <i2> | mov DWORD PTR [rip+0x54ac],0x6 # 14000d250 <i2> | Все операции переприсваивания были убраны, так как компилятор вычислил итоговое значение переменной и сразу присвоил его, без лишних операций. |
| Размножение констант и копий | j4 = 2;  if (i2 < j4 && i4 < j4)  {  i2 = 2;  printf("Hello");  }  j4 = k5;  if (i2 < j4 && i4 < j4)  {  i5 = 3;  printf("Hello");  } | mov DWORD PTR [rip+0x2eb2],0x2 # 4078 <j4>  mov edx,DWORD PTR [rip+0x2e88] # 4054 <i2>  mov eax,DWORD PTR [rip+0x2ea6] # 4078 <j4>  cmp edx,eax  jge 1204 <main+0xbb>  mov edx,DWORD PTR [rip+0x2e98] # 4074 <i4>  mov eax,DWORD PTR [rip+0x2e96] # 4078 <j4>  cmp edx,eax  jge 1204 <main+0xbb>  mov DWORD PTR [rip+0x2e64],0x2 # 4054 <i2>  lea rax,[rip+0xe11] # 2008 <\_IO\_stdin\_used+0x8>  mov rdi,rax  mov eax,0x0  call 1040 <printf@plt>  1204:  mov eax,DWORD PTR [rip+0x2e7a] # 4084 <k5>  mov DWORD PTR [rip+0x2e68],eax # 4078 <j4>  mov edx,DWORD PTR [rip+0x2e3e] # 4054 <i2>  mov eax,DWORD PTR [rip+0x2e5c] # 4078 <j4>  cmp edx,eax  jge 124e <main+0x105>  mov edx,DWORD PTR [rip+0x2e4e] # 4074 <i4>  mov eax,DWORD PTR [rip+0x2e4c] # 4078 <j4>  cmp edx,eax  jge 124e <main+0x105>  mov DWORD PTR [rip+0x2e42],0x3 # 407c <i5>  lea rax,[rip+0xdc7] # 2008 <\_IO\_stdin\_used+0x8>  mov rdi,rax  mov eax,0x0  call 1040 <printf@plt> | mov eax,DWORD PTR [rip+0x5486] # 14000d220 <k5>  mov DWORD PTR [rip+0x54ac],0x6 # 14000d250 <i2>  mov DWORD PTR [rip+0x5482],eax # 14000d22c <j4>  cmp eax,0x6  jle 140007dbb <main+0x5b>  cmp eax,DWORD PTR [rip+0x547b] # 14000d230 <i4>  jg 140007f59 <main+0x1f9>    140007f59:  mov DWORD PTR [rip+0x52c5],0x3 # 14000d228 <i5>  mov rcx,rbx  call 1400014a0 <printf.constprop.0>  jmp 140007dbb <main+0x5b> | Первый условный оператор был удален, поскольку его невыполнение было очевидно. Во втором условном операторе сравнения с переменной было преобразовано в сравнение с константой, а само условие осталось прежним. |
| Свертка констант, арифметические тождества и излишние операции загрузки/сохранения | i3 = 1 + 2;  flt\_1 = 2.4 + 6.3;  i2 = 5;  j2 = i + 0;  k2 = i / 1;  i4 = i \* 1;  i5 = i \* 0; | mov DWORD PTR [rip+0x2e10],0x3 # 4068 <i3>  movsd xmm0,QWORD PTR [rip+0xdf8] # 2058 <\_IO\_stdin\_used+0x58>  movsd QWORD PTR [rip+0x2e20],xmm0 # 4088 <flt\_1>  mov DWORD PTR [rip+0x2de2],0x5 # 4054 <i2>  mov eax,DWORD PTR [rip+0x2dc8] # 4040 <i>  mov DWORD PTR [rip+0x2dda],eax # 4058 <j2>  mov eax,DWORD PTR [rip+0x2dbc] # 4040 <i>  mov DWORD PTR [rip+0x2dd2],eax # 405c <k2>  mov eax,DWORD PTR [rip+0x2db0] # 4040 <i>  mov DWORD PTR [rip+0x2dde],eax # 4074 <i4>  mov DWORD PTR [rip+0x2ddc],0x0 # 407c <i5> | mov rax,QWORD PTR [rip+0x1276] # 140009038 <.rdata+0x38>  mov DWORD PTR [rip+0x5470],0x3 # 14000d23c <i3>  mov DWORD PTR [rip+0x547a],0x5 # 14000d250 <i2>  mov QWORD PTR [rip+0x543b],rax # 14000d218 <flt\_1>  mov eax,DWORD PTR [rip+0x5481] # 14000d264 <i>  mov DWORD PTR [rip+0x5459],eax # 14000d24c <j2>  mov DWORD PTR [rip+0x5437],eax # 14000d230 <i4>  mov DWORD PTR [rip+0x53fb],0x0 # 14000d228 <i5> | На самом деле свертка констант произошла уже в неоптимизированном листинге.  Операция mov для 64-битного значения в rax заменяет более медленное обращение с плавающей запятой. Было убрано присваивание переменной ‘k2’, поскольку она переопределялась впоследствии.  Также после оптимизации инструкции только один раз обращались к переменной ‘i’. |
| Лишнее присваивание | k3 = 1;  k3 = 1; | mov DWORD PTR [rip+0x2dc2],0x1 # 406c <k3>  mov DWORD PTR [rip+0x2db8],0x1 # 406c <k3> | mov DWORD PTR [rip+0x544b],0x1 # 14000d238 <k3> | Было убрано лишнее присваивание. |
| Снижение мощности | k2 = 4 \* j5;  for (i = 0; i <= 5; i++)  ivector4[i] = i \* 2; | mov eax,DWORD PTR [rip+0x2dc6] # 4080 <j5>  shl eax,0x2  mov DWORD PTR [rip+0x2d99],eax # 405c <k2>  mov DWORD PTR [rip+0x2d73],0x0 # 4040 <i>  jmp 1300 <main+0x1b7>  12cf:  mov eax,DWORD PTR [rip+0x2d6b] # 4040 <i>  lea edx,[rax+rax\*1] // Вычисление i \* 2  mov eax,DWORD PTR [rip+0x2d62] # 4040 <i>  mov ecx,edx  cdqe  lea rdx,[rax+rax\*1]  lea rax,[rip+0x2ddb] # 40c8 <ivector4>  mov WORD PTR [rdx+rax\*1],cx  mov eax,DWORD PTR [rip+0x2d49] # 4040 <i>  add eax,0x1  mov DWORD PTR [rip+0x2d40],eax # 4040 <i>  1300:  mov eax,DWORD PTR [rip+0x2d3a] # 4040 <i>  cmp eax,0x5  jle 12cf <main+0x186> | mov eax,DWORD PTR [rip+0x5425] # 14000d224 <j5>  mov DWORD PTR [rip+0x53cf],0xa0008 # 14000d1d8 <ivector4+0x8>  shl eax,0x2  mov DWORD PTR [rip+0x542c],eax # 14000d248 <k2>  mov rax,QWORD PTR [rip+0x121d] # 140009040 <.rdata+0x40>  mov QWORD PTR [rip+0x539c],rax # 14000d1d0 <ivector4> | Снижение мощности обусловлено отказом от цикла. Был просто вычислен адрес массива ‘ivector4’. |
| Простой цикл | j5 = 0;  k5 = 10000;  do  {  k5 = k5 - 1;  j5 = j5 + 1;  i5 = (k5 \* 3) / (j5 \* constant5);  } while (k5 > 0); | mov DWORD PTR [rip+0x2d6b],0x0 # 4080 <j5>  mov DWORD PTR [rip+0x2d65],0x2710 # 4084 <k5>  131f:  mov eax,DWORD PTR [rip+0x2d5f] # 4084 <k5>  sub eax,0x1  mov DWORD PTR [rip+0x2d56],eax # 4084 <k5>  mov eax,DWORD PTR [rip+0x2d4c] # 4080 <j5>  add eax,0x1  mov DWORD PTR [rip+0x2d43],eax # 4080 <j5>  mov edx,DWORD PTR [rip+0x2d41] # 4084 <k5>  mov eax,edx  add eax,eax  lea ecx,[rax+rdx\*1]  mov edx,DWORD PTR [rip+0x2d30] # 4080 <j5>  mov eax,edx  shl eax,0x2  lea esi,[rdx+rax\*1]  mov eax,ecx  cdq  idiv esi  mov DWORD PTR [rip+0x2d19],eax # 407c <i5>  mov eax,DWORD PTR [rip+0x2d1b] # 4084 <k5>  test eax,eax  jg 131f <main+0x1d6> | mov DWORD PTR [rip+0x5411],0x2710 # 14000d224 <j5>  mov DWORD PTR [rip+0x53fb],0x0 # 14000d228 <i5> | Простой цикл позволяет отказаться от него в пользу вычисления значения переменных ‘j5’ и ‘i5’. |
| Управление переменной индукции цикла | for (i = 0; i < 100; i++)  ivector5[i \* 2 + 3] = 5; | mov DWORD PTR [rip+0x2cc9],0x0 # 4040 <i>  jmp 13ab <main+0x262>  1379:  mov eax,DWORD PTR [rip+0x2cc1] # 4040 <i>  add eax,eax  add eax,0x3  cdqe  lea rdx,[rax\*4+0x0]  lea rax,[rip+0x2d4b] # 40e0 <ivector5>  mov DWORD PTR [rdx+rax\*1],0x5  mov eax,DWORD PTR [rip+0x2c9e] # 4040 <i>  add eax,0x1  mov DWORD PTR [rip+0x2c95],eax # 4040 <i>  13ab:  mov eax,DWORD PTR [rip+0x2c8f] # 4040 <i>  cmp eax,0x63  jle 1379 <main+0x230> | lea rax,[rip+0x5211] # 14000d04c <ivector5+0xc>  lea rdx,[rax+0x320]  140007e42:  mov DWORD PTR [rax],0x5  add rax,0x10  mov DWORD PTR [rax-0x8],0x5  cmp rdx,rax  jne 140007e42 <main+0xe2>  mov DWORD PTR [rip+0x5402],0x64 # 14000d264 <i> | Компилятор изначально вычисляет значение для указателя на начальный элемент ivector5[3] и на последний элемент, который будет изменён. Указатель без вычисления следующего адреса перемещается на 16 байт, пропуская 1 элемент. |
| Глубокие подвыражения | if (i < 10)  j5 = i5 + i2;  else  k5 = i5 + i2; | mov eax,DWORD PTR [rip+0x2c84] # 4040 <i>  cmp eax,0x9  jg 13d7 <main+0x28e>  mov edx,DWORD PTR [rip+0x2cb5] # 407c <i5>  mov eax,DWORD PTR [rip+0x2c87] # 4054 <i2>  add eax,edx  mov DWORD PTR [rip+0x2cab],eax # 4080 <j5>  jmp 13eb <main+0x2a2>  13d7:  mov edx,DWORD PTR [rip+0x2c9f] # 407c <i5>  mov eax,DWORD PTR [rip+0x2c71] # 4054 <i2>  add eax,edx  mov DWORD PTR [rip+0x2c99],eax # 4084 <k5> | mov DWORD PTR [rip+0x53ae],0x5 # 14000d220 <k5> | Отказ от условного оператора обусловлен очевидностью его результата, поэтому оператор и его операторы были заменены на оператор из ветви else. |
| Проверка того, как компилятор генерирует адрес переменной с константным индексом, размножает  копии и регистры | ivector[0] = 1;  ivector[i2] = 2;  ivector[i2] = 2;  ivector[2] = 3; | mov DWORD PTR [rip+0x2cc3],0x1 # 40b8 <ivector>  mov eax,DWORD PTR [rip+0x2c59] # 4054 <i2>  cdqe  lea rdx,[rax\*4+0x0]  lea rax,[rip+0x2cac] # 40b8 <ivector>  mov DWORD PTR [rdx+rax\*1],0x2  mov eax,DWORD PTR [rip+0x2c3b] # 4054 <i2>  cdqe  lea rdx,[rax\*4+0x0]  lea rax,[rip+0x2c8e] # 40b8 <ivector>  mov DWORD PTR [rdx+rax\*1],0x2  mov DWORD PTR [rip+0x2c85],0x3 # 40c0 <ivector+0x8> | mov DWORD PTR [rip+0x5364],0x1 # 14000d1e0 <ivector>  lea edx,[rcx+0x1]  mov DWORD PTR [rip+0x536b],0x2 # 14000d1f4 <flt\_6+0x4>  mov DWORD PTR [rip+0x5355],0x3 # 14000d1e8 <ivector+0x8> | После данной оптимизации обращение происходит только по константному значению и сдвигу. |
| Удаление общих подвыражений | if ((h3 + k3) < 0 || (h3 + k3) > 5)  printf("Common subexpression elimination\n");  else  {  m3 = (h3 + k3) / i3;  g3 = i3 + (h3 + k3);  } | mov DWORD PTR [rip+0x2e10],0x3 # 4068 <i3>  mov edx,DWORD PTR [rip+0x2c23] # 4064 <h3>  mov eax,DWORD PTR [rip+0x2c25] # 406c <k3>  add eax,edx  test eax,eax  js 1460 <main+0x317>  mov edx,DWORD PTR [rip+0x2c11] # 4064 <h3>  mov eax,DWORD PTR [rip+0x2c13] # 406c <k3>  add eax,edx  cmp eax,0x5  jle 1471 <main+0x328>  lea rax,[rip+0xba9] # 2010 <\_IO\_stdin\_used+0x10>  mov rdi,rax  call 1030 <puts@plt>  jmp 14aa <main+0x361>  mov edx,DWORD PTR [rip+0x2bed] # 4064 <h3>  mov eax,DWORD PTR [rip+0x2bef] # 406c <k3>  add eax,edx  mov esi,DWORD PTR [rip+0x2be3] # 4068 <i3>  cdq  idiv esi  mov DWORD PTR [rip+0x2be2],eax # 4070 <m3>  mov edx,DWORD PTR [rip+0x2bd0] # 4064 <h3>  mov eax,DWORD PTR [rip+0x2bd2] # 406c <k3>  add edx,eax  mov eax,DWORD PTR [rip+0x2bc6] # 4068 <i3>  add eax,edx  mov DWORD PTR [rip+0x2bb6],eax # 4060 <g3> | mov DWORD PTR [rip+0x5470],0x3 # 14000d23c <i3>  cmp edx,0x5  ja 140007f48 <main+0x1e8>  movsxd rax,edx  sar edx,0x1f  add ecx,0x4  imul rax,rax,0x55555556  mov DWORD PTR [rip+0x5392],ecx # 14000d244 <g3>  shr rax,0x20  sub eax,edx  mov DWORD PTR [rip+0x5376],eax # 14000d234 <m3> | Операция вычисления выражения `h3+k3` была упразднена и вместе с этим ускорены некоторые математические вычисления |
| Вынесение инвариантного кода | for (i4 = 0; i4 <= max\_vector; i4++)  {  printf("Hello");  ivector2[i4] = j \* k;  } | lea rax,[rip+0xb4b] # 2008 <\_IO\_stdin\_used+0x8>  mov rdi,rax  mov eax,0x0  call 1040 <printf@plt>  mov eax,DWORD PTR [rip+0x2b74] # 4044 <j>  mov esi,eax  mov eax,DWORD PTR [rip+0x2b70] # 4048 <k>  mov edx,eax  mov ecx,DWORD PTR [rip+0x2b94] # 4074 <i4>  mov eax,esi  imul eax,edx  mov edx,eax  movsxd rax,ecx  lea rcx,[rip+0x2bd3] # 40c4 <ivector2>  mov BYTE PTR [rax+rcx\*1],dl  mov eax,DWORD PTR [rip+0x2b7a] # 4074 <i4>  add eax,0x1  mov DWORD PTR [rip+0x2b71],eax # 4074 <i4>  mov eax,DWORD PTR [rip+0x2b6b] # 4074 <i4>  cmp eax,0x2  jle 14b6 <main+0x36d> | lea rsi,[rip+0x530d] # 14000d1dc <ivector2>  nop  mov rcx,rbx  call 1400014a0 <printf.constprop.0>  movsxd rcx,DWORD PTR [rip+0x5351] # 14000d230 <i4>  movzx eax,BYTE PTR [rip+0x537a] # 14000d260 <j>  mul BYTE PTR [rip+0x5370] # 14000d25c <k>  mov rdx,rcx  add edx,0x1  mov BYTE PTR [rsi+rcx\*1],al  mov DWORD PTR [rip+0x5335],edx # 14000d230 <i4>  cmp edx,0x2  jle 140007ed0 <main+0x170> | Вычисление `j \* k` не было вынесено из цикла, но были ускорены все вычисления |
| Вызов функции с аргументами | dead\_code(1, "This line should not be printed"); | lea rax,[rip+0xb23] # 2038 <\_IO\_stdin\_used+0x38>  mov rsi,rax  mov edi,0x1  call 1561 <dead\_code> | <empty> | Так как данный блок не имеет никакого смысла, то и соответственно его вызова в ассемблерном листинге мы не увидим |
| Вызов функции без аргументов | unnecessary\_loop(); | ...  call 1575 <unnecessary\_loop>  ... | mov eax,DWORD PTR [rip+0x531e] # 14000d224 <j5>  ...  mov DWORD PTR [rip+0x534f],0x5 # 14000d264 <i>  mov DWORD PTR [rip+0x5305],eax # 14000d220 <k5> | Так как определение функции было заменено на простое присваивание, так как цикл в ее описание лишний, то и вызов функции не генерируется, а генерируется лишь присваивание значения |
| dead\_code | void dead\_code(a, b) int a; char \*b;  {  int idead\_store;  idead\_store = a;  if (0)  printf("%s\n", b);  } | push rbp  mov rbp,rsp  mov DWORD PTR [rbp-0x14],edi  mov QWORD PTR [rbp-0x20],rsi  mov eax,DWORD PTR [rbp-0x14]  mov DWORD PTR [rbp-0x4],eax  nop  pop rbp  ret | ret  data16 cs nop WORD PTR [rax+rax\*1+0x0]  00 00 00 00  nop DWORD PTR [rax+0x0] | Никакой код не генерировался, при определении функции сразу производится выход |
| unnecessary\_loop | void unnecessary\_loop()  {  int x;  x = 0;  for (i = 0; i < 5; i++) /\* Цикл не должен генерироваться\*/  k5 = x + j5;  } | push rbp  mov rbp,rsp  mov DWORD PTR [rbp-0x4],0x0  mov DWORD PTR [rip+0x2ab6],0x0 # 4040 <i>  00 00 00  jmp 15ac <unnecessary\_loop+0x37>  mov edx,DWORD PTR [rip+0x2aee] # 4080 <j5>  mov eax,DWORD PTR [rbp-0x4]  add eax,edx  mov DWORD PTR [rip+0x2ae7],eax # 4084 <k5>  mov eax,DWORD PTR [rip+0x2a9d] # 4040 <i>  add eax,0x1  mov DWORD PTR [rip+0x2a94],eax # 4040 <i>  mov eax,DWORD PTR [rip+0x2a8e] # 4040 <i>  cmp eax,0x4  jle 158c <unnecessary\_loop+0x17>  nop  nop  pop rbp  ret | mov eax,DWORD PTR [rip+0xbd1e] # 14000d224 <j5>  mov DWORD PTR [rip+0xbd54],0x5 # 14000d264 <i>  00 00 00  mov DWORD PTR [rip+0xbd0a],eax # 14000d220 <k5>  ret  nop WORD PTR [rax+rax\*1+0x0]  00 00 | Цикл не был сгенерирован, вместо этого в переменную `i` заносится 5, как будто цикл полностью прошел и выполняется статичное суммирование `x` и `j5` |
| loop\_jamming | void loop\_jamming(x) int x;  {  for (i = 0; i < 5; i++)  k5 = x + j5 \* i;  for (i = 0; i < 5; i++)  i5 = x \* k5 \* i;  } | push rbp  mov rbp,rsp  mov DWORD PTR [rbp-0x4],edi  mov DWORD PTR [rip+0x2a74],0x0 # 4040 <i>  jmp 15f7 <loop\_jamming+0x3c>  mov edx,DWORD PTR [rip+0x2aac] # 4080 <j5>  mov eax,DWORD PTR [rip+0x2a66] # 4040 <i>  imul edx,eax  mov eax,DWORD PTR [rbp-0x4]  add eax,edx  mov DWORD PTR [rip+0x2a9c],eax # 4084 <k5>  mov eax,DWORD PTR [rip+0x2a52] # 4040 <i>  add eax,0x1  mov DWORD PTR [rip+0x2a49],eax # 4040 <i>  mov eax,DWORD PTR [rip+0x2a43] # 4040 <i>  cmp eax,0x4  jle 15ce <loop\_jamming+0x13>  mov DWORD PTR [rip+0x2a34],0x0 # 4040 <i>  jmp 1638 <loop\_jamming+0x7d>  mov eax,DWORD PTR [rip+0x2a70] # 4084 <k5>  imul eax,DWORD PTR [rbp-0x4]  mov edx,eax  mov eax,DWORD PTR [rip+0x2a20] # 4040 <i>  imul eax,edx  mov DWORD PTR [rip+0x2a53],eax # 407c <i5>  mov eax,DWORD PTR [rip+0x2a11] # 4040 <i>  add eax,0x1  mov DWORD PTR [rip+0x2a08],eax # 4040 <i>  mov eax,DWORD PTR [rip+0x2a02] # 4040 <i>  cmp eax,0x4  jle 160e <loop\_jamming+0x53>  nop  nop  pop rbp  ret | mov eax,DWORD PTR [rip+0xbcfe] # 14000d224 <j5>  mov DWORD PTR [rip+0xbd34],0x5 # 14000d264 <i>  lea eax,[rcx+rax\*4]  mov DWORD PTR [rip+0xbce7],eax # 14000d220 <k5>  imul eax,ecx  shl eax,0x2  mov DWORD PTR [rip+0xbce3],eax # 14000d228 <i5>  ret  cs nop WORD PTR [rax+rax\*1+0x0] | При оптимизации циклы были удалены вовсе, так как их присутствие не изменяет конечное значение переменных `k5` и `i5`. Если присвоить `i` максимально достигаемое значение в цикле = 5, то можно сразу вычислить итоговые значения, не прибегая к использованию циклов |
| loop\_unrolling | void loop\_unrolling(x) int x;  {  for (i = 0; i < 6; i++)  ivector4[i] = 0;  } | push rbp  mov rbp,rsp  mov DWORD PTR [rbp-0x4],edi  mov DWORD PTR [rip+0x29e8],0x0 # 4040 <i>  jmp 1682 <loop\_unrolling+0x3b>  mov eax,DWORD PTR [rip+0x29e0] # 4040 <i>  cdqe  lea rdx,[rax+rax\*1]  lea rax,[rip+0x2a5b] # 40c8 <ivector4>  mov WORD PTR [rdx+rax\*1],0x0  mov eax,DWORD PTR [rip+0x29c7] # 4040 <i>  add eax,0x1  mov DWORD PTR [rip+0x29be],eax # 4040 <i>  mov eax,DWORD PTR [rip+0x29b8] # 4040 <i>  cmp eax,0x5  jle 165a <loop\_unrolling+0x13>  nop  nop  pop rbp  ret | mov QWORD PTR [rip+0xbc75],0x0 # 14000d1d0 <ivector4>  mov DWORD PTR [rip+0xbc73],0x0 # 14000d1d8 <ivector4+0x8>  mov DWORD PTR [rip+0xbcf5],0x6 # 14000d264 <i>  ret | Цикл в этом блоке программы был заменён на 3 присваивания, два из них для обнуления `ivector4` и одно из них для присваивания `i` значения `6` |
| jump\_compression | int jump\_compression(i, j, k, l, m)  int i, j, k, l, m;  {  beg\_1:  if (i < j)  if (j < k)  if (k < l)  if (l < m)  l += m;  else  goto end\_1;  else  k += l;  else  {  j += k;  end\_1:  goto beg\_1;  }  else  i += j;  return (i + j + k + l + m); | push rbp  mov rbp,rsp  mov DWORD PTR [rbp-0x4],edi  mov DWORD PTR [rbp-0x8],esi  mov DWORD PTR [rbp-0xc],edx  mov DWORD PTR [rbp-0x10],ecx  mov DWORD PTR [rbp-0x14],r8d  mov eax,DWORD PTR [rbp-0x4]  cmp eax,DWORD PTR [rbp-0x8]  jge 16e0 <jump\_compression+0x4f>  mov eax,DWORD PTR [rbp-0x8]  cmp eax,DWORD PTR [rbp-0xc]  jge 16d5 <jump\_compression+0x44>  mov eax,DWORD PTR [rbp-0xc]  cmp eax,DWORD PTR [rbp-0x10]  jge 16cd <jump\_compression+0x3c>  mov eax,DWORD PTR [rbp-0x10]  cmp eax,DWORD PTR [rbp-0x14]  jge 16dd <jump\_compression+0x4c>  mov eax,DWORD PTR [rbp-0x14]  add DWORD PTR [rbp-0x10],eax  jmp 16e6 <jump\_compression+0x55>  mov eax,DWORD PTR [rbp-0x10]  add DWORD PTR [rbp-0xc],eax  jmp 16e6 <jump\_compression+0x55>  mov eax,DWORD PTR [rbp-0xc]  add DWORD PTR [rbp-0x8],eax  jmp 16a5 <jump\_compression+0x14>  nop  jmp 16a5 <jump\_compression+0x14>  mov eax,DWORD PTR [rbp-0x8]  add DWORD PTR [rbp-0x4],eax  mov edx,DWORD PTR [rbp-0x4]  mov eax,DWORD PTR [rbp-0x8]  add edx,eax  mov eax,DWORD PTR [rbp-0xc]  add edx,eax  mov eax,DWORD PTR [rbp-0x10]  add edx,eax  mov eax,DWORD PTR [rbp-0x14]  add eax,edx  pop rbp  ret | mov r10d,DWORD PTR [rsp+0x28]  cmp edx,ecx  jle 140001597 <jump\_compression+0x27>  cmp r8d,r9d  jge 1400015b7 <jump\_compression+0x47>  cmp r9d,r10d  jl 1400015d7 <jump\_compression+0x67>  cmp r8d,edx  jle 140001590 <jump\_compression+0x20>  cmp ecx,edx  jge 140001597 <jump\_compression+0x27>  jmp 14000158c <jump\_compression+0x1c>  xchg ax,ax  add edx,r8d  cmp ecx,edx  jl 140001583 <jump\_compression+0x13>  add ecx,edx  lea eax,[rcx+rdx\*1]  add eax,r8d  add eax,r9d  add eax,r10d  ret  cs nop WORD PTR [rax+rax\*1+0x0]  add edx,r8d  cmp ecx,edx  jge 140001597 <jump\_compression+0x27>  cmp r8d,edx  jle 1400015b0 <jump\_compression+0x40>  add r8d,r9d  lea eax,[rcx+rdx\*1]  add eax,r8d  add eax,r9d  add eax,r10d  ret  nop DWORD PTR [rax+0x0]  add edx,r8d  cmp ecx,edx  jge 140001597 <jump\_compression+0x27>  cmp r8d,edx  jle 1400015d0 <jump\_compression+0x60>  lea eax,[rcx+rdx\*1]  add r9d,r10d  add eax,r8d  add eax,r9d  add eax,r10d  ret  nop  nop  nop  nop | Математичсекие операции были ускорены использованием дополнительных регистров, так же была изменена структура переходов. Переход в некоторый адрес для перехода в другой адрес был убран, т.е. `goto end\_1` переход был изменен. |

При рассмотрении оптимизации переходов, для получения наглядной картины внесенных изменений был рассмотрен граф переходов, сгенерированный с помощью IDA



*Рис. 1. Оптимизированные переходы.*

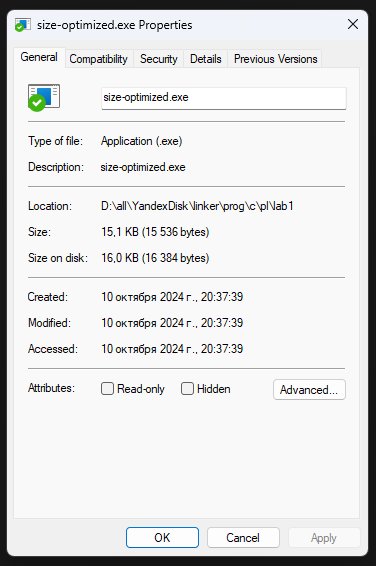
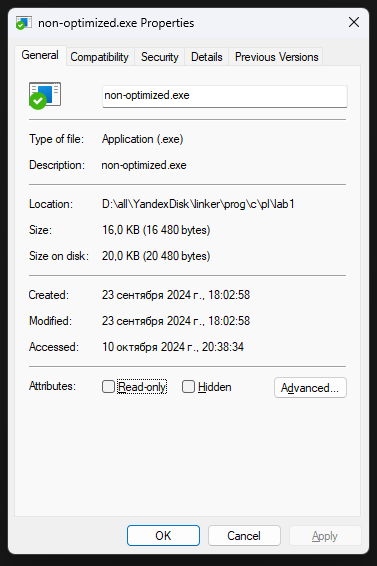


*Рис. 2. Неоптимизированные переходы.*

В результате, после рассмотрения графов с оптимизацией переходов можно сделать вывод. **GCC** убрал ненужный переход в **end\_1,** но увеличил общее количество переходов, разделив блоки с операциями на более мелкие.

## **Оптимизация файла по размеру, сравнение полученных бинарных файлов.**

После компиляции обоих файлов с помощью ***GCC*** с и без флагов оптимизации по размеру, сравнив полученные бинарные файлы, были получены следующие результаты:

*Рис. 3, 4. Бинарные файлы после оптимизации по размеру*

Как можно заметить, бинарный файл, который был скомпилирован с флагами оптимизации по размеру имеет меньший итоговый размер **(15,1kB < 16.0kB).**

Сравнив ассемблерные листинги полученных бинарных файлов, можно сказать, что листинг файла с оптимизацией по размеру получился гораздо меньше по объему **(7800 символов < 28000 символов),** и как следствие ухудшилась читаемость листинга, так как очень много команд было пропущено компилятором.

ВЫВОД

В ходе работы были изучены принципы работы оптимизирующих компиляторов, произведены оптимизации по скорости и по памяти, а также произведены сравнения полученных ассемблерных листингов.

ПРИЛОЖЕНИЕ А

Репозиторий с файлами лабораторной работы - [репозиторий на GitHub](https://github.com/clowixdev/pl_lab1)