1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. **«Институт Компьютерных наук и кибербезопасности»**
5. **Высшая школа кибербезопасности и защиты информациия**

**ЛАБОРАТОРНАЯ РАБОТА № 1**

1. «**Исследование оптимизаций компилятора**»
2. по дисциплине «Языки программирования»
3. Выполнил
4. студент гр. 5131001/20003 Черникова В.М.

<*подпись*>

1. Преподаватель
2. Семьянов. П. В.

<*подпись*>

1. Санкт-Петербург
2. 2023
3. Начало

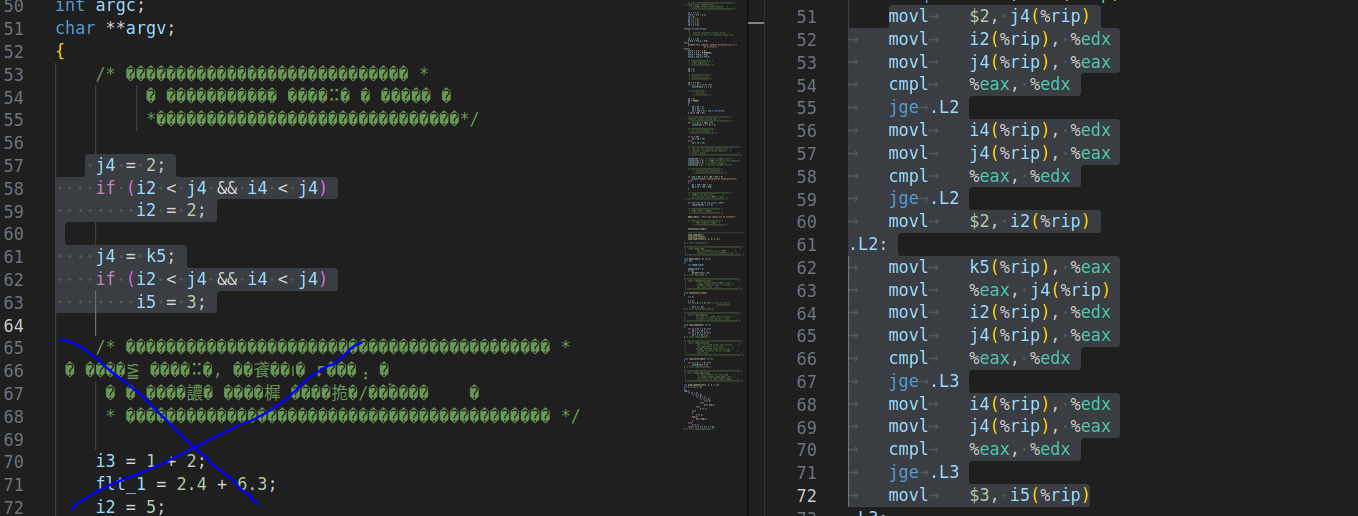
Выбранный компилятор – gcc. В изначальном файле представлены такие тесты оптимизации компилятора, как:

1. Размножение констант и копий
2. Свертка констант, арифметические тождества и излишние операции загрузки/сохранения
3. Некоторые компиляторы распознают ошибку деления на нуль и не генерируют объектный код
4. Лишнее присваивание
5. Снижение мощности
6. Простой цикл
7. Управление переменной индукции цикла
8. Глубокие подвыражения
9. Проверка того, как компилятор генерирует адрес переменной с константным индексом, размножает копии и регистры
10. Удаление общих подвыражений
11. Вынесение инвариантного кода (j \* k) может быть вынесено из цикла
12. Вызов функции с аргументами
13. Вызов функции без аргументов
14. Функция: dead\_code : Проверка недостижимого кода и лишних присваиваний. Не должен генерироваться код.
15. Функция: unnecessary\_loop : Цикл в следующей функции ненужен, так как значение присваивания постоянно. В идеале цикл должен быть удален.
16. Функция: loop\_jamming : Два цикла в этой функции имеют одинаковые заголовки и могут быть слиты в один.
17. ­Функция: loop\_unrolling : Цикл в этой функции должен быть заменен тремя присваиваниями с использованием константной индексации массива или машинно-зависимыми командами для инициализации блока памяти.
18. Функция: jump\_compression : Эта программа полезна для демонстрации сжатия цепочки переходов. goto end\_1 может быть заменен на прямой переход на beg\_1.

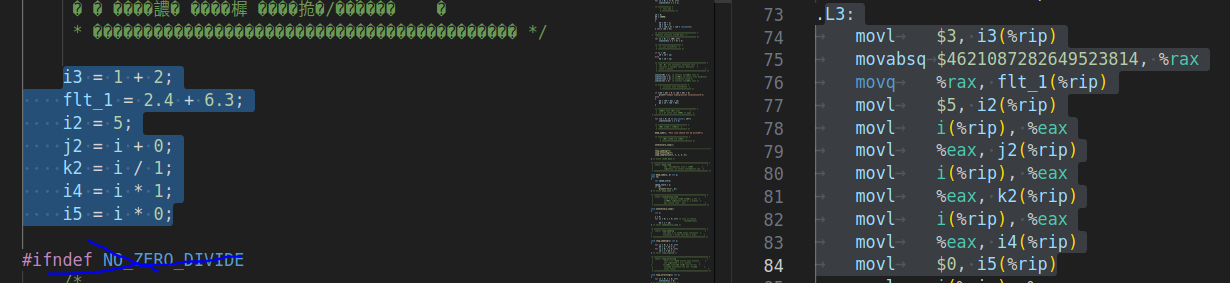
И еще несколько добавленных оптимизаций:

1. Неправильное вложение циклов
2. Основная часть
   1. Исследование ассемблера (флаг О0)

* Код преобразовался без оптимизаций



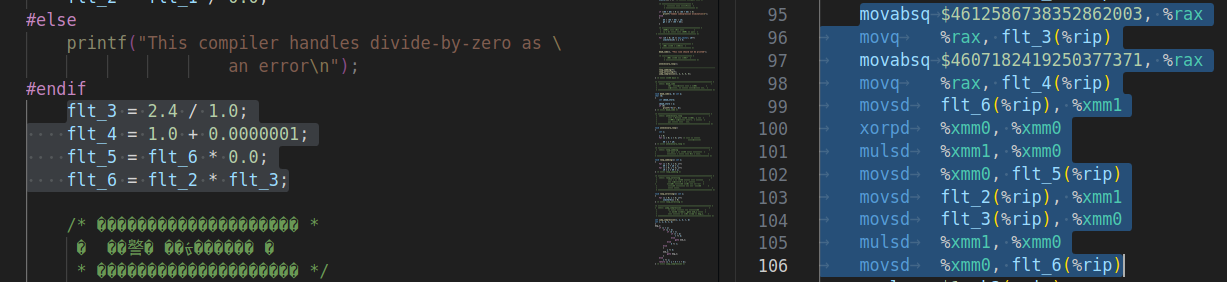
* Компилятор сразу посчитал арифметические операции в первых трех строчках. В следующих выполнил операцию присвоения сразу значения i и 0 для каждой переменной.



* Деление на ноль в компиляторе не распозналось как ошибка, и код сгенерировался так:



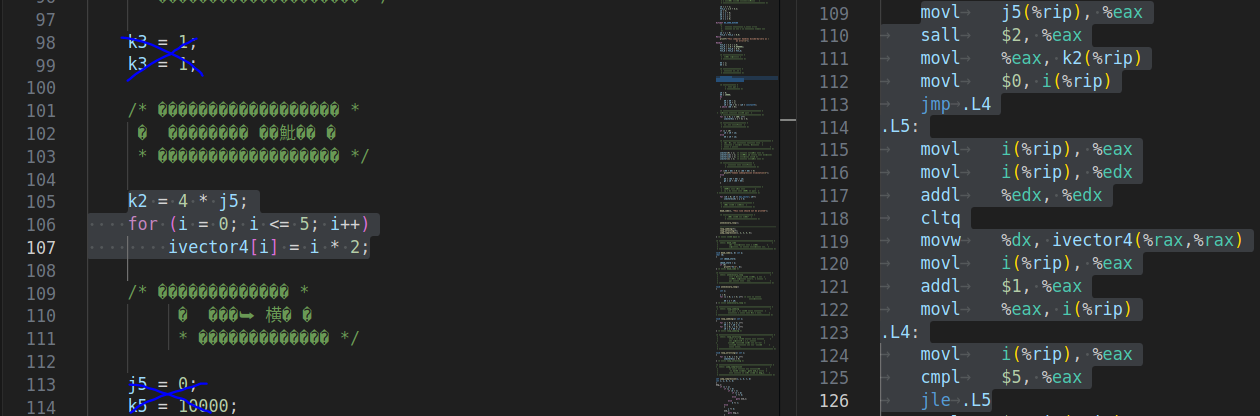
* Просто деление произошло через регистры



* Лишнее присваивание не оптимизировалось



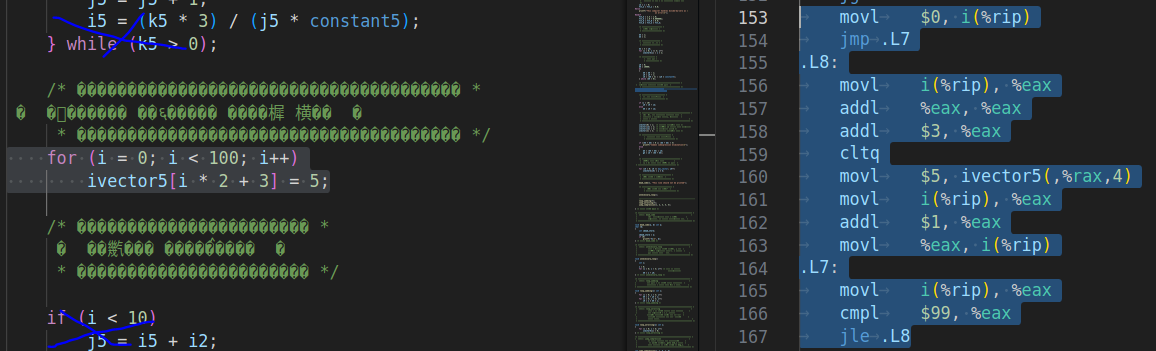
* Снижение мощности. Умножение на 4 заменилось на сдвиг на вторую степень двойки, а цикл for заменился на while{..} do (..), а умножение i на 2 заменилось на сложение.



* Простой цикл не оптимизировался



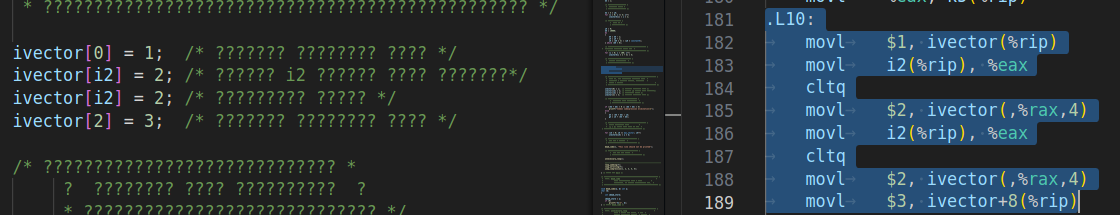
* Управление переменной индукции цикла не оптимизировалось



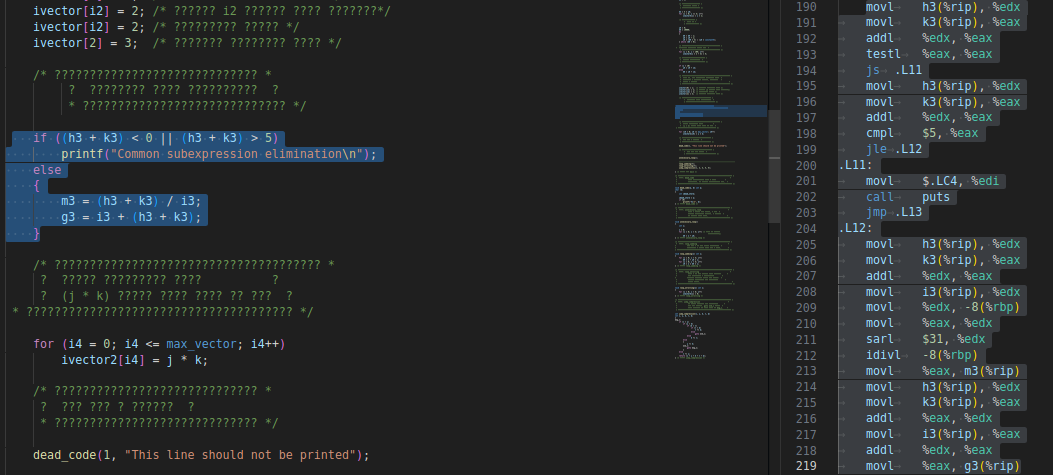
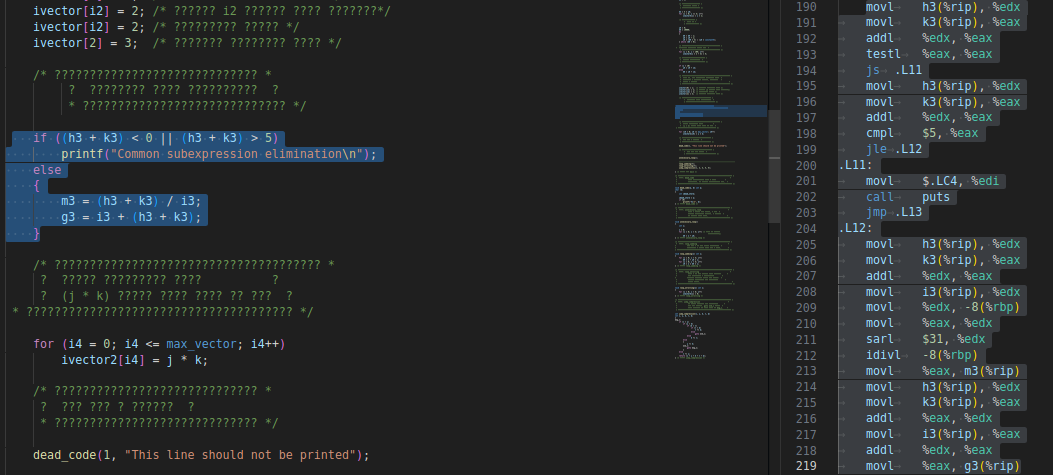
* Глубокие подвыражения не оптимизировалось



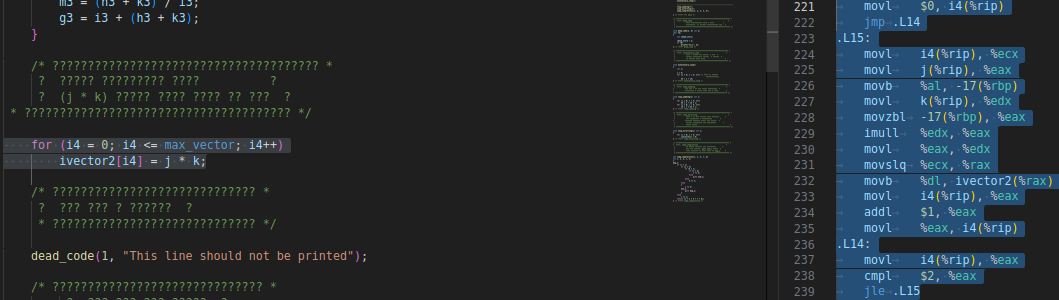
* Проверка того, как компилятор генерирует адрес переменной с константным индексом, размножает копии и регистры



* Удаление общих подвыражений



* Вынесение инвариантного кода (j \* k) может быть вынесено из цикла



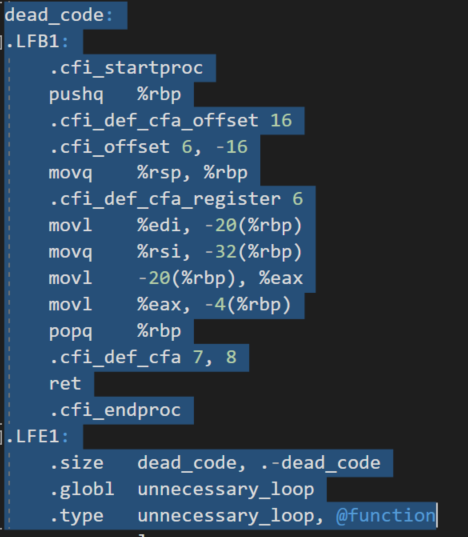
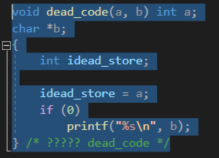
* Вызов функции с аргументами



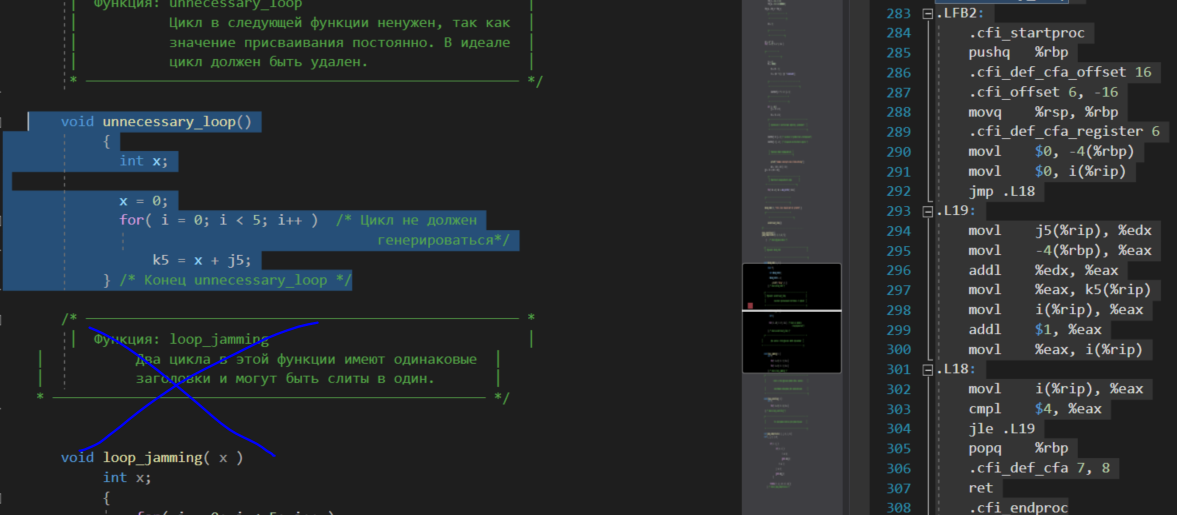
* Вызов функции без аргументов



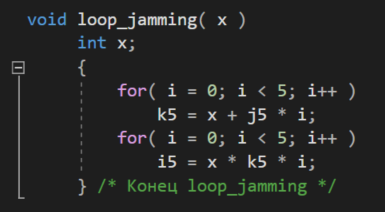
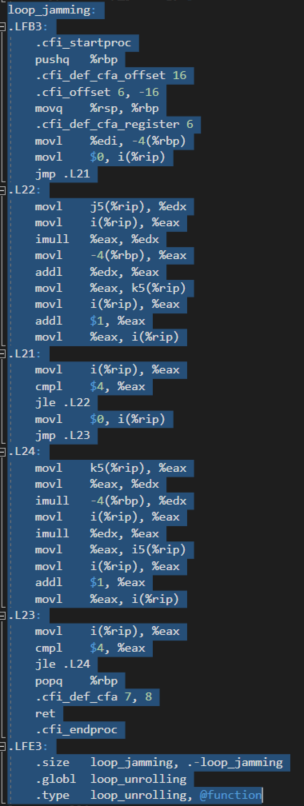
* Функция: dead\_code : Проверка недостижимого кода и лишних присваиваний. Не должен генерироваться код.



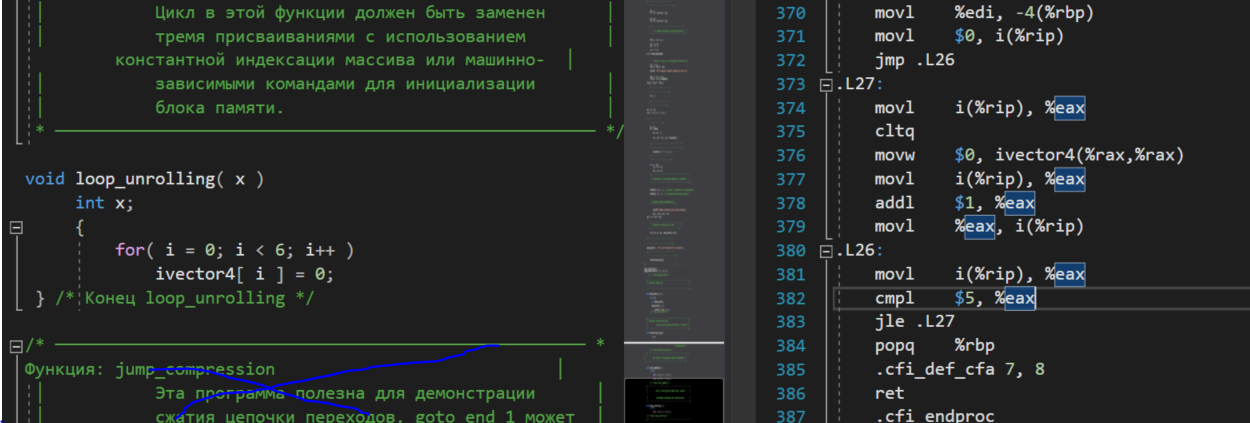
* Функция: unnecessary\_loop : Цикл в следующей функции не нужен, так как значение присваивания постоянно. В идеале цикл должен быть удален.



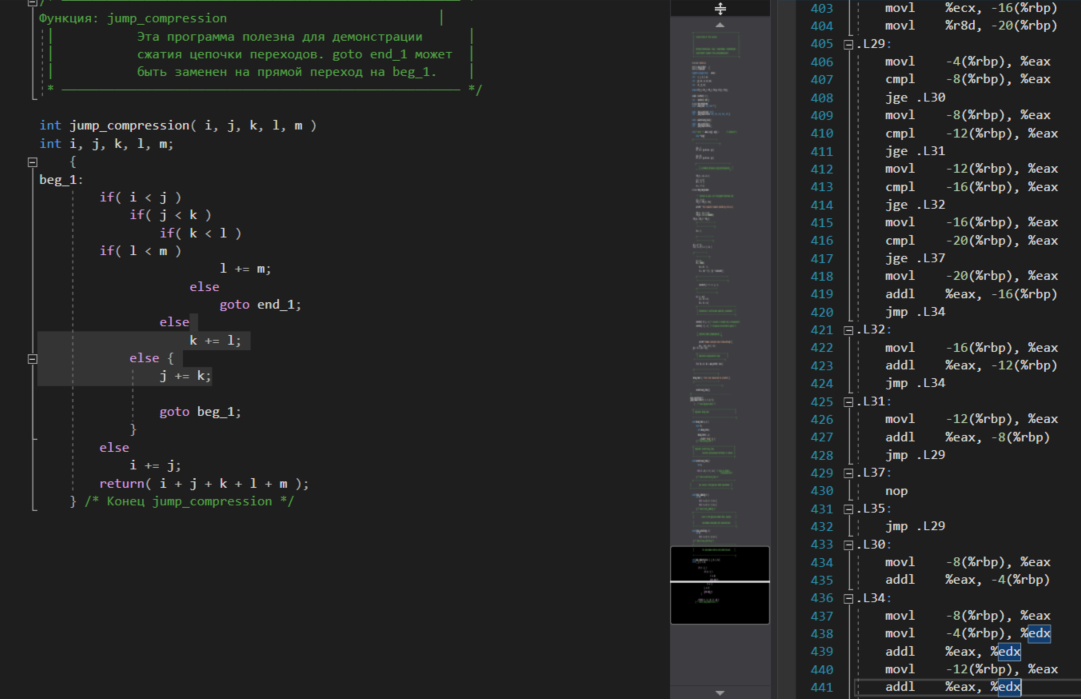
* Функция: loop\_jamming : Два цикла в этой функции имеют одинаковые заголовки и могут быть слиты в один.

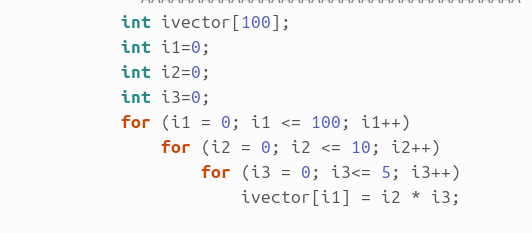
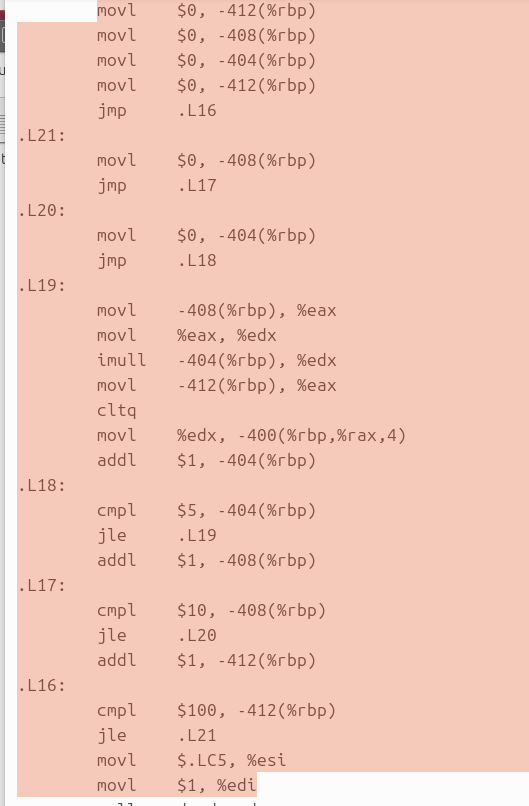
* ­Функция: loop\_unrolling : Цикл в этой функции должен быть заменен тремя присваиваниями с использованием константной индексации массива или машинно-зависимыми командами для инициализации блока памяти.



* Функция: jump\_compression : Эта программа полезна для демонстрации сжатия цепочки переходов. goto end\_1 был заменен на nop (нет операций).



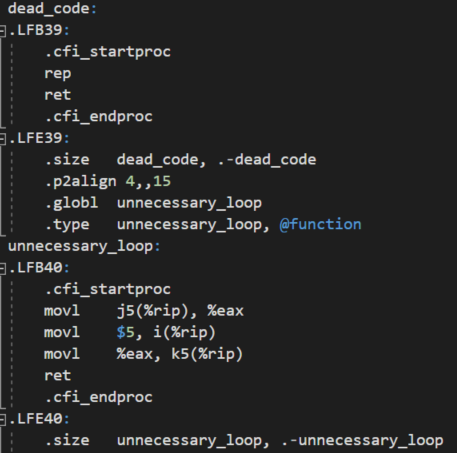
* Неправильное вложение циклов

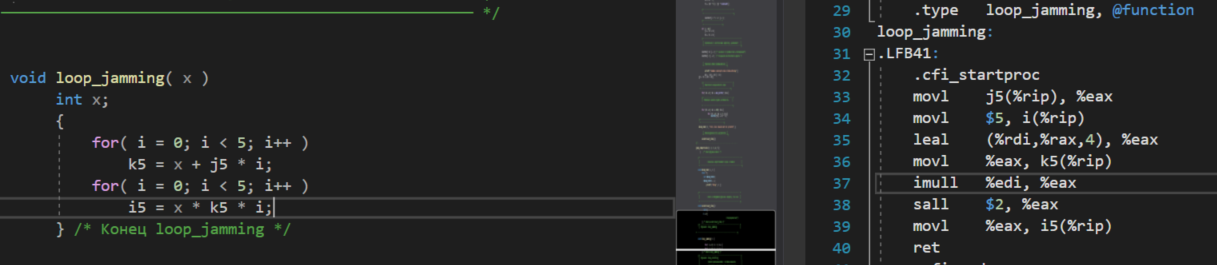
* 1. Исследование оптимизаций

В результате сравнения настроек компилятора была выбрана опция O3 (оптимизация с Ofast не сильно отличается), так как:

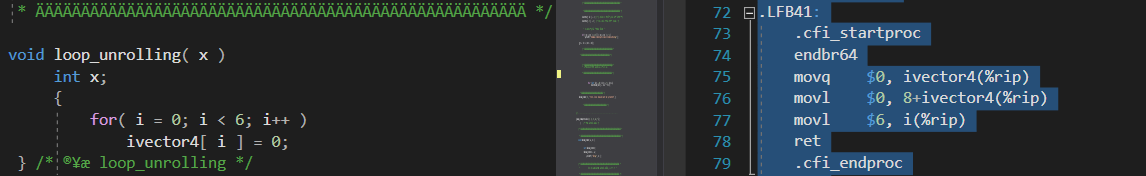
* Функция dead\_code не генерируется и unnecessary\_loop не использует цикл



* В loop\_jamming цикл не используется вообще



* Loop\_unrolling развернул цикл и записал 0 в память оптимальным способом



* В функции jump\_compression компилятор выставил условия таким образом, чтобы совершалось меньше переходов

|  |  |  |
| --- | --- | --- |
| xorl %eax, %eax  movl $2, j4(%rip)  movl i2(%rip), %edx  movl j4(%rip), %eax  cmpl %eax, %edx  jge .L2  movl i4(%rip), %edx  movl j4(%rip), %eax  cmpl %eax, %edx  jge .L2  movl $2, i2(%rip)  .L2:  movl k5(%rip), %eax  movl %eax, j4(%rip)  movl i2(%rip), %edx  movl j4(%rip), %eax  cmpl %eax, %edx  jge .L3  movl i4(%rip), %edx  movl j4(%rip), %eax  cmpl %eax, %edx  jge .L3  movl $3, i5(%rip)  .L3: | j4 = 2;  if( i2 < j4 && i4 < j4 )  i2 = 2;   j4 = k5;  if( i2 < j4 && i4 < j4 )  i5 = 3; | movl k5(%rip), %eax  movl %eax, j4(%rip) |
| movl $3, i3(%rip)  movsd .LC0(%rip), %xmm0  movsd %xmm0, flt\_1(%rip)  movl $5, i2(%rip)  movl i(%rip), %eax  movl %eax, j2(%rip)  movl i(%rip), %eax  movl %eax, k2(%rip)  movl i(%rip), %eax  movl %eax, i4(%rip)  movl $0, i5(%rip) | i3 = 1 + 2;  flt\_1 = 2.4 + 6.3;  i2 = 5;  j2 = i + 0;  k2 = i / 1;  i4 = i \* 1;  i5 = i \* 0; | movl $3, i3(%rip)  movq .LC0(%rip), %rax  movq %rax, flt\_1(%rip)  movl i(%rip), %eax  movl $5, i2(%rip)  movl %eax, j2(%rip)  movl %eax, i4(%rip) |
| movsd .LC1(%rip), %xmm0  movsd %xmm0, flt\_3(%rip)  movsd .LC2(%rip), %xmm0  movsd %xmm0, flt\_4(%rip)  movsd flt\_6(%rip), %xmm1  pxor %xmm0, %xmm0  mulsd %xmm1, %xmm0  movsd %xmm0, flt\_5(%rip)  movsd flt\_2(%rip), %xmm1  movsd flt\_3(%rip), %xmm0  mulsd %xmm1, %xmm0  movsd %xmm0, flt\_6(%rip) | flt\_3 = 2.4 / 1.0;  flt\_4 = 1.0 + 0.0000001;  flt\_5 = flt\_6 \* 0.0;  flt\_6 = flt\_2 \* flt\_3; | movsd .LC1(%rip), %xmm0  pxor %xmm1, %xmm1  mulsd flt\_6(%rip), %xmm1  movsd %xmm0, flt\_3(%rip)  mulsd flt\_2(%rip), %xmm0  movq .LC2(%rip), %rax  movq %rax, flt\_4(%rip)  movsd %xmm1, flt\_5(%rip)  movsd %xmm0, flt\_6(%rip) |
| movl $1, k3(%rip)  movl $1, k3(%rip) | k3 = 1;  k3 = 1; | movl $1, k3(%rip) |
| movl j5(%rip), %eax  sall $2, %eax  movl %eax, k2(%rip)  movl $0, i(%rip)  jmp .L4  .L5:  movl i(%rip), %eax  leal (%rax,%rax), %edx  movl i(%rip), %eax  movl %edx, %ecx  cltq  leaq (%rax,%rax), %rdx  leaq ivector4(%rip), %rax  movw %cx, (%rdx,%rax)  movl i(%rip), %eax  addl $1, %eax  movl %eax, i(%rip)  .L4:  movl i(%rip), %eax  cmpl $5, %eax  jle .L5 | k2 = 4 \* j5;  for( i = 0; i <= 5; i++ )  ivector4[ i ] = i \* 2; | movl j5(%rip), %eax  movl $655368, 8+ivector4(%rip)  sall $2, %eax  movl %eax, k2(%rip)  movq .LC4(%rip), %rax  movq %rax, ivector4(%rip) |
| movl $0, j5(%rip)  movl $10000, k5(%rip)  .L6:  movl k5(%rip), %eax  subl $1, %eax  movl %eax, k5(%rip)  movl j5(%rip), %eax  addl $1, %eax  movl %eax, j5(%rip)  movl k5(%rip), %edx  movl %edx, %eax  addl %eax, %eax  leal (%rax,%rdx), %ecx  movl j5(%rip), %edx  movl %edx, %eax  sall $2, %eax  leal (%rdx,%rax), %esi  movl %ecx, %eax  cltd  idivl %esi  movl %eax, i5(%rip)  movl k5(%rip), %eax  testl %eax, %eax  jg .L6 | j5 = 0;  k5 = 10000;  do {  k5 = k5 - 1;  j5 = j5 + 1;  i5 = (k5 \* 3) / (j5 \* constant5);  } while ( k5 > 0 ); | movl $10000, j5(%rip)  movl $0, i5(%rip) |
| movl $0, i(%rip)  jmp .L7  .L8:  movl i(%rip), %eax  addl %eax, %eax  addl $3, %eax  cltq  leaq 0(,%rax,4), %rdx  leaq ivector5(%rip), %rax  movl $5, (%rdx,%rax)  movl i(%rip), %eax  addl $1, %eax  movl %eax, i(%rip)  .L7:  movl i(%rip), %eax  cmpl $99, %eax  jle .L8 | for( i = 0; i < 100; i++ )  ivector5[ i \* 2 + 3 ] = 5; | leaq 12+ivector5(%rip), %rax  leaq 800(%rax), %rdx  .L6:  movl $5, (%rax)  addq $16, %rax  movl $5, -8(%rax)  cmpq %rax, %rdx  jne .L6  movl $100, i(%rip) |
| movl i(%rip), %eax  cmpl $9, %eax  jg .L9  movl i5(%rip), %edx  movl i2(%rip), %eax  addl %edx, %eax  movl %eax, j5(%rip)  jmp .L10  .L9:  movl i5(%rip), %edx  movl i2(%rip), %eax  addl %edx, %eax  movl %eax, k5(%rip)  .L10: | if( i < 10 )  j5 = i5 + i2;  else  k5 = i5 + i2; | movl $5, k5(%rip) |
| movl $1, ivector(%rip)  movl i2(%rip), %eax  cltq  leaq 0(,%rax,4), %rdx  leaq ivector(%rip), %rax  movl $2, (%rdx,%rax)  movl i2(%rip), %eax  cltq  leaq 0(,%rax,4), %rdx  leaq ivector(%rip), %rax  movl $2, (%rdx,%rax)  movl $3, 8+ivector(%rip) | ivector[ 0 ] = 1; /\* генерация константного адреса \*/  ivector[ i2 ] = 2; /\* значение i2 должно быть скопировано\*/  ivector[ i2 ] = 2; /\* копирование регистров \*/  ivector[ 2 ] = 3; /\* генарация константного адреса \*/ | movl $1, ivector(%rip)    movl $2, 20+ivector(%rip)  movl $3, 8+ivector(%rip) |
| movl h3(%rip), %edx  movl k3(%rip), %eax  addl %edx, %eax  testl %eax, %eax  js .L11  movl h3(%rip), %edx  movl k3(%rip), %eax  addl %edx, %eax  cmpl $5, %eax  jle .L12  .L11:  leaq .LC4(%rip), %rax  movq %rax, %rdi  call puts@PLT  jmp .L13  .L12:  movl h3(%rip), %edx  movl k3(%rip), %eax  addl %edx, %eax  movl i3(%rip), %edi  cltd  idivl %edi  movl %eax, m3(%rip)  movl h3(%rip), %edx  movl k3(%rip), %eax  addl %eax, %edx  movl i3(%rip), %eax  addl %edx, %eax  movl %eax, g3(%rip)  .L13: | if(( h3 + k3 ) < 0 || ( h3 + k3 ) > 5)  printf("Common subexpression elimination\n");  else {  m3 = ( h3 + k3 ) / i3;  g3 = i3 + (h3 + k3);  } | movl h3(%rip), %ecx  movslq %edx, %rax  sarl $31, %edx  addl $4, %ecx  movl $10000, %esi  imulq $1431655766, %rax, %rax  movl %ecx, g3(%rip)  shrq $32, %rax  subl %edx, %eax  movl %eax, m3(%rip)  leal 1(%rcx), %edx  cmpl $5, %edx  ja .L12  .L12:  …  call puts@PLT  …  jmp .L8 |
| movl $0, i4(%rip)  jmp .L14  .L15:  movl j(%rip), %eax  movl %eax, %esi  movl k(%rip), %eax  movl %eax, %edx  movl i4(%rip), %ecx  movl %esi, %eax  imull %edx, %eax  movl %eax, %edx  movslq %ecx, %rax  leaq ivector2(%rip), %rcx  movb %dl, (%rax,%rcx)  movl i4(%rip), %eax  addl $1, %eax  movl %eax, i4(%rip)  .L14:  movl i4(%rip), %eax  cmpl $2, %eax  jle .L15 | for( i4 = 0; i4 <= max\_vector; i4++)  ivector2[ i4 ] = j \* k; | movzbl j(%rip), %eax  mulb k(%rip)  movzbl %al, %ecx  movb %al, 2+ivector2(%rip)  sall $8, %edx  addl %ecx, %edx  movw %dx, ivector2(%rip) |
| leaq .LC5(%rip), %rax  movq %rax, %rsi  movl $1, %edi  call dead\_code  call unnecessary\_loop  movl $7, %edi  call loop\_jamming  movl $7, %edi  call loop\_unrolling  movl $5, %r8d  movl $4, %ecx  movl $3, %edx  movl $2, %esi  movl $1, %edi  call jump\_compression | dead\_code( 1, "This line should not be printed" );  unnecessary\_loop();  loop\_jamming(7);  loop\_unrolling(7);  jump\_compression(1, 2, 3, 4, 5); | movl j5(%rip), %esi  movl %esi, k5(%rip)  movl $5, i(%rip)  call loop\_jamming  movq $0, ivector4(%rip)  movl $0, 8+ivector4(%rip)  movl $6, i(%rip) |
| movl $0, -60(%rbp)  jmp .L16  .L21:  movl $0, -56(%rbp)  jmp .L17  .L20:  movl $0, -52(%rbp)  jmp .L18  .L19:  movl -56(%rbp), %eax  imull -52(%rbp), %eax  movl %eax, %edx  movl -60(%rbp), %eax  cltq  movl %edx, -48(%rbp,%rax,4)  addl $1, -52(%rbp)  .L18:  cmpl $1, -52(%rbp)  jle .L19  addl $1, -56(%rbp)  .L17:  cmpl $5, -56(%rbp)  jle .L20  addl $1, -60(%rbp)  .L16:  cmpl $10, -60(%rbp)  jle .L21 | int ivectorm[10];  for (int ii1 = 0; ii1 <= 10; ii1++)  for (int ii2 = 0; ii2 <= 5; ii2++)  for (int ii3 = 0; ii3 <= 1; ii3++)  ivectorm[ii1] = ii2 \* ii3; |  |

1. Вывод

В ходе работы был изучен и сравнен ассемблерный код неоптимизированной и оптимизированной программы. В результате сравнения получено, что код с оптимизацией по скорости будет работать быстрее, в некоторых случаях требует меньше памяти. Для оптимизации кода программы (оптимизации по скорости) в основном. Для оптимизации код просматривается компилятором заранее для удаления лишних присваиваний, замены конструкций там, где это возможно, удалялся недостижимый код, а также производится слияние и развертка циклов.

1. Приложение

* Отличия O3 и Ofast

