1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Высшая школа кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 3**

1. «**Оптимизация программы**»
2. по дисциплине «Языки программирования»
3. Выполнил
4. студент гр. 5131001/20001 Маронова К.Д.  
    Тайманов Д.М.

<*подпись*>

1. Преподаватель

Малышев Е.В.

<*подпись*>

1. Санкт-Петербург
2. 2023
3. **1. Цель работы**

В работе необходимо изучить принципы работы оптимизирующих компиляторов.

**2. Теоретические исследования**

*Оптимизирующий компилятор* - это программное обеспечение, которое преобразует исходный код программы, написанной на языке программирования, в машинный код, оптимизированный для выполнения на конкретной аппаратной платформе.

*Основная цель* оптимизирующего компилятора - повысить производительность программы. Для этого он применяет различные оптимизации, такие как удаление ненужных операций, реорганизацию кода для лучшего использования памяти и другие техники. В результате работы оптимизирующего компилятора программа может выполняться быстрее и используется ресурсы компьютера более эффективно.

В ходе работы был использован оптимизирующий компилятор visual studio 2022. Оптимизирующий компилятор в Visual Studio 2022 выполняет ряд оптимизаций, чтобы улучшить производительность создаваемых программ. Компилятор может оптимизировать использование памяти и производить различные трансформации кода для улучшения его эффективности.

Флаги оптимизации в Visual Studio - это настройки, позволяющие улучшить производительность и эффективность компиляции программного кода.

Некоторые примеры флагов оптимизации в Visual Studio:

1. /O1, /O2, /Ox: Эти флаги включают оптимизацию кода. /O1 применяет базовую оптимизацию, /O2 применяет оптимизацию более высокого уровня, а /Ox включает все доступные оптимизации.

2. /GL: Данный флаг включает объединение кода в пределах единого исполняемого файла. Это может улучшить производительность, но также может увеличить время компиляции.

В ходе работы необходимо было настроить проект на вывод дизассемблера и различные виды оптимизации (по скорости/по размеру). Все действия производились в свойствах проекта:

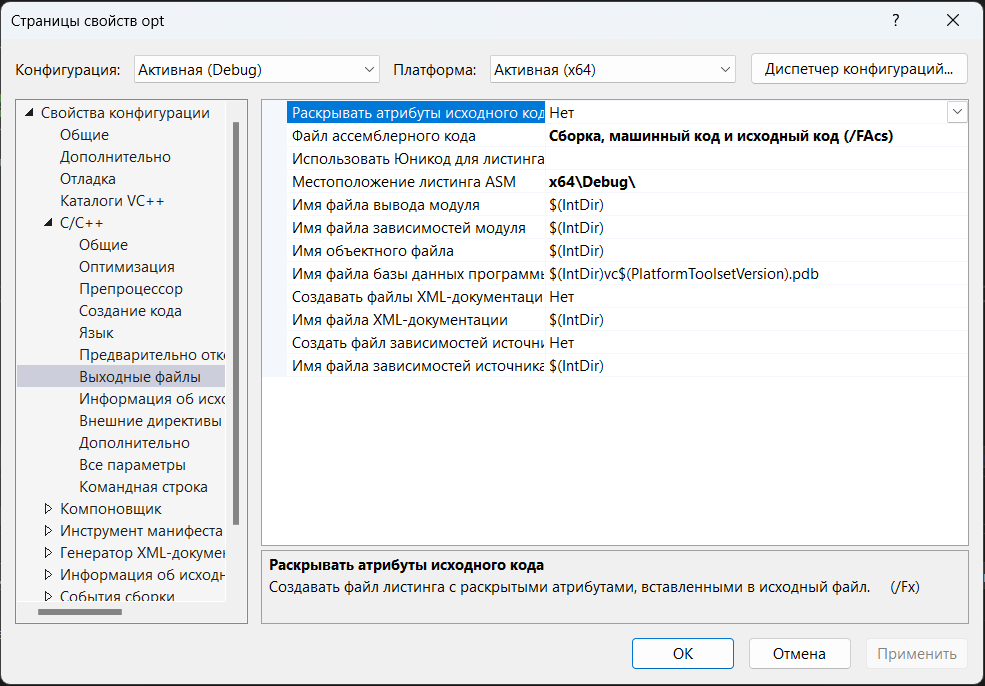


Рис.1 «Установка выходных данных (включение дизассемблирования)»

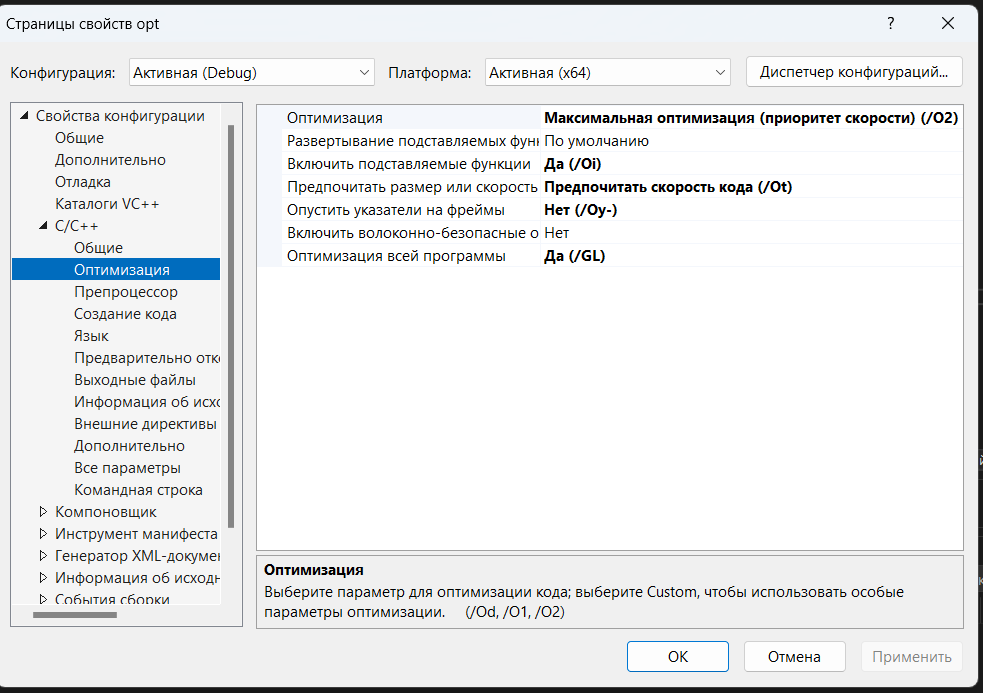


Рис.2 «Включение оптимизации по скорости»

Для компиляции кода пришлось установить метку «C7 совместимо (Z7)» в графе «Формат отладочной информации», а также метку «По умолчанию» в графе «Основные проверки времени выполнения» (Рис.3 и 4).

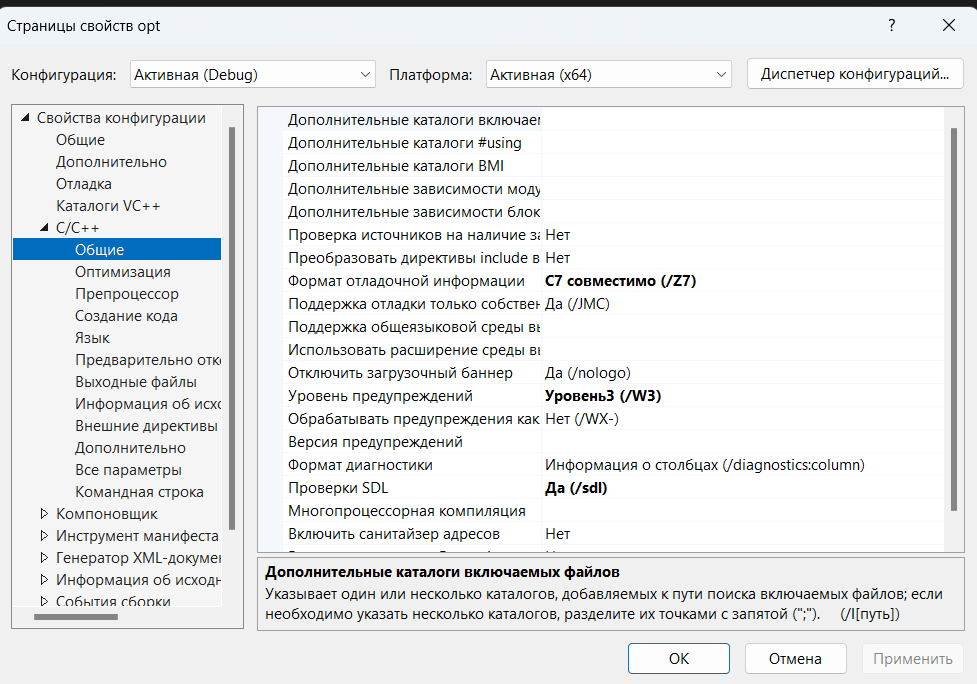


Рис. 3 «Раздел «Общие»»

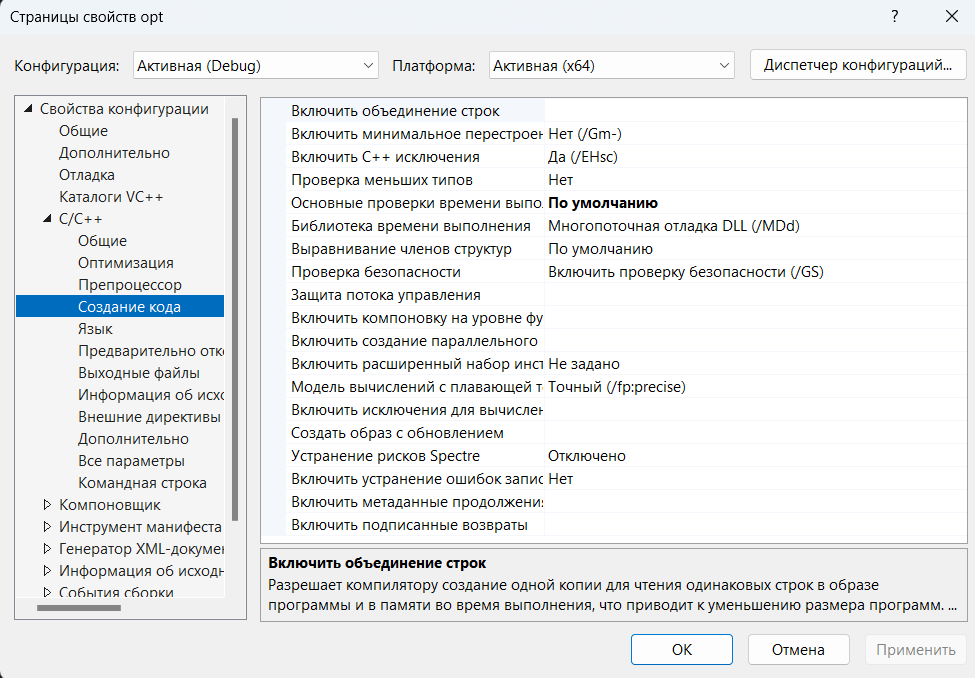
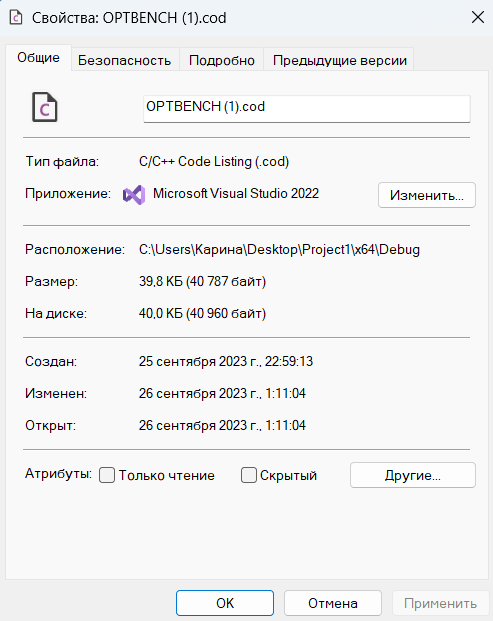
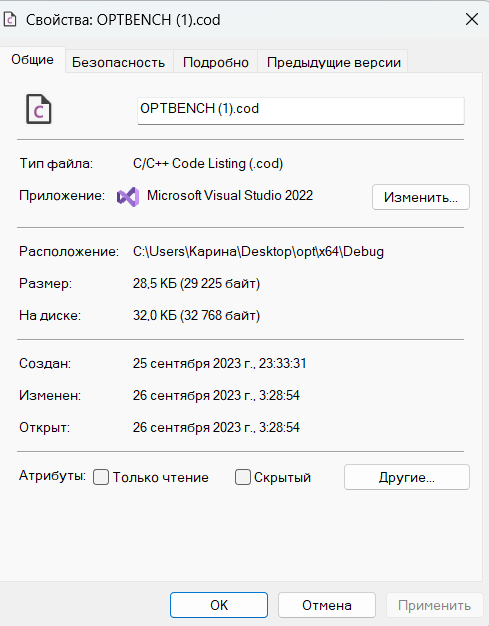


Рис.4 «Раздел «Создание кода»»

**3. Ход работы**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Оптимизация** | **Код на языке С** | **Неоптимизированный код** | **Оптимизированный код** | **Комментарий** |
| Размножение констант и копий | j4 = 2;  if(i2<j4&& i4< j4 )  i2 = 2;  j4 = k5;  if(i2<j4&&i4 < j4 )  i5 = 3; | mov DWORD PTR j4[rip], 2  mov edx, DWORD PTR i2[rip]  mov eax, DWORD PTR j4[rip]  cmp edx, eax  jge .L2  mov edx, DWORD PTR i4[rip]  mov eax, DWORD PTR j4[rip]  cmp edx, eax  jge .L2  mov DWORD PTR i2[rip], 2  .L2:  mov eax, DWORD PTR k5[rip]  mov DWORD PTR j4[rip], eax  mov edx, DWORD PTR i2[rip]  mov eax, DWORD PTR j4[rip]  cmp edx, eax  jge .L4  mov edx, DWORD PTR i4[rip]  mov eax, DWORD PTR j4[rip]  cmp edx, eax  jge .L4  mov DWORD PTR i5[rip], 3  .L4: | cmp DWORD PTR i2[rip], 1  jle .L5  .L2:  mov eax, DWORD PTR k5[rip]  cmp eax, DWORD PTR i2[rip]  mov DWORD PTR j4[rip], eax  jle .L3  cmp eax, DWORD PTR i4[rip]  jle .L3  mov DWORD PTR i5[rip], 3  .L3:  xor eax, eax  ret  .L5:  cmp DWORD PTR i4[rip], 1  jg .L2  mov DWORD PTR i2[rip], 2  jmp .L2 | Компилятор убрал присваивание значения переменной j4, т.к. она далее не используется, подставив в аргумент функции cmp соответствующую константу. |
| Отказ от циклов | for(i = 0; i < 3 ; i++) ivector[ i ] = 1; | mov dword ptr [i (07FF71822D788h)],0  jmp main+39h (07FF718226229h)  mov eax,dword ptr [i (07FF71822D788h)]  inc eax  mov dword ptr [i (07FF71822D788h)],eax  cmp dword ptr [i (07FF71822D788h)],3  jge main+59h (07FF718226249h)  movsxd rax,dword ptr [i (07FF71822D788h)]  lea rcx,[ivector (07FF71822D710h)]  mov dword ptr [rcx+rax\*4],1  jmp main+2Bh (07FF71822621Bh) | mov dword ptr [rdi+rax\*4+6660h],1  inc ecx+  cmp ecx,3  jl main+30h (07FF79D021110h) | Компилятор не использует переменную i. Обращение к массиву происходит с помощью суммы и доступа по адресам смещения. Цикл компилятором удален не был |
| Переприсваивание | i2 = 5; j4 = 6;  i2 = j4; | mov dword ptr [i2 (07FF71822D774h)],5  mov dword ptr [j4 (07FF71822D750h)],6  mov eax,dword ptr [j4 (07FF71822D750h)]  mov dword ptr [i2 (07FF71822D774h)],eax | mov dword ptr [i2 (07FF79D026810h)],6 | Отсутствует «лишнее» присваивание i2=5 и j4=6, вместо этого сразу выполняется присваивание i2=6. |
| 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 [i3 (07FF71822D760h)],3  movsd xmm0,mmword ptr [\_\_real@4021666666666666 (07FF71822AC88h)]  movsd mmword ptr [flt\_1 (07FF71822D748h)],xmm0  mov dword ptr [i2 (07FF71822D774h)],5  mov eax,dword ptr [i (07FF71822D788h)]  mov dword ptr [j2 (07FF71822D770h)],eax  mov eax,dword ptr [i (07FF71822D788h)]  mov dword ptr [k2 (07FF71822D76Ch)],eax  mov eax,dword ptr [i (07FF71822D788h)]  mov dword ptr [i4 (07FF71822D754h)],eax  imul eax,dword ptr [i (07FF71822D788h)],0  mov dword ptr [i5 (07FF71822D71Ch)],eax  mov eax,dword ptr [i (07FF764B5D788h)]  cdq  xor ecx,ecx  idiv eax,ecx  mov dword ptr [i2 (07FF764B5D774h)],eax  movsd xmm0,mmword ptr [flt\_1 (07FF764B5D748h)]  divsd xmm0,mmword ptr [\_\_real@0000000000000000 (07FF764B5AC88h)]  movsd mmword ptr [flt\_2 (07FF764B5D740h)],xmm0  movsd xmm0,mmword ptr [\_\_real@4003333333333333 (07FF764B5ABC8h)]  movsd mmword ptr [flt\_3 (07FF764B5D738h)],xmm0  movsd xmm0,mmword ptr [\_\_real@3ff000001ad7f29b (07FF764B5ABC0h)]  movsd mmword ptr [flt\_4 (07FF764B5D730h)],xmm0  movsd xmm0,mmword ptr [flt\_6 (07FF764B5D720h)]  mulsd xmm0,mmword ptr [\_\_real@0000000000000000 (07FF764B5AC88h)]  movsd mmword ptr [flt\_5 (07FF764B5D728h)],xmm0  movsd xmm0,mmword ptr [flt\_2 (07FF764B5D740h)]  mulsd xmm0,mmword ptr [flt\_3 (07FF764B5D738h)]  movsd mmword ptr [flt\_6 (07FF764B5D720h)],xmm0 | mov ecx,dword ptr [i (07FF79D026834h)]  mov edx,ebx  mov ecx,ebx  mov dword ptr [i2 (07FF79D026810h)],5  movsd xmm1,mmword ptr [\_\_real@3ff0000000000000 (07FF6F5354298h)]  xorps xmm0,xmm0  movsd xmm2,mmword ptr [\_\_real@4021666666666666 (07FF6F53542A8h)]  mov eax,ecx  divsd xmm1,xmm0  cdq  mov dword ptr [i4 (07FF6F5356838h)],ecx  mov ecx,esi  mov dword ptr [i3 (07FF6F5356668h)],3 | Исключаются расчеты, не меняющие значения переменной (i + 0; i / 1; i \* 1;) переменным сразу присваиваются конечные значения (i3 = 1 + 2; заменяется на i3=3). Значение переменной i помещается в регистр, для быстрого доступа и ускорения выполнения операций. Значение i5 не присваивается, тк оно не используется в дальнейшем. |
| Лишнее присваивание | k3 = 1; | mov dword ptr [k3 (07FF71822D75Ch)],1  mov dword ptr [k3 (07FF71822D75Ch)],1 | mov dword ptr [k3 (07FF6F535683Ch)],1 | Лишнее присваивание не выполняется |
| Снижение мощности | k2 = 4 \* j5;  for( i = 0; i <= 5; i++ ) ivector4[ i ] = i \* 2; | mov eax,dword ptr [j5 (07FF71822D70Ch)]  shl eax,2  mov dword ptr [k2 (07FF71822D76Ch)],eax  mov dword ptr [i (07FF71822D788h)],0  jmp main+185h (07FF718226375h)  mov eax,dword ptr [i (07FF71822D788h)]  inc eax  mov dword ptr [i (07FF71822D788h)],eax  cmp dword ptr [i (07FF71822D788h)],5  jg main+1AAh (07FF71822639Ah)  mov eax,dword ptr [i (07FF71822D788h)]  add eax,eax  movsxd rcx,dword ptr [i (07FF71822D788h)]  lea rdx,[ivector4 (07FF71822D700h)]  mov word ptr [rdx+rcx\*2],ax  jmp main+177h (07FF718226367h) | add cx,cx  movsxd rax,edx  mov word ptr [rdi+rax\*2+6820h],cx  lea ecx,[rdx+1]  mov edx,ecx  cmp ecx,5  jle main+100h (07FF6F53511E0h) | Умножение заменено сложением (shl на add).  Переменная i в заголовке цикла принимается за регистр счетчика cx. Значение i\*2 считается в начале цикла. Значение k2 больше не сохраняется, так как нигде далее оно не используется. |
| Простой цикл | j5 = 0;  k5 = 10000;  do {  k5 = k5 - 1;  j5 = j5 + 1;  i5 = (k5 \* 3) / (j5 \* constant5);  } while ( k5 > 0 ); | mov dword ptr [j5 (07FF71822D70Ch)],0  mov dword ptr [k5 (07FF71822D6FCh)],2710h  mov eax,dword ptr [k5 (07FF71822D6FCh)]  dec eax  mov dword ptr [k5 (07FF71822D6FCh)],eax  mov eax,dword ptr [j5 (07FF71822D70Ch)]  inc eax  mov dword ptr [j5 (07FF71822D70Ch)],eax  imul eax,dword ptr [k5 (07FF71822D6FCh)],3  imul ecx,dword ptr [j5 (07FF71822D70Ch)],5  cdq  idiv eax,ecx  mov dword ptr [i5 (07FF71822D71Ch)],eax  cmp dword ptr [k5 (07FF71822D6FCh)],0  jg main+1BEh (07FF7182263AEh) | Mov ecx,752Dh  mov ecx,2710h  mov r9d,5  sub r10d,3  add r9d,5  mov dword ptr [i5 (07FF691106654h)],eax  mov r11d,eax  test ecx,ecx  jg main+0E0h (07FF6911011C0h)  mov dword ptr [k5 (07FF69110664Ch)],ecx  mov edx,ebx  mov ecx,ebx  mov dword ptr [j5 (07FF691106830h)],r8d | Цикл был упрощен. Некоторые значения посчитаны перед циклом: 10000 и 29997 Обращение происходит через регистры. Некоторые операции были заменены упрощенными вариантами (cmp и test, например).  Выражение для i5 было заменено более оптимальным с математической точки зрения |
| Управление переменной индукции цикла | for( i = 0; i < 100; i++ )  ivector5[ i \* 2 + 3 ] = 5; | mov dword ptr [i (07FF71822D788h)],0  jmp main+214h (07FF718226404h)  mov eax,dword ptr [i (07FF71822D788h)]  inc eax  mov dword ptr [i (07FF71822D788h)],eax  cmp dword ptr [i (07FF71822D788h)],64h  jge main+239h (07FF718226429h)  mov eax,dword ptr [i (07FF71822D788h)]  lea eax,[rax+rax+3]  cdqe  lea rcx,[ivector5 (07FF71822D8E0h)]  mov dword ptr [rcx+rax\*4],5  jmp main+206h (07FF7182263F6h) | movsxd rax,edx  inc ecx  add edx,2  mov dword ptr [rdi+rax\*4+668Ch],5  cmp ecx,64h  jl main+112h (07FF6911011F2h) | Вместо переменной используется регистр ecx. Присваивание элемента массива в цикле осуществляется через прибавление смещения к адресу массива, смещение высчитано компилятором заранее. |
| Глубокие подвыражения | if( i < 10 )  j5 = i5 + i2;  else  k5 = i5 + i2; | cmp dword ptr [i (07FF71822D788h)],0Ah  jge main+25Ah (07FF71822644Ah)  mov eax,dword ptr [i2 (07FF71822D774h)]  mov ecx,dword ptr [i5 (07FF71822D71Ch)]  add ecx,eax  mov eax,ecx  mov dword ptr [j5 (07FF71822D70Ch)],eax  jmp main+270h (07FF718226460h)  mov eax,dword ptr [i2 (07FF71822D774h)]  mov ecx,dword ptr [i5 (07FF71822D71Ch)]  add ecx,eax  mov eax,ecx  mov dword ptr [k5 (07FF71822D6FCh)],eax | lea eax,[r11+5]  mov dword ptr [i (07FF691106834h)],ecx  mov dword ptr [k5 (07FF69110664Ch)],eax | Значение переменной i хранится в регистре ecx и считается в предыдущем цикле. Условие в if не проверяется, так как оно наверняка будет не верным. Переменной k5 присваивается значение, заранее высчитанное компилятором (5). |
| 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 eax,4  imul rax,rax,0  lea rcx,[ivector (07FF71822D710h)]  mov dword ptr [rcx+rax],1  movsxd rax,dword ptr [i2 (07FF71822D774h)]  lea rcx,[ivector (07FF71822D710h)]  mov dword ptr [rcx+rax\*4],2  movsxd rax,dword ptr [i2 (07FF71822D774h)]  lea rcx,[ivector (07FF71822D710h)]  mov dword ptr [rcx+rax\*4],2  mov eax,4  imul rax,rax,2  lea rcx,[ivector (07FF71822D710h)]  mov dword ptr [rcx+rax],3 | mov dword ptr [ivector  (09053C0h)], 1  mov dword ptr ds:  (9053D4h],2  mov dword ptr ds:  (9053C8h],3 | Убрано лишнее присваивания, а оставшиеся выполняются не на прямую по адресу, а по заранее вычисленному смещению. |
| 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 eax,dword ptr [k3 (07FF71822D75Ch)]  mov ecx,dword ptr [h3 (07FF71822D764h)]  add ecx,eax  mov eax,ecx  test eax,eax  jl main+2F1h (07FF7182264E1h)  mov eax,dword ptr [k3 (07FF71822D75Ch)]  mov ecx,dword ptr [h3 (07FF71822D764h)]  add ecx,eax  mov eax,ecx  cmp eax,5  jle main+2FFh (07FF7182264EFh)  lea rcx,[string "Common subexpression eliminatio@"... (07FF71822AC30h)]  call printf (07FF71822119Ah)  jmp main+33Ch (07FF71822652Ch)  mov eax,dword ptr [k3 (07FF71822D75Ch)]  mov ecx,dword ptr [h3 (07FF71822D764h)]  add ecx,eax  mov eax,ecx  cdq  idiv eax,dword ptr [i3 (07FF71822D760h)]  mov dword ptr [m3 (07FF71822D758h)],eax  mov eax,dword ptr [k3 (07FF71822D75Ch)]  mov ecx,dword ptr [h3 (07FF71822D764h)]  add ecx,eax  mov eax,ecx  mov ecx,dword ptr [i3 (07FF71822D760h)]  add ecx,eax  mov eax,ecx  mov dword ptr [g3 (07FF71822D768h)],eax  } | mov eax,dword ptr [h3 (07FF69110666Ch)]  inc eax  mov dword ptr [ivector (07FF691106660h)],1  mov dword ptr [h3+8h (07FF691106674h)],2  mov dword ptr [ivector+8h (07FF691106668h)],3  cmp eax,5  jbe main+171h (07FF691101251h)  lea rcx,[string "Common subexpression eliminatio@"... (07FF691104248h)]  call printf (07FF691101070h) | Общее выражение h3+k3 вычисляется заранее, сначала в регистр eax записываем значение h3, а затем прибавляем к значению регистра eax 1(тк k3==1). Для более быстрой работы используются регистры. |
| Вынесение инвариантного кода | for( i4 = 0; i4 <= max\_vector; i4++){  printf("Hello");  ivector2[ i4 ] = j \* k;  } | mov dword ptr [i4 (07FF71822D754h)],0  jmp main+356h (07FF718226546h)  mov eax,dword ptr [i4 (07FF71822D754h)]  inc eax  mov dword ptr [i4 (07FF71822D754h)],eax  cmp dword ptr [i4 (07FF71822D754h)],2  jg main+38Bh (07FF71822657Bh)  lea rcx,[string "Hello" (07FF71822AC24h)]  call printf (07FF71822119Ah)  mov eax,dword ptr [j (07FF71822D784h)]  imul eax,dword ptr [k (07FF71822D780h)]  movsxd rcx,dword ptr [i4 (07FF71822D754h)]  lea rdx,[ivector2 (07FF71822D6F8h)]  mov byte ptr [rdx+rcx],al | mov dword ptr [i4 (07FF691106838h)],ebx  lea rcx,[string "Hello" (07FF691104240h)]  call printf (07FF691101070h)  movsxd rdx,dword ptr [i4 (07FF691106838h)]  imul ecx,eax  inc edx  mov dword ptr [i4 (07FF691106838h)],edx  cmp edx,2  jle main+180h (07FF691101260h)  mov edx,dword ptr [j5 (07FF691106830h)]  mov ecx,ebx  mov eax,7 | Компилятор не выполнил оптимизацию этой части кода |
| Вызов функции с аргументами | dead\_code( 1, "This line should not be printed" ); | lea rdx,[string "This line should not be printed" (07FF71822AC58h)]  mov ecx,1  call dead\_code (07FF7182211A4h) | Не вызывается | Не вызывается |
| Вызов функции без аргументов | unnecessary\_loop(); | call unnecessary\_loop (07FF718221014h) | mov r8d,eax  mov dword ptr [k5 (07FF69110664Ch)],eax  add eax,edx  inc ecx  cmp ecx,5  jl main+1D0h (07FF6911012B0h) | Вызов функции был заменен на тело функции. Это уменьшает время выполнения программы (при компиляции функции для хранения передаваемых функции параметров, а также любых локальных переменных, используемых этой функцией, компилятор С использует стек, а когда происходит вызов функции, то в стек помещается еще и адрес возврата в вызывающую программу, поэтому благодаря применению линейного кода были исключены расходы на выполнение вызывающей и возвращающей последовательностей). |
| unnecessary\_loop  Цикл в следующей функции не нужен, так как значение присваивания постоянно. В идеале цикл должен быть удален. | void unnecessary\_loop()  {  int x;  x = 0;  for( i = 0; i < 5; i++ ) /\* Цикл не должен  генерироваться\*/  k5 = x + j5;  } | mov dword ptr [rbp],0  mov dword ptr [i (07FF634EFA768h)],0  jmp unnecessary\_loop+0A4h (07FF634EF18D4h)  mov eax,dword ptr [i (07FF634EFA768h)]  mov es,word ptr [rax]  add byte ptr [rbx-37FCFFB3h],cl  mov eax,ecx  mov dword ptr [k5 (07FF634EFA6DCh)],eax  jmp unnecessary\_loop+96h (07FF634EF18C6h) | mov ecx,dword ptr [j5 (07FF6B97E6830h)]  xor eax,eax  mov dword ptr [k5 (07FF6B97E664Ch)],ecx | Цикл был удален, присваиваемое значение переменной k5 постоянно. Так как х = 0, то k5 просто присваивается значение j5, посчитанное ранее. |
| Два цикла в этой функции имеют одинаковые заголовки и могут быть слиты в один | 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;  } /\* Конец loop\_jamming \*/ | mov dword ptr [i (07FF634EFA768h)],0  jmp loop\_jamming+0B8h (07FF634EF17C8h)  mov eax,dword ptr [i (07FF634EFA768h)]  inc eax  mov dword ptr [i (07FF634EFA768h)],eax  cmp dword ptr [i (07FF634EFA768h)],5  jge loop\_jamming+0DDh (07FF634EF17EDh)  mov eax,dword ptr [j5 (07FF634EFA6ECh)]  imul eax,dword ptr [i (07FF634EFA768h)]  mov ecx,dword ptr [x]  add ecx,eax  mov eax,ecx  mov dword ptr [k5 (07FF634EFA6DCh)],eax  jmp loop\_jamming+0AAh (07FF634EF17BAh)  mov dword ptr [i (07FF634EFA768h)],0  jmp loop\_jamming+0F7h (07FF634EF1807h)  mov eax,dword ptr [i (07FF634EFA768h)]  inc eax  mov dword ptr [i (07FF634EFA768h)],eax  cmp dword ptr [i (07FF634EFA768h)],5  jge loop\_jamming+119h (07FF634EF1829h)  mov eax,dword ptr [x]  imul eax,dword ptr [k5 (07FF634EFA6DCh)]  imul eax,dword ptr [i (07FF634EFA768h)]  mov dword ptr [i5 (07FF634EFA6FCh)],eax  jmp loop\_jamming+0E9h (07FF634EF17F9h) | inc ecx  mov r9d,edx  mov dword ptr [k5 (07FF6B97E664Ch)],r8d  add edx,r10d  cmp ecx,5  00007FF6B97E13C6 jl loop\_jamming+30h (07FF6B97E13B0h)  imul r8d,r8d,7  mov edx,eax  mov ecx,r8d  mov dword ptr [i5 (07FF6B97E6654h)],edx  inc eax  mov edx,ecx  add ecx,r8d  cmp eax,5  jl loop\_jamming+60h (07FF6B97E13E0h) | Циклы должны были слиться в один, так как имеют одинаковые заголовки. Однако слияния не произошло, поскольку переменная k5 для второго цикла высчитывается в первом. Была проведена работа с регистрами |
| Цикл в этой функции должен быть заменен тремя присваиваниями с использованием константной индексации массива или машинно-зависимыми командами для инициализации блока памяти. | void loop\_unrolling( x )  int x;  {  for( i = 0; i < 6; i++ )  ivector4[ i ] = 0;  } | push ebp  mov ebp,esp  sub esp,44h  push ebx  push esi  push edi  mov dword ptr [i (0C4A824h)],0  jmp loop\_unrolling+2Ch (0C41AAEh)  mov eax,dword ptr [i (0C4A824h)]  add eax,1  mov dword ptr [i (0C4A824h)],eax  cmp dword ptr [i (0C4A824h)],6  jge loop\_unrolling+5Ah (0C41AE5h)  mov eax,dword ptr [i (0C4A824h)]  shl eax,1  mov dword ptr [ebp-0C8h],eax  cmp dword ptr [ebp-0C8h],0Ch  jae loop\_unrolling+47h (0C41ACFh)  jmp loop\_unrolling+4Ch (0C41AD4h)  call \_\_\_report\_rangecheckfailure (0C41046h)  xor ecx,ecx  mov edx,dword ptr [ebp-0C8h]  mov word ptr ivector4 (0C4A790h)[edx],cx  jmp jmp loop\_unrolling+1Fh (0C41AA1h) | movsxd rcx,eax  add rcx,rcx  cmp rcx,0Ch  jae loop\_unrolling+56h (07FF6B97E1456h)  lea eax,[rdx+1]  mov word ptr [rcx+r9],r8w  mov dword ptr [i (07FF6B97E6834h)],eax  mov edx,eax  cmp eax,6  jl loop\_unrolling+30h (07FF6B97E1430h) | Эта оптимизация уменьшает количество итераций цикла за счёт того, что значение внешнего счётчика цикла rcx удваивается на каждой итерации. Это позволяет увеличить шаг выполнения цикла, то есть количество обрабатываемых элементов на каждой итерации, что ускоряет выполнение программы. Проверка на выход за пределы цикла также позволяет сократить количество итераций. |
| Эта программа полезна для демонстрации сжатия цепочки переходов. goto end\_1 может быть заменен на прямой переход на beg\_1. | int jump\_compression( i, j, k, l, m )  int i, j, k, l, m;  {  int p;  beg\_1:  p = 4;  goto beg\_2;  beg\_2:  goto beg\_3;  beg\_3:  printf("%d", p);  goto beg\_1;  } /\* Конец jump\_compression \*/ |  | mov edx,4  lea rcx,[string "%d" (07FF6220D4294h)]  call printf (07FF6220D1070h)  jmp $beg\_3 (07FF6220D13F0h) | Компилятор игнорирует цепочку меток, ссылаясь сразу на последнюю упомянутую метку. |

1. Скомпилируем полученный файл с оптимизацией по размеру и сравним размер выходных бинарных файлов.
2. Размер файла до оптимизации – 39,8кб (Рис.5)
3. Размера файла после оптимизации – 28,5кб (Рис.6)
4. Таким образом мы можем сделать вывод, что результат оптимизации файла с использованием компилятора является достаточно успешным. Исходный файл весил 39,8, а после оптимизации его размер сократился до 28,5, что составляет снижение размера на 28% (примерно в 1,5 раза).
5. Это говорит о том, что оптимизация привела к удалению или сокращению ненужного, или избыточного кода и данных, что улучшает эффективность работы программы и снижает объем занимаемой памяти.
6. 
7. Рис. 5 «Размер файла до оптимизации»



1. Рис.6 «Размер файла после оптимизации»

Нами было замечено, что при оптимизации по размеру количество строк кода в некоторых местах сильно сократилось (Рис. 7 и 8). Это может негативно повлиять на читаемость кода в ассемблерном виде. Тем не менее, все комментарии остались на своих местах и не были удалены.

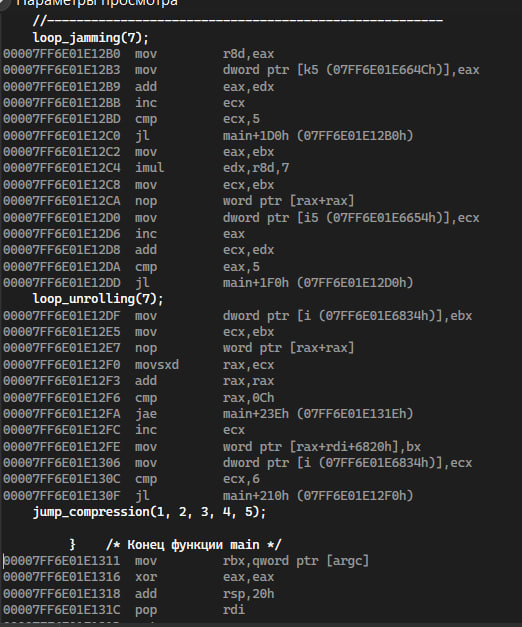


Рис.7 «Часть кода при оптимизации по скорости»

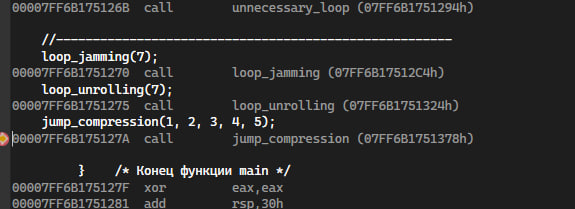


Рис.8 «Идентичная часть кода при оптимизации по размеру»

**4. Вывод**

В ходе исследования был проанализирован ассемблерный код двух программ - неоптимизированной и оптимизированной. Сравнение показало, что оптимизированная версия работает быстрее и может потреблять меньше памяти в некоторых случаях.

Компилятор Visual Studio 2022 пытается максимально использовать регистры для временных переменных и промежуточных результатов, что может значительно сократить время выполнения программы.

Кроме того, компилятор Visual Studio 2022 проводит оптимизации на уровне кода, просматривая программу и применяя различные трансформации и оптимизации. Например, компилятор может удалять лишние присваивания, заменять конструкции на более эффективные альтернативы, удалять недостижимый код и сливать или развертывать циклы для повышения эффективности выполнения.

В целом, компилятор Visual Studio 2022 предоставляет мощные возможности для оптимизации времени выполнения и размера программы, которые могут привести к улучшению производительности и эффективности программы.