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

**ЛАБОРАТОРНАЯ РАБОТА № 1**

1. «**Оптимизация программы**»
2. по дисциплине «Языки программирования»
3. Выполнил
4. студент гр. №4851003/00002 Скрипко И.А.

<*подпись*>

1. Преподаватель
2. Малышев Е.В.

<*подпись*>

1. Санкт-Петербург
2. 2021
3. **Цель работы**

Изучить принципы работы оптимизирующих компиляторов.

1. **Постановка задачи**
2. Скомпилировать файл, выданный преподавателем, без оптимизаций. Для компилируемого файла получить листинг на языке ассемблер;
3. Скомпилировать полученный файл с оптимизацией по скорости, для компилируемого файла получить листинг на языке ассемблер;
4. Сравнить листинги оптимизированного и неоптимизированного файлов, сделать выводы о примененных оптимизациях;
5. Скомпилировать полученный файл с оптимизацией по размеру и сравнить размеры выходных файлов (без оптимизации по размеру и с оптимизацией).
6. **Ход работы**

В данной работе будет использоваться компилятор Microsoft Visual C++ (MSVC). Параметры компилятора можно задать для каждого проекта в диалоговом окне страницы свойств Visual Studio (свойства конфигурации, C/C++ , параметр компилятора).

Был скомпилирован файл OPTBENCH.c, для данного кода был получен листинг на языке ассемблер. Для этого была поставлена точка остановы и запущено компилирование. Во время компилирования программы на панели в «откладка/окна» был выбран «дизассемблированный код». Далее в параметрах компилятора был установлен флаг «O2» для оптимизации кода по скорости. Параметры компилятора с включенной оптимизацией по скорости показаны на рисунке 1.

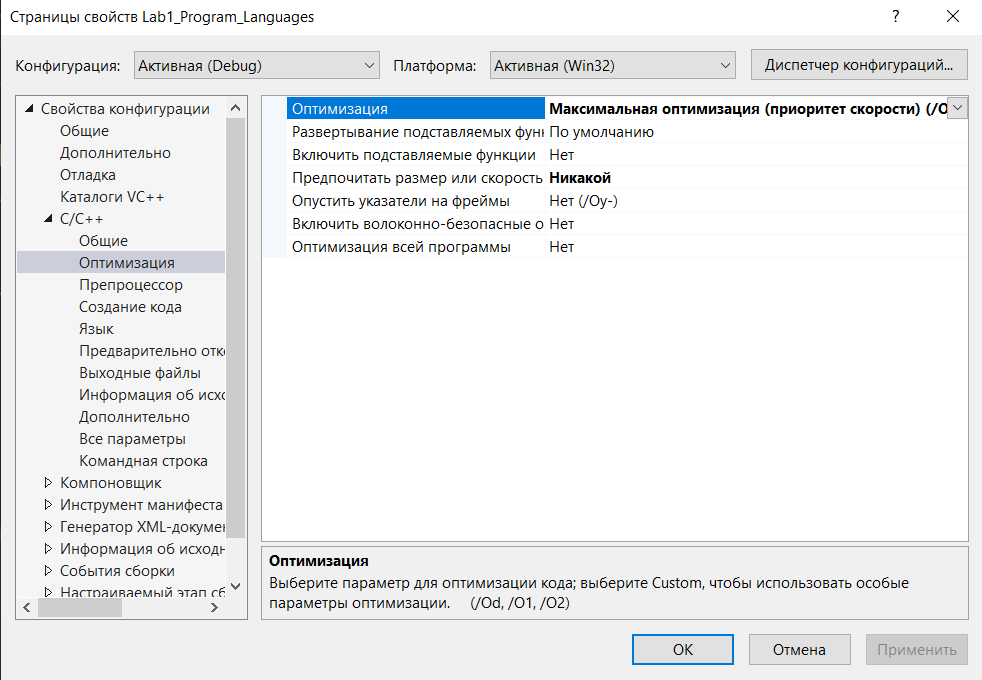


Рисунок 1 – Параметры компилятора с включенной оптимизацией по скорости

Далее были сравнены полученные листинги. Результаты сравнений представлены в таблице 1.

Таблица 1 – Сравнение листингов оптимизированной по скорости программы и неоптимизированной

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Оптими-зация** | **Код на языке С** | **Неоптимизированный код** | **Оптимизированный код** | **Комментарий** |
| Отказ от циклов | for (i = 0; i < 3; i++) ivector[i] = 1; | 00CE1B38 mov dword ptr [\_i (0CEA4F4h)],0  00CE1B42 jmp main+41h (0CE1B51h)  00CE1B44 mov eax,dword ptr [\_i (0CEA4F4h)]  00CE1B49 add eax,1  00CE1B4C mov dword ptr [\_i (0CEA4F4h)],eax  00CE1B51 cmp dword ptr [\_i (0CEA4F4h)],3  00CE1B58 jge main+5Ch (0CE1B6Ch)  00CE1B5A mov eax,dword ptr [\_i (0CEA4F4h)]  00CE1B5F mov dword ptr \_ivector (0CEA480h)[eax\*4],1  00CE1B6A jmp main+34h (0CE1B44h) | 00AC1B20 mov dword ptr \_ivector (0ACA480h)[eax\*4],1  00AC1B2B inc eax  00AC1B2C cmp eax,3  00AC1B2F jl main+10h (0AC1B20h)  00AC1B37 mov dword ptr [\_i (0ACA4F4h)],eax | Не используется переменная i, из-за чего не нужно присваивать ей каждый раз новое значение. В качестве i используется значение из регистра eax напрямую. Потом i один раз присваивается значение из регистра eax, то есть i получает только конечное значение |
| Перепри-сваивание | i2 = 5;  j4 = 6;  i2 = j4; | 00CE1B6C mov dword ptr [\_i2 (0CEA4E0h)],5  00CE1B76 mov dword ptr [\_j4 (0CEA4BCh)],6  00CE1B80 mov eax,dword ptr [\_j4 (0CEA4BCh)]  00CE1B85 mov dword ptr [\_i2 (0CEA4E0h)],eax | 00AC1B3C mov dword ptr [\_i2 (0ACA4E0h)],6 | Не происходит лишних присваиваний. Сразу присваивается итоговое значение переменной i2 |
| Размноже-ние 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");  } | 00CE1B8A mov dword ptr [\_j4 (0CEA4BCh)],2  00CE1B94 mov eax,dword ptr [\_i2 (0CEA4E0h)]  00CE1B99 cmp eax,dword ptr [\_j4 (0CEA4BCh)]  00CE1B9F jge main+0B5h (0CE1BC5h)  00CE1BA1 mov eax,dword ptr [\_i4 (0CEA4C0h)]  00CE1BA6 cmp eax,dword ptr [\_j4 (0CEA4BCh)]  00CE1BAC jge main+0B5h (0CE1BC5h)  00CE1BAE mov dword ptr [\_i2 (0CEA4E0h)],2  00CE1BB8 push offset string "Hello" (0CE7B30h)  00CE1BBD call \_printf (0CE1046h)  00CE1BC2 add esp,4  00CE1BC5 mov eax,dword ptr [\_k5 (0CEA47Ch)]  00CE1BCA mov dword ptr [\_j4 (0CEA4BCh)],eax  00CE1BCF mov eax,dword ptr [\_i2 (0CEA4E0h)]  00CE1BD4 cmp eax,dword ptr [\_j4 (0CEA4BCh)]  00CE1BDA jge main+0F0h (0CE1C00h)  00CE1BDC mov eax,dword ptr [\_i4 (0CEA4C0h)]  00CE1BE1 cmp eax,dword ptr [\_j4 (0CEA4BCh)]  00CE1BE7 jge main+0F0h (0CE1C00h)  00CE1BE9 mov dword ptr [\_i5 (0CEA4B8h)],3  00CE1BF3 push offset string "Hello" (0CE7B30h)  00CE1BF8 call \_printf (0CE1046h)  00CE1BFD add esp,4 | 00AC1B31 mov ecx,dword ptr [\_k5 (0ACA47Ch)]  00AC1B46 mov dword ptr [\_j4 (0ACA4BCh)],ecx  00AC1B4C cmp ecx,6  00AC1B4F jle main+65h (0AC1B75h)  00AC1B51 cmp dword ptr [\_i4 (0ACA4C0h)],ecx  00AC1B57 jge main+65h (0AC1B75h)  00AC1B59 push offset string "Hello" (0AC7B30h)  00AC1B5E mov dword ptr [\_i5 (0ACA4B8h)],3  00AC1B68 call \_printf (0AC1046h)  00AC1B6D mov eax,dword ptr [\_i (0ACA4F4h)]  00AC1B72 add esp,4 | Первое условие не будет истинным, и оно не проверяется, действия блока if не выполняются. j4 до первого if не меняет значение. В проверке второго условия сравнивается сначала значение в регистре с числом, а не регистр с переменной. Во втором условии не происходит лишних присваиваний, так как в регистре ecx уже лежит нужное значение (ранее из этого регистра присваивалось значение к j4). |
| Свертка констант, арифмети-ческие 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; | 00CE1C00 mov dword ptr [\_i3 (0CEA4CCh)],3  00CE1C0A movsd xmm0,mmword ptr [\_\_real@4021666666666666 (0CE7B90h)]  00CE1C12 movsd mmword ptr [\_flt\_1 (0CEA4B0h)],xmm0  00CE1C1A mov dword ptr [\_i2 (0CEA4E0h)],5  00CE1C24 mov eax,dword ptr [\_i (0CEA4F4h)]  00CE1C29 mov dword ptr [\_j2 (0CEA4DCh)],eax  00CE1C2E mov eax,dword ptr [\_i (0CEA4F4h)]  00CE1C33 mov dword ptr [\_k2 (0CEA4D8h)],eax  00CE1C38 mov eax,dword ptr [\_i (0CEA4F4h)]  00CE1C3D mov dword ptr [\_i4 (0CEA4C0h)],eax  00CE1C42 imul eax,dword ptr [\_i (0CEA4F4h)],0  00CE1C49 mov dword ptr [\_i5 (0CEA4B8h)],eax | 00AC1B91 mov dword ptr [\_i3 (0ACA4CCh)],3  00AC1B75 movsd xmm0,mmword ptr [\_\_real@4021666666666666(0AC7B90h)]  00AC1B9B movsd mmword ptr [\_flt\_1 (0ACA4B0h)],xmm0  00AC1BA3 mov dword ptr [\_i2 (0ACA4E0h)],5  00AC1B7D mov dword ptr [\_j2 (0ACA4DCh)],eax  00AC1B82 mov dword ptr [\_i4 (0ACA4C0h)],eax | k2 и i5 не принимают значения, так как в последствии не будут использоваться и просто поменяют значение снова – переприсваивание. В eax уже лежит i, поэтому нет смысла ее снова туда класть. Также операции +0, /1 и \*1 не меняют i, поэтому в регистр не перезаписывается значение и j2 и i4 просто принимают значение из регистра eax. |
| Лишнее присваивание | k3 = 1;  k3 = 1; | 00CE1C4E mov dword ptr [\_k3 (0CEA4C8h)],1  00CE1C58 mov dword ptr [\_k3 (0CEA4C8h)],1 | 00AC1BAD mov dword ptr [\_k3 (0ACA4C8h)],1 | Не происходит лишнего присваивания |
| Снижение мощности | k2 = 4 \* j5;  for (i = 0; i <= 5; i++) ivector4[i] = i \* 2; | 00CE1C62 mov eax,dword ptr [\_j5 (0CEA48Ch)]  00CE1C67 shl eax,2  00CE1C6A mov dword ptr [\_k2 (0CEA4D8h)],eax  00CE1C6F mov dword ptr [\_i (0CEA4F4h)],0  00CE1C79 jmp main+178h (0CE1C88h)  00CE1C7B mov eax,dword ptr [\_i (0CEA4F4h)]  00CE1C80 add eax,1  00CE1C83 mov dword ptr [\_i (0CEA4F4h)],eax  00CE1C88 cmp dword ptr [\_i (0CEA4F4h)],5  00CE1C8F jg main+198h (0CE1CA8h)  00CE1C91 mov eax,dword ptr [\_i (0CEA4F4h)]  00CE1C96 shl eax,1  00CE1C98 mov ecx,dword ptr [\_i (0CEA4F4h)]  00CE1C9E mov word ptr \_ivector4 (0CEA470h)[ecx\*2],eax  00CE1CA6 jmp main+16Bh (0CE1C7Bh) | 00AC1B87 mov eax,dword ptr [\_j5 (0ACA48Ch)]  00AC1B8C shl eax,2  00AC1B8F xor ecx,ecx  00AC1BB7 mov dword ptr [\_k2 (0ACA4D8h)],eax  00AC1BBC nop dword ptr [eax]  00AC1BC0 lea eax,[ecx+ecx]  00AC1BC3 mov word ptr \_ivector4 (0ACA470h)[ecx\*2],eax  00AC1BCB inc ecx  00AC1BCC cmp ecx,5  00AC1BCF jle main+0B0h (0AC1BC0h) | Вместо mov используется lea, что быстрее. Вместо операции умножения на 2 в цикле используется операция сложения, что быстрее. Вместо операции сложения числа с единицей используется операция инкремент, что тоже быстрее. Также в качестве переменных в основном используются регистры, а с ними работа происходит быстрее. Таким образом, были заменены неэффективные по времени операции на эффективные. |
| Простой цикл | j5 = 0;  k5 = 10000;  do {  k5 = k5 - 1;  j5 = j5 + 1;  i5 = (k5 \* 3) / (j5 \* constant5);  } while (k5 > 0); | 00CE1CA8 mov dword ptr [\_j5 (0CEA48Ch)],0  00CE1CB2 mov dword ptr [\_k5 (0CEA47Ch)],2710h  00CE1CBC mov eax,dword ptr [\_k5 (0CEA47Ch)]  00CE1CC1 sub eax,1  00CE1CC4 mov dword ptr [\_k5 (0CEA47Ch)],eax  00CE1CC9 mov eax,dword ptr [\_j5 (0CEA48Ch)]  00CE1CCE add eax,1  00CE1CD1 mov dword ptr [\_j5 (0CEA48Ch)],eax  00CE1CD6 imul eax,dword ptr [\_k5 (0CEA47Ch)],3  00CE1CDD imul ecx,dword ptr [\_j5 (0CEA48Ch)],5  00CE1CE4 cdq  00CE1CE5 idiv eax,ecx  00CE1CE7 mov dword ptr [\_i5 (0CEA4B8h)],eax  00CE1CEC cmp dword ptr [\_k5 (0CEA47Ch)],0  00CE1CF3 jg main+1ACh (0CE1CBCh) | 00AC1BD1 push ebx  00AC1BD2 push esi  00AC1BD3 xor esi,esi  00AC1BD5 mov ecx,752Dh  00AC1BDA push edi  00AC1BDB mov edi,2710h  00AC1BE0 lea ebx,[esi+5]  00AC1BE3 mov eax,ecx  00AC1BE5 dec edi  00AC1BE6 sub ecx,3  00AC1BE9 cdq  00AC1BEA mov dword ptr [ebp-4],ecx  00AC1BED inc esi  00AC1BEE mov ecx,ebx  00AC1BF0 add ebx,5  00AC1BF3 idiv eax,ecx  00AC1BF5 mov ecx,dword ptr [ebp-4]  00AC1BF8 mov dword ptr [\_i5 (0ACA4B8h)],eax  00AC1BFD test edi,edi  00AC1BFF jg main+0D3h (0AC1BE3h) | Были заранее высчитаны значения (k5-1)\*3 (=ecx=752D) и (j+1)\*constant5 (=edi=2710). Далее идет цикл в котором не высчитываются эти значения каждый раз по новой. Теперь в цикле каждый раз первое значение уменьшается на 3, а второе увеличивается на 5 .j5 и k5 впоследствии не будут использоваться, но потом снова поменяют значения, поэтому на данном этапе им не присваивается никаких значений. Значение переменной k хранится в регистре и вместо операции вычитания используется операция декремента. Значение j5 хранится в регистре и для увеличения используется инкремент, а не сложение. |
| Управление переменной индукции цикла | for (i = 0; i < 100; i++) ivector5[i \* 2 + 3] = 5; | 00CE1CF5 mov dword ptr [\_i (0CEA4F4h)],0  00CE1CFF jmp main+1FEh (0CE1D0Eh)  00CE1D01 mov eax,dword ptr [\_i (0CEA4F4h)]  00CE1D06 add eax,1  00CE1D09 mov dword ptr [\_i (0CEA4F4h)],eax  00CE1D0E cmp dword ptr [\_i (0CEA4F4h)],64h  00CE1D15 jge main+21Bh (0CE1D2Bh)  00CE1D17 mov eax,dword ptr [\_i (0CEA4F4h)]  00CE1D1C shl eax,1  00CE1D1E mov dword ptr [eax\*4+0CEA58Ch],5  00CE1D29 jmp main+1F1h (0CE1D01h) | 00AC1C01 pop edi  00AC1C08 xor ecx,ecx  00AC1C0A pop esi  00AC1C0B pop ebx  00AC1C0C nop dword ptr [eax]  00AC1C10 mov dword ptr [ecx\*8+0ACA58Ch],5  00AC1C1B inc ecx  00AC1C1C cmp ecx,64h  00AC1C1F jl main+100h (0AC1C10h)  00AC1C21 mov dword ptr [\_i (0ACA4F4h)],ecx | После прошлого цикла в регистре ecx лежит значение 0, данный регистр будет использоваться вместо i. Теперь при указании адреса, куда сохранить значение 5, используется умножение на 8. Таким образом, адрес будет нужным, но перед ним не будет происходить лишнего бинарного сдвига. Вместо операции сложения используется операция инкремента. В конце i один раз принимает значение (итоговое). |
| Глубокие подвыражения | if (i < 10)  j5 = i5 + i2;  else  k5 = i5 + i2; | 00CE1D2B cmp dword ptr [\_i (0CEA4F4h)],0Ah  00CE1D32 jge main+236h (0CE1D46h)  00CE1D34 mov eax,dword ptr [\_i5 (0CEA4B8h)]  00CE1D39 add eax,dword ptr [\_i2 (0CEA4E0h)]  00CE1D3F mov dword ptr [\_j5 (0CEA48Ch)],eax  00CE1D44 jmp main+246h (0CE1D56h)  00CE1D46 mov eax,dword ptr [\_i5 (0CEA4B8h)]  00CE1D4B add eax,dword ptr [\_i2 (0CEA4E0h)]  00CE1D51 mov dword ptr [\_k5 (0CEA47Ch)],eax | 00AC1C27 add eax,5  00AC1C31 mov dword ptr [\_k5 (0ACA47Ch)],eax | В eax лежит значение i5, значение i2 на данный момент 5, поэтому к eax добавляется 5. Проверка условий не выполняется, сразу выполняется нужное действие. |
| 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; | 00CE1D56 mov eax,4  00CE1D5B imul ecx,eax,0  00CE1D5E mov dword ptr \_ivector (0CEA480h)[ecx],1  00CE1D68 mov eax,dword ptr [\_i2 (0CEA4E0h)]  00CE1D6D mov dword ptr \_ivector (0CEA480h)[eax\*4],2  00CE1D78 mov eax,dword ptr [\_i2 (0CEA4E0h)]  00CE1D7D mov dword ptr \_ivector (0CEA480h)[eax\*4],2  00CE1D88 mov eax,4  00CE1D8D shl eax,1  00CE1D8F mov dword ptr \_ivector (0CEA480h)[eax],3 | 00AC1C36 mov dword ptr [\_ivector (0ACA480h)],1  00AC1C40 mov dword ptr ds:[0ACA494h],2  00AC1C4A mov dword ptr ds:[0ACA488h],3 | При генерировании константного индекса в регистр кладется число 4, потом преобразуется. В данных случаях оно преобразовывалось с помощью операций умножения и бинарного сдвига. В оптимизированной коде сразу указывается адрес, куда нужно положить значение. |
| Удаление общих подвыражений | if ((h3 + k3) < 0 || (h3 + k3) > 5) printf("Common subexpression elimination\n");  else {  m3 = (h3 + k3) / i3;  g3 = i3 + (h3 + k3);  } | 00CE1D99 mov eax,dword ptr [\_h3 (0CEA4D0h)]  00CE1D9E add eax,dword ptr [\_k3 (0CEA4C8h)]  00CE1DA4 js main+2A6h (0CE1DB6h)  00CE1DA6 mov eax,dword ptr [\_h3 (0CEA4D0h)]  00CE1DAB add eax,dword ptr [\_k3 (0CEA4C8h)]  00CE1DB1 cmp eax,5  00CE1DB4 jle main+2B5h (0CE1DC5h)  00CE1DB6 push offset string "Common subexpression eliminatio@"... (0CE7B38h)  00CE1DBB call \_printf (0CE1046h)  00CE1DC0 add esp,4  00CE1DC3 jmp main+2E2h (0CE1DF2h)  00CE1DC5 mov eax,dword ptr [\_h3 (0CEA4D0h)]  00CE1DCA add eax,dword ptr [\_k3 (0CEA4C8h)]  00CE1DD0 cdq  00CE1DD1 idiv eax,dword ptr [\_i3 (0CEA4CCh)]  00CE1DD7 mov dword ptr [\_m3 (0CEA4C4h)],eax  00CE1DDC mov eax,dword ptr [\_h3 (0CEA4D0h)]  00CE1DE1 add eax,dword ptr [\_k3 (0CEA4C8h)]  00CE1DE7 add eax,dword ptr [\_i3 (0CEA4CCh)]  00CE1DED mov dword ptr [\_g3 (0CEA4D4h)],eax | 00AC1C2A mov ecx,dword ptr [\_h3 (0ACA4D0h)]  00AC1C30 inc ecx  00AC1C54 cmp ecx,5  00AC1C57 ja main+166h (0AC1C76h)  00AC1C59 mov eax,55555556h  00AC1C5E imul ecx  00AC1C60 mov eax,edx  00AC1C62 shr eax,1Fh  00AC1C65 add eax,edx  00AC1C67 mov dword ptr [\_m3 (0ACA4C4h)],eax  00AC1C6C lea eax,[ecx+3]  00AC1C6F mov dword ptr [\_g3 (0ACA4D4h)],eax  00AC1C74 jmp main+173h (0AC1C83h)  00AC1C76 push offset string "Common subexpression eliminatio@"... (0AC7B38h)  00AC1C7B call\_printf (0AC1046h)  00AC1C80 add esp,4 | k3 равен 1, поэтому вместо операции сложения двух переменных сохраняется в регистр значение h3, после чего делается инкремент. Значение h3+k3 не считается по несколько раз. Оно хранится в регистре ecx и используется в нужных местах. |
| Вынесение инвариантного кода (j \* k) может быть вынесено из цикла | for (i4 = 0; i4 <= max\_vector; i4++) { printf("Hello"); ivector2[i4] = j \* k;  } | 00CE1DF2 mov dword ptr [\_i4 (0CEA4C0h)],0  00CE1DFC jmp main+2FBh (0CE1E0Bh)  00CE1DFE mov eax,dword ptr [\_i4 (0CEA4C0h)]  00CE1E03 add eax,1  00CE1E06 mov dword ptr [\_i4 (0CEA4C0h)],eax  00CE1E0B cmp dword ptr [\_i4 (0CEA4C0h)],2  00CE1E12 jg main+32Bh (0CE1E3Bh)  00CE1E14 push offset string "Hello" (0CE7B30h)  00CE1E19 call \_printf (0CE1046h)  00CE1E1E add esp,4  00CE1E21 mov eax,dword ptr [\_j (0CEA4F0h)]  00CE1E26 imul eax,dword ptr [\_k (0CEA4ECh)]  00CE1E2D mov ecx,dword ptr [\_i4 (0CEA4C0h)]  00CE1E33 mov byte ptr \_ivector2 (0CEA45Ch)[ecx],al  00CE1E39 jmp main+2EEh (0CE1DFEh) | 00AC1C83 mov dword ptr [\_i4 (0ACA4C0h)],0  00AC1C8D nop dword ptr [eax]  00AC1C90 push offset string "Hello" (0AC7B30h)  00AC1C95 call \_printf (0AC1046h)  00AC1C9A movzx eax,byte ptr [\_k (0ACA4ECh)]  00AC1CA1 add esp,4  00AC1CA4 movzx ecx,byte ptr [\_j (0ACA4F0h)]  00AC1CAB imul ecx,eax  00AC1CAE mov eax,dword ptr [\_i4 (0ACA4C0h)]  00AC1CB3 mov byte ptr \_ivector2 (0ACA45Ch)[eax],cl  00AC1CB9 inc eax  00AC1CBA mov dword ptr [\_i4 (0ACA4C0h)],eax  00AC1CBF cmp eax,2  00AC1CC2 jle main+180h (0AC1C90h) | Не происходит первого сравнения. Вместо сложения с 1 используется инкремент. В данном случае j\*k не меняется сколько бы раз не выполнялся цикл и их можно было бы вынести, но этого не произошло при оптимизации. |
| Вызов функции с аргументами | dead\_code(1, "This line should not be printed");  void dead\_code(a, b)  int a;  char\* b;  {  int idead\_store; idead\_store = a;  if (0) printf("%s\n", b);  } | 00AC1CC4 push offset string "This line should not be printed"  00AC1CC9 push 1  00AC1CCB call \_dead\_code (0AC117Ch)  00CE1E47 add esp,8  000D186D clc  000D186E xor eax,eax  000D1870 je dead\_code+43h (0D1883h)  000D1872 mov eax,dword ptr [b]  000D1875 push eax  000D1876 push offset string "%s\n" (0D7B5Ch)  000D187B call \_printf (0D1046h)  000D1880 add esp,8  000D1896 ret | 00AC1CC4 push offset string "This line should not be printed"  00AC1CC9 push 1  00AC1CCB call \_dead\_code (0AC117Ch) | Код функции не генерируется, ибо она никогда не станет что-либо делать, ведь там условие всегда будет ложным. |
| Вызов функции без аргументов | unnecessaryloop();  void unnecessaryloop()  {  int x;  x = 0;  for (i = 0; i < 5; i++)  k5 = x + j5;  } | 00CE1E4A call \_unnecessary\_loop  00F72010 push ebp  00F72011 mov ebp,esp  00F72013 sub esp,0CCh  00F72019 push ebx  00F7201A push esi  00F7201B push edi  00F7201C lea edi,[ebp-0CCh]  00F72022 mov ecx,33h  00F72027 mov eax,0CCCCCCCCh  00F7202C rep stos dword ptr es:[edi]  00F7202E mov ecx,offset \_03E453DF\_lab1@c (0F7C006h)  00F72033 call @\_\_CheckForDebuggerJustMyCode@4 (0F7122Bh)  00F72038 mov dword ptr [x],0  00F7203F mov dword ptr [\_i (0F7A4F4h)],0  00F72049 jmp unnecessary\_loop+48h (0F72058h)  00F7204B mov eax,dword ptr [\_i (0F7A4F4h)]  00F72050 add eax,1  00F72053 mov dword ptr [\_i (0F7A4F4h)],eax  00F72058 cmp dword ptr [\_i (0F7A4F4h)],5  00F7205F jge unnecessary\_loop+61h (0F72071h)  00F72061 mov eax,dword ptr [x]  00F72064 add eax,dword ptr [\_j5 (0F7A48Ch)]  00F7206A mov dword ptr [\_k5 (0F7A47Ch)],eax  00F7206F jmp unnecessary\_loop+3Bh (0F7204Bh)  00F72071 pop edi  00F72072 pop esi  00F72073 pop ebx  00F72074 add esp,0CCh  00F7207A cmp ebp,esp  00F7207C call \_\_RTC\_CheckEsp (0F71235h)  00F72081 mov esp,ebp  00F72083 pop ebp  00F72084 ret | 00AC1CD0 call \_unnecessary\_loop  00582010 mov ecx,58C006h  00582015 call @\_\_CheckForDebuggerJustMyCode@4 (058122Bh)  0058201A mov eax,dword ptr [\_j5 (058A48Ch)]  0058201F mov dword ptr [\_k5 (058A47Ch)],eax  00582024 mov dword ptr [\_i (058A4F4h)],5  0058202E ret | Не происходит работы с переменной x, так как она равна 0 и добавлять 0 нет смысла. Цикл не выполняется, переменная k5 один раз принимает нужное значение. i сразу принимает итоговое значение. |
| Вызов функции с аргументами | loop\_jamming(7);  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;  } | 00CE1E4F push 7  00CE1E51 call \_loop\_jamming  00CE1E56 add esp,4  00781980 push ebp  00781981 mov ebp,esp  00781983 sub esp,0C0h  00781989 push ebx  0078198A push esi  0078198B push edi  0078198C lea edi,[ebp-0C0h]  00781992 mov ecx,30h  00781997 mov eax,0CCCCCCCCh  0078199C rep stos dword ptr es:[edi]  0078199E mov ecx,offset \_03E453DF\_lab1@c (078C006h)  007819A3 call @\_\_CheckForDebuggerJustMyCode@4 (078122Bh)  007819A8 mov dword ptr [\_i (078A4F4h)],0  007819B2 jmp loop\_jamming+41h (07819C1h)  007819B4 mov eax,dword ptr [\_i (078A4F4h)]  007819B9 add eax,1  007819BC mov dword ptr [\_i (078A4F4h)],eax  007819C1 cmp dword ptr [\_i (078A4F4h)],5  007819C8 jge loop\_jamming+60h (07819E0h)  007819CA mov eax,dword ptr [\_j5 (078A48Ch)]  007819CF imul eax,dword ptr [\_i (078A4F4h)]  007819D6 add eax,dword ptr [x]  007819D9 mov dword ptr [\_k5 (078A47Ch)],eax  007819DE jmp loop\_jamming+34h (07819B4h)  007819E0 mov dword ptr [\_i (078A4F4h)],0  007819EA jmp loop\_jamming+79h (07819F9h)  007819EC mov eax,dword ptr [\_i (078A4F4h)]  007819F1 add eax,1  007819F4 mov dword ptr [\_i (078A4F4h)],eax  007819F9 cmp dword ptr [\_i (078A4F4h)],5  00781A00 jge loop\_jamming+9Ah (0781A1Ah)  00781A02 mov eax,dword ptr [x]  00781A05 imul eax,dword ptr [\_k5 (078A47Ch)]  00781A0C imul eax,dword ptr [\_i (078A4F4h)]  00781A13 mov dword ptr [\_i5 (078A4B8h)],eax  00781A18 jmp loop\_jamming+6Ch (07819ECh)  00781A1A pop edi  00781A1B pop esi  00781A1C pop ebx  00781A1D add esp,0C0h  00781A23 cmp ebp,esp  00781A25 call \_\_RTC\_CheckEsp (0781235h)  00781A2A mov esp,ebp  00781A2C pop ebp  00781A2D ret | 00AC1CD5 push 7  00AC1CD7 call \_loop\_jamming  00000 mov edx, DWORD PTR \_j5  00006 mov eax, 7  0000b push esi  0000c lea ecx, DWORD PTR [eax-2]  0000f npad 1  00010 mov esi, eax  00012 mov DWORD PTR \_k5, eax  00017 add eax, edx  00019 sub ecx, 1  0001c jne SHORT  0001e lea edx, DWORD PTR [esi\*8]  00025 xor eax, eax  00027 mov ecx, 5  0002c sub edx, esi  0002e mov DWORD PTR \_i, ecx  00034 pop esi  00035 mov DWORD PTR \_i5, eax  0003a add eax, edx  0003c sub ecx, 1  0003f jne SHORT  00041 ret 0 | Вместо объединения действий циклов программа кладет итоговые значения переменных k5 и i5 в их ячейки памяти, то есть их значения при i=4. Также присваивает i ее итоговое значение – 4. |
| Вызов функции с аргументами | loop\_unrolling(7);  void loop\_unrolling(x)  int x;  {  for (i = 0; i < 6; i++) ivector4[i] = 0;  } | 00CE1E59 push 7  00CE1E5B call \_loop\_unrolling  00CE1E60 add esp,4    000D1A64 in al,dx  000D1A65 int 3  000D1A66 add byte ptr [eax],al  000D1A68 add byte ptr [ebx+56h],dl  000D1A6B push edi  000D1A6C lea edi,[ebp-0CCh]  000D1A72 mov ecx,33h  000D1A77 mov eax,0CCCCCCCCh  000D1A7C rep stos dword ptr es:[edi]  000D1A7E mov ecx,offset \_03E453DF\_lab1@c (0DC006h)  000D1A83 call @\_\_CheckForDebuggerJustMyCode@4 (0D122Bh)  000D1A88 mov dword ptr [\_i (0DA4F4h)],0  000D1A92 jmp loop\_unrolling+41h (0D1AA1h)  000D1A94 mov eax,dword ptr [\_i (0DA4F4h)]  000D1A99 add eax,1  000D1A9C mov dword ptr [\_i (0DA4F4h)],eax  000D1AA1 cmp dword ptr [\_i (0DA4F4h)],6  000D1AA8 jge loop\_unrolling+78h (0D1AD8h)  000D1AAA mov eax,dword ptr [\_i (0DA4F4h)]  000D1AAF shl eax,1  000D1AB1 mov dword ptr [ebp-0C8h],eax  000D1AB7 cmp dword ptr [ebp-0C8h],0Ch  000D1ABE jae loop\_unrolling+62h (0D1AC2h)  000D1AC0 jmp loop\_unrolling+67h (0D1AC7h)  000D1AC2 call \_\_\_report\_rangecheckfailure (0D1159h)  000D1AC7 xor ecx,ecx  000D1AC9 mov edx,dword ptr [ebp-0C8h]  000D1ACF mov word ptr \_ivector4 (0DA470h)[edx],cx  000D1AD6 jmp loop\_unrolling+34h (0D1A94h)  000D1AD8 pop edi  000D1AD9 pop esi  000D1ADA pop ebx  000D1ADB add esp,0CCh  000D1AE1 cmp ebp,esp  000D1AE3 call \_\_RTC\_CheckEsp (0D1235h)  000D1AE8 mov esp,ebp  000D1AEA pop ebp  000D1AEB ret | 00AC1CDC push 7  00AC1CDE call \_loop\_unrolling    00341A60 mov ecx,34C006h  00341A65 call @\_\_CheckForDebuggerJustMyCode@4 (034122Bh)  00341A6A xor eax,eax  00341A6C xor ecx,ecx  00341A6E mov dword ptr [\_i (034A4F4h)],eax  00341A73 cmp ecx,6  00341A76 jae loop\_unrolling+32h (0341A92h)  00341A78 xor edx,edx  00341A7A mov word ptr \_ivector4 (034A470h)[eax\*2],dx  00341A82 lea eax,[ecx+1]  00341A85 mov dword ptr [\_i (034A4F4h)],eax  00341A8A mov ecx,eax  00341A8C cmp eax,6  00341A8F jl loop\_unrolling+13h (0341A73h)  00341A91 ret | Цикл не пропал, но уменьшилось количество присваиваний между переменными и регистрами, также была заменена команда mov на lea, что увеличивает скорость работы. |
| Вызов функции с аргументами | jump\_compression(1, 2, 3, 4, 5);  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);  } | 00CE1E63 push 5  00CE1E65 push 4  00CE1E67 push 3  00CE1E69 push 2  00CE1E6B push 1  00CE1E6D call \_jump\_compression  00CE1E72 add esp,14h  000D18B0 push ebp  000D18B1 mov ebp,esp  000D18B3 sub esp,0C0h  000D18B9 push ebx  000D18BA push esi  000D18BB push edi  000D18BC lea edi,[ebp-0C0h]  000D18C2 mov ecx,30h  000D18C7 mov eax,0CCCCCCCCh  000D18CC rep stos dword ptr es:[edi]  000D18CE mov ecx,offset \_03E453DF\_lab1@c (0DC006h)  000D18D3 call @\_\_CheckForDebuggerJustMyCode@4 (0D122Bh)  000D18D8 mov eax,dword ptr [i]  000D18DB cmp eax,dword ptr [j]  000D18DE jge end\_1+4h (0D191Fh)  000D18E0 mov eax,dword ptr [j]  000D18E3 cmp eax,dword ptr [k]  000D18F9 inc ebp  000D18FA adc al,3  000D18FC inc ebp  000D18FD sbb byte ptr [ecx+2EB1445h],cl  000D1903 jmp end\_1 (0D191Bh)  000D1905 jmp jump\_compression+60h (0D1910h)  000D1907 mov eax,dword ptr [k]  000D190A add eax,dword ptr [l]  000D190D mov dword ptr [k],eax  000D1910 jmp end\_1+2h (0D191Dh)  000D1912 mov eax,dword ptr [j]  000D1915 add eax,dword ptr [k]  000D1918 mov dword ptr [j],eax  000D191B jmp jump\_compression+28h (0D18D8h)  000D191D jmp end\_1+0Dh (0D1928h)  000D191F mov eax,dword ptr [i]  000D1922 add eax,dword ptr [j]  000D1925 mov dword ptr [i],eax  000D1928 mov eax,dword ptr [i]  000D192B add eax,dword ptr [j]  000D192E add eax,dword ptr [k]  000D1931 add eax,dword ptr [l]  000D1934 add eax,dword ptr [m]  000D1937 pop edi  000D1938 pop esi  000D1939 pop ebx  000D193A add esp,0C0h  000D1940 cmp ebp,esp  000D1942 call \_\_RTC\_CheckEsp (0D1235h)  000D1947 mov esp,ebp  000D1949 pop ebp  000D194A ret | 00BB18B6 push es  00BB18B7 sar byte ptr [ebx-6931800h],0FFh  00BB18BE dec dword ptr [ebx+458B0875h]  00BB18C4 or al,8Bh  00BB18D1 and byte ptr [ebx],bh  00BB18D3 sar dword ptr [ebp+16h],3Bh  00BB18D7 retf 267Dh  00BB18DA cmp edx,edi  00BB18DC jge end\_1 (0BB18EEh)  00BB18DE add eax,esi  00BB18E0 add edx,edi  00BB18E2 add eax,ecx  00BB18E4 add eax,edx  00BB18E6 add eax,edi  00BB18E8 pop edi  00BB18E9 pop esi  00BB18EA pop ebp  00BB18EB ret  00BB18EC add eax,ecx  00BB18EE cmp esi,eax  00BB18F0 jl jump\_compression+22h (0BB18D2h)  00BB18F2 add esi,eax  00BB18F4 add eax,esi  00BB18F6 add eax,ecx  00BB18F8 add eax,edx  00BB18FA add eax,edi  00BB18FC pop edi  00BB18FD pop esi  00BB18FE pop ebp  00BB18FF ret  00BB1900 add eax,esi  00BB1902 add ecx,edx  00BB1904 add eax,ecx  00BB1906 add eax,edx  00BB1908 add eax,edi  00BB190A pop edi  00BB190B pop esi  00BB190C pop ebp  00BB190D ret  00341904 add eax,ecx  00341906 add eax,edx  00341908 add eax,edi  0034190A pop edi  0034190B pop esi  0034190C pop ebp  0034190D ret | В коде переход производится к имени end\_1, но если перейти по адресу, то повторного перехода не происходит, а начинается сравнение. То есть по факту переходит сразу к beg\_1. |

После этого была выключена оптимизация и проверен размер неоптимизированной программы. Он составил 41475 Б. Затем была включена оптимизация по размеру с помощью установления флага /O1. После оптимизации он стал весить 29696 Б. Код стал короче, начало использоваться много новых команд из-за чего удобочитаемость кода понизилась.

1. **Выводы**

В ходе лабораторной работы были сравнены оптимизированный по скорости и неоптимизированные коды. Проделав сравнения, было выявлено, что оптимизированный код старается больше работать с регистрами и числами, чем с переменными. Также он не присваивает переменным значения, если в последствии переменная не будет использоваться и снова поменяет свое значение – проблема переприсваивания. В циклах переменная, которая меняется каждый раз при входе в цикл, принимает свое значение единожды, в конце цикла. Вместо нее используется регистр. Заранее ложные проверки условий не проверяются. Сложные команды по возможности заменяются на более быстрые. Лишние действия, такие как x=x\*1 не выполняются. Если функция никогда ничего не делает, то ее код не генерируется. Если много раз проверяется одно и тоже значение в условии (например, g+k), то компьютер сохраняет его значение в регистр и больше не считает, а каждый раз берет значение из регистра и сравнивает. В оптимизированном коде не происходит косвенных переходов – они заменяются на прямые. Также компьютер может заранее посчитать итоговое значение в цикле и сразу присвоить его переменной, тем самым отказавшись от выполнения цикла.