МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Вычислительная техника»

Дисциплина «Организация ЭВМ и систем»

**Лабораторная работа №4.**

**Применение методов оптимизации ассемблерного кода, полученного компилятором из Си программ**

**3 вариант**

Выполнил:

студент группы ИВТАПбд-31.

Вершинин Д. В.

Проверила:

Лылова А. В.

Ульяновск, 2018

**Техническое задание:** Запрограммировать на языке Си алгоритм умножения матриц. Откомпилировать полученную программу в любом компиляторе, позволяющем получить исходный код программы на языке Assembler. Провести анализ ассемблерного кода и изменить его таким образом, чтобы в нем не прослеживались следы Си компиляции

**Ход работы:** Алгоритм перемножения матриц заключается в следующем: заведем матрицы A и B, в которых будем хранить собственно перемножаемые матрицы. Заведем матрицу C, в которой будем хранить ответ на задачу. Основное тело проекта состоит из трех вложенных циклов for, в которых мы изменяем соответствующие счетчики от 0 до 3 и на место элемента с индексами i и j матрицы C записываем значение A[i][k]\*B[k][j].

**Анализ кода на языке ассемблер:**

|  |  |
| --- | --- |
| #include <stdio.h>  int main() {  00F51680 push ebp  00F51681 mov ebp,esp  00F51683 sub esp,16Ch  00F51689 push ebx  00F5168A push esi  00F5168B push edi  00F5168C lea edi,[ebp-16Ch]  00F51692 mov ecx,5Bh  00F51697 mov eax,0CCCCCCCCh  00F5169C rep stos dword ptr es:[edi]  00F5169E mov eax,dword ptr [\_\_security\_cookie (0F58004h)]  00F516A3 xor eax,ebp  00F516A5 mov dword ptr [ebp-4],eax | В данном участке кода происходит подключение библиотеки <stdio.h>, и вызов точки входа в приложение функции main().  При написании программы на ассемблере первой строкой мы бы указали формат исполняемого файла, а в качестве точки входа указали метку, с которой начинается выполнение программы.  *format MZ*  *entry code\_seg:start* |
| int i, j, k;  int a[3][3] = { { 1, 2, 3 }, { 1, 2, 3 }, { 1, 2, 3 } };  00F516A8 mov dword ptr [a],1  00F516AF mov dword ptr [ebp-4Ch],2  00F516B6 mov dword ptr [ebp-48h],3  00F516BD mov dword ptr [ebp-44h],1  00F516C4 mov dword ptr [ebp-40h],2  00F516CB mov dword ptr [ebp-3Ch],3  00F516D2 mov dword ptr [ebp-38h],1  00F516D9 mov dword ptr [ebp-34h],2  00F516E0 mov dword ptr [ebp-30h],3  int b[3][3] = { { 1, 2, 3 }, { 1, 2, 3 }, { 1, 2, 3 } };  00F516E7 mov dword ptr [b],1  00F516EE mov dword ptr [ebp-78h],2  00F516F5 mov dword ptr [ebp-74h],3  00F516FC mov dword ptr [ebp-70h],1  00F51703 mov dword ptr [ebp-6Ch],2  00F5170A mov dword ptr [ebp-68h],3  00F51711 mov dword ptr [ebp-64h],1  00F51718 mov dword ptr [ebp-60h],2  00F5171F mov dword ptr [ebp-5Ch],3  int c[3][3] = {};  00F51726 xor eax,eax  00F51728 mov dword ptr [c],eax  00F5172E mov dword ptr [ebp-0A4h],eax  00F51734 mov dword ptr [ebp-0A0h],eax  00F5173A mov dword ptr [ebp-9Ch],eax  00F51740 mov dword ptr [ebp-98h],eax  00F51746 mov dword ptr [ebp-94h],eax  00F5174C mov dword ptr [ebp-90h],eax  00F51752 mov dword ptr [ebp-8Ch],eax  00F51758 mov dword ptr [ebp-88h],eax | Далее объявляются двумерные массивы. Их инициализация происходит через смещение указателя. Но в реальной программе на ассемблере массив скорее всего будет проинициализирован иначе.  *segment data\_seg*  *a db 1, 2, 3, 1, 2, 3, 1, 2, 3*  *b db 1, 2, 3, 1, 2, 3, 1, 2, 3*  *c db 9 dup(0)* |
| for (i = 0; i < 3; i++)  00F5175E mov dword ptr [i],0  00F51765 jmp main+0F0h (0F51770h)  00F51767 mov eax,dword ptr [i]  00F5176A add eax,1  00F5176D mov dword ptr [i],eax  00F51770 cmp dword ptr [i],3  00F51774 jge main+16Eh (0F517EEh)  for (j = 0; j < 3; j++)  00F51776 mov dword ptr [j],0  00F5177D jmp main+108h (0F51788h)  00F5177F mov eax,dword ptr [j]  00F51782 add eax,1  00F51785 mov dword ptr [j],eax  00F51788 cmp dword ptr [j],3  00F5178C jge main+169h (0F517E9h)  for (k = 0; k < 3; k++)  00F5178E mov dword ptr [k],0  00F51795 jmp main+120h (0F517A0h)  00F51797 mov eax,dword ptr [k]  00F5179A add eax,1  00F5179D mov dword ptr [k],eax  00F517A0 cmp dword ptr [k],3  00F517A4 jge main+167h (0F517E7h)  c[i][j] += a[i][k] \* b[k][j];  00F517A6 imul eax,dword ptr [i],0Ch  00F517AA lea ecx,c[eax]  00F517B1 imul edx,dword ptr [i],0Ch  00F517B5 lea eax,a[edx]  00F517B9 imul edx,dword ptr [k],0Ch  00F517BD lea edx,b[edx]  00F517C1 mov esi,dword ptr [k]  00F517C4 mov edi,dword ptr [j]  00F517C7 mov eax,dword ptr [eax+esi\*4]  00F517CA imul eax,dword ptr [edx+edi\*4]  00F517CE mov edx,dword ptr [j]  00F517D1 add eax,dword ptr [ecx+edx\*4]  00F517D4 imul ecx,dword ptr [i],0Ch  00F517D8 lea edx,c[ecx]  00F517DF mov ecx,dword ptr [j]  00F517E2 mov dword ptr [edx+ecx\*4],eax  00F517E5 jmp main+117h (0F51797h)  00F517E7 jmp main+0FFh (0F5177Fh)  00F517E9 jmp main+0E7h (0F51767h) | Далее идет основное тело программы. В ассемблере мы поместим его внутрь сегмента кода.  *segment data\_seg*  Команды jmp и jge используются для прыжка по коду программы. Но такая реализация прыжка неудобна, так как необходимо знать длину каждой команды, описанной выше. Заменим такие прыжки на прыжок по метке  *jmp label*  *…*  *label:* |
| }  00F517EE xor eax,eax  00F517F0 push edx  00F517F1 mov ecx,ebp  00F517F3 push eax  00F517F4 lea edx,ds:[0F51814h]  00F517FA call @\_RTC\_CheckStackVars@8 (0F51244h)  00F517FF pop eax  00F51800 pop edx  00F51801 pop edi  00F51802 pop esi  00F51803 pop ebx  00F51804 mov ecx,dword ptr [ebp-4]  00F51807 xor ecx,ebp  00F51809 call @\_\_security\_check\_cookie@4 (0F51258h)  00F5180E mov esp,ebp  00F51810 pop ebp  00F51811 ret  00F51812 xchg ax,ax  00F51814 add eax,dword ptr [eax]  00F51816 add byte ptr [eax],al  00F51818 sbb al,18h  00F5181A cmc  00F5181B add byte ptr [eax+24FFFFFFh],dh  00F51821 add byte ptr [eax],al  00F51823 add byte ptr [eax+ebx-0Bh],al  00F51827 add byte ptr [edi+edi\*8+24FFFFh],al  00F5182E add byte ptr [eax],al  00F51830 inc edx  00F51831 sbb ch,dh  00F51833 add byte ptr [eax-1],bl  00F51836 ?? ??  00F51837 jmp dword ptr [eax+eax]  00F5183A add byte ptr [eax],al  00F5183C inc eax  00F5183D sbb ch,dh  00F5183F add byte ptr [ebx],ah  00F51842 bound eax,qword ptr [eax]  00F51844 popad  00F51845 add ah,cl | Код в левом столбце производит обнуление и очистку данных, а затем выходит из программы. Заменим весь этот код на  *mov ah, 4ch*  *int 21 h* |

**Вывод**

В данной лабораторной работе было рассмотрен дизассемблированный код, выявлена разница между ассемблерным кодом, выданным компилятором и ассемблерным кодом, написанным программистом. Были получены основы в области написания ассемблерного кода.

**Приложение – программа на языке Си**

#include <stdio.h>

int main() {

int i, j, k;

int a[3][3] = { { 1, 2, 3 }, { 1, 2, 3 }, { 1, 2, 3 } };

int b[3][3] = { { 1, 2, 3 }, { 1, 2, 3 }, { 1, 2, 3 } };

int c[3][3] = {};

for (i = 0; i < 3; i++)

for (j = 0; j < 3; j++)

for (k = 0; k < 3; k++)

c[i][j] += a[i][k] \* b[k][j];

}