

# HASH TABLE

(исследовательская работа)

Баринов Денис

Б05-931

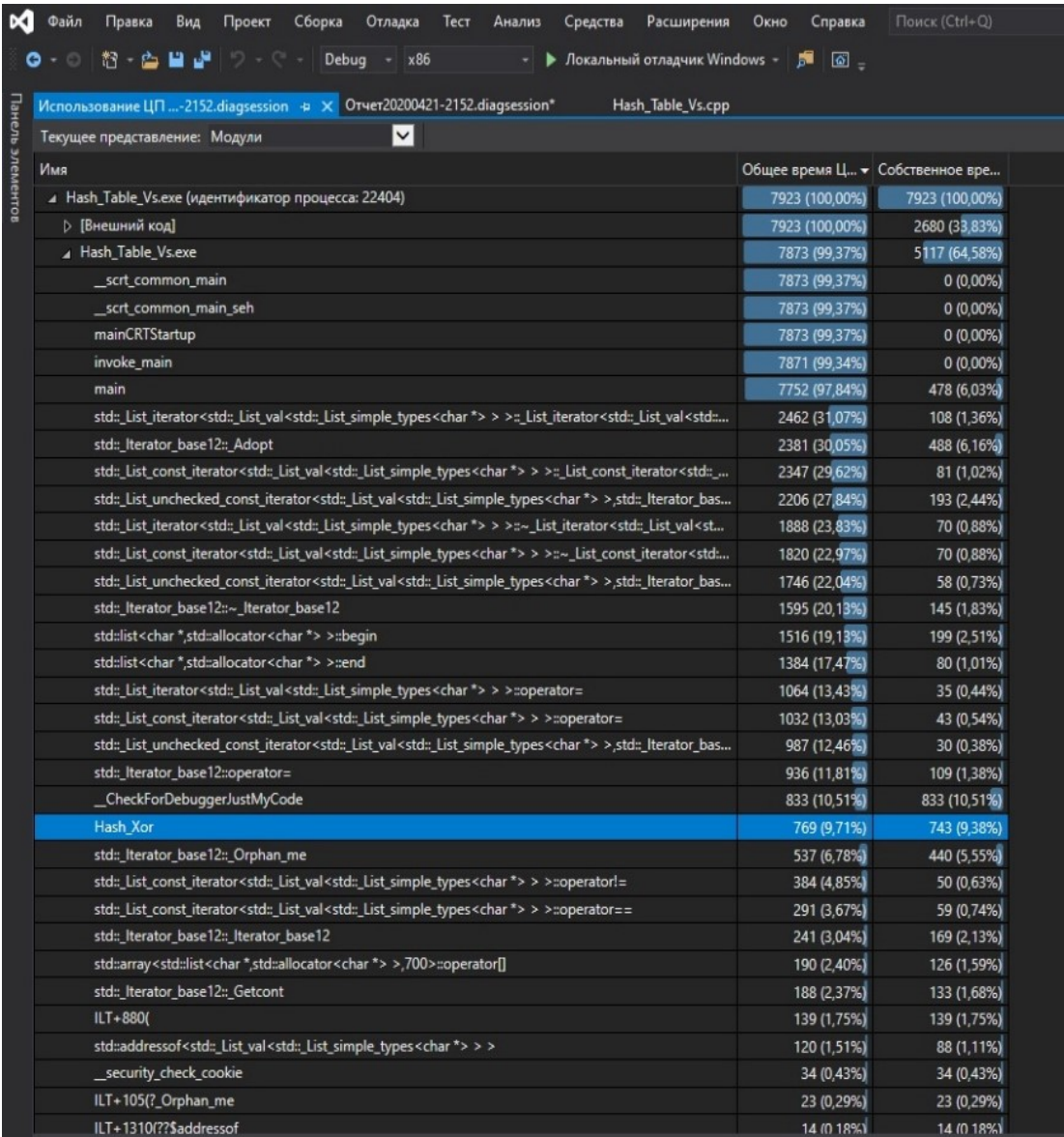
22.04.202

## ЦЕЛИ

- Исследовать hash table на скорость при -Od, -O1, -O2
- Написать асемблерную оптимизацию
- Сравнить полученное время со стандартными оптимизациями

# Версия с std::list и std::find

Как мы видим, в версии со стандартным list  
Наша функция Hash\_Xor занимает меньше 10%,  
Поэтому я решил переписать на свой list на массивах  
Из первого семестра



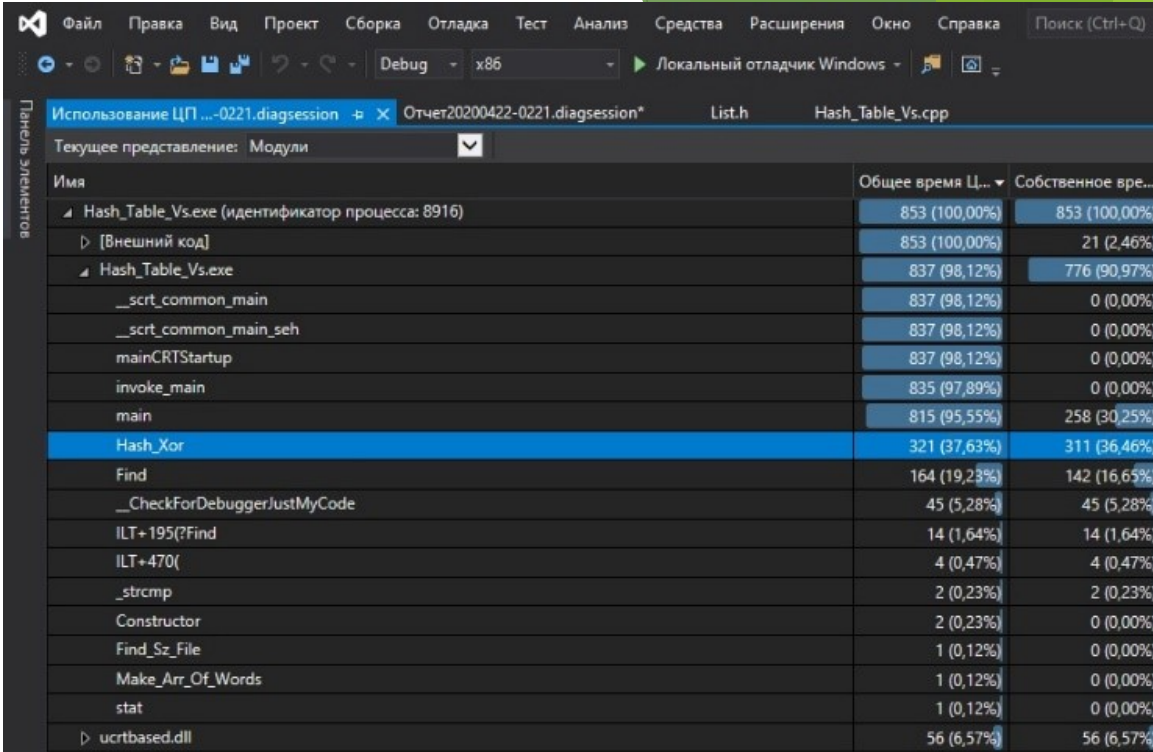
Имя	Общее время ЦП...	Собственное вре...
Hash_Table_Vs.exe (идентификатор процесса: 22404)	7923 (100,00%)	7923 (100,00%)
> [Внешний код]	7923 (100,00%)	2680 (33,83%)
Hash_Table_Vs.exe	7873 (99,37%)	5117 (64,58%)
_scr_t_common_main	7873 (99,37%)	0 (0,00%)
_scr_t_common_main_seh	7873 (99,37%)	0 (0,00%)
mainCRTStartup	7873 (99,37%)	0 (0,00%)
invoke_main	7871 (99,34%)	0 (0,00%)
main	7752 (97,84%)	478 (6,03%)
std::_List_iterator<std::_List_val<std::_List_simple_types<char*>>>::_List_iterator<std::_List_val<std::...	2462 (31,07%)	108 (1,36%)
std::_Iterator_base12::_Adopt	2381 (30,05%)	488 (6,16%)
std::_List_const_iterator<std::_List_val<std::_List_simple_types<char*>>::_List_const_iterator<std::...	2347 (29,62%)	81 (1,02%)
std::_List_unchecked_const_iterator<std::_List_val<std::_List_simple_types<char*>>::_List_iterator_bas...	2206 (27,84%)	193 (2,44%)
std::_List_iterator<std::_List_val<std::_List_simple_types<char*>>::_List_iterator<std::_List_val<st...	1888 (23,83%)	70 (0,88%)
std::_List_const_iterator<std::_List_val<std::_List_simple_types<char*>>::_List_const_iterator<std::...	1820 (22,97%)	70 (0,88%)
std::_List_unchecked_const_iterator<std::_List_val<std::_List_simple_types<char*>>::_List_iterator_bas...	1746 (22,04%)	58 (0,73%)
std::_Iterator_base12::_Iterator_base12	1595 (20,13%)	145 (1,83%)
std::list<char*,std::allocator<char*>>::_begin	1516 (19,13%)	199 (2,51%)
std::list<char*,std::allocator<char*>>::_end	1384 (17,47%)	80 (1,01%)
std::_List_iterator<std::_List_val<std::_List_simple_types<char*>>::_operator=	1064 (13,43%)	35 (0,44%)
std::_List_const_iterator<std::_List_val<std::_List_simple_types<char*>>::_operator=	1032 (13,03%)	43 (0,54%)
std::_List_unchecked_const_iterator<std::_List_val<std::_List_simple_types<char*>>::_List_iterator_bas...	987 (12,46%)	30 (0,38%)
std::_Iterator_base12::_operator=	936 (11,81%)	109 (1,38%)
__CheckForDebuggerJustMyCode	833 (10,51%)	833 (10,51%)
Hash_Xor	769 (9,71%)	743 (9,38%)
std::_Iterator_base12::_Orphan_me	537 (6,78%)	440 (5,55%)
std::_List_const_iterator<std::_List_val<std::_List_simple_types<char*>>::_operator!=	384 (4,85%)	50 (0,63%)
std::_List_const_iterator<std::_List_val<std::_List_simple_types<char*>>::_operator==	291 (3,67%)	59 (0,74%)
std::_Iterator_base12::_Iterator_base12	241 (3,04%)	169 (2,13%)
std::array<std::list<char*,std::allocator<char*>>,700>::_operator[]	190 (2,40%)	126 (1,59%)
std::_Iterator_base12::_Getcont	188 (2,37%)	133 (1,68%)
ILT+880{	139 (1,75%)	139 (1,75%)
std::addressof<std::_List_val<std::_List_simple_types<char*>>::_>	120 (1,51%)	88 (1,11%)
__security_check_cookie	34 (0,43%)	34 (0,43%)
ILT+105{?_Orphan_me	23 (0,29%)	23 (0,29%)
ILT+1310{???\$addressof	14 (0,18%)	14 (0,18%)

Переписав получаем, что теперь Hash\_Xor занимает практически 40%! Что очень удобно для оптимизации. Также стоит отметить, что Hash\_Xor выбрана из-за довольно хорошего распределения и возможности удобной асемблерной оптимизации. (см. ниже)

```
int Hash_Xor (char *elem, int len)
{
    int res = 0;
    int bit = 0;

    for (int i = 0; i < len; i++)
    {
        res ^= elem[i];
        bit = (res >> 31) & 1;
        res <<= 1;
        res |= bit;
    }

    return res;
}
```

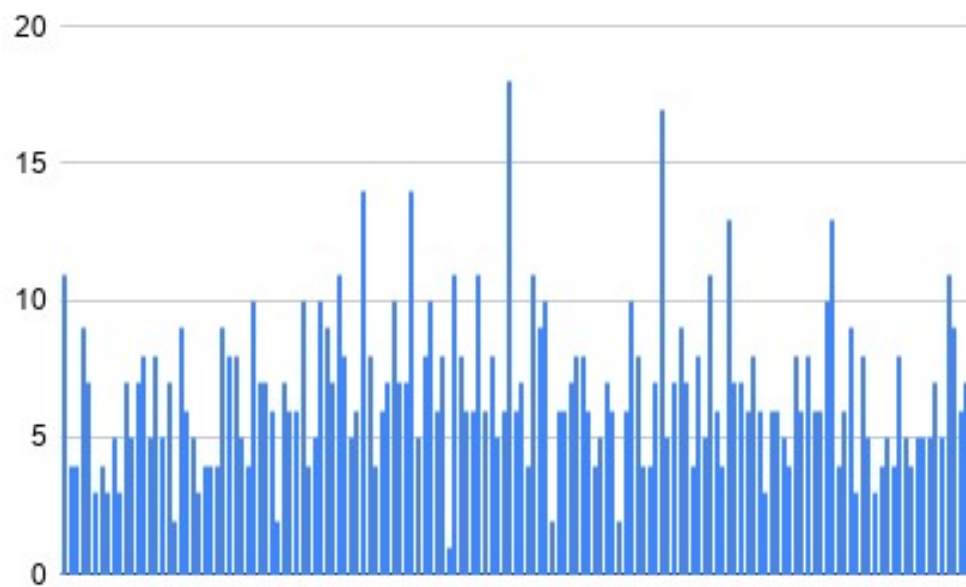


Использование ЦП ...-0221.diagsession -> x Отчет20200422-0221.diagsession\* List.h Hash\_Table\_Vs.cpp

Текущее представление: Модули

Имя	Общее время ЦП	Собственное вре...
Hash_Table_Vs.exe (идентификатор процесса: 8916)	853 (100,00%)	853 (100,00%)
[Внешний код]	853 (100,00%)	21 (2,46%)
Hash_Table_Vs.exe	837 (98,12%)	776 (90,97%)
__scrt_common_main	837 (98,12%)	0 (0,00%)
__scrt_common_main_seh	837 (98,12%)	0 (0,00%)
mainCRTStartup	837 (98,12%)	0 (0,00%)
invoke_main	835 (97,89%)	0 (0,00%)
main	815 (95,55%)	258 (30,25%)
Hash_Xor	321 (37,63%)	311 (36,46%)
Find	164 (19,23%)	142 (16,65%)
__CheckForDebuggerJustMyCode	45 (5,28%)	45 (5,28%)
ILT+195(?Find	14 (1,64%)	14 (1,64%)
ILT+470(	4 (0,47%)	4 (0,47%)
_strcmp	2 (0,23%)	2 (0,23%)
Constructor	2 (0,23%)	0 (0,00%)
Find_Sz_File	1 (0,12%)	0 (0,00%)
Make_Arr_Of_Words	1 (0,12%)	0 (0,00%)
stat	1 (0,12%)	0 (0,00%)
ucrtbased.dll	56 (6,57%)	56 (6,57%)

# РАСПРЕДЕЛЕНИЕ



# Ассемблерная оптимизация:

- Создаём отдельный файл .asm
- Пишем там реализацию функции
- В коде на С пишем:

```
extern "C"  
{  
    int Hash_Xor(char* elem, int len);  
}
```

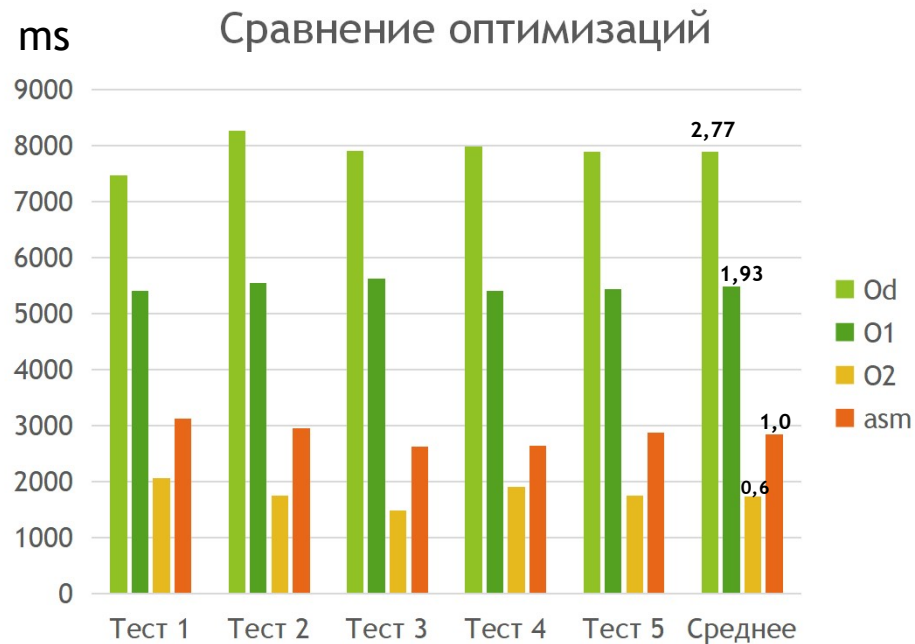
```
1  .686  
2  .MODEL FLAT, C  
3  .CODE  
4  
5  Hash_Xor proc arg1:ptr byte, len:WORD  
6      mov ax, 0  
7      movzx bx, byte ptr [arg1]  
8      mov cx, len  
9  
10 again:  
11     test cx, cx  
12     je return  
13     xor ax, bx  
14     inc bx  
15     rol ax, 1  
16     dec cx  
17     jmp again  
18 return:  
19     ret  
20 Hash_Xor endp  
21  
22 END
```

# Из 37% стало 19!

Hash_Table_Vs.exe (ИП: 25908)	3333 (100,00%)	0 (0,00%)	Hash_Table_Vs.exe
[Внешний код]	3333 (100,00%)	41 (1,23%)	Несколько модул...
_sclr_common_main	3295 (98,86%)	0 (0,00%)	Hash_Table_Vs.exe
_sclr_common_main_seh	3295 (98,86%)	0 (0,00%)	Hash_Table_Vs.exe
mainCRTStartup	3295 (98,86%)	0 (0,00%)	Hash_Table_Vs.exe
invoke_main	3294 (98,83%)	0 (0,00%)	Hash_Table_Vs.exe
main	3133 (94,00%)	2080 (62,41%)	Hash_Table_Vs.exe
Hash_Xor	638 (19,14%)	637 (19,11%)	Hash_Table_Vs.exe
Find	453 (13,59%)	313 (9,39%)	Hash_Table_Vs.exe
_CheckForDebuggerJustMyCode	183 (5,49%)	183 (5,49%)	Hash_Table_Vs.exe
ILT+200(?Find	67 (2,01%)	67 (2,01%)	Hash_Table_Vs.exe
ILT+490(	10 (0,30%)	10 (0,30%)	Hash_Table_Vs.exe
[Системный вызов] ntoskrnl.exe	2 (0,06%)	2 (0,06%)	ntoskrnl.exe
Constructor	1 (0,03%)	0 (0,00%)	Hash_Table_Vs.exe
Find_Sz_File	1 (0,03%)	0 (0,00%)	Hash_Table_Vs.exe
Make_Arr_Of_Words	1 (0,03%)	0 (0,00%)	Hash_Table_Vs.exe
stat	1 (0,03%)	0 (0,00%)	Hash_Table_Vs.exe



## Время выполнения:



Тип:	-Od	-O1	-O2	Asm
Тест 1 (ms)	7465	5401	2057	3130
Тест 2 (ms)	8266	5548	1755	2958
Тест 3 (ms)	7904	5621	1476	2629
Тест 4 (ms)	7987	5405	1906	2637
Тест 5 (ms)	7886	5439	1750	2867
Среднее(ms)	7901	5482	1728	2844

По формуле относительного ускорения:  
коэф.ускор/кол-во строк \* 1000 получаем:  
 $(7901 / 2844) / 22 * 1000 = 126$



## ВЫВОДЫ

Написанная асемблерная реализация быстрее чем

-Od в среднем в 2,77

-O1 в среднем в 1,93

но медленнее чем -O2 в 0,6