МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
« ХАРЬКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ»

А.Н. Рысованый

РЕВЕРСНОЕ ПРОГРАММИРОВАНИЕ

Антиотладочные приемы защиты от реверса. Среда программирования masm64

Учебное пособие
для студентов специальности 123 – «Компьютерная инженерия»
и специальности 125 – «Кибербезопасность»

Утверждено
редакционно-издательским
советом университета,
протокол №1.14 от
19.02.2020

Харьков
2020

ББК 32.973-018.1
УДК 004.42
Р54

Рецензенты:

А.В. Шостак, канд. техн. наук, доцент Національний аерокосмічний університет ім. М. Є. Жуковського "Харківський авіаційний інститут";

Г.А. Кучук, доктор техн. наук, профессор НТУ ХПИ

Рысованый А.Н.

Р54 Реверсное программирование. Антиотладочные приемы защиты от реверса. Среда программирования masm64 : учебное пособие для студентов специальностей 123 «Компьютерная инженерия», 125 «Кибербезопасность»/ А.Н. Рысованый. – Харьков : НТУ «ХПИ», 2020. – 112 с.– На рус. яз.

Рассмотрены вопросы использования противодействия отладке в ОС Windows 10 при программировании на языке Ассемблер в среде masm64. Уделено внимание трассированию по времени выполнения инструкций, использованию API-функций мультимедийного таймера, функций обнаружения отладчика и других приемов препятствования отладке.

Приведены материалы, особенности использования отладчика x64Dbg и лабораторные работы с заданиями и примерами выполнения в среде masm64.

Предназначено для студентов специальности 123 – «Компьютерная инженерия», 125 «Кибербезопасность».

Ил. 29. Библиогр. 10 назв.

УДК 004.42
ББК 32.973-018.1
© А.Н. Рысованый, 2020

ВСТУПЛЕНИЕ

Это учебное пособие не претендует на изложении полного списка антиотладочных методов, поскольку оно содержит только простые и часто используемые способы при работе в ОС Windows 10. Но применение десятка таких даже простых приемов очень сильно затруднит процесс реверсинга программ. В этом случае взлом программы становится совсем нерентабельным и единственным стимулом хакера остается только спортивный интерес [1-5].

Вопросы антиотладки интересны не только исследователям вирусов, но и программистам, которые заботятся о безопасности своих коммерческих программ. Чтобы хоть как-то защитить свою программу без применения антиотладочных приемов надо применять такой стиль написания, который требует от хакера слишком много времени. К такому стилю можно отнести применение очень большого количества процедур и макросов, запутывание кода мусорным кодом, применением глобальных переменных, применением инструкций новых технологий (например, AVX) и других приемов защиты. Естественно, что защита увеличивает объем кода и время выполнения. Но все зависит от целей автора и важности его программного продукта.

Хакера не страшит низкоуровневое программирование [6-10]. Иначе он не хакер. Хакер непосредственно занимается анализом исполняемого файла как раз на этом низкоуровневом языке – ассемблере. **Хакер сам является программистом и если ему удается изменить чужой код, то его уровень программирования должен быть больше, чем уровень разработчика программного продукта.**

Антиотладочные приемы часто применяются в протекторах, пакерах и вредоносном программном обеспечении (малвари - malware), замедляя или предотвращая процесс реверс-инжениринга.

1. АНТИОТЛАДОЧНЫЕ ПРИЕМЫ. ОБЩИЕ СВЕДЕНИЯ

Для архитектуры x64 многие ранее известные средства обнаружения отладчиков устарели. Методы обнаружения отладчика, которые применялись для архитектуры x32 требуют корректировки или отмены.

Существует огромное количество способов противостоять отладке программ. С каждым выходом новой версии ОС Windows таких способов становится меньше. Это происходит в силу того, что борясь за безопасность своей ОС аппаратные возможности определения отладки исчезают, а новые программные не успевают разрабатываться за время выхода нового обновления ОС.

Можно выделить следующую классификацию методов определения отладки:

– обработка исключений;

– трассировка по времени выполнения инструкций;

– API-функции определения отладчика;

– отладочные регистры;

– флаги отладки, память приложений (куча – название структуры данных, с помощью которой реализована динамически распределяемая память приложения);

– другие.

Основная цель различных антиотладочных приемов – усложнение процесса анализа приложений.

2. ОБРАБОТКА ИСКЛЮЧЕНИЙ

Обработка исключительных ситуаций (англ. exception handling) — это механизм программирования, предназначенный для описания реакции программы на ошибки времени выполнения и другие возможные проблемы (исключения), которые могут возникнуть при выполнении программы и приводят к невозможности (бессмысленности) дальнейшей отработки программой её базового алгоритма. В русском языке также применяется более короткая форма термина: «обработка исключений».

Другими словами, исключение – это событие при выполнении программы, которое приводит к её ненормальному или неправильному поведению.

Существует два вида исключений: аппаратные и программные.

Аппаратные исключения (структурные, SE-Structured Exception) генерируются процессором. К ним относятся, например,

– деление на 0;

– выход за границы массива;

– обращение к невыделенной памяти;

– переполнение разрядной сетки.

Программные исключения, которые генерируются операционной системой и прикладными программами возникают тогда, когда программа их явно инициирует. Когда встречается аномальная ситуация, то та часть программы, которая ее обнаружила может сгенерировать или возбудить исключение.

Механизм структурной обработки исключений позволяет однотипно обрабатывать как программные, так и аппаратные исключения.

Существует два механизма функционирования обработчиков исключений:

– обработка с возвратом;

– обработка без возврата.

Если происходит исключение, то выполнение основного кода программы прерывается и начинается выполнение обработчика. По завершении обработчика управление передаётся либо в некоторую наперёд заданную точку программы, либо обратно в точку

возникновения исключения (в зависимости от заданного способа обработки — с возвратом или без).

Некоторые команды ассемблера, а также некоторые API-функции вызывают исключения выполнения программы. При этом управление передается заранее установленному обработчику исключений. Если такую программу запустить под отладчиком, то эти же функции и команды исключений вызывать не будут.

В 64-битных версиях Windows для платформ x64 и Itanium применяется иной способ обработки исключений, нежели в x86-версиях. Способ основан на таблицах, содержащих всю необходимую для диспетчеризации исключения информацию, включая смещения начала и конца блока кода, для которого производится обработка исключения. Поэтому в коде, скомпилированном для этих платформ, нет никаких операций по установке и снятию обработчика для каждого блока. Статичная таблица исключений располагается в Exception Directory PE-файла и представляет собой массив элементов структур RUNTIME_FUNCTION

В качестве таких команд или API могут быть использованы:

- флаг трассировки (trap flag);
- int 0x3 (одним байтом 0xCC);
- int 0x3 (двумя байтами 0xCD, 0x03);
- int 0x2d;
- int 0x2c (работает только под Win Vista и выше);
- так называемая точка заморозки (команда с опкодом 0xf1);
- API-функция DebugBreak (или DbgBreakPoint из ntdll.dll);
- API-функция RaiseException с некоторыми входными значениями.

Наиболее просто реализовать проверку флага трассировки TF (но отладчик x64Dbg обходит проверку этого флага).

2.1. Трассировка по флагу TF

Флаг (признак) TF разрешает пошаговый режим отладки. При установке этого флага после каждой выполненной инструкции происходит прерывание программы и вызов специального обработчика прерывания (Int3). Команда Int (interrupt) – инструкция на языке ассемблера для процессора архитектуры x86, генерирующая программное прерывание.

INT 3 – команда процессоров семейства x86, которая выполняет функцию программного breakpoint (точка останова). Исполнение команды приводит к вызову обработчика прерывания номер 3,

зарезервированного для отладочных целей. В отличие от остальных команд вида INT N, которые кодируются двумя байтами, команда INT 3 кодируется только одним байтом с кодом 0xCC, хотя двухбайтная инструкция 0xCD 0x03 тоже будет работать. Эта команда используется главным образом при отладке программ. При этом отладчик может вставлять INT 3 в код отлаживаемой программы в точках останова.

В режиме трассировки программист видит последовательность выполнения команд и значения переменных на каждом шаге выполнения программы. Во время трассировки программист сам управляет выполнением очередной команды программы.

Способ трассировки по флагу TF часто применялся при отладке 32-разрядных программ различными отладчиками. В настоящее время одним из лучших отладчиков под 64-разрядное программирование является отладчик x64Dbg. Однако, в отладчике x64Dbg способ трассировки по флагу TF **не позволяет обнаруживать отладку**, т.к TF-флаг в этом отладчике при трассировке **не устанавливается**. Этот способ обнаружения трассировки состоит в чтении регистра флагов (EFLAGS) и проверки состояния бита TF. Если этот бит не равен нулю – программу трассируют (программа 2.1).

Программа 2.1. Трассировка по флагу TF:

```
        title отладчик x64Dbg даже не устанавливает бит TF
        include \masm64\include64\masm64rt.inc ; библиотеки
.data
        szTest db "Программу НЕ трассируют",0
        szTest2 db "Программу трассируют",0
.code
entry_point proc
        pushfq    ; сохраняем флаги в стеке, включая и TF;
        pop rax      ; выталкиваем сохраненные флаги в rax
        and rax,100h   ; проверяем состояние TF-бита
         jnz debug1   ; если TF взведен, нас трассируют

         invoke MessageBox,0,addr szTest,0,MB_OK
        jmp m1
        debug1:
        invoke MessageBox,0,addr szTest2,0,MB_OK
m1: invoke ExitProcess,0
entry_point endp
        end
```

И хотя программа 2.1 для архитектуры x64 **уже не является актуальной**, она позволяет рассмотреть классический прием

трассировки – трассировка по специально предназначенному для этих целей флагу трассировки TF.

Трассировка по флагу TF – это самый простой классический способ обнаружения отладки. Если TF-флаг (признак), хранящийся в регистре EFLSGS взведен (установлен), то микропроцессор перед каждой инструкцией генерирует исключение, сбрасывает признак TF и приостанавливает поток. Отладчик функцией WaitForDebugEvent возвращает информацию о событии отладки EXCEPTION_DEBUG_EVENT + EXCEPTION_SINGLE_STEP. Функция WaitForDebugEvent ожидает событие отладки, которое произойдет в отлаживаемом процессе. Причем отладчик устанавливает бит регистра EFLAGS с помощью функции SetThreadContext, которая позволяет в любом потоке изменить любой из регистров. Функция SetThreadContext устанавливает контекст для заданного потока.

2.2. Трассировка по маске

Исключение составляют команды, которые модифицируют регистр SS (селектор стека) и маскирующие прерывание на выполнение **следующей** команды. На этот шаг разработчики микропроцессоров пошли потому, что в 16/32-разрядном коде часто встречаются конструкции вида MOV SS, MOV ss,ax/MOV esp, new_ESP. Это сделано для случая, чтобы прерывание не произошло раньше инициализации вершины стека (программа 2.2).

Программа 2.2. Трассировка по маске:

```
      include \masm64\include64\masm64rt.inc ; библиотеки
.data
      szTest db "Программу НЕ трассируют",0
      szTest2 db "Программу трассируют",0

.code
entry_point proc
      mov ax,ss ; (вроде) маскируем трассировочное прерывание...
      mov ss,ax ; ...на время выполнения команды PUSHFQ
pushfq ; сохраняем флаги в стеке, включая и TF (флаг не установился)
      pop rax ; выталкиваем сохраненные флаги в rax
       and rax,100h ; проверяем состояние TF-бита
       jnz debug1   ; если TF взведен, нас трассируют

 invoke MessageBox,0,addr szTest,0,MB_OK
      jmp m1
debug1:
```

```
  invoke MessageBox,0,addr szTest2,0,MB_OK
m1: invoke ExitProcess,0
      entry_point endp
      end
```

Однако, в отладчике x64Dbg и этот классический способ уже не поддерживается, и, естественно, отладка не обнаруживается.

3. ТРАССИРОВКА ПО ВРЕМЕНИ ВЫПОЛНЕНИЯ ИНСТРУКЦИЙ

Для определения времени выполнения части кода или всей программы могут применяться различные способы измерения временных промежутков, как при помощи команды RDTSC, так и при помощи, например, API-функций:

GetTickCount – извлекает число миллисекунд, которые истекли с момента запуска ОС;

TimeGetTime (из winmm.dll) – извлекает системное время в миллисекундах;

QueryPerformanceCounter, QueryPerformanceFrequency – извлекает текущее значение счетчика производительности;

GetSystemTimeAsFileTime – получает текущую системную дату и время;

GetSystemTimePreciseAsFileTime – извлекает текущую системную дату и время с максимально возможным уровнем точности (<1us);

KeTickCount, KeQueryTimeIncrement (или вызов прерывания int 0x2A) – измерение системного времени (NtosKrnl.lib);

KeQueryPerformanceCounter – извлекает текущее значение и частоту счетчика производительности;

KeQuerySystemTime – получает текущее системное время;

KeQueryInterruptTime – возвращает текущее значение счетчика времени системного прерывания с точностью до отметки системных часов;

KeQueryTickCount – поддерживает количество прерываний таймера интервала, которые произошли с момента загрузки системы;

KeQueryUnbiasedInterruptTime – возвращает текущее значение счетчика времени системного прерывания;

GetProcessTimes – получает информацию о времени для указанного процесса;

NtQueryInformationProcess (ProcessInformationClass = ProcessTimes (0x04);

NtQueryInformationThread (ThreadInformationClass = ThreadTimes (0x01);

NtSetInformationThread – устанавливает приоритет потока

поля структуры KUSER_SHARED_DATA.

3.1. Использование команды rdtsc

Каждая инструкция выполняется микропроцессором за определенное количество тактов. Время выполнения зависит также от частоты микропроцессора, от количества ядер микропроцессора, от загруженности ОС, от технологии оптимизации выполнения кода и от других причин. При измерении линейного кода без цикла получаемый результат почти стабилен. А при измерении времени выполнения цикла результаты будут сильно отличаться друг от друга (за счет измерения большого промежутка времени).

Команда rdtsc (Read Time Stamp Counter) – ассемблерная инструкция, которая читает счетчик TSC (Time Stamp Counter) и возвращает в регистрах RDX:RAX 64-битное количество тактов с момента последнего сброса процессора. Опкод команды rdtsc: 0F 31.

Для того чтобы использовать инструкцию RDTSC в антиотладочных целях, необходимо выполнить ее дважды: до и после выполнения кода или части кода, для которого будет производиться измерение времени (программа 3.1) и из большего значения вычесть меньшее. Полученный результат будет указывать на число тактов, затраченные на выполнение блока, который ограничен командами rdtsc.

Программа 3.1. Измерение времени выполнения кода при помощи команд rdtsc:

```
        include \masm64\include64\masm64rt.inc ; библиотеки
.data
        titl1 db "Исследование трассировки",0
        szTest db "Программу НЕ трассируют",0
        szTest2 db "Программу трассируют",0
.code
entry_point proc
        rdtsc
          xchg eax, ecx
        rdtsc
          sub eax, ecx
          cmp eax, 1000 ; сравнение с 1000 тактов
          ja debug1 ; если больше
  invoke MessageBox,0,addr szTest,0,MB_OK
        jmp @f
debug1:
 invoke MessageBox,0,addr szTest2,0,MB_OK
@@:
        invoke ExitProcess,0
```

entry_point endp
end

В рассматриваемой программе первая команда rdtsc заносит в регистры RDX:RAX число тактов с момента последнего сброса процессора. Через одну команду выполняется вторая команда rdtsc. В этой программе нет математического смысла использовать регистры RDX, которые хранят старшие части числа тактов – они будут одинаковыми (из-за небольшого числа тактов на выполнение кода между командами rdtsc). А вот младшие части за время выполнения команды xchg естественно изменятся. Поэтому, уместной является команда sub eax, ecx, которая и высчитывает время выполнения команды в тактах микропроцессора.

Результат трассирования этой программы в отладчике x64Dbg приведен на рис. 3.1.

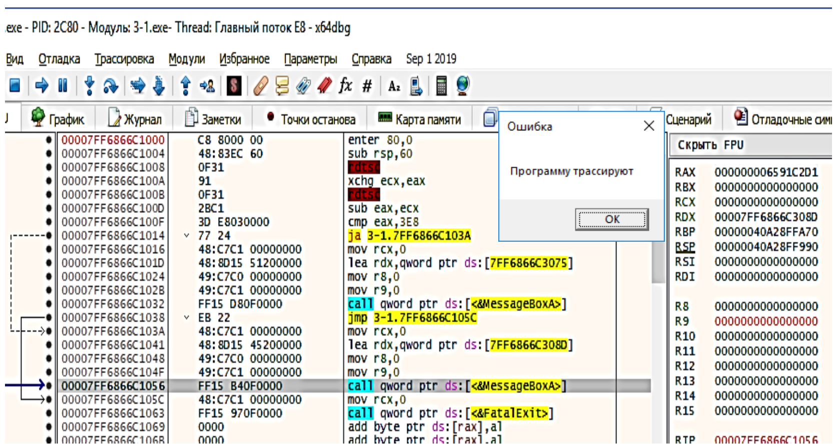


Рис. 3.1. Результат трассирования программы

Сообщение «Программу трассируют» вызывает функция MessageBox при ее вызове в пошаговом режиме в отладчике.

Рассмотрим пример использования антиотладочного кода на основе применения команды rdtsc в программе передачи целочисленных значений одного массива данных в другой (программа 3.2).

Программа 3.2. Использование антиотладочного кода на основе применения команды rdtsc в программе передачи целочисленных значений одного массива данных в другой:

```
        title передача массива; masm64
        include \masm64\include64\masm64rt.inc ; библиотеки
.data
        titl db "Вывод массива через функцию MessageBox",0; назв. окна
        buf1 dq 3 dup(0) ; буфер вывода сообщения
ifmt1 db "Задание: данные одного массива записать в другой массив.",0ah,0ah,
"Вывод массива:", 0dh,"mas2[0] = %d",10,"mas2[8] = %d",10,"mas2[16] = %d",10,
"mas2[24] = %xh",0ah,"mas2[32] = %xh",0ah,0ah,"Автор:         НТУ ХПИ",0
        mas1 db 1,4,6,3ah,3bh
        mas2 dq 5 dup(?)
        len1 equ $-mas2

.code
entry_point proc

.data
        titl1 db "Исследование трассировки",0
        szTest db "Программу НЕ трассируют",0
        szTest2 db "Программу трассируют",0
.code
rdtsc
         xchg eax, ecx
rdtsc
         sub eax, ecx
         cmp eax, 1000 ; сравнение с 1000 тактов
         ja debug1 ; если больше
         invoke MessageBox,0,addr szTest,0,MB_OK
        jmp @f
debug1:
        invoke MessageBox,0,addr szTest2,0,MB_OK
        jmp @2

@@:
        mov rcx,len1 ; len1=5
lea rsi,byte ptr mas1; занесение адреса начала элементов массива mas1
lea rdi,byte ptr mas2; занесение адреса массива результата mas2
@1: mov al,[rsi]
          mov [rdi],al
          inc rsi
          add rdi,8
```

```
    dec rcx
  jnz @1
    mov r10,mas2
    mov r11,mas2[8]
    mov r12,mas2[16]
    mov r13,mas2[24]
    mov r14,mas2[32]
    invoke wsprintf,ADDR buf1,ADDR ifmt1,r10,r11,r12,r13,r14;
    invoke MessageBox,0,addr buf1,addr titl, MB_ICONINFORMATION;
@2:
    invoke ExitProcess,0
    entry_point endp
    end
```

Результат трассирования программы приведен на рис. 3.2.

Рис. 3.2. Результат трассирования программы

В программе применена упрощенная сегментация (несколько одинаковых сегментов) – так легче встраивать одну программу в другую и встраиваемые данные расположены недалеко от встраиваемого кода.

Антиотладочный код вставлен в начало программы. Это сделано для того, чтобы не дать оттрассировать основную часть кода, которая отвечает за передачу целочисленных массивов.

После вывода сообщения о трассировке следующей командой выполняется команда безусловной передачи управления jmp @2, которая в отладчике отображается, как

jmp rl1-41m.7FF6B19410CD.

Эта команда передает управление на функцию завершения программы

invoke RtlExitUserProcess,0

в самом конце программы.

Иногда, для запутывания отладки применяют дополнительную ветвь вычисления ложного выражения.

3.2. Использование функции GetTickCount

Для измерения времени выполнения части кода можно использовать функцию GetTickCount. Функция GetTickCount извлекает число **миллисекунд**, которые истекли с тех пор как система была запущена. Она ограничивается разрешающей способностью системного таймера. Чтобы получить данные о разрешающей способности системного таймера, используют функцию GetSystemTimeAdjustment.

Возвращаемое значение в функции GetTickCount – число миллисекунд, которые истекли после того, как система была запущена.

Рассмотрим программу 3.3, в которой сравниваются два значения чисел миллисекунд, полученных функциями GetTickCount за определенный интервал времени. Если разница между ними больше 1000 мС, то выводится сообщение о том, что программа трассируется в отладчике.

Программа 3.3. Использование функции GetTickCount:

```
    title   masm64
    include \masm64\include64\masm64rt.inc ; библиотеки
.data ;
    szTest  db 'Отладчик не обнаружен',0
    szTest2 db 'Отладчик обнаружен',0
.code
entry_point proc
    invoke GetTickCount
    mov rbx,rax
```

```
db 10 dup(90h) ; 10 nop’ов (90h = опкод nop’a)
    invoke GetTickCount
    sub rax,rbx
    cmp rax,1000
    Ja DEBUG1  ; если >
invoke MessageBox,0,addr szTest,0,MB_OK
    jmp exit1
    DEBUG1:
    invoke MessageBox,0,addr szTest2,0,MB_OK
exit1:
    invoke ExitProcess,0
    entry_point endp
    end
```

Окно отладчика x64Dbg с отслеживаемой программой представлено на рис. 3.3.

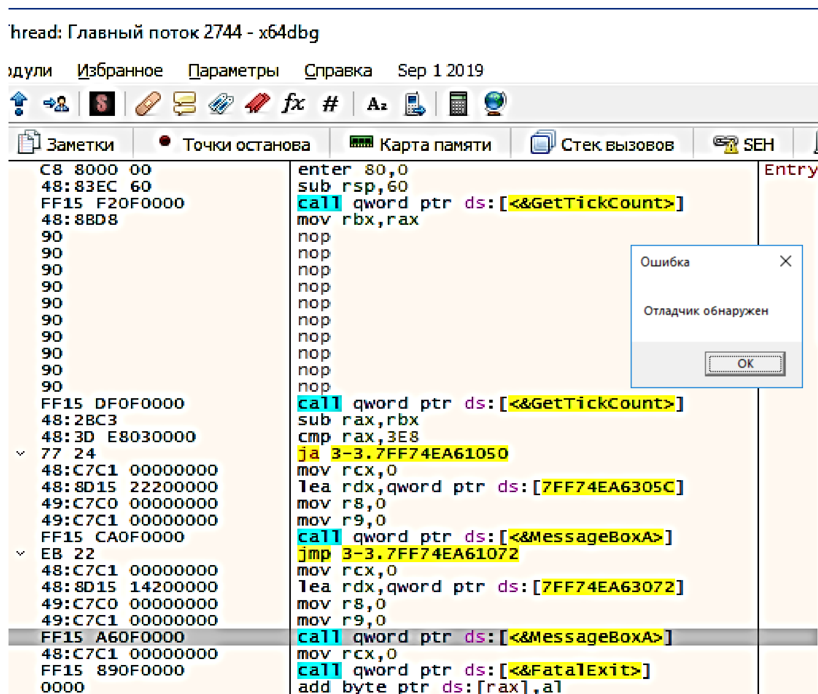


Рис. 3.3. Результат трассирования программы

Пустых команд nop может вообще и не быть, т.к. присутствует

хотя бы одна команда. Это команда mov rbx,rax. А разница значений, полученных двумя функциями GetTickCount определяется количеством тактов на выполнение этой команды mov rbx,rax.

Рассмотрим пример использования антиотладочного кода на основе применения функций GetTickCount в программе выполнения параллельного сложения двух массивов чисел с помощью команды vpaddd набора AVX.

Программа 3.4:

```
      title    AVX,masm64
      include \masm64\include64\masm64rt.inc ; библиотеки
.data ;
      mas1 dd 10, 5, 2, 1
      mas2 dd 16,10, 9, 8
      res dd 4 dup(0),0
      buf dd 4 DUP(0),0
info1 db "Пример работы команды AVX vpaddd",10,10,
 "(циклическое сложение).",10,10,"Результат работы: ",4 dup(" %d "),10,10,
 "Автор: Рысованый А.Н., каф. ВТП, фак. КИТ, НТУ ХПИ",10,
9,"Сайт:  http://blogs.kpi.kharkov.ua/v2/asm/",0
      titl1 db "Вывод чисел через извлечение из XMM",0
.code
      entry_point proc
.data
      szTest  db 'Отладчик не обнаружен',0
      szTest2 db 'Отладчик обнаружен. Уравнение не выполняется',0
.code
    invoke GetTickCount
      mov rbx,rax
      db 10 dup(90h) ; 10 nop'ов (90h = опкод nop'а)
    invoke GetTickCount
         sub rax,rbx
         cmp rax,1000
         Ja DEBUGGED  ; если >
      invoke MessageBox,0,addr szTest,0,MB_OK
      jmp exit1
DEBUGGED:
         invoke MessageBox,0,addr szTest2,0,MB_OK
      jmp @f

exit1:
vmovups xmm1, mas1 ; переслать невыровненные упакованные значения
vmovups xmm2, mas2 ; переслать невыровненные упакованные значения
```

```
vpaddd xmm3, xmm1, xmm2 ; операция циклического сложения
        vpextrd r10d,xmm3,0   ; извлечение 0-го числа з xmm
        vpextrd r11d,xmm3,1   ; извлечение 1-го числа з xmm
        vpextrd r12d,xmm3,2   ; извлечение 2-го числа з xmm
        vpextrd r13d,xmm3,3   ; извлечение 3-го числа з xmm

        mov dword ptr res, r10d ;
        mov dword ptr res[4],r11d ;
        mov dword ptr res[8],r12d ;
        mov dword ptr res[12],r13d ;

        movsxd r10,r10d
        movsxd r11,r11d
        movsxd r12,r12d
        movsxd r13,r13d
      invoke wsprintf,ADDR buf,ADDR info1,r10,r11,r12,r13 ;
      invoke MessageBox,0,addr buf,addr titl1, 0 ;
@@:
      invoke ExitProcess,0
        entry_point endp
        end
```

В случае обнаружения отладки выводится соответствующее сообщение (рис. 3.4) и передача управления на завершение программы:

```
invoke MessageBox,0,offset szTest2,0,MB_OK
 jmp @f
```

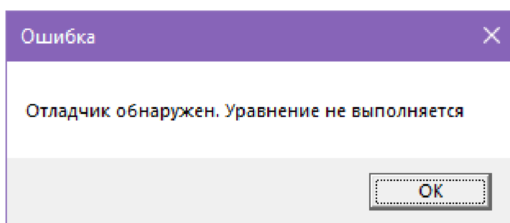


Рис. 3.4. Сообщение об обнаружении отладчика

А в случае отсутствия отладки с метки exit1: выполняется основная часть программы, по завершению которой выводится сообщение (рис. 3.5).

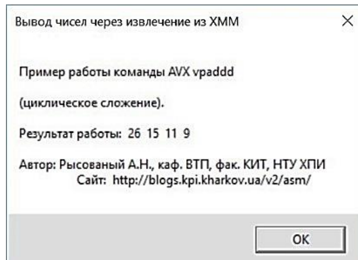


Рис. 3.5. Результат выполнения программы

В настоящее время не все компьютеры поддерживают команды набора AVX. Поэтому, необходимо это учитывать и делать соответствующие проверки (программа 3.5).

Программа 3.5. Проверка поддержки команд набора AVX:

```
    include \masm64\include64\masm64rt.inc ;
.data
titl db "Проверка микропроцессора на поддержку команд AVX",0 ;
название окна
    szInf db "Команды AVX ПОДДЕРЖИВАЮТСЯ!!!",0  ;
    inf db "Команды AVX микропроцессором НЕ поддерживаются",0

.code
entry_point proc

    mov   rax,1
    cpuid ; по rax производится идентификация микропроцессора
    and  rcx,10000000h  ; rcx:= rcx v 1000 0000h  (28 разряд)
    jnz  exit1                ; перейти на exit, если не ноль
    invoke MessageBox,0,addr inf,addr titl,MB_OK
    jmp exit2
  exit1:
    invoke MessageBox,0,addr szInf,addr titl,
         MB_ICONINFORMATION
  exit2:
.data
tit2 db "Проверка микропроцессора на поддержку команд AVX2",0 ;
  szInf2 db "Команды AVX2 ПОДДЕРЖИВАЮТСЯ!!!",0  ;
  inf2 db "Команды AVX2 микропроцессором НЕ поддерживаются",0
.code
```

```
    mov eax,7
    mov ecx,0
    cpuid ; по содержимому rax производится идентификация МП
      and  rbx,20h ;   (5 разряд)
    jnz  @f       ; перейти, если не нуль
      invoke MessageBox,0,addr inf2,addr tit2,MB_OK
    jmp exit12
@@:
      invoke MessageBox,0,addr szInf2,addr tit2,MB_ICONINFORMATION

exit12:
      invoke ExitProcess,0
      entry_point endp
      end
```

Программа 2.6 проверяет поддержку микропроцессором наборов AVX и AVX2. Удобнее эту программу откомпилировать, получить exe-файл и в начале кода программы вызывать этот исполняемый файл, как показано на фрагменте кода:

```
.data
szAVX db 'AVX-12-64-0.exe'
…
invoke WinExec,addr szAVX,SW_HIDE
```

Вызов внешних файлов

Для вызова внешних файлов существуют несколько вариантом.

Чаще всего применяют следующие варианты:

Вариант 1:

```
include \masm64\include64\masm64rt.inc
.data
szRunDLL db "путь и имя exe-шника",0
.code
…
 invoke WinExec,ADDR szRunDLL,SW_HIDE
```

Вариант 2: оформление как inc-файл:

В основном файле:

```
include \masm64\include64\masm64rt.inc
include floatToString.inc
…
invoke  floatToString
…
```

inc-файл:

```
.code
floatToString proc
…
ret
floatToString endp
```

В inc- файле секцию .data лучше исключить, а использовать параметры процедуры.

Вариант 3:

```
.data
;programname db  "c:\windows\system32\calc.exe",0
programname db  "1-1.exe",0
startInfo dd ?
processInfo PROCESS_INFORMATION <> ;
…
invoke CreateProcess,ADDR programname,0,0,0,FALSE,\
  NORMAL_PRIORITY_CLASS, 0,0,\
   ADDR startInfo,ADDR processInfo
```

Вариант 4:

```
invoke ShellExecute, 0, chr$("open"), chr$("hello_world.exe"), 0, 0, SW_SHOWNORMAL
```

3.3 Функция TimeGetTime

Функция timeGetTime извлекает системное время в **миллисекундах**. Системное время – это время, истекшее с момента старта Windows. Эта функция относится к группе функций мультимедийного таймера.

Для измерения коротких интервалов времени с высокой точностью используют функции QueryPerformanceCounter и QueryPerformanceFrequency.

3.4. Использование функции QueryPerformanceCounter

Функция QueryPerformanceCounter извлекает текущее значение счетчика производительности, которое представляет собой метку времени с высоким разрешением (< 1 us), которую можно использовать для измерений с временным интервалом.

Особенность применения этой функции состоит в том, что возвращаемые ею значения вещественные. Поэтому, чтобы вычислить

значение интервала выполнения части кода необходимо воспользоваться командами FPU или SSE/AVX. Один из вариантов применения этой функции приведен в программе 3.6.

Программа 3.6. Применение функции QueryPerformanceCounter:

```
        title      masm64
        include \masm64\include64\masm64rt.inc
.data
        Time1 DQ ?
        Time2 DQ ?
        szTest  db 'Отладчик не обнаружен',0
        szTest2 db 'Отладчик обнаружен',0
        tit1 db "Функция QueryPerformanceCounter",0
        buf1 dq 0,0 ; буфер вывода сообщения
ifmt db "masm64. Вывод метки времени с высоким разрешением:",10,
"a1 = %d",10,10,"Автор программы:   НТУ ХПИ, каф. ВТП",0
.code
entry_point proc

; QueryPerformanceCounter - метка времени с высоким разрешением
invoke QueryPerformanceCounter,ADDR Time1
invoke QueryPerformanceCounter,ADDR Time2
        ;finit
         fild  Time2
         fild  Time1
         fsubp
         fistp Time1
invoke wsprintf, ADDR buf1, ADDR ifmt,Time1
invoke MessageBox,0,addr buf1,addr tit1,MB_OK
           cmp Time1,1000
           Ja DEBUG1  ; если >
invoke MessageBox,0,addr szTest,0,MB_OK
        jmp exit1

DEBUG1:
invoke MessageBox,0,addr szTest2,0,MB_OK
exit1:
        invoke ExitProcess,0
        entry_point endp
        end
```

В программе показан пример применения двух последовательно выполняемых функций. Но на завершение первой и вызова второй функции все-таки затрачивается несколько тактов, которые затем и

вычисляются командами сопроцессора.

В программе сначала выводится сообщение (рис. 3.6) с числом тактов, потраченных на выполнение функции.

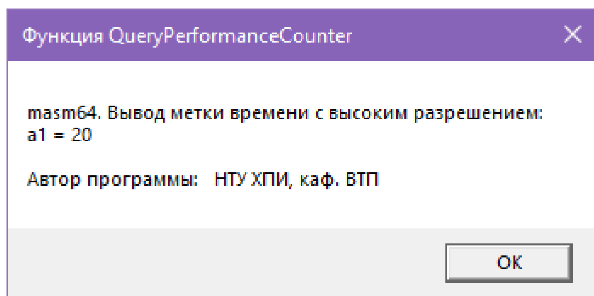


Рис. 3.6. Сообщение с числом тактов, потраченных на выполнение функции

Затем выводится сообщение о наличии или отсутствии отладчика.

3.5. Функция QueryPerformanceFrequency

Функция QueryPerformanceFrequenc получает частоту счетчика производительности. Частота счетчика производительности фиксируется при загрузке системы и одинакова для всех процессоров. Следовательно, частота должна запрашиваться только при инициализации приложения, и результат может быть кэширован.

Синтаксис:

```
BOOL QueryPerformanceFrequency( LARGE_INTEGER *lpFrequency );
```

Параметр lpFrequency – указатель на переменную, которая получает текущую частоту счетчика производительности в секундах.

3.6. Использование функции GetSystemTimeAsFileTime

Функция **GetSystemTimeAsFileTime** извлекает дату и время текущей операционной системы.

Синтаксис:

```
void GetSystemTimeAsFileTime(
  LPFILETIME lpSystemTimeAsFileTime
);
```

Параметр lpSystemTimeAsFileTime – указатель на структуру FILETIME, которая получит дату и время текущей операционной системы в формате универсального глобального времени (UTC).

Эта функция значений не возвращает.

Структура FILETIME – 64-битовое значение, представляющее число интервалов по 100 наносекунд с 1 января 1601 (универсальное глобальное время (UTC)).

Синтаксис:

```
typedef struct _FILETIME {
  DWORD dwLowDateTime;
  DWORD dwHighDateTime;
} FILETIME,
*PFILETIME;
```

Параметры:

dwLowDateTime – младшая часть файлового времени;

dwHighDateTime – старшая часть файлового времени.

Программа 3.7. Применение функции GetSystemTimeAsFileTime без организации вывода информации:

```
        ; Создание файла
        ; Извлечение даты и времени создания файла
        ; Извлечение даты и время текущей операционной системы
        ; Установка даты и времени, когда файл создавался,
        ; последний доступ или изменение
include \masm64\include64\masm64rt.inc ; библиотеки

.data
        file db "1.txt",0
.data?
        hFile   HANDLE ?
        fCRTime FILETIME <> ; время создания
        fACTime FILETIME <> ; время доступа
        fWRTime FILETIME <> ; время записи
        .code
        entry_point proc
invoke CreateFile,addr file,GENERIC_WRITE,0,0, OPEN_ALWAYS,
FILE_ATTRIBUTE_NORMAL, 0
        .if rax == INVALID_HANDLE_VALUE
        jmp m1 ;invoke ExitProcess,0
        .endif
        mov    hFile,rax
```

```
; извлекает данные о дате и времени создания файла
invoke GetFileTime,hFile,addr fCRTime,addr fACTime,addr fWRTime

; извлекает дату и время текущей операционной системы
invoke GetSystemTimeAsFileTime,ADDR fCRTime

; устанавливает дату и время, когда файл создавался, последний доступ или изменение
invoke SetFileTime,hFile,addr fCRTime,addr fACTime,addr fWRTime
	; …
	invoke CloseHandle,hFile ;
m1: invoke ExitProcess,0
	entry_point endp
	end
```

3.7. Функция GetSystemTimePreciseAsFileTime

Функция GetSystemTimePreciseAsFileTime извлекает текущую системную дату и время с максимально возможным уровнем точности (<1us). Полученная информация представлена в формате UTC.

Синтаксис:

```
void GetSystemTimePreciseAsFileTime( LPFILETIME lpSystemTimeAsFileTime );
```

Параметр:

lpSystemTimeAsFileTime – тип: LPFILETIME – указатель на структуру FILETIME, которая содержит текущую системную дату и время в формате UTC.

Эта функция не возвращает значение.

Эта функция лучше всего подходит для измерений времени суток с высоким разрешением или для отметок времени, синхронизированных с UTC. Но для измерений интервала с высоким разрешением используются функции QueryPerformanceCounter или KeQueryPerformanceCounter.

3.8. Функции KeTickCount, KeQueryTimeIncrement

Функции KeTickCount, KeQueryTimeIncrement необходимы для измерения системного времени.

Функция KeQueryTimeIncrement возвращает количество единиц по 100 наносекунд, которые добавляются к системному времени каждый

раз, когда прерывается интервал. Функция KeTickCount не документирована, но выполняет аналогичные действия и находится внутри обработчика int 2Ah.

Функции KeQueryTimeIncrement не имеет параметров.

3.9. Функция KeQueryPerformanceCounter

Функция KeQueryPerformanceCounter извлекает текущее значение и частоту счетчика производительности. Используется функция KeQueryPerformanceCounter для получения меток времени с высоким разрешением (<1 мкс) при измерении временных интервалов.

Синтаксис:

```
NTHALAPI LARGE_INTEGER KeQueryPerformanceCounter( PLARGE_INTEGER PerformanceFrequency );
```

Параметр: PerformanceFrequency – указатель на переменную, в которую KeQueryPerformanceCounter записывает частоту счетчика производительности в тиках в секунду. Этот параметр является необязательным и может иметь значение NULL, если вызывающей стороне не нужно значение частоты счетчика.

Функция KeQueryPerformanceCounter возвращает 64-разрядное целое число, которое представляет текущее значение монотонно неубывающего счетчика высокого разрешения.

3.10. Функция KeQuerySystemTime

Функция KeQuerySystemTime получает текущее системное время.

Синтаксис:

```
void KeQuerySystemTime( PLARGE_INTEGER CurrentTime );
```

Параметр: CurrentTime – указатель на текущее время по возвращении из KeQuerySystemTime.

Системное время – это счетчик интервалов в 100 наносекунд с 1 января 1601 года. Системное время обычно обновляется примерно каждые десять миллисекунд. Это значение рассчитывается для часового пояса GMT. Чтобы настроить это значение для местного часового пояса, используется функция ExSystemTimeToLocalTime.

3.11. Функция KeQueryInterruptTime

Функция KeQueryInterruptTime возвращает текущее значение счетчика времени системного прерывания с точностью до отметки системных часов.

Синтаксис:
ULONGLONG KeQueryInterruptTime( );
Эта функция не имеет параметров.

Функция KeQueryInterruptTime возвращает текущий счетчик времени прерывания в единицах по 100 наносекунд. Обновление этого возвращаемого значения обычно происходит, по крайней мере, один раз за такт системы.

Эта функция возвращает время системного прерывания, которое представляет собой время, прошедшее с момента последнего запуска операционной системы Счетчик времени прерывания начинается с нуля, когда запускается операционная система, и увеличивается при каждом прерывании часов на длину такта. По разным причинам, например из-за различий в оборудовании, длительность системного тактового сигнала может отличаться для разных компьютеров. Для определения величины системного тактового сигнала используют функцию KeQueryTimeIncrement.

3.12. Функция KeQueryTickCount

Функция KeQueryTickCount поддерживает количество прерываний таймера интервала, которые произошли с момента загрузки системы.

Синтаксис:
VOID KeQueryTickCount (_Out_ PLARGE_INTEGER CurrentCount);

Параметр: urrentCount – указатель на значение счетчика тиков по возвращении из KeQueryTickCount.

Значение TickCount увеличивается на единицу при каждом интервале прерывания таймера во время работы системы.

Чтобы определить абсолютное истекшее время необходимо

умножить возвращаемое значение TickCount на возвращаемое значение KeQueryTimeIncrement, используя поддержку компилятора для 64-битных целочисленных операций.

3.13. Функция KeQueryUnbiasedInterruptTime

Функция KeQueryUnbiasedInterruptTime возвращает текущее значение счетчика времени системного прерывания .

Синтаксис:

```
ULONGLONG KeQueryUnbiasedInterruptTime( );
```

Функция KeQueryUnbiasedInterruptTime возвращает текущий счетчик времени прерывания в единицах по 100 наносекунд. Счет начинается с нуля, когда компьютер запускается. Обновление этого счетчика приостанавливается, когда компьютер переходит в состояние сна, и возобновляется, когда компьютер просыпается.

3.14. Функция GetProcessTimes

Функция GetProcessTimes получает информацию о распределении интервалов времени для заданного процесса.

Синтаксис:

```
BOOL GetProcessTimes(
HANDLE hProcess,       // дескриптор процесса
LPFILETIME lpCreationTime, // время создания процесса
LPFILETIME lpExitTime,  // время выхода из работы процесса
LPFILETIME lpKernelTime, // время работы процесса в режиме ядра
LPFILETIME lpUserTime // время работы процесса в режиме пользователя
);
```

Параметры:

hProcess – дескриптор процесса, информация о распределении интервалов времени которого разыскивается. Этот дескриптор должен быть создан с правами доступа PROCESS_QUERY_INFORMATION. Для получения дополнительной информации, см. статью Защита процесса и права доступа.

lpCreationTime – [out], указатель на структуру FILETIME, которая

принимает время создания процесса.

lpExitTime – [out], указатель на структуру FILETIME, которая принимает время выхода из работы процесса. Если процесс не вышел из работы, содержание этой структуры не определенное.

lpKernelTime – [out], указатель на структуру FILETIME, которая принимает величину времени, в течение которого процесс выполнялся в привилегированном режиме (режиме ядра). Чтобы получить это значение, определяется время, в ходе которого каждый из потоков процесса выполнялся в режиме ядра, а затем все эти периоды суммируются вместе.

lpUserTime – [out], указатель на структуру FILETIME, которая принимает величину времени, в течение которого процесс выполнялся в непривилегированном (пользовательском) режиме. Чтобы получить это значение, определяется время, в ходе которого каждый из потоков процесса выполнялся в режиме ядра, а затем все эти периоды суммируются вместе.

Если функция завершается успешно, величина возвращаемого значения – не ноль. Если функция завершается с ошибкой, величина возвращаемого значения – ноль. Чтобы получать расширенные данные об ошибках необходимо вызывать функцию GetLastError.

Все периоды определяются при помощи структур данных FILETIME. Такая структура содержит два значения, которые объединяются в счетчик тактов, размер которых 100 наносекунд.

Дескриптор процесса может ссылаться как на процесс, который продолжает выполняться, так и на процесс, выполнение которого прекратилось. Вычитая время создания процесса (creation time) из времени завершения процесса (exit time) определяется истекшее время.

3.15. Функция NtQueryInformationProcess

Функция NtQueryInformationProcess получает информацию об указанном процессе, в том числе, данных, связанных с отладкой.

Функция NtQueryInformationProcess находится в ntdll.dll.

При описании функции использован параметр из http://solutionmes.wikidot.com/ad-api-3:

```
hmod = LoadLibrary(L"ntdll.dll");
_NtQueryInformationProcess = GetProcAddress(hmod,
"NtQueryInformationProcess");
```

```
status = (_NtQueryInformationProcess) (-1, 0x07, &retVal, 4, NULL);
if (retVal != 0) {
    MessageBox(NULL, L"Debugger Detected Via
NtQueryInformationProcess ProcessDebugPort", L"Debugger Detected",
MB_OK);
} else {
    MessageBox(NULL, L"No Debugger Detected", L"No Debugger
Detected", MB_OK);
}
```

Функция принимает пять параметров, первые два из которых представляют наибольший интерес.

Первый параметр является дескриптором запрашиваемого процесса. В нашем примере используется значение -1. Это соответствует текущему дескриптору запущенного процесса.

Вторым параметром является константа (значение из перечисления ProcessInformationClass), указывающая на тип данных, которые хотим получить из целевого процесса.

Вызов этой функции с дескриптором нашего текущего процесса, вместе со значением ProcessDebugPort (0x07) вернет номер порта для отладки процесса (EPROCESS -> DebugPort).

Если текущий процесс в данный момент отлаживается, то порт будет назначен, и номер порта будет возвращен (значение 0xFFFFFFFF). А если текущий процесс не имеет отладчика, то вернется нулевое значение. Т.к. функция NtQueryInformationProcess используется для внутренних служб операционной системы, поэтому необходимо использовать динамическое связывание: загрузка модуля с помощью LoadLibrary и получение указателя на функцию с помощью GetProcAddress.

Пример использования этой функции на ассемблере рассмотрен в статье «Получение информации о другом процессе» на http://www.manhunter.ru.

3.16. Функция NtQueryInformationThread

Функция **NtQueryInformationThread** получает информацию об указанном потоке.

```
__kernel_entry NTSTATUS NtQueryInformationThread( IN HANDLE
ThreadHandle, IN THREADINFOCLASS ThreadInformationClass,
```

OUT PVOID ThreadInformation, IN ULONG ThreadInformationLength, OUT PULONG ReturnLength);

Параметры:

ThreadHandle – дескриптор потока, информация о котором запрашивается.

ThreadInformationClass – если этот параметр является значением ThreadIsIoPending перечисления THREADINFOCLASS, то функция определяет, ожидает ли поток каких-либо ожидающих операций ввода-вывода.

ThreadInformation – указатель на буфер, в который функция записывает запрошенную информацию.

ThreadInformationLength – размер буфера, на который указывает параметр ThreadInformation , в байтах.

ReturnLength – указатель на переменную, в которой функция возвращает размер запрашиваемой информации.

3.17. Функция NtSetInformationThread

Функция NtSetInformationThread устанавливает приоритет потока.

Синтаксис:

```
NtSetInformationThread(
  IN HANDLE               ThreadHandle,
  IN THREAD_INFORMATION_CLASS ThreadInformationClass,
  IN PVOID               ThreadInformation,
  IN ULONG               ThreadInformationLength );
```

Параметры:

ThreadHandle – хэндл потока, открытый с доступом THREAD_SET_INFORMATION;

ThreadInformationClass – класс информации для установки у потока;

ThreadInformation – указатель на значение для установки;

ThreadInformationLength – длина значения для установки.

3.18. Лабораторная работа «Антиотладка с использованием функций времени»

Цель занятия: приобрести практические навыки составления, отладки и выполнения программ с использованием антиотладочных приемов в среде masm64.

В отчете представить:

– номер, название лабораторной работы и фамилию исполнителя;

– задание;

– текст исходной программы с комментариями к каждой строке программы;

– результат выполнения программы (скриншоты упрощенного окна MessageBox и отладчика);

– особенности выполнения.

Постановка задачи

В работе необходимо:

– запрограммировать уравнение при помощи команд AVX;

– применить не менее 2-х различных функций времени для отслеживания отладки;

– запустить внешний файл для проверки возможности выполнения команд AVX.

Задание

В среде masm64 решить уравнение для целых и вещественных чисел при помощи команд AVX. Рассчитать уравнение не менее чем для 3-х значений любой из переменных. Вывести одной функцией MessageBox фамилию, группу и результаты.

Применить не менее 2-х различных функций времени, одну из которых выбрать из перечня функций:

1. GetSystemTimePreciseAsFileTime;
2. GetSystemTimeAsFileTime;
3. GetSystemTimePreciseAsFileTime;
4. KeTickCount;
5. KeQueryTickCount;
6. KeQueryTimeIncrement;
7. KeQuerySystemTime;
8. KeQueryInterruptTime;
9. KeQueryPerformanceCounter;
10. KeQueryTimeIncrement;
11. QueryPerformanceFrequency.

Некоторые из предлагаемых функций применить не удастся.

Для случая обнаружения отладки вывести через функцию MessageBox соответствующее сообщение.

В программе запустить внешний файл для проверки возможности выполнения команд AVX.

Варианты заданий:

1. 32 – √(b)/ac;	8. 16/b + a√c;	15. a/√b – 8c;	22. 4ab – 8/√c;
2. a√b – 16c;	9. 16a + 16b√c;	16. 32c/7b + √a;	23. √a+ 8b/c2;
3. ab + 32/√c;	10. 8b + 16a/√c;	17. 32 – b/a √c;	24. 32a + b/√c;
4. a/√b – 8c;	11. ab + 8√c;	18. a√c – 8b;	25. 16b + √a/c;
5. 32/b + a√/c;	12. √c + 2b/a;	19. ac + 23√b;	26. b4 + a/√d;
6. ab + 32√c;	13. ac/√b + 8;	20. a√ (b/c) +16;	27. a + (√b)c;
7. a√b — c + b;	14. 32a + √ (bc);	21. a + √(b/c);	28. a/b/√c.

Программа 3.8. Решение уравнения 3^3 – √b (acd) при помощи команд AVX с целыми числами:

```
	include \masm64\include64\masm64rt.inc ; библиотеки
.data ;
	mas1 dd 1,4,2,3 ; a, c, d,3
	masb dd 25,9,16
	titl1 db "Обработка чисел. AVX команды.",0
	buf1 dq 3 dup(0),0		; буфер
	res1 dd 3 dup(0),0
    fmt1 db "masm64.",10,10,"Уравнение 3^3 – sqrt b (acd)",10,
    "Результат = ",3 dup(" %d "),10,10,
    "Автор: Рысованый А.Н., каф. ВТП, НТУ ХПИ",0

.code ;
entry_point proc	; 3^3 – sqrt b (acd)
  lea rbx,masb   ; адр. masb
  lea rdx,res1   ; адр. res1
  mov rcx,3
    vmovd xmm1,mas1[0]    ; a
    vmovd xmm2,mas1[4]    ; c
    vmovd xmm3,mas1[8]    ; d
    vmovd xmm4,mas1[12]   ; 3
    vpmulld xmm5,xmm4,xmm4 ; 3 x 3
    vpmulld xmm6,xmm5,xmm5 ; 3^3
;movss res1,xmm6 ; для проверки результата
;mov r10d,res1 ;
	vpmulld xmm7,xmm1,xmm2 ; ac
```

```
        vpmulld xmm8,xmm7,xmm3 ; acd
@@:
        vmovd xmm0,dword ptr[rbx] ;
        vmovd r10d,xmm0 ;
        vcvtsi2ss xmm0,xmm0,r10d ;
        vsqrtss xmm0,xmm0,xmm0 ;
        cvttss2si r10d,XMM0           ;
        vmovd xmm0,r10d
        vpmulld xmm9,xmm0,xmm8 ;
        vsubss xmm10,xmm6,xmm9
        movss res1,xmm10 ;
        mov r10d,res1 ;

  mov [rdx],r10d
  add rbx,4
  add rdx,4
  dec rcx
  jnz @b
        movsxd r10,res1
        movsxd r11,res1[4]
        movsxd r12,res1[8]

  invoke wsprintf,addr buf1,ADDR fmt1,r10,r11,r12
  invoke MessageBox,0,addr buf1,addr titl1,MB_OK
        invoke ExitProcess,0
        entry_point endp
        end
```

Результат выполнения программы представлен на рис. 3.7.

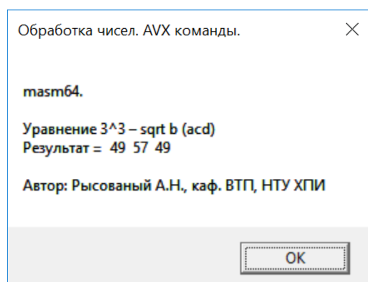


Рис. 3.7. Результат выполнения программы

Программа 3.9. Вычисление уравнения ab + c/d – sqrt(e) для целых и для вещественных чисел:

```
        include \masm64\include64\masm64rt.inc
.data
        res1 dd 0,0
        titl1 db "Уравнение. Решение уравнения. AVX команды.",0;
        buf1 dq 0,0; буфер
fmt1 db "masm64. Числа целые: 9, 1, 4, 2, 64 ",10,10,"Уравнение ab + c/d –
sqrt(e);",10,
"Результат = %d",10,10,
  "Автор: каф. ВТП, НТУ ХПИ",0
        mas1 dd 9, 1, 4, 2, 64 ; a, b, c, d, e    - целые

.data
          titl2 db "masm64. Решение уравнения. AVX команды.",0
          buf2 dq 0,0;
fmt2 db "masm64. Числа вещественные: 9.,1.,4.,2.,64.",10,10,"Уравнение
ab + c/d – sqrt(e);",10,
"Результат = %d",10,10,
"Автор:  каф. ВТП, НТУ ХПИ",0
        mas2 DQ 9., 1., 4., 2., 64. ; a, b, c, d, e   - вещественные
.code
proc1 proc            ;процедура ab + c/d – sqrt(e)
        vmovd xmm1,mas1[0]  ;a
        vmovd xmm2,mas1[4]  ;b
        vmovd xmm3,mas1[8]  ;c
        vmovd xmm4,mas1[12] ;d
        vmovd xmm5,mas1[16] ;e

        vmovd r10d,xmm5       ;  e
vcvtsi2ss xmm0,xmm0,r10d ; преобр. 32-разр. целого в вещественное
vsqrtss xmm0,xmm0,xmm0  ; корень вещественного числа
cvttss2si r10d,XMM0    ; преобразование  в 32-разрядное целое число
        vmovd xmm0,r10d       ; сохранение sqrt (e) в хмм0

        vpmulld xmm6,xmm2,xmm1 ;ab
        vmovd eax,xmm3  ;  подготовка к делению
        vmovd ebx,xmm4  ;
        xor edx,edx     ;
        div ebx;
        vmovd xmm7,eax;  сохранение результата деления
        vaddss xmm8,xmm7,xmm6; ab + c/d
        vsubss xmm9,xmm8,xmm0; ab + c/d - sqrt(e)
```

```
        movss res1,xmm9;
invoke wsprintf,addr buf1,ADDR fmt1,res1
invoke MessageBox,0,ADDR buf1,ADDR titl1,MB_ICONINFORMATION
  ret
        proc1 endp;
        ;________________________________________
proc2 proc; процедура  ; ab + c/d
        vmovsd xmm1,mas2[0]    ; a
        vmovsd xmm2,mas2[8]    ; b
        vmovsd xmm3,mas2[16]   ; c
        vmovsd xmm4,mas2[24]   ; d
        vmovsd xmm5,mas2[32]   ; e

        vsqrtsd xmm5,xmm5,xmm5  ; корень вещественного числа

        vmulsd xmm6,xmm2,xmm1; ab
        vdivsd xmm7,xmm3,xmm4; c/d
        vaddsd xmm8,xmm6,xmm7; ab + c/d
        vsubsd xmm9,xmm8,xmm5; ab + c/d - sqrt(e)
        vcvttsd2si r15d,xmm9 ;
        movsxd r15,r15d

invoke wsprintf,addr buf2,ADDR fmt2,r15
invoke MessageBox,0,addr buf2,addr titl2,MB_ICONINFORMATION
  ret
proc2 endp;

entry_point proc
        invoke proc1
        invoke proc2
        invoke ExitProcess,0
entry_point endp
        end
```

Результат выполнения программы представлен на рис. 3.8.

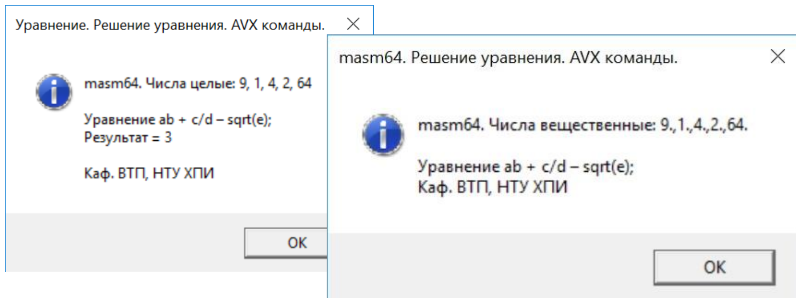


Рис. 3.8. Результат выполнения программы

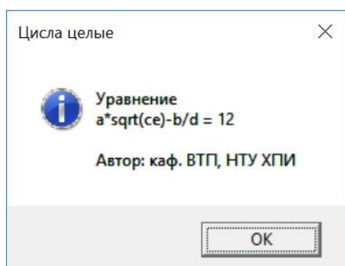
Программа 3.10. Вычисление уравнения a*sqrt(ce)-b/d для целых и для вещественных чисел и запуском внешней программы на выполнение:

```
    include \masm64\include64\masm64rt.inc
.data
    decemal_array dq 4,11,1,32,9
    lenth1 equ ($-array)/type array
    titl db "Цисла целые",0
    buf1 dq 10 dup(0,0) ; буфер вывода сообщения
    ifmt db "Уравнение",10,"a*sqrt(ce)-b/d = %d",10,10,
    "Автор: каф. ВТП, НТУ ХПИ",0
    res dq 0
  szRun db "C:\Masm64\bin64\avx.exe",0

.data
    float_array dq 4.0,11.0,1.0,32.0,9.0
    lenth2 equ ($-array)/type array
    titl2 db "Цисла вещественные",0
    buf2 dq 10 dup(0,0); буфер вывода сообщения
    ifmt2 db "a*sqrt(ce)-b/d = %d",10,10,
    "Автор: каф. ВТП, НТУ ХПИ",0
    res2 dq 0

.code
    count1 proc; a*sqrt(ce)-b/d
    vcvtsi2sd xmm0,xmm0,decemal_array[0] ; a
    vcvtsi2sd xmm1,xmm1,decemal_array[8] ; b
    vcvtsi2sd xmm2,xmm2,decemal_array[16]; c
    vcvtsi2sd xmm3,xmm3,decemal_array[24]; d
    vcvtsi2sd xmm4,xmm4,decemal_array[32]; e
    vmulsd xmm2,xmm2,xmm4
```

```
        vsqrtsd xmm2,xmm2,xmm2
        vmulsd xmm0,xmm0,xmm2
        vdivsd xmm1,xmm1,xmm3
        vsubsd xmm0,xmm0,xmm1
        vcvtsd2si r10,xmm0
        mov res,r10
invoke wsprintf,addr buf1,ADDR ifmt,res
invoke MessageBox,0,addr buf1,addr titl,MB_ICONINFORMATION
        ret
        count1 endp

        count2 proc; a*sqrt(ce)-b/d
        vmovsd xmm0,float_array[0] ; a
        vmovsd xmm1,float_array[8] ; b
        vmovsd xmm2,float_array[16]; c
        vmovsd xmm3,float_array[24]; d
        vmovsd xmm4,float_array[32]; e
        vmulsd xmm2,xmm2,xmm4
        vsqrtsd xmm2,xmm2,xmm2
        vmulsd xmm0,xmm0,xmm2
        vdivsd xmm1,xmm1,xmm3
        vsubsd xmm0,xmm0,xmm1
        vcvtsd2si r10,xmm0
invoke wsprintf,addr buf2,ADDR ifmt2,r10
invoke MessageBox,0,addr buf2,addr titl2,MB_OK
        ret
        count2 endp

entry_point proc
invoke WinExec,ADDR szRun,SW_HIDE
        invoke count1
        invoke count2

        invoke ExitProcess,0
        entry_point endp
        end
```

Содержимое внешней программы с проверкой поддержки микропроцессором команд AVX не приведено, т.к. не однократно рассматривалось (см. программу 3.5).

Результат выполнения программы представлен на рис. 3.9.



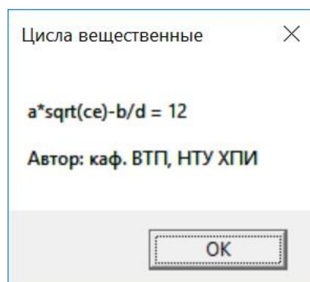


Рис. 3.9. Результат выполнения программы

4. ФУНКЦИИ МУЛЬТИМЕДИЙНОГО ТАЙМЕРА

Основным недостатком обычных таймеров Windows является невысокая предельная частота (18,2 Гц) и, следовательно, низкое временное разрешение (55 мс). При использовании этим таймером для перемещения объекта по экрану с разрешением экрана 800 x 600 пикселов объект затратит на прохождение экрана по горизонтали более 40 с, т. е. движение его будет медленным. Для получения более высоких скоростей перемещения объектов на мониторе и для отсчета интервалов времени с более высокой точностью можно использовать мультимедийные таймеры, предельное разрешение которых составляет 1 мс, что соответствует частоте 1 кГц.

Основной смысл применения мультимедийных таймеров – это синхронизация рассчитанного движения и анимации.

Для работы с мультимедийными таймерами предусмотрена небольшая группа специальных функций, имена которых начинаются со слова time (начинающегося со строчной буквы, что нетипично для имен функций Windows).

Следующие функции используются с мультимедийными таймерами:

– timeBeginPeriod;
– timeEndPeriod;
– timeGetDevCaps;
– timeGetSystemTime;
– timeGetTime;
– timeKillEvent;
– TimeProc;
– timeSetEvent.

Если необходима большая точность определения времени (большая точность), то вызывается мультимедийный таймер Windows. Его точность составляет 1 мс. Вначале вызывается функция timeBeginPeriod, которая сообщает операционной системой о необходимости в высокоточном хронометрировании, а затем вызывается функция timeSetEvent для запуска мультимедийного таймера.

Чтобы остановить таймер вызывается функция timeEndPeriod.

Для окончания определения времени используется функция timeKiilEvent. Эта функция информирует операционную систему о том, что высокая точность измерения времени больше не нужна.

4.1. Использование функции timeBeginPeriod

Функция timeBeginPeriod запрашивает минимальное разрешение для периодических таймеров.

Синтаксис:

MMRESULT timeBeginPeriod(UINT uPeriod)

Параметр: uPeriod – минимальное разрешение таймера в миллисекундах для приложения или драйвера устройства. Более низкое значение указывает более точное разрешение.

Возвращаемое значение: возвращает TIMERR_NOERROR в случае успеха или TIMERR_NOCANDO, если разрешение, указанное в uPeriod, выходит за пределы диапазона.

Располагается в библиотеке winmm.lib.

Минимальное разрешение таймера можно определить после выполнения программы 4.1.

Программа 4.1. Определение минимального разрешения таймера:

```
    include \masm64\include64\masm64rt.inc ; библиотеки
   include \masm64\include64\winmm.inc
   includelib \masm64\lib64\winmm.lib
.data
  titl1 db "Исследование ф-и timeBeginPeriod",0
  ifmt1 db "Минимальное разрешение таймера = %d",0
    buf1 dq 0
    uPeriod dq 0
.code
  entry_point proc
    invoke timeBeginPeriod,uPeriod
invoke wsprintf,ADDR buf1,ADDR ifmt1,rax;
invoke MessageBox,0,addr buf1,addr titl1,MB_ICONINFORMATION;
    invoke ExitProcess,0
    entry_point endp
    end
```

При использовании мультимедийных функций необходимо присоединить библиотеки, где эти функции располагаются. Вторую и третью строчки программы можно перенести в файл masm64rt.inc или непосредственно расположить в программе.

Результат выполнения программы представлен на рис. 4.1.

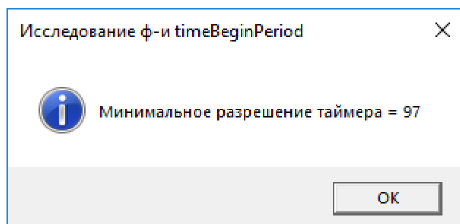


Рис. 4.1. Результат выполнения программы

4.2. Функция timeEndPeriod

Функция timeEndPeriod очищает предварительно установленное минимальное разрешение таймера.

Синтаксис:

MMRESULT timeEndPeriod(UINT uPeriod)

Параметр: uPeriod – минимальное разрешение таймера, указанное в предыдущем вызове функции timeBeginPeriod.

Возвращаемое значение: возвращает TIMERR_NOERROR в случае успеха или TIMERR_NOCANDO, если разрешение, указанное в uPeriod, выходит за пределы диапазона.

4.3. Функция timeGetDevCaps

Функция timeGetDevCaps запрашивает устройство таймера для определения его разрешения.

Синтаксис:

MMRESULT timeGetDevCaps(LPTIMECAPS ptc, UINT cbtc);

Параметры:

ptc –указатель на структуру TIMECAPS. Эта структура заполнена

информацией о разрешающей способности устройства таймера;
cbtc – размер в байтах структуры TIMECAPS .

Возвращаемое значение – возвращает MMSYSERR_NOERROR в случае успеха или код ошибки в противном случае. Возможные коды ошибок включают следующее:

MMSYSERR_ERROR – общий код ошибки;

TIMERR_NOCANDO – параметр ptc имеет значение NULL, либо параметр cbtc недействителен, либо произошла какая-то другая ошибка.

4.4. Функция timeGetSystemTime

Функция timeGetSystemTime извлекает системное время в **миллисекундах**. Системное время – это время, прошедшее с момента запуска Windows. Эта функция очень похожа на функцию timeGetTime.

Синтаксис:

```
MMRESULT timeGetSystemTime( LPMMTIME pmmt, UINT cbmmt );
```

Параметры:

Pmmt – указатель на структуру MMTIME;

Cbmmt – размер в байтах структуры MMTIME.

Возвращаемое значение – в случае успеха возвращает TIMERR_NOERROR. В противном случае возвращает код ошибки.

4.5. Использование функции timeGetTime

Функция timeGetTime извлекает системное время в **миллисекундах**. Системное время – это время, прошедшее с момента запуска Windows.

Синтаксис:

```
DWORD timeGetTime();
```

Функция не имеет параметров. Функция возвращает системное время в миллисекундах. Единственной разницей между этой функцией и функцией timeGetSystemTime является использование timeGetSystemTime структуры типа MMTIME для возвращения системного времени.

Возвращаемое значение сбрасывается в нуль каждые 2^32

миллисекунд, что составляет примерно 49.71 дней.

Для увеличения точности используют функции timeBeginPeriod и timeEndPeriod. В этом случае минимальная разница между двумя успешно возвращенными функцией timeGetTime значениями может быть меньше минимального периода, установленного функциями timeBeginPeriod и timeEndPeriod.

Разница между функцией timeGetTime и функцией timeGetSystemTime заключается в том, что функция timeGetSystemTime использует структуру MMTIME для возврата системного времени. Функция timeGetTime имеет меньше накладных расходов, чем timeGetSystemTime.

Пример получения системного времени в миллисекундах рассмотрено в программе 4.2.

Программа 4.2. Получение системного времени:

```
      include \masm64\include64\masm64rt.inc ; библиотеки
      include \masm64\include64\winmm.inc
      includelib \masm64\lib64\winmm.lib
.data
  titl1 db "Исследование ф-и  timeGetTime",0
  ifmt1 db 'Системное время  = %d мс',10,
  "Время, прошедшее с момента запуска Windows",0
  buf1 dq 0
.code
entry_point proc
      invoke timeGetTime
       invoke wsprintf,ADDR buf1,ADDR ifmt1,rax;
invoke MessageBox,0,addr buf1,addr titl1, MB_ICONINFORMATION;
      invoke ExitProcess,0
      entry_point endp
      end
```

Результат выполнения программы с использованием функции timeGetTime приведен на рис. 4.2.

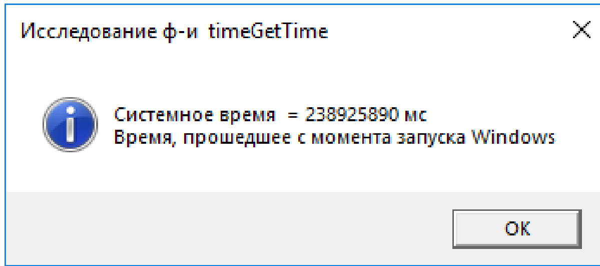


Рис. 4.2. Результат выполнения программы

Для того, чтобы обнаружить процесс отладки необходимо в начале и в конце блока кода использовать функции timeGetTime. И если разница времени выполнения этого блока кода превышает, например, 1 мс, то файл находится под отладкой. Пример вычисления времени выполнения части кода в миллисекундах представлен в программе 4.3.

Программа 4.3. Вычисление времени выполнения части кода в миллисекундах:

```
        include \masm64\include64\masm64rt.inc ; библиотеки
        include \masm64\include64\winmm.inc
        includelib \masm64\lib64\winmm.lib
.data
titl1 db "masm64. Исследование трассировки",0
ifmt1 db "Использование мультимедийной функции timeGetTime.",10,
        "Время выполнения кода %d мс",0
        buf1 dq 0

.code
  entry_point proc
        mov r15,0fffffffh
invoke timeGetTime
        xchg rax,rbx
m1:
        dec r15
        jnz m1
        nop
invoke timeGetTime
        sub rax,rbx
 invoke wsprintf,ADDR buf1,ADDR ifmt1,rax;
```

```
invoke MessageBox,0,addr buf1,addr titl1, MB_ICONINFORMATION;
    invoke ExitProcess,0
    entry_point endp
    end
```

Результат выполнения программы с использованием функции timeGetTime вычисления времени выполнения части кода в миллисекундах приведен на рис. 4.3.

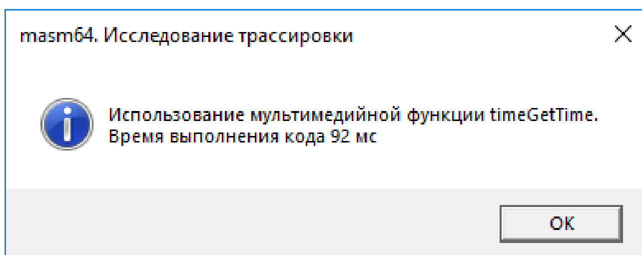


Рис. 4.3. Результат выполнения программы

В связи с тем, что функция timeGetTime вычисляет время в мс, а самая простая ассемблерная команда выполняется за единицы нс и меньше, то в цикле применяется константа большой величины 0fffffffh (для того, чтобы цикл выполнялся не за нс, а за мс).

4.6. Функция timeKillEvent

Функция timeKillEvent отменяет указанное событие таймера.

Синтаксис:

```
MMRESULT timeKillEvent( UINT uTimerID );
```

Параметр uTimerID – идентификатор события таймера для отмены. Этот идентификатор был возвращен функцией timeSetEvent, когда было установлено событие таймера.

Возвращает TIMERR_NOERROR в случае успеха или MMSYSERR_INVALPARAM, если указанное событие таймера не существует.

4.7. Функция TimeProc

Функция обратного вызова функции TimeProc, которая вызывается один раз по истечении одного события или периодически по истечении периодических событий.

Синтаксис:

```
void CALLBACK TimeProc(
  UINT  uID,
  UINT  uMsg,
  DWORD dwUser,
  DWORD dw1,
  DWORD dw2
);
```

Параметры:

uID – идентификатор события таймера. Этот идентификатор был возвращен функцией timeSetEvent, когда было установлено событие таймера;

uMsg – зарезервировано;

dwUser – данные экземпляра пользователя передаются параметру dwUser timeSetEvent;

dw1 – зарезервировано;

dw2 – зарезервировано.

4.8. Использование функции timeSetEvent

Функция timeSetEvent запускает указанное событие таймера. Мультимедийный таймер работает в своем собственном потоке. После того, как событие активировано, оно вызывает указанную функцию обратного вызова или устанавливает или запускает указанный объект события.

Эта функция устарела. В новых приложениях желательно использовать функцию CreateTimerQueueTimer для создания таймера в ожидании (в очереди таймера).

Синтаксис функции timeSetEvent:

```
MMRESULT timeSetEvent(
  UINT           uDelay,
  UINT           uResolution,
  LPTIMECALLBACK lpTimeProc,
```

```
  DWORD_PTR     dwUser,
  UINT          fuEvent
);
```

Параметры:

uDelay – задержка события в миллисекундах. Если это значение не находится в диапазоне минимальной и максимальной задержек событий, поддерживаемых таймером, функция возвращает ошибку;

uResolution – разрешение события таймера в миллисекундах. Разрешение увеличивается с меньшими значениями; разрешение 0 указывает, что периодические события должны происходить с максимально возможной точностью. Однако, чтобы уменьшить нагрузку на систему желательно использовать максимальное значение, соответствующее вашему приложению;

lpTimeProc – указатель на функцию обратного вызова, которая вызывается один раз по истечении одного события или периодически по истечении периодических событий.

Если параметр fuEvent указывает флаг TIME_CALLBACK_EVENT_SET или TIME_CALLBACK_EVENT_ PULSE, то параметр lpTimeProc интерпретируется как дескриптор объекта события. Событие будет установлено или пульсировано после завершения одного события или периодически после завершения периодических событий.

Для любого другого значения fuEvent параметр lpTimeProc является указателем на функцию обратного вызова типа LPTIMECALLBACK;

dwUser – данные? определяемые пользователем;

fuEvent – тип события таймера. Этот параметр может включать одно из следующих значений:

TIME_ONESHOT – событие происходит один раз, через миллисекунды.

TIME_PERIODIC – событие происходит каждые миллисекунды.

Параметр fuEvent также может включать одно из следующих значений.

TIME_CALLBACK_FUNCTION – при истечении таймера Windows вызывает функцию, указанную параметром lpTimeProc (по умолчанию).

TIME_CALLBACK_EVENT_SET – при истечении таймера Windows вызывает функцию SetEvent, чтобы установить событие, на которое указывает параметр lpTimeProc. Параметр dwUser

игнорируется.

TIME_CALLBACK_EVENT_PULSE – при истечении таймера Windows вызывает функцию PulseEvent для отправки события, на которое указывает параметр lpTimeProc. Параметр dwUser игнорируется.

TIME_KILL_SYNCHRONOUS – передача этого флага предотвращает возникновение события после вызова функции timeKillEvent.

Программа 4.4. Запуск события таймера:

```
include \masm64\include64\masm64rt.inc ; библиотеки
include \masm64\include64\winmm.inc
includelib \masm64\lib64\winmm.lib

.code
TimerProc proc uTimerID,uMsg,dwUser,dw1,dw2
  invoke Beep,440,100 ; частота, длительность в мс
     ret
TimerProc endp

entry_point proc
invoke timeSetEvent,1000,0,addr TimerProc,0,TIME_PERIODIC
      test rax,rax ; логическое И без изменения результата
      jz error
      mov  rbx,rax
      invoke Sleep,7200  ; задержка кода
      invoke timeKillEvent,rbx
error:
      invoke Beep,1000,1000 ; частота, длительность в мс
rcall MessageBox,0,"Проверка выполнена","Проверка ф-и timeSetEvent",
MB_OK
         invoke ExitProcess,0

      entry_point endp
      end
```

В первой строке программы выполняется функция timeSetEvent, которая вызывает функцию TimerProc с задержкой выполнения 1000 мс (приблизительно 1 с). Таким образом, через 1000 мс звучит сигнал динамика, который вызывается строкой invoke Beep,440,100.

За 7000 мс с задержкой 1000 мс может прозвучать 7 сигналов

динамика. Дополнительные 200 мс из 7200 мс необходимы на выполнения строк

```
invoke timeKillEvent,rbx
invoke Beep,1000,1000 ;
```

Если не зарезервировать дополнительное время (200 мс), то не прозвучит окончательный сигнал, который определяется строкой

```
invoke Beep,1000,1000 ; частота, длительность в мс
```

После вывода всех сигналов выводится сообщение, представленное на рис. 4.4.

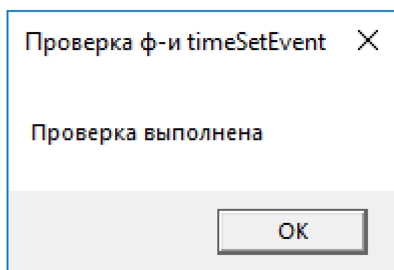


Рис. 4.4. Результат выполнения программы

Разрешение системного таймера можно определить при помощи функции **GetSystemTimeAdjustment**:

```
BOOL GetSystemTimeAdjustment(
  PDWORD lpTimeAdjustment,
  PDWORD lpTimeIncrement,
  PBOOL lpTimeAdjustmentDisabled
);
```

Параметры:

– lpTimeAdjustment – [out], указатель на значение, в которое функция устанавливает число 100 наносекундных блоков, добавленных к часам времени дня при каждой периодической корректировке времени.

– lpTimeIncrement – [out], указатель на значение, в которое функция устанавливает интервалы между периодическими корректировками времени 100 наносекундными блоками. Этот интервал

– период времени между прерываниями часов системы.

– lpTimeAdjustmentDisabled – [out], указатель на значение, в которое функция устанавливает индикатор того, в действительности ли действует периодическая корректировка времени.

Значение TRUE указывает, что периодическая корректировка времени заблокирована. При каждом прерывании часов система просто добавляет интервал между прерываниями часов в часы времени дня. Однако система является свободной, чтобы настраивать ее часы времени дня, используя другие методы. Эти другие методы могут стать причиной того, что часы времени дня заметно перепрыгивают, когда делаются корректировки.

Значение FALSE указывает на то, что периодическая корректировка времени используется, чтобы настроить часы времени дня. При каждом прерывании часов система добавляет приращение времени, определенное параметром dwTimeIncrement функции

4.9. Лабораторная работа «Антиотладка с использованием функций мультимедийного таймера»

Цель занятия: приобрести практические навыки составления, отладки и выполнения программ с использованием антиотладочных приемов в среде masm64.

В отчете представить:

– номер, название лабораторной работы и фамилию исполнителя;

– задание;

– текст исходной программы с комментариями к каждой строке программы;

– результат выполнения программы (скриншоты упрощенного окна (MessageBoxIndirect) и отладчика);

– особенности выполнения.

Постановка задачи

В работе необходимо:

– запрограммировать уравнение для целых и для вещественных чисел при помощи команд AVX и в разных потоках;

– использовать **три** различных проверки на отладку с использованием функций мультимедийного таймера;

– выполнить проверку возможности выполнения команд AVX.

Задание

Варианты заданий:

1. 32 – √(b)/ac;	8. 16/b + a√c;	15. a/√b – 8c;	22. 4ab – 8/√c;
2. a√b – 16c;	9. 16a + 16b√c;	16. 32c/7b + √a;	23. √a+ 8b/c2;
3. ab + 32/√c;	10. 8b + 16a/√c;	17. 32 – b/a √c;	24. 32a + b/√c;
4. a/√b – 8c;	11. ab + 8√c;	18. a√c – 8b;	25. 16b + √a/c;
5. 32/b + a√/c;	12. √c + 2b/a;	19. ac + 23√b;	26. b4 + a/√d;
6. ab + 32√c;	13. ac/√b + 8;	20. a√ (b/c) +16;	27. a + (√b)c;
7. a√b — c + b;	14. 32a + √ (bc);	21. a + √(b/c);	28. a/b/√c.

Примеры основной части

Программа 4.5:

```
; Решить уравнение для целых и для вещественных чисел
; при помощи команд AVX, оформленных в разных процедурах.
; Числа заданы массивами.
; Результат вывести через функцию MessageBox:
; 3.c √e + a/b;
include \masm64\include64\masm64rt.inc
.data?
        buf BYTE 64 dup (?)
        tmp BYTE 64 dup (?)
        result REAL8 ?
.data
        buf_ptr QWORD buf
        temp_ptr QWORD tmp
        arr1 REAL8 1.0, 2.0, 4.0, 16.0
        ;               0.a  1.b  2.c  3.e
        arr2 DWORD 1, 2, 4, 16

.code
; Основная процедура вычисления уравнения
; Вызывается процедурами calculateInt и calculateFloat
; для вычисления целых и вещественных чисел соответственно

calculate proc          ; c * sqrt(e) +  a / b
vsqrtsd xmm0, xmm0, REAL8 PTR[rsi + sizeof REAL8 * 3] ; xmm0 = sqrt(e)
vmovsd xmm1, REAL8 PTR[rsi + sizeof REAL8 * 0]         ;  xmm1 = a
vdivsd xmm2, xmm1, REAL8 PTR[rsi + sizeof REAL8 * 1] ; xmm2 = a / b
vmovsd xmm1, REAL8 PTR[rsi + sizeof REAL8 * 2]         ; xmm1 = c
vfmadd213sd  xmm0, xmm1, xmm2       ; xmm0 = (xmm0 * xmm1) + xmm2
movsd result, xmm0  ; 4 * 4 + 1/2 = 16.5
        ret
```

```
calculate endp


; Принимает массив целых чисел,
; преобразует его в массив вещественных чисел
; и вычисляет уравнение с помощью процедуры calculate
; Результат-строка в RAX
calculateInt proc
            LOCAL new_arr[LENGTHOF arr2]: REAL8
            lea rsi, arr2
            vcvtdq2pd xmm0, qword ptr[rsi + (sizeof DWORD * 2) * 0]
            vcvtdq2pd xmm1, qword ptr[rsi + (sizeof DWORD * 2) * 1]
            ; занести каждое число в новый масив
            lea rsi, new_arr
            vmovlpd REAL8 ptr[rsi + sizeof REAL8 * 0], xmm0
            vmovhpd REAL8 ptr[rsi + sizeof REAL8 * 1], xmm0
            vmovlpd REAL8 ptr[rsi + sizeof REAL8 * 2], xmm1
            vmovhpd REAL8 ptr[rsi + sizeof REAL8 * 3], xmm1
        call calculate
            vcvtsd2si rax, result
            mcat buf_ptr, cfm$("Danylevych, CIT-117G \n "), str$(rax)
            mov rax, buf_ptr
   ret
calculateInt endp


; Принимает масив вещественных чисел,
; и вычисляет уравнение с помощью процедуры calculate
; Результат-строка в RAX
calculateFloat proc
            lea rsi, arr1
  call calculate
  rcall fptoa, result, temp_ptr
            mcat buf_ptr, cfm$("Danylevych, CIT-117G \n "), temp_ptr
            mov rax, buf_ptr
            ret
calculateFloat endp

; Проверка наличия AVX
check_avx proc
            mov rax, 1
            cpuid
            bt ecx, 28
            jb AVXpresent
```

```
invoke MessageBox, 0, "Процессор не поддерживает расширения инструкций AVX", "Проверка команд AVX", MB_ICONINFORMATION
            jmp toExit
AVXpresent:
invoke MessageBox, 0, "Процессор поддерживает расширения инструкций AVX", "Проверка команд AVX", MB_ICONINFORMATION
            mov rax, 7
   mov rcx, 0
   cpuid
   bt ebx, 5
            jb AVX2present
invoke MessageBox, 0, "Процессор не поддерживает расширения инструкций AVX2", "Проверка команд AVX2", MB_ICONINFORMATION
            jmp toExit
AVX2present:
invoke MessageBox, 0, "Процессор поддерживает расширения инструкций AVX2", "Проверка команд AVX2", MB_ICONINFORMATION
toExit:
            ret
check_avx endp

       ;  Основная программа
       ; Проверяет наличие AVX
       ; Вычисляет уравнение для целых,
       ; а затем и для вещественных чисел,
       ; при этом выводя результаты в отдельных окнах
entry_point proc
            call check_avx
            call calculateInt
       invoke MsgboxI,0,buf_ptr, "Results Int", MB_OK,10
            call calculateFloat
       invoke MsgboxI,0,buf_ptr, "Results Float", MB_OK,10
            .exit
       entry_point endp
         end
```

Программа 4.6:

```
       include \masm64\include64\masm64rt.inc ; библиотеки
       include \masm64\include64\winmm.inc
       includelib \masm64\lib64\winmm.lib
.data
       res1 dd 0,0
       mas1 dd 2,2,4,16,4 ; a, b, c, e, d  - ЦЕЛЫЕ
       mas2 DQ 2.,2.,4.,16.,4. ; a, b, c, d  - ВЕЩЕСТВЕННЫЕ
```

```
        titl1 db "Решение уравнения. AVX команды.ЦЕЛЫЕ",0
        buf1 dq 0,0; буфер
        fmt1 db "masm64.",10,10,"Уравнение  ab?c + e/d",10,
        "Результат = %d",10,10, " каф. ВТП, НТУ ХПИ",0

        titl2 db "Решение уравнения. AVX команды.ВЕЩЕСТВЕННЫЕ",0
        buf2 dq 0,0; буфер
        fmt2 db "masm64.",10,10,"Уравнение ab√c + e/d",10,
        "Результат = %d",10,10,
        "Автор: Матвєєв Микита КИТ-117Д, каф. ВТП, НТУ ХПИ",0

.code ; ab√?c + e/d
  proc1 proc                    ;ЦЕЛЫЕ PROC1
        vmovd xmm1,mas1[0]    ; a
        vmovd xmm2,mas1[4]    ; b
        vmovd xmm3,mas1[8]    ; c
        vmovd xmm4,mas1[12]   ; e
        vmovd xmm5,mas1[16]   ; d

        vpmulld xmm6,xmm1,xmm2  ; ab

vmovd r10d,xmm3                 ;  c
vcvtsi2ss xmm0,xmm0,r10d  ; преобр. 32-разр. целого в вещественное
vsqrtss xmm0,xmm0,xmm0  ; корень вещественного числа
cvttss2si r10d,XMM0             ; преобраз.  в 32-разрядное целое число
vmovd xmm0,r10d                 ; сохранение sqrt (c) в хмм0

        vpmulld xmm7,xmm6,xmm0             ;  ab√c

        vmovd eax,xmm4 ; подг. к делению
        vmovd ebx,xmm5 ; подг. к делению
        xor edx,edx ; подг. к делению
        div ebx                          ; e/d
        vmovd xmm8,eax ; сохранение результата деления

        vaddss xmm9,xmm8,xmm7 ; ab√c + e/d
        movss res1,xmm9 ;

invoke wsprintf,addr buf1,ADDR fmt1,res1
invoke MessageBox,0,addr buf1,addr titl1, MB_ICONINFORMATION
        ret
    proc1 endp
```

```
proc2 proc                                ; ВЕЩЕСТВЕННЫЕ PROC2
        vmovsd xmm1,mas2[0]    ; a
        vmovsd xmm2,mas2[8]    ; b
        vmovsd xmm3,mas2[16]    ; c
        vmovsd xmm4,mas2[24]   ; e
        vmovsd xmm5,mas2[32]   ; d

        vmulsd xmm6,xmm1,xmm2  ; ab
        vsqrtsd xmm3,xmm3,xmm3
        vmulsd xmm7,xmm6,xmm3            ;  ab√c
        vdivsd xmm8,xmm4,xmm5; e/d

        vaddsd xmm9,xmm8,xmm7;
        vcvttsd2si r15d,xmm9 ;
        movsxd r15,r15d

invoke wsprintf,addr buf2,ADDR fmt2,r15
invoke MessageBox,0,addr buf2,addr titl2,MB_ICONINFORMATION
        ret
        proc2 endp


        entry_point proc
        invoke proc1
        invoke proc2
        invoke ExitProcess,0
        ret
        entry_point endp
        end
```

5. API-ФУНКЦИИ ДЛЯ ОБНАРУЖЕНИЯ ОТЛАДЧИКА

Для определения отладки используются следующие API-функции:

– IsDebuggerPresent – определяет, отлаживается ли вызывающий процесс отладчиком пользовательского режима;

– CheckRemoteDebuggerPresent – определяет, выполняется ли отладка указанного процесса;

– ContinueDebugEvent – позволяет отладчику продолжить поток, который ранее сообщал о событии отладки;

– DebugActiveProcess – позволяет отладчику подключаться к активному процессу и отлаживать его;

– DebugActiveProcessStop – останавливает отладчик для отладки указанного процесса;

– DebugBreak – вызывает исключение точки останова в текущем процессе;

– DebugBreakProcess – вызывает исключение точки останова в указанном процессе;

– DebugSetProcessKillOnExit – устанавливает действие, которое будет выполнено при выходе из вызывающего потока;

– FatalExit – передает управление выполнением отладчику;

– FlushInstructionCache – очищает кэш инструкций для указанного процесса;

– GetThreadContext – получает контекст указанного потока;

– GetThreadSelectorEntry – извлекает запись таблицы дескрипторов для указанного селектора и потока;

– OutputDebugString – отправляет строку в отладчик для отображения;

– ReadProcessMemory – читает данные из области памяти в указанном процессе;

– SetThreadContext – устанавливает контекст для указанного потока;

– WaitForDebugEvent – ожидание события отладки в процессе отладки;

– Wow64GetThreadContext – получает контекст указанного потока WOW64;

– Wow64GetThreadSelectorEntry – извлекает запись таблицы дескрипторов для указанного селектора и потока WOW64;

– Wow64SetThreadContext – устанавливает контекст указанного потока WOW64;

– WriteProcessMemory – записывает данные в область памяти в указанном процессе.

5.1. Использование функции IsDebuggerPresent

Функция IsDebuggerPresent определяет, отлаживается ли вызывающий процесс отладчиком пользовательского режима. Эта функция не имеет параметров. Если текущий процесс выполняется в контексте отладчика, возвращаемое значение отлично от нуля. Если текущий процесс не запущен, возвращаемое значение равно нулю.

Пример применения этой функции приведен в программе 5.1.

Программа 5.1:

```
include \masm64\include64\masm64rt.inc ;
.data
titl1 db "Исследование трассировки",0
szTest db "Программу НЕ трассируют.",10,
"Исследование функции IsDebuggerPresent",0
szTest2 db "Программу трассируют.",10,
"Исследование функции IsDebuggerPresent",0
.code
entry_point proc

invoke IsDebuggerPresent ;
cmp rax,0 ;
  jnz debug1 ;
  invoke MessageBox,0,addr szTest,addr titl1,MB_OK
jmp @f
debug1:
invoke MessageBox,0,addr szTest2,addr titl1,MB_OK
@@:
invoke ExitProcess,0
entry_point endp
end
```

В случае, если функция IsDebuggerPresent выполняется под отладчиком, то возвращаемое значение в регистре rax=1.

Установка признаков выполнения предыдущей операции выполняется командой

cmp rax,0,

после которой осуществляются ветвления на одно из соответствующих сообщений.

Окно отладчика x64Dbg с отслеживаемой программой

представлено на рис. 5.1.

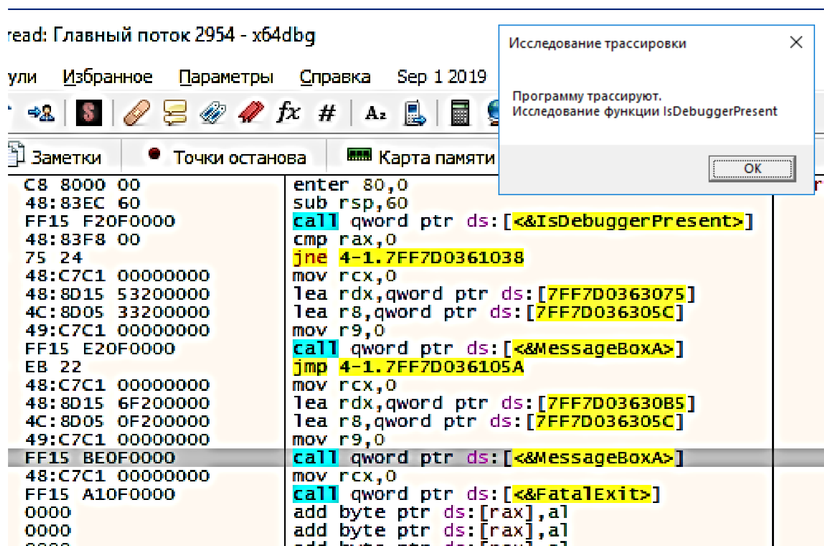


Рис. 5.1. Результат трассирования программы

5.2. Использование функции CheckRemoteDebuggerPresent

Функция CheckRemoteDebuggerPresent, как и функция IsDebuggerPresent, кросс-платформенная и проверяет наличие отладчика. Ее отличие от IsDebuggerPresent в том, что она умеет проверять не только свой процесс, но и другие по их хендлу.

Синтаксис:

```
BOOL CheckRemoteDebuggerPresent(
  HANDLE hProcess,
  PBOOL pbDebuggerPresent
);
```

Параметры:

– hProcess – [in], дескриптор процесса.

– pbDebuggerPresent – [in, out], указатель на переменную, которую функция устанавливает в значение ИСТИНА (TRUE), если

указанный процесс отлаживался или ЛОЖЬ (FALSE) в противном случае.

Если функция завершается успешно, возвращаемое значение не нуль.

Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке, надо вызвать GetLastError.

Вариант применения этой функции приведен в программе 5.2.

Программа 5.2. Применение функции CheckRemoteDebuggerPresent:

```
    include \masm64\include64\masm64rt.inc ;
.data
   tit1 db "Исследование функции CheckRemoteDebuggerPresent",0
   szTest db 'Отладчик не обнаружен',10,
   'Автор: Рысованый А.Н,, каф. ВТП, НТУ ХПИ',0
   szTest2 db 'Отладчик обнаружен',10,
   'Исследование функции CheckRemoteDebuggerPresent',0
   dwResult dq  ?
.code
entry_point proc

invoke GetCurrentProcessId ; извлекает идентификатор вызывающего
процесса
invoke OpenProcess,PROCESS_ALL_ACCESS,0,rax
invoke CheckRemoteDebuggerPresent,rax,addr dwResult
        cmp qword ptr dwResult,0
        jz @1
        jmp exit2
@1:
invoke MessageBox,0,addr szTest,addr tit1,
MB_ICONINFORMATION+180000h
       jmp exit1
 exit2:
    invoke MessageBox,0,addr szTest2,addr tit1,MB_ICONWARNING
exit1:
    invoke ExitProcess,0
    entry_point endp
    end
```

Функция OpenProcess открывает существующий объект локального процесса по идентификатору, полученному функцией GetCurrentProcessId.

При выполнении функции CheckRemoteDebuggerPresent под отладчиком в rax заносится 1 и это же значение переписывается в

ячейку с именем **dwResult**.

Командой

```
cmp qword ptr dwResult,0
```

устанавливаются признаки сравнения, на основании признака ZF и происходит ветвление для вывода сообщения.

Результаты выполнения программы для двух ветвей анализа признака равенства ZF приведены на рис. 5.2.

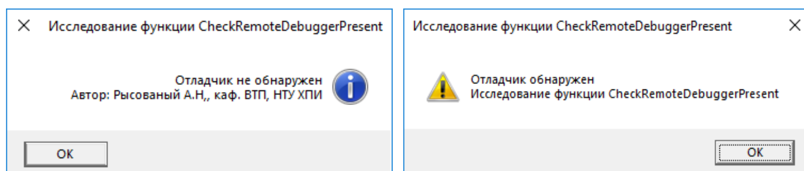


Рис. 5.2. Результат выполнения программы

5.3. Функция ContinueDebugEvent

Функция ContinueDebugEvent дает возможность отладчику оставить поток, который предварительно сообщил о событии отладки.

Синтаксис:

```
BOOL ContinueDebugEvent(
  DWORD dwProcessId,
  DWORD dwThreadId,
  DWORD dwContinueStatus
);
```

Параметры:

– dwProcessId – [in], дескриптор остающегося процесса.

– dwThreadId – [in], дескриптор остающегося потока. Комбинация идентификатора процесса и идентификатора потока должна идентифицировать поток, который предварительно сообщил о событии отладки.

– dwContinueStatus – [in], опции остающегося потока, который сообщил о событии отладки.

Если для этого параметра устанавливается флажок DBG_CONTINUE, а поток, указанный параметром dwThreadId предварительно сообщил об отлаживающем событии EXCEPTION_DEBUG_EVENT, то функция останавливает всякую

обработку исключений и оставляет поток. Для любого другого события отладки, этот флажок просто оставляет поток.

Если для этого параметра устанавливается флажок DBG_EXCEPTION_NOT_HANDLED, а поток, заданный в dwThreadId предварительно сообщил об отлаживающем событии EXCEPTION_DEBUG_EVENT, то функция продолжает обработку исключений. Если это событие первой случайной исключительной ситуации , используется логика поиска и распределения структурного обработчика исключительных ситуаций; в противном случае, процесс завершается. Для любого другого события отладки, этот флажок просто оставляет поток.

Если функция завершается успешно, возвращаемое значение не нуль.

Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке необходимо вызывать GetLastError.

5.4. Функция DebugActiveProcess

Функция DebugActiveProcess дает возможность отладчику подключиться к активному процессу и отлаживать его.

Синтаксис:

```
BOOL DebugActiveProcess(
  DWORD dwProcessId
);
```

Параметр:

dwProcessId – [in], идентификатор процесса, который будет отлаживаться. Отладчику предоставляется доступ к отлаживаемому процессу, как если бы он создавал процесс с флажком DEBUG_ONLY_THIS_PROCESS.

Если функция завершается успешно, возвращаемое значение не нуль.

Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке необходимо вызывать GetLastError.

Чтобы остановить отладку процесса необходимо выйти из него или вызвать функцию DebugActiveProcessStop. Выход из отладчика также порождает выход из процесса, если не используется функция DebugSetProcessKillOnExit.

Отладчик должен иметь соответствующий доступ к целевому процессу; он должен быть в состоянии открыть процесс для доступа PROCESS_ALL_ACCESS.

Если процессу отладки предоставили привилегию SE_DEBUG_NAME и включили в работу, он может отладить любой процесс.

5.5. Функция DebugActiveProcessStop

Останавливает отладчик для отладки указанного процесса.

Синтаксис:

```
BOOL DebugActiveProcessStop( DWORD dwProcessId );
```

Параметр dwProcessId – идентификатор процесса, чтобы остановить отладку.

Если функция завершается успешно, возвращаемое значение отлично от нуля.

Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке необходимо вызывать GetLastError.

5.6. Функция DebugBreak

Функция DebugBreak заставляет в контрольной точке текущего процесса произойти исключительной ситуации. Это дает возможность вызывающему потоку подать сигнал отладчику, чтобы он обработал исключительную ситуацию.

Чтобы вызывать исключительную ситуацию в контрольной точке в другом процессе, используют функцию DebugBreakProcess.

Синтаксис:

```
void DebugBreak(void);
```

Параметров нет.

Функция не возвращает значений.

Если процесс не отлаживается, функция использует логику поиска стандартного обработчика исключительных ситуаций. В большинстве случаев, это заставляет вызывающий процесс завершать работу из-за необработанного исключения контрольной точки.

5.7. Функция DebugBreakProcess

Функция DebugBreakProcess заставляет в контрольной точке указанного процесса произойти исключительной ситуации. Это дает возможность вызывающему потоку подать сигнал отладчику на обработку исключительной ситуации.

Синтаксис:

```
BOOL DebugBreakProcess(
  HANDLE Process
);
```

Параметр: Process – [in], дескриптор процесса.

Если функция завершается успешно, возвращаемое значение отлично от нуля.

Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке необходимо вызывать GetLastError.

5.8. Функция DebugSetProcessKillOnExit

Функция DebugSetProcessKillOnExit определяет действие, которое будет выполняться, когда отлаживаемый поток завершает работу.

Синтаксис:

```
BOOL DebugSetProcessKillOnExit(
  BOOL KillOnExit
);
```

Параметры: KillOnExit – [in], если этот параметр - ИСТИНА (TRUE), поток отладки убьет отлаживаемый процесс при завершении работы. В противном случае, отлаживаемый поток отделится от отлаживаемого процесса при завершении работы.

Если функция завершается успешно, возвращаемое значение отлично от нуля.

Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке необходимо вызывать GetLastError.

5.9. Функция FatalExit

Функция FatalExit передает управление исполнением кода

отладчику. Режим работы отладчика после этого является специфичным для типа используемого отладчика.

Синтаксис:

```
void FatalExit(
  int ExitCode
);
```

Параметры: ExitCode – [in], код ошибки связанный с завершением выполнения.

Эта функция не возвращает значение.

5.10. Функция FlushInstructionCache

Функция FlushInstructionCache освобождает кэш команд для заданного процесса.

Синтаксис:

```
BOOL FlushInstructionCache(
  HANDLE hProcess,
  LPCVOID lpBaseAddress,
  SIZE_T dwSize
);
```

Параметры:

– hProcess – [in], дескриптор процесса, кэш команд которого должен освободиться.

– lpBaseAddress – [in], указатель на базу освобождаемой зоны. Этот параметр может быть ПУСТО (NULL).

– dwSize – [in], размер освобождаемой зоны, если параметр lpBaseAddress - не ПУСТО (NULL), в байтах

Если функция завершается успешно, возвращаемое значение отлично от нуля.

Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке необходимо вызывать GetLastError.

5.11. Функция GetThreadContext

Функция GetThreadContext извлекает данные о контексте заданного потока.

Синтаксис:

```
BOOL GetThreadContext(
```

```
  HANDLE hThread,
  LPCONTEXT lpContext
);
```

Параметры:

– hThread – [in], дескриптор потока, контекст которого должен извлечься. Дескриптор должен иметь доступ к потоку THREAD_QUERY_INFORMATION.

– lpContext – [in, out], указатель на структуру CONTEXT, которая получает соответствующий контекст заданного потока. Значение члена структуры ContextFlags этой структуры определяет, какие части контекста потока извлекаются. Для процессоров Intel существует своя структура CONTEXT.

Функция GetThreadContext используется, чтобы извлечь данные о контексте заданного потока. Функция дает возможность отбирать контекст, который будет извлекаться, основываясь на значении члена структуры ContextFlags структуры CONTEXT. Дескриптор потока, идентифицированный параметром hThread обычно отлаживаемый, но функция может также работать и тогда, когда отладки он не делает.

Нельзя получить допустимый контекст для запущенного потока. Для этого используют функцию SuspendThread, чтобы приостановить поток перед вызовом GetThreadContext.

Если функция завершается успешно, возвращаемое значение отлично от нуля.

Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке необходимо вызывать GetLastError.

5.12. Функция GetThreadSelectorEntry

Функция GetThreadSelectorEntry извлекает данные о записи таблицы дескрипторов для заданного селектора и потока.

Синтаксис:

```
BOOL GetThreadSelectorEntry(
  HANDLE hThread,
  DWORD dwSelector,
  LPLDT_ENTRY lpSelectorEntry
);
```

Параметры:

– hThread – [in], дескриптор потока, содержащего заданный селектор. Дескриптор должен иметь доступ THREAD_QUERY_INFORMATION. Дополнительную информацию, см. в статье Защита потока и права доступа.

– dwSelector – [in], значение глобального или локального селектора, которое ищется в дескрипторных таблицах потока.

– lpSelectorEntry – [out], указатель на структуру LDT_ENTRY, которая получает копию записи дескрипторной таблицы, если указанный селектор имеет запись в дескрипторной таблице указанного потока. Эта информация, может быть использована, чтобы преобразовывать адрес относительно начала сегмента в линейный виртуальный адрес.

Если функция завершается успешно, возвращаемое значение не нуль. В этом случае, структура, на которую указывает параметр lpSelectorEntry получает копию указанной записи дескрипторной таблицы.

Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке необходимо вызывать GetLastError.

5.13. Функция OutputDebugString

Функция OutputDebugString отправляет символьную строку отладчику программы для вывода на экран.

Синтаксис:

```
void OutputDebugString(
  LPCTSTR lpOutputString
);
```

Параметр: lpOutputString – [in], указатель на символьную строку с нулем в конце, которая появится на экране.

Эта функция не возвращает значение.

Если у приложения нет отладчика, системный отладчик показывает на экране символьную строку. Если приложение не имеет отладчика, а системный отладчик не активен, функция OutputDebugString ничего не делает.

5.14. Функция ReadProcessMemory

Функция ReadProcessMemory читает данные из области памяти

в заданном процессе. Вся область, которая читается, должна быть доступна, а иначе операция завершается ошибкой.

Синтаксис:

```
BOOL ReadProcessMemory(
  HANDLE hProcess,
  LPCVOID lpBaseAddress,
  LPVOID lpBuffer,
  SIZE_T nSize,
  SIZE_T* lpNumberOfBytesRead
);
```

Параметры:

– hProcess – [in], дескриптор процесса, память которого читается. Дескриптор должен иметь доступ к процессу PROCESS_VM_READ.

– lpBaseAddress – [in], указатель на базовый адрес в заданном процессе из которого делается чтение. Прежде, чем происходит какая-либо передача данных, система проверяет, все ли данные в базовом адресе и памяти в заданном размере доступны для чтения. Если дело обстоит так, функции приступает к работе; в противном случае функция завершается ошибкой.

– lpBuffer – [out], уУказатель на буфер, который получает содержание из адресного пространства заданного процесса.

– nSize – [in], число байтов, которое читается из заданного процесса.

– lpNumberOfBytesRead – [out], указатель на переменную, которая получает число байтов, переданное в указанный буфер. Если lpNumberOfBytesRead - ПУСТО (NULL), этот параметр игнорируется.

Если функция завершается успешно, возвращаемое значение отлично от нуля.

Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке необходимо вызывать GetLastError.

5.15. Функция SetThreadContext

Функция SetThreadContext устанавливает контекст для заданного потока.

Синтаксис:

```
BOOL SetThreadContext(
  HANDLE hThread,
  const CONTEXT* lpContext
```

);

Параметры:

– hThread – [in], дескриптор потока, контекст которого должен установиться. Дескриптор должен иметь право доступа к потоку THREAD_SET_CONTEXT. Дополнительную информацию, см. в статье Защита потока и права доступа.

– lpContext –[in], указатель на структуру CONTEXT, содержащую контекст, который устанавливается в заданном потоке. Значение члена структуры ContextFlags этой структуры определяет, какие части контекста потока устанавливаются. Некоторые значения в структуре CONTEXT, которые не могут быть определены, устанавливаются без объявления в правильное значение. Они включают в себя биты состояния регистров центрального процессора, которые устанавливают привилегированный процессорный режим, глобальную переменную, включающую в работу биты в регистре отладки программы и другие состояния, которые должны управляться операционной системой.

Если функция завершается успешно, возвращаемое значение отлично от нуля.

Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке необходимо вызывать GetLastError.

5.16. Функция WaitForDebugEvent

Функция WaitForDebugEvent ожидает событие отладки, которое произойдет в отлаживаемом процессе.

Синтаксис:

```
BOOL WaitForDebugEvent(
  LPDEBUG_EVENT lpDebugEvent,
  DWORD dwMilliseconds
);
```

Параметры:

– lpDebugEvent – [out], указатель на структуру DEBUG_EVENT, которая получает информацию о событии отладки.

– dwMilliseconds – [in], число миллисекунды, в течение которых ожидается событие отладки. Если этот параметр равняется нулю, функция проверяет событие отладки и возвращает значение немедленно. Если параметр INFINITE (БЕСКОНЕЧНО), функция не возвращает значение до тех пор, пока не совершится событие отладки.

Если функция завершается успешно, возвращаемое значение отлично от нуля.

Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке необходимо вызывать GetLastError.

5.17. Функция Wow64GetThreadContext

Получает контекст указанного потока WOW64.

Синтаксис:

BOOL Wow64GetThreadContext(HANDLE hThread, PWOW64_CONTEXT lpContext);

Параметры:

– hThread – дескриптор потока, контекст которого должен быть получен. Дескриптор должен иметь доступ THREAD_GET_CONTEXT к потоку.

– lpContext – структура WOW64_CONTEXT . Вызывающая сторона должна инициализировать член ContextFlags этой структуры.

Если функция завершается успешно, возвращаемое значение отлично от нуля.

Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке необходимо вызывать GetLastError.

5.18. Функция Wow64GetThreadSelectorEntry

Извлекает запись таблицы дескрипторов для указанного селектора и потока WOW64.

Синтаксис:

BOOL Wow64GetThreadSelectorEntry(HANDLE hThread, DWORD dwSelector, PWOW64_LDT_ENTRY lpSelectorEntry);

Параметры:

– hThread –дескриптор потока, содержащего указанный селектор. Дескриптор должен быть создан с доступом THREAD_QUERY_INFORMATION к потоку.

– dwSelector – значение глобального или локального селектора для поиска в таблицах дескрипторов потока.

– lpSelectorEntry – указатель на структуру WOW64_LDT_ENTRY, которая получает копию записи таблицы дескрипторов, если указанный

селектор имеет запись в таблице дескрипторов указанного потока. Эта информация может использоваться для преобразования относительного сегмента адреса в линейный виртуальный адрес.

Если функция завершается успешно, возвращаемое значение отлично от нуля. В этом случае структура, на которую указывает параметр lpSelectorEntry, получает копию указанной записи таблицы дескрипторов.

Если функция завершается ошибкой, возвращаемое значение равно нулю. Для получения расширенной информации об ошибке необходимо вызвать функцию GetLastError .

5.19. Функция Wow64SetThreadContext

Устанавливает контекст указанного потока WOW64.

Синтаксис:

BOOL Wow64SetThreadContext(HANDLE hThread, const WOW64_CONTEXT *lpContext);

Параметры:

– hThread – дескриптор потока, контекст которого должен быть установлен.

– lpContext – структура WOW64_CONTEXT . Вызывающая сторона должна инициализировать член ContextFlags этой структуры.

Если функция завершается успешно, возвращаемое значение отлично от нуля.

Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке необходимо вызывать GetLastError.

5.20. Лабораторная работа «Антиотладка с использованием функций обнаружения отладчика»

Цель занятия: приобрести практические навыки обнаружения отладчика специально предназначенными функциями в среде masm64.

В отчете представить:

– номер, название лабораторной работы и фамилию исполнителя;

– задание;

– текст исходной программы с комментариями к каждой строке программы;

– результат выполнения программы (скриншоты упрощенного окна (MessageBoxIndirect) и отладчика);

– особенности выполнения.

Постановка задачи

В работе можно использовать только одну из функций (IsDebuggerPresent или CheckRemoteDebuggerPresent). Вторую функцию выбрать самостоятельно из перечня функций, перечисленные в разделе 4 или из справочника (https://docs.microsoft.com).

Задание

Создать диалоговое окно с ресурсами:

– иконкой (иконками) на диалоговом окне;

– кнопками;

– текстом на окне;

– статическими рамками и др.

Использовать функцию MessageBoxIndirect.

Вывести фамилию, группу и результаты.

В соответствии с номером студента в группе выбрать вариант задания и написать в среде masm64 на ассемблере программу вычисления одного из выражений:

1. Найти количество и сумму элементов массива, принадлежащих интервалу.
2. Удалить повторяющиеся элементы из массива.
3. Найти два максимальных элемента массива.
4. Удалить из массива четные элементы.
5. Найти количество отрицательных и положительных элементов в массиве.
6. Найти количество элементов в массиве, отличающихся от минимального на 5.
7. Найти максимальный элемент в массиве.
8. Вычислить сумму элементов массива.
9. Найти первый положительный элемент массива.
10. Поменять местами минимальный и максимальный элементы массива.
11. Найти разность между максимальным и минимальным элементами массива.

12. Найти сумму четных отрицательных элементов массива.

13. Найти минимальный из элементов массива с нечетными индексами.

14. Вывести элементы массива, которые больше среднего арифметического.

15. Найти сумму положительных элементов массива.

16. Найти количество положительных элементов массива.

17. Найти разность между максимальным и минимальным элементами массива.

18. Удалить повторяющиеся элементы из массива.

19. Найти сумму и произведение элементов одномерного массива.

20. Разделить элементы массива на максимальный.

21. Найти первый отрицательный элемент массива.

22. Заменить элементы массива на противоположные.

23. Найти три минимальных элемента массива.

24. Найти минимальный элемент в массиве.

25. Найти сумму четных положительных элементов массива.

Методические указания

Для создания ресурсов (например, вставка иконки, JPG-файла и прочее) будем использовать программу ResEd. При создании проекта будут использованы файлы:

1. Prog_user.asm – исходный код программы пользователя;
2. Prog_user.rc – файл ресурсов пользователя;
3. Prog_user.ico – иконка пользователя;
4. makeit.bat – для использования **rc**.exe.

Программа 5.3. Файл makeit.bat:

```
:: вывод выполняющихся строк на экран
@echo off
:: установка переменной
set appname=%1

\masm64\bin64\ml64.exe /c %appname%.asm
\masm64\bin64\rc.exe %appname%.rc
\masm64\bin64\link.exe /SUBSYSTEM:WINDOWS /ENTRY:entry_point
/nologo /LARGEADDRESSAWARE %appname%.obj %appname%.res
:: вывод списка файлов
dir %appname%.*
:: удаление файла *.obj
del %appname%.obj
```

```
del %appname%.res
pause
```

Рассмотрим последовательность создания **диалогового окна** с предъявленными требованиями:

1. В программе ResEd создаем новый файл ресурсов (File – New Project и вводим имя). В правой верхней панели появляется значок папки и имя файла (рис. 5.3).

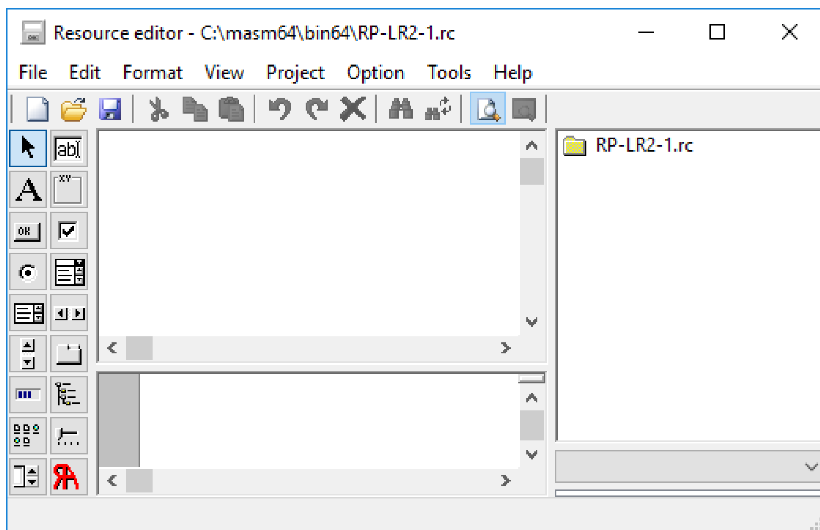
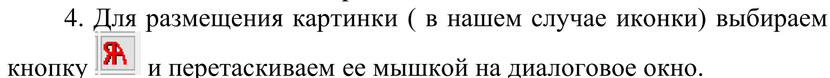


Рис. 5.3. Создание окна в программе ResEd

2. Нажимаем на нее правой кнопкой мыши и выбираем «Add Dialog».

3. Выставляем характеристики окна:

- выставляем мышкой необходимый размер окна;
- в поле свойств окна Caption меняем название диалогового окна.

4. Для размещения картинки (в нашем случае иконки) выбираем кнопку  и перетаскиваем ее мышкой на диалоговое окно.

5. Для размещения кнопки выбираем в поле инструментов кнопку с изображением ОК и перетаскиваем ее мышкой на выбранное место. В поле Caption вводим название кнопки.

6. Для написания текста на окне выбираем в поле инструментов кнопку с изображением буквы «A» и перетаскиваем ее мышкой на выбранное место. В поле Caption вводим текст.

7. Для создания рамки (если необходимо) выбираем поле с изображением рамки. В поле Caption вводим название рамки.

8. Добавляем файл ресурсов.

Добавление файла констант. Для этого, нажимаем правой кнопкой мыши по значку папки, выбираем «Include file», затем «Add». Выбираем в окне программы три точки и вводим путь, например C:\masm64\include64\RESOURCE.H.

9. Сохраняем программу.

10. Открываем созданный rc-файл и добавляем строку с ресурсом курсора:

10 ICON DISCARDABLE "smail5.ico”

В этой строке число **10** обозначает цифровой идентификатор иконки (ресурс). Он может быть любым, но и в rc- и asm-файлах он должен быть одним и тем же.

В результате выполненных действий окно программы должно быть таким, как приведено на рис. 5.4.

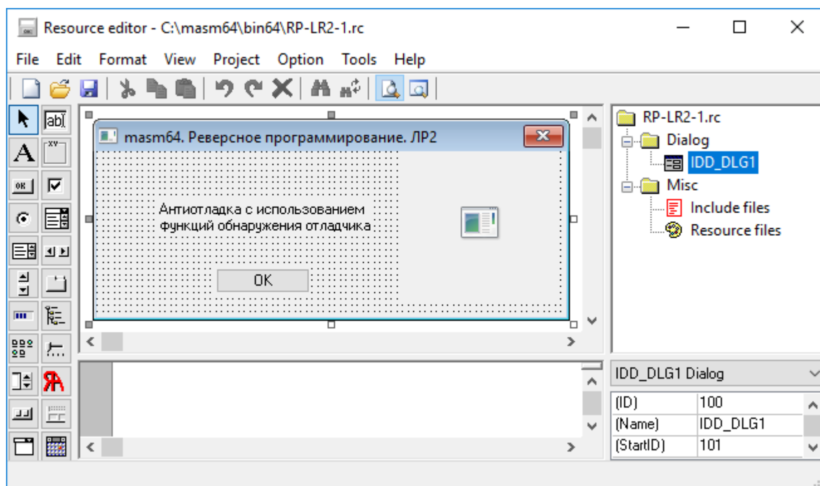


Рис. 5.4. Результат создания окна в программе ResEd

Программа 5.4. Файл ресурсов rc:

```
#define IDD_DLG1 100
#define IDC_BTN1 101
#define IDC_IMG1 102
#define IDC_STC1 103

#include "/masm64/include64/resource.h"

10 ICON DISCARDABLE "smail5.ico"
11 ICON DISCARDABLE "smail4.ico"

IDD_DLG1 DIALOGEX 10,10,249,81
CAPTION "masm64. Реверсное программирование. ЛР2"
FONT 8,"MS Sans Serif",0,0,0
STYLE
WS_POPUP|WS_VISIBLE|WS_CAPTION|WS_SYSMENU|DS_CENTER
BEGIN
CONTROL "OK",IDC_BTN1,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,63,57,50,12
CONTROL "",IDC_IMG1,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTERIMAGE|SS_ICON,162,3,
84,75
CONTROL "Антиотладка с использованием функций обнаружения
отладчика",IDC_STC1,"Static",WS_CHILDWINDOW|WS_VISIBLE,33,24,11
4,18
END
```

Программа 5.5. Файл asm:

```
include \masm64\include64\masm64rt.inc
.data?
   hInstance dq ?
   hIcon     dq ?
.code
entry_point proc
                    ; вставка кода антиотладки
.data
titl1 db "Исследование трассировки",0
szTest db "Программу НЕ трассируют.",10,
"Исследование функции IsDebuggerPresent",0
szTest2 db "Программу трассируют.",10,
"Исследование функции IsDebuggerPresent",0
.code
invoke IsDebuggerPresent ;
      cmp rax,0 ;
        jnz debug1 ;
  invoke MessageBox,0,addr szTest,addr titl1,MB_OK
```

```
        jmp @f
debug1:
invoke MessageBox,0,addr szTest2,addr titl1,MB_OK
        invoke ExitProcess,0
@@:
                                ; основная часть программы
         mov hInstance,rv(GetModuleHandle,0)
 mov hIcon, rv(LoadImage,hInstance,10,IMAGE_ICON,128,128,
LR_DEFAULTCOLOR)
        invoke DialogBoxParam,hInstance,100,0,ADDR main,hIcon
        invoke ExitProcess,0
            ret
        entry_point endp

main proc hWin:QWORD,uMsg:QWORD,wParam:QWORD,lParam:QWORD
.switch uMsg
.case WM_INITDIALOG
   ; установить значок для диалога
invoke SendMessage,hWin,WM_SETICON,1,lParam
   ; установить значок в клиентской области
invoke SendMessage,rv(GetDlgItem,hWin,102),\
                      STM_SETIMAGE,IMAGE_ICON,lParam
invoke SetWindowText,hWin,"masm64. Лабораторная 2. Антиотладка"
        .case WM_COMMAND
         .switch wParam
             .case 101
invoke MsgboxI,0,"Вывод информации","Вывод информации",MB_OK,11
         .endsw
.case WM_CLOSE
         invoke EndDialog,hWin,0 ; exit from system menu
            .endsw
            xor rax, rax
   ret
main endp
        end
```

В начале основной программы полностью вставлена программа, которая отвечает за антиотладку с применением функции IsDebuggerPresent и функции окончания программы ExitProcess. В результате такого приема в основной части программы не внесено никаких изменений.

Внесение в программу других частей с применением других функций антиотладки можно осуществить в любое место. Но, естественно, лучше это выполнить до вывода диалогового окна.

Результат выполнения программы приведен на рис. 5.5.

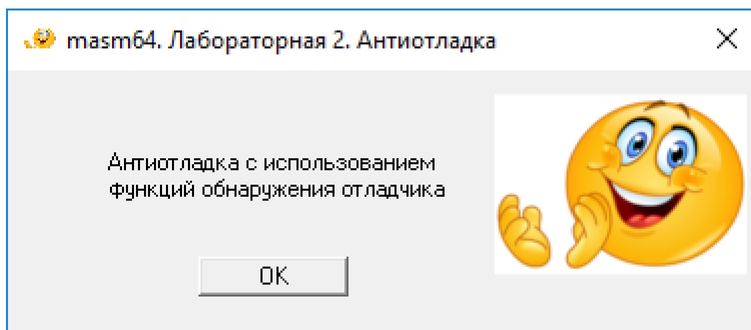


Рис. 5.5. Результат выполнения программы

6. ОТЛАДОЧНЫЕ РЕГИСТРЫ

Признаком выполнения программы под отладчиком являются данные в отладочных регистрах. Но отладочные регистры – это **привилегированный** ресурс и получить к ним доступ напрямую можно только в режиме ядра. Но к ним можно получить доступ и при помощи функции GetThreadContext и прочитать эти данные.

Всего отладочных регистров восемь – DR0-DR7. Первые четыре регистра DR0–DR3 содержат информацию о точках останова, регистры DR4–DR5 – зарезервированные, регистр DR6 заполняется, когда сработал брейк-пойнт (точка останова) отладчика, и содержит информацию об этом событии. Регистр DR7 содержит биты управления отладкой. Можно проверить информацию о точках останова в регистрах DR0–DR3.

Регистры отладки являются привилегированным ресурсом и доступны программе либо в режиме реальной адресации, либо в защищённом режиме с наивысшим уровнем привилегий (CPL = 0).

Когда же бит DE в CR4 установлен, попытки выполнить команду MOV, использующую DR4 или DR5, вызывают генерацию особой ситуации #UD.

Команды доступа к отладочным регистрам должны выполняться при уровне привилегий 0 или в реальном режиме, иначе будет сгенерирована особая ситуация #GP. Существует еще и сверхзащита отладчика, которая может быть включена при программировании отладочных регистров (бит DR7.GD).

Чтобы создать процесс для отладки, необходимо вызвать функцию CreateProcess с флагом DEBUG_PROCESS. Windows будет посылать уведомления о важных отладочных событиях, которые происходят в отлаживаемом процессе. Он будет немедленно заморожен, пока программа не выполнит то, что должна. Отключить эту возможность можно заменив флаг DEBUG_PROCESS на DEBUG_ONLY_THIS_PROCESS.

Подсоединиться к уже выполняющемуся процессу возможно с помощью функции DebugActiveProcess. Когда процесс создан или присоединён, он замораживается, пока программа не вызовет WaitForDebugEvent. Функция блокирует вызывающий поток, пока не произойдет ожидаемое событие или не истечёт заданный временной интервал.

Для продолжения выполнения отлаживаемого процесса служит функция ContinueDebugEvent. Она продолжает выполнение потока, который был остановлен произошедшим событием. Подобный цикл должен повторяться пока отлаживаемый процесс не завершится. Как только начнётся отладка программы, отсоединиться от процесса будет невозможно до его завершения. Так как в одно и тоже время может быть запущено несколько программ, Windows даёт каждому потоку небольшой временной интервал, по истечению которого поток замораживается и запускается следующий, согласно приоритету. Но перед тем как запустится другой поток, Windows сохраняет значения регистров текущего. Когда случается отладочное событие, Windows замораживает отлаживаемый процесс, сохраняя значения регистров. Получить эти значения возможно с помощью функции GetThreadContext, а изменить их функцией SetThreadContext

7. ФЛАГИ ОТЛАДКИ, ПАМЯТЬ ПРИЛОЖЕНИЙ

Отладчик имеет имя процесса, открытые окна, загруженные библиотеки и драйверы. Все это можно выявить, задействовав функции операционной системой.

7.1. Поиск отладчика по названию класса

Функция FindWindow извлекает данные о дескрипторе окна верхнего уровня, чье имя класса и имя окна соответствуют определенным строкам. Эта функция не ищет дочерние окна.

Синтаксис:

```
HWND FindWindow
(
LPCTSTR lpClassName,    // указатель на имя класса
LPCTSTR lpWindowName   // указатель на имя окна
);
```

Параметры:

– lpClassName – указывает на строку с нулевым символом в конце, которая определяет имя класса или – атом, который идентифицирует строку имени класса. Если этот параметр – атом, он должен быть общим атомом, созданным предыдущим вызовом функции GlobalAddAtom.

– lpWindowName – указывает на строку с нулевым символом в конце, которая определяет имя окна (заголовок окна). Если этот параметр – ПУСТО (NULL), то это – полное соответствие имени окна.

Возвращаемые значения:

– если функция завершилась успешно, то возвращаемое значение – дескриптор окна, которое имеет определенное имя класса и имя окна. Если функция терпит неудачу, возвращаемое значение – ПУСТО (NULL). Чтобы получить дополнительные данные об ошибках, необходимо вызывать функцию GetLastError;

– если в программе указать имя другой программы, которую необходимо обнаружить, то в случае, если разыскиваемая программа будет найдена – выведется соответствующее сообщение (программа 7.1).

Программа 7.1. Применение функции FindWindow:

```
	include \masm64\include64\masm64rt.inc ;
.data
	ClassName1 db 'x64dbg',0
	titl1 db "Поиск программы по имени",0
szTest db "Отладчик x64Dbg НЕ обнаружен",0
szTest2 db "Отладчик x64Dbg ОБНАРУЖЕН",0

.code
entry_point proc
invoke FindWindow,0,addr ClassName1,0 ;
	cmp eax,0 ;
	  jnz m1 ; если /=0 - отладчик обнаружен
invoke MessageBox,0,addr szTest, addr titl1,MB_OK
	jmp @f
m1:
invoke MessageBox,0,addr szTest2, addr titl1,MB_OK
@@:
	invoke ExitProcess,0
entry_point endp
	end
```

Результаты выполнения программы без выполнения процесса отладки в отладчике x64dbg приведены на рис. 7.1.

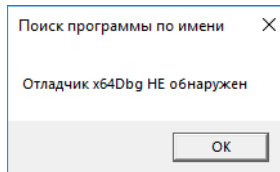


Рис. 7.1. Результат выполнения программы

В программе ищется отладчик по имени строки:
ClassName1 db '**x64dbg**',0

В случае, если запустить программу 7.1 в отладчике, имя которого приведено в программе (ClassName1 db 'x64dbg',0), то этот отладчик будет обнаружен. Таким образом можно последовательно перечислить все известные отладчики.

Рассмотрим пример перемещения массивов через FPU и SSE с

выводом числа тактов на эти операции в разных потоках (программа 7.2).

Программа 7.2:

```
      include \masm64\include64\masm64rt.inc ;
.data ;
      mas1 DD 20 dup(0,2,4,6) ;
          dd 16 dup(1)      ; 80+16=96
      len1 equ ($-mas1)/type mas1
      mas2 DD len1 dup(0) ; резервирование ячеек памяти для mas2
      buf1 dq 0,0   ; буфер
      tFpu dq 0
      tSse dq 0
  ifmt db "Число тиков FPU =  %d ",10,10,"Число тиков SSE =  %d ",10,
  "Автор программы:  Рысованый А.Н., каф. ВТП, НТУ ХПИ",0
      h1 dq  ? ; идентификатор потока
      h2 dq  ?  ; идентификатор потока
      hEventStart HANDLE ?   ; хэндл события

.code
proc1 proc ;  процедура
       rdtsc
      xchg r14,rax
      finit
      mov rcx,len1   ; количество чисел массива mas1
       lea rsi,mas1  ; адрес начала массива mas1
       lea rdi,mas2  ; адрес начала массива mas2
@@: fild dword ptr [rsi] ; загрузка целого числа
      fistp dword ptr [rdi]
      add rsi,type mas1;
      add rdi,type mas1;
    loop @b
      rdtsc
      sub rax,r14
      mov tFpu,rax
      ret
proc1 endp ;

proc2 proc  ; процедура
      rdtsc
      xchg r14,rax
       mov rcx,len1   ; количество чисел массива mas1
       lea rsi,mas1   ; адрес начала массива mas1
       lea rdi,mas2   ; адрес начала массива mas2
@@: movups XMM0,mas1 ; пересылка 4-х целых чисел
```

```
        movups mas2,xmm0
        add rsi,type mas1;
        add rdi,type mas1;
     loop @b
        rdtsc
        sub rax,r14
        mov tSse,rax
        ret
proc2 endp  ;

entry_point proc
.data
        ClassName1 db 'x64dbg',0
        titl1 db "Исследование трассировки",0
        szTest db "Программу НЕ трассируют",0
        szTest2 db "Программу трассируют.",
        "Поэтому, вычисления не производятся.",0
.code
        invoke FindWindow,0,addr ClassName1,0 ;
        cmp eax,0 ;
          jnz debugging ; если происходит отладка
          invoke MessageBox,0,addr szTest,0,MB_ICONINFORMATION
     jmp @f
        debugging:
        invoke MessageBox,0,addr szTest2,0,MB_ICONERROR
        jmp exit1
@@:

        lea rax, proc1   ; загрузка адреса процедуры
invoke CreateThread,0,0,rax,0,0,addr h1 ; создать процесс
        ;invoke proc1
        lea rax, proc2 ; загрузка адреса процедуры
invoke CreateThread,0,0,rax,0,0,addr h2; создать процесс
        ;invoke proc2
invoke CreateEvent,0,FALSE,FALSE,0; создание события
        mov hEventStart,rax    ; сохранение хендла события
invoke WaitForSingleObject,hEventStart,300
        invoke wsprintf,addr buf1,ADDR ifmt,tFpu,tSse;
        invoke MessageBox,0,addr buf1,ADDR titl1,MB_ICONINFORMATION
exit1: invoke ExitProcess,0
        entry_point endp
        end
```

Выполнение перемещения массивов выполнено в двух потоках: для перемещения при помощи команд FPU и команд SSE.

Для случая, когда программа в отладчике не трассируется, выводятся последовательно два сообщения (рис. 7.2 и рис. 7.3).

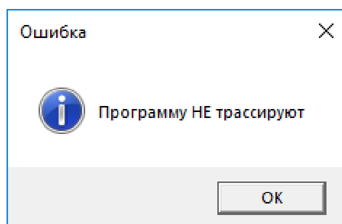


Рис. 7.2. Первое сообщение программы

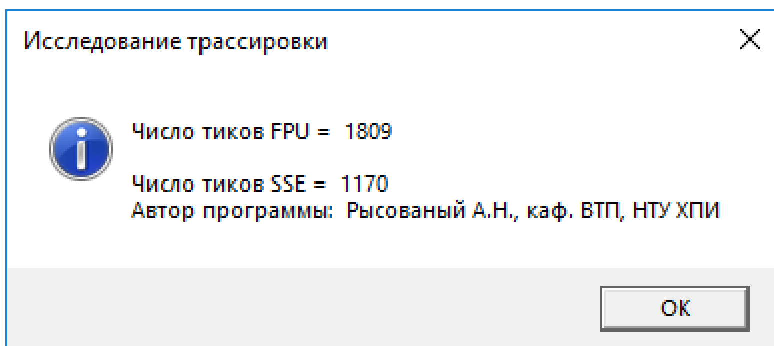


Рис. 7.3. Второе сообщение программы

7.2. Глобальная переменная NtGlobalFlag

В Windows NT существует набор флагов, которые хранятся в глобальной переменной NtGlobalFlag в структуре PEB и является общей для всей системы.

При загрузке глобальная системная переменная NtGlobalFlag инициализируется значением из ключа системного реестра:

[HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Control \ Session Manager \ GlobalFlag]

Это значение переменной используется для отслеживания системы, ее отладки и управления. Переменные флаги недокументированы, но в SDK включена утилита gflags, которая позволяет редактировать значение глобального флага:

FLG_HEAP_ENABLE_TAIL_CHECK (0x10)
FLG_HEAP_ENABLE_FREE_CHECK (0x20)
FLG_HEAP_VALIDATE_PARAMETERS (0x40)

Чтобы проверить, был ли запущен процесс с отладчиком, необходимо проверить значение поля NtGlobalFlag в структуре PEB. Это поле расположено по смещениям 0x068 и 0x0bc для систем x32 и x64 соответственно, относительно начала структуры PEB. PEB – закрытая структура, используемая внутри операционной системы Windows.

7.3. Лабораторная работа «Антиотладка с использованием названия отладчика»

Цель занятия: приобрести практические навыки составления, отладки и выполнения программ с использованием антиотладочных приемов в среде masm64.

В отчете представить:

– номер, название лабораторной работы и фамилию исполнителя;

– задание;

– текст исходной программы с комментариями к каждой строке программы;

– результат выполнения программы (скриншоты упрощенного окна (MessageBoxIndirect) и отладчика);

– особенности выполнения, где описать общий алгоритм исправления вашей программы: какие клавиши в отладчике использовались для получения исправленной программы.

Постановка задачи

Написать программу с применением разных процессов, функции FindWindow, функции мультимедийного таймера и команд AVX для совмещенного выполнения операций умножения и сложения.

Задание

Создать Windows окно с ресурсами:

– меню;
– иконкой (иконками или jmp-файлом) на диалоговом окне.
Использовать функцию MessageBoxIndirect.
Вывести фамилию, группу и результаты.

В соответствии с номером студента в группе выбрать вариант задания и написать в среде masm64 на ассемблере программу вычисления одного из выражений:

1. Умножить матрицу [2 x 2] на матрицу [2 x 2].
2. Умножить матрицу [2 x 2] на матрицу [2 x 2] по mod2.
3. Умножить матрицу [2 x 2] на матрицу [2 x 2] по mod3.
4. Умножить матрицу [3 x 3] на матрицу [3 x 1].
5. Умножить матрицу [3 x 3] на матрицу [3 x 1] по mod2.
6. Умножить матрицу [3 x 3] на матрицу [3 x 1] по mod3.
7. Умножить матрицу [3 x 3] на матрицу [3 x 3].
8. Умножить матрицу [3 x 3] на матрицу [3 x 3] по mod2.
9. Умножить матрицу [3 x 3] на матрицу [3 x 3] по mod3.
10. Умножить матрицу [4 x 4] на матрицу [4 x 1].
11. Умножить матрицу [4 x 4] на матрицу [4 x 1] по mod2.
12. Умножить матрицу [4 x 4] на матрицу [4 x 1] по mod3.
13. Умножить матрицу [4 x 4] на матрицу [4 x 4].
14. Умножить матрицу [4 x 4] на матрицу [4 x 4] по mod2.
15. Умножить матрицу [4 x 4] на матрицу [4 x 4] по mod3.
16. Умножить матрицу [5 x 5] на матрицу [5 x 1].
17. Умножить матрицу [5 x 5] на матрицу [5 x 1] по mod2.
18. Умножить матрицу [5 x 5] на матрицу [5 x 1] по mod3.
19. Умножить матрицу [5 x 5] на матрицу [5 x 5].
20. Умножить матрицу [5 x 5] на матрицу [5 x 5] по mod3.
21. Умножить матрицу [6 x 6] на матрицу [6 x 6].
22. Умножить матрицу [6 x 6] на матрицу [6 x 6] по mod3.
23. Умножить матрицу [7 x 7] на матрицу [7 x 7].
24. Умножить матрицу [7 x 7] на матрицу [7 x 7] по mod2.
25. Умножить матрицу [7 x 7] на матрицу [7 x 7] по mod3.
26. Умножить матрицу [8 x 8] на матрицу [8 x 8].
27. Умножить матрицу [9 x 9] на матрицу [9 x 9].

Программа 7.3.

```
; Умножить матрицу [2 x 2] на матрицу [2 x 2] по mod3.
include \masm64\include64\masm64rt.inc
; timeGetTime
include \masm64\include64\winmm.inc
```

```
        includelib \masm64\lib64\winmm.lib
.data?
            buf BYTE 64 dup (?)
            matrix_AB DWORD 4 dup (?)
.data
            matrix_A WORD  1, 2,
                           3, 4

            matrix_B WORD  5, 6,
                           7, 8
            buf_ptr QWORD buf
startup_info STARTUPINFO {2 dup(0),0,0,0,0,0,0,0,0,0,0,0,3 dup(0),0,0,0,0}
            process_info PROCESS_INFORMATION {0, 0, 0, 0}

.code
; Умножение по модулю 3 две матрицы с размерностью 2 x 2
; и возвращает результат в виде строки
;
; Алгоритм:
; Пусть A = |a1     b1|
;           |a2     b2|,
;
;         B = |c1   d1|
;             |c2   d2|
;
; тогда   (A * B) mod 3 =   |(a1c1 + b1c2) mod 3 (a1d1 + b1d2) mod 3|
;                           |(a2c1 + b2c2) mod 3 (a2d1 + b2d2) mod 3|
calculate proc
    LOCAL cloc: QWORD
            lea rsi, matrix_A
            ; xmm0 = a1, b1
            pinsrd xmm0, dword ptr[rsi+sizeof WORD*0],0
            ; xmm0 = a1, b1, a1, b1,
            pinsrd xmm0, dword ptr[rsi+sizeof WORD*0],1
            ; xmm0 = a1, b1, a1, b1, a2, b2,
            pinsrd xmm0, dword ptr[rsi + sizeof WORD*2],2
            ; xmm0 = a1, b1, a1, b1, a2, b2, a2, b2
            pinsrd xmm0, dword ptr[rsi + sizeof WORD*2],3

            lea rsi, matrix_B
            ; xmm1 = c1
pinsrw xmm1, word ptr[rsi+sizeof WORD*0],0;копир. мл. слова в позицию
            ; xmm1 = c1, c2
            pinsrw xmm1, word ptr[rsi + sizeof WORD * 2], 1
            ; xmm1 = c1, c2, d1
```

```
	pinsrw xmm1, word ptr[rsi + sizeof WORD * 1], 2
	; xmm1 = c1, c2, d1, d2
	pinsrw xmm1, word ptr[rsi + sizeof WORD * 3], 3
	; xmm1 = c1, c2, d1, d2, c1, c2, d1, d2
	movlhps xmm1, xmm1


	; xmm0 = A * B
	pmaddwd xmm0, xmm1 ; умножить и сложить


	lea rsi, matrix_AB
	; matrix_AB = xmm0
	pextrd dword ptr[rsi+sizeof DWORD*0],xmm0,0
	pextrd dword ptr[rsi+sizeof DWORD*1],xmm0,1
	pextrd dword ptr[rsi+sizeof DWORD*2],xmm0,2
	pextrd dword ptr[rsi+sizeof DWORD*3],xmm0,3

	; foreach dword in matrix_AB do
	;                               dword %= 3
	mov rcx, 4
	mov ebx, 3
@@:
	mov eax, dword ptr[rsi]
	cdq
	idiv ebx
	mov dword ptr[rsi], edx
	add rsi, sizeof dword
	dec rcx
	jnz @b
	sub rsi, sizeof dword * 4

	; создать строку с результатами
	mov cloc, 0
	invoke szappend, buf_ptr, cfm$("(A * B) mod 3 = \t|"), cloc
	mov cloc, rax
	mov eax, dword ptr[rsi + sizeof DWORD * 0]
	cdqe                            ; eax в rax
	invoke szappend,buf_ptr,str$(rax),cloc;добавление строк в буфер
	mov cloc, rax
	invoke szappend,buf_ptr," ",cloc
	mov cloc, rax
	mov eax,dword ptr[rsi + sizeof DWORD * 1]
	cdqe
	invoke szappend,buf_ptr,str$(rax),cloc
```

```
            mov cloc, rax
            invoke szappend, buf_ptr, cfm$("| \n \t\t|"), cloc
            mov cloc, rax
            mov eax, dword ptr[rsi + sizeof DWORD * 2]
            cdqe
            invoke szappend, buf_ptr, str$(rax), cloc
            mov cloc, rax
            invoke szappend, buf_ptr, " ", cloc
            mov cloc, rax
            mov eax, dword ptr[rsi + sizeof DWORD * 3]
            cdqe
            invoke szappend, buf_ptr, str$(rax), cloc
            mov cloc, rax
invoke szappend, buf_ptr, cfm$("| \n\n CIT-117G, Danylevych R.I."), cloc
            mov rax, buf_ptr
            ret
calculate endp


; Проверяет отлаживается ли программа,
; выводя при этом соответствующее сообщение
;
; Алгоритм:
; Если на выполнение кода уходит больше чем 1 миллисекунда
; - возможно программа отлаживается
is_debugger_present proc
            LOCAL _tmp: QWORD
            call timeGetTime
            mov _tmp, rax
            call timeGetTime
            sub rax, _tmp
            .If rax le 1
invoke MessageBox,0,"Приложение не отлаживается", "Anti-
debugging",MB_OK
            .Else
invoke MessageBox,0,"Выполнение кода занимает много времени!","Anti-
debugging",MB_OK
            .EndIf
             ret
is_debugger_present endp

; Запускает процесс "check_debugger", который проверяет запущен ли
отладчик x64dbg
; Проверяет отлаживается ли программа путем оценки времени на
выполнение кода
```

```
; Вычисляет произведение матриц и выводит результат

entry_point proc
            mov startup_info.cb, sizeof STARTUPINFO
invoke CreateProcess,0,"check_debugger.exe",0,0,0,0,0,0,addr startup_info,
addr process_info
  fn CloseHandle, process_info.hProcess
  fn CloseHandle, process_info.hThread
            call is_debugger_present
            call calculate
            invoke MsgboxI,0,rax,"Результат операции",MB_OK,10
   .exit
entry_point endp
        end
```

Программа 7.4. Файл с именем check_debugger:

```
        include \masm64\include64\masm64rt.inc
.code
; Проверяет запущен ли отладчик x64dbg,
; выводя при этом соответствующее сообщение
entry_point proc
            invoke FindWindow, 0, "x64dbg"

            .If rax gt 0
invoke MessageBox,0,"Found x64dbg! Приложение может отлаживаться",
"Anti-debugging", MB_OK
            .Else
invoke MessageBox, 0, "Приложение не отлаживается", "Anti-debugging",
MB_OK
            .EndIf
.exit
        entry_point endp
        end
```

В рассмотренной программе применена команда совмещенного умножения и сложения VPMADDWD, которая имеет синтаксис:

```
VPMADDWD xmm1, xmm2, xmm3/mem128
VPMADDWD ymm1, ymm2, ymm3/mem256
```

и выполняет операции над 16-разрядными данными.

Рассмотрим в качестве примера программу 7.5, в которой осуществляется умножение матрицы [3 x 3] на матрицу [3 x 1] без использования команды VPMADDWD совмещенного умножения и

сложения 16-разрядных данных и без использования модуля числа. В рассматриваемой программе данные 32-разрядные.

Программа 7.5. Умножение матрицы [3 x 3] на матрицу [3 x 1]:

```
      include \masm64\include64\masm64rt.inc ;
.data
      mas1 dd 1,2,3 ; a1, b1, c1
      mas2 dd 4,5,6 ; a2, b2, c2
      mas3 dd 3,2,1 ; a3, b3, c3

      mas4 dd 10,; d1
            11,; d2
            12 ; d3
      buf1 dq 3 dup(0); буфер
      fmt1 db "masm64.",10,10,
      "|1  2  3|     |10|",10,"|4  5  6|  x  |11|", 10,"|3  2  1|     |12|",10,10,
      "Результат = |%d  %d  %d|T",10,10,
      "Автор: Рысованый А.Н., каф. ВТП, НТУ ХПИ",0
      titl1 db "Перемножение матриц 3x3 на 3x1.",0

.code ;
entry_point proc
;;;; Умножить матрицу [3 x 3] на матрицу [3 x 1]
      vmovd xmm1,mas1[0]    ; a1 [3x3]
      vmovd xmm2,mas1[4]    ; b1
      vmovd xmm3,mas1[8]    ; c1

      vmovd xmm4,mas4[0]    ; d1
      vmovd xmm5,mas4[4]    ; d2
      vmovd xmm6,mas4[8]    ; d3
      vpmulld xmm10,xmm1,xmm4 ;[a1xd1]
      vpmulld xmm11,xmm2,xmm5 ;[b1xd1]
      vpmulld xmm12,xmm3,xmm6 ;[c1xd3]
      vaddsd xmm10,xmm10,xmm11
      vaddsd xmm10,xmm10,xmm12
      vmovd r10d,xmm10
      movsxd r10,r10d ; [3x1] result

      vmovd xmm1,mas2[0]    ; a2 [3x3]
      vmovd xmm2,mas2[4]    ; b2
      vmovd xmm3,mas2[8]    ; c2
      vpmulld xmm7,xmm1,xmm4 ;[a2xd1]
      vpmulld xmm8,xmm2,xmm5 ;[b2xd2]
      vpmulld xmm9,xmm3,xmm6 ;[c3xd3]
      vaddsd xmm11,xmm7,xmm8  ;
```

```
        vaddsd xmm11,xmm11,xmm9 ;
        vmovd r11d,xmm11
        movsxd r11,r11d ;a2 [3x1] result

        vmovd xmm1,mas3[0]    ; a3 [3x3]
        vmovd xmm2,mas3[4]    ; b3
        vmovd xmm3,mas3[8]    ; c3
        vpmulld xmm1,xmm1,xmm4 ;[a3xd1]
        vpmulld xmm2,xmm2,xmm5 ;[b3xd2]
        vpmulld xmm3,xmm3,xmm6 ;[c3xd3]
        vaddsd xmm12,xmm1,xmm2
        vaddsd xmm12,xmm12,xmm3
        vmovd r12d,xmm12
        movsxd r12,r12d ;a3 [3x1] result

invoke wsprintf,addr buf1,ADDR fmt1,r10,r11,r12
invoke MessageBox,0,addr buf1,addr titl1,MB_OK
        invoke ExitProcess,0
        entry_point endp
        end
```

Результат выполнения программы представлен на рис. 7.4.

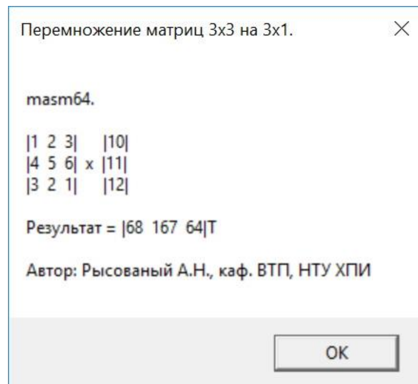


Рис. 7.4. Результат выполнения программы

В программе 7.6 32-разрядные данные представлены в виде структур. Поэтому операции умножения и сложения выполняются с элементами этих структур командами AVX.

Программа 7.6. Умножение матрицы [3 x 3] на матрицу [3 x 1]:

```
include \masm64\include64\masm64rt.inc ; библиотеки
COLUMN STRUCT
    El1 dd ?
    El2 dd ?
    El3 dd ?
COLUMN ENDS

MATRIX STRUCT
    L1 COLUMN <?,?,?>
    L2 COLUMN <?,?,?>
    L3 COLUMN <?,?,?>
MATRIX ENDS

.data ;
  matrix2 COLUMN <10,11,12>
  matrix1 MATRIX <<1,2,3>,<4,5,6>,<3,2,1>>
  outM1 db "Первая матрица:",10," 1,  2,  3",10," 4,  5,  6",10," 3,  2,  1",0
  outM2 db "Вторая матрица:",10,9,"10",10,9,"11",10,9,"12",0
    titl1 db "Перемножение матриц",0
    buf1 dq 3 dup(0),0; буфер
    buffer dd ?
      fmt1 db "masm64.",10,10,
      "Результат = ",3 dup(" %d "),10,10,
      "Автор:  каф. ВТП, НТУ ХПИ",0

.code ;
entry_point proc
    invoke MessageBox,0,addr outM1,addr titl1,MB_ICONINFORMATION
    invoke MessageBox,0,addr outM2,addr titl1,MB_ICONINFORMATION
;get element C(1,1)
      vmovups xmm2, matrix1.L1.El1
      vmovups xmm3, matrix2.El1
      vpmulld xmm6, xmm2, xmm3

        vmovups xmm2, matrix1.L1.El2
        vmovups xmm3, matrix2.El2
        vpmulld xmm7, xmm2, xmm3

      vmovups xmm2, matrix1.L1.El3
      vmovups xmm3, matrix2.El3
      vpmulld xmm8, xmm2, xmm3

        vaddss xmm6, xmm6, xmm7
        vaddss xmm6, xmm6, xmm8
```

```
    movss buffer, xmm6
    movsxd r10,buffer

;get element C(2,1)
   vmovups xmm2, matrix1.L2.El1
   vmovups xmm3, matrix2.El1
   vpmulld xmm6, xmm2, xmm3

   vmovups xmm2, matrix1.L2.El2
   vmovups xmm3, matrix2.El2
   vpmulld xmm7, xmm2, xmm3

   vmovups xmm2, matrix1.L2.El3
   vmovups xmm3, matrix2.El3
   vpmulld xmm8, xmm2, xmm3

   vaddss xmm6, xmm6, xmm7
   vaddss xmm6, xmm6, xmm8
   movss buffer, xmm6
   movsxd r11,buffer

;get element C(3,1)
   vmovups xmm2, matrix1.L3.El1
   vmovups xmm3, matrix2.El1
   vpmulld xmm6, xmm2, xmm3

   vmovups xmm2, matrix1.L3.El2
   vmovups xmm3, matrix2.El2
   vpmulld xmm7, xmm2, xmm3

   vmovups xmm2, matrix1.L3.El3
   vmovups xmm3, matrix2.El3
   vpmulld xmm8, xmm2, xmm3

   vaddss xmm6, xmm6, xmm7
   vaddss xmm6, xmm6, xmm8
   movss buffer, xmm6
   movsxd r12,buffer

   ;output result
   invoke wsprintf,addr buf1,ADDR fmt1,r10,r11,r12
   invoke MessageBox,0,addr buf1,addr titl1,MB_OK
   invoke ExitProcess,0
   entry_point endp
   end
```

Результат выполнения программы представлен на рис. 7.5.

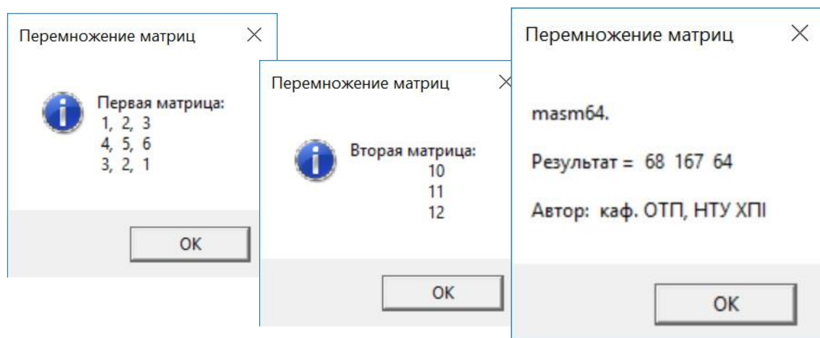


Рис. 7.5. Результат выполнения программы

В качестве еще одного варианта умножения матриц в программе 7.7 приведено умножение при помощи структур и обычных команд общего назначения.

Программа 7.7. Умножение матрицы [5 x 5] на матрицу [5 x 1]:

```
include \masm64\include64\masm64rt.inc ; библиотеки
DATE1 STRUCT ;  СТРУКТУРА с именем DATE1
    col1 dd ? ; имя первого поля структуры
    col2 dd ? ; имя второго поля структуры
    col3 dd ? ; имя третьего поля структуры
    col4 dd ? ; имя четвертого поля структуры
    col5 dd ? ; имя пятого поля структуры
DATE1 ENDS ; окончание структуры с именем Date1

DATE2 STRUCT ; СТРУКТУРА с именем DATE2
   col01 dd ? ;
DATE2 ENDS ; окончание структуры

.data
titl1 db "Умножение матриц 5*5 на 5*1",0;
buf1 dq 0,0; буфер
fmt1 db "Первый массив – в виде матрицы:",0dh,0ah,\
"01  02 -01  03  04",10,\
"02 -02  02  02  02",10,\
```

```
	"03  03  04  05  10",10,\
	"04  04  10 -11  05",10,\
	"05  04  03  02  01",0
	fmt2 db "Второй массив – в виде матрицы:",0dh,0ah,\
	9,"05",10,9,"06",10,9,"07",10,9,"08",10,9,"09",0
	fmt3 db "Результат – в виде матрицы:",0dh,0ah,\
	9,"%d",10,9,"%d",10,9,"%d",10,9,"%d",10,9,"%d",10,10,\
"Кафедра <Вычислительная техника и программирование>, НТУ ХПИ",0

	res1 dq ?     ; ячейки для результата
	res2 dq ?
	res3 dq ?
	res4 dq ?
	res5 dq ?

	; задание имен строк структуры
	Car1 DATE1 <1,4,-2,3,5> ; структура с именем Car1
	Car2 DATE1 <2,-11,6,7,5> ; структура с именем Car2
	Car3 DATE1 <3,12,10,11,5> ; структура с именем Car3
	Car4 DATE1 <4,13,14,-15,5> ; структура с именем Car4
	Car5 DATE1 <4,3,14,15,5> ; структура с именем Car5

	Car6 DATE2 <5> ; структура с именем Car6
	Car7 DATE2 <6> ; структура с именем Car7
	Car8 DATE2 <7> ; структура с именем Car8
	Car9 DATE2 <8> ; структура с именем Car9
	Car10 DATE2 <9> ; структура с именем Car10

.code
 proc1 proc
	mov ecx,Car1.col1
	mov ebx, Car6.col01
	imul ecx,ebx ; 1*1
	movsxd r15,ecx

	mov ecx,Car1.col2
	mov ebx, Car7.col01
	imul ecx,ebx ; 1*2
	movsxd r14,ecx
	add r15,r14

	mov ecx,Car1.col3
	mov ebx, Car8.col01
	imul ecx,ebx ; 1*3
	movsxd r13,ecx
```

```
        add r15,r13

        mov ecx,Car1.col4
        mov ebx, Car9.col01
        imul ecx,ebx ; 1*4
        movsxd r12,ecx
        add r15,r12

        mov ecx,Car1.col5
        mov ebx, Car10.col01
        imul ecx,ebx ; 1*5
        movsxd r11,ecx
        add r15,r11
        mov res1,r15
invoke wsprintf,addr buf1,ADDR fmt1
invoke MessageBox,0,ADDR buf1,ADDR titl1,MB_ICONINFORMATION
        ret
         proc1 endp

proc2 proc
        mov ecx,Car2.col1
        mov ebx, Car6.col01
        imul ecx,ebx ; 2*1
        movsxd r15,ecx

        mov ecx,Car2.col2
        mov ebx, Car7.col01
        imul ecx,ebx ; 2*2
        movsxd r14,ecx
        add r15,r14

        mov ecx,Car2.col3
        mov ebx, Car8.col01
        imul ecx,ebx ; 2*3
        movsxd r13,ecx
        add r15,r13

        mov ecx,Car2.col4
        mov ebx, Car9.col01
        imul ecx,ebx ; 2*4
        movsxd r12,ecx
        add r15,r12

        mov ecx,Car2.col5
        mov ebx, Car10.col01
```

```
        imul ecx,ebx ; 2*5
        movsxd r11,ecx
        add r15,r11

        mov res2,r15
invoke wsprintf,addr buf1,ADDR fmt2
invoke MessageBox,0,ADDR buf1,ADDR titl1,MB_ICONINFORMATION
        ret
        proc2 endp

proc3 proc
        mov ecx,Car3.col1
        mov ebx, Car6.col01
        imul ecx,ebx ; 3*1
        movsxd r15,ecx

        mov ecx,Car3.col2
        mov ebx, Car7.col01
        imul ecx,ebx ; 3*2
        movsxd r14,ecx
        add r15,r14

        mov ecx,Car3.col3
        mov ebx, Car8.col01
        imul ecx,ebx ; 3*3
        movsxd r13,ecx
        add r15,r13

        mov ecx,Car3.col4
        mov ebx, Car9.col01
        imul ecx,ebx ; 3*4
        movsxd r12,ecx
        add r15,r12

        mov ecx,Car3.col5
        mov ebx, Car10.col01
        imul ecx,ebx ; 3*5
        movsxd r11,ecx
        add r15,r11

        mov res3,r15
        ret
        proc3 endp

proc4 proc
```

```
        mov ecx,Car4.col1
        mov ebx, Car6.col01
        imul ecx,ebx ; 4*1
        movsxd r15,ecx

        mov ecx,Car4.col2
        mov ebx, Car7.col01
        imul ecx,ebx ; 4*2
        movsxd r14,ecx
        add r15,r14

        mov ecx,Car4.col3
        mov ebx, Car8.col01
        imul ecx,ebx ; 4*3
        movsxd r13,ecx
        add r15,r13

        mov ecx,Car4.col4
        mov ebx, Car9.col01
        imul ecx,ebx ; 4*4
        movsxd r12,ecx
        add r15,r12

        mov ecx,Car4.col5
        mov ebx, Car10.col01
        imul ecx,ebx ; 4*5
        movsxd r11,ecx
        add r15,r11

        mov res4,r15
        ret
        proc4 endp

proc5 proc
        mov ecx,Car5.col1
        mov ebx, Car6.col01
        imul ecx,ebx ; 5*1
        movsxd r15,ecx

        mov ecx,Car5.col2
        mov ebx, Car7.col01
        imul ecx,ebx ; 5*2
        movsxd r14,ecx
        add r15,r14
```

```
	mov ecx,Car5.col3
	mov ebx, Car8.col01
	imul ecx,ebx ; 5*3
	movsxd r13,ecx
	add r15,r13

	mov ecx,Car5.col4
	mov ebx, Car9.col01
	imul ecx,ebx ; 5*4
	movsxd r12,ecx
	add r15,r12

	mov ecx,Car5.col5
	mov ebx, Car10.col01
	imul ecx,ebx ; 5*5
	movsxd r11,ecx
	add r15,r11

	mov res5,r15
	ret
	proc5 endp

entry_point proc
	invoke proc1
	invoke proc2
	invoke proc3
	invoke proc4
	invoke proc5
invoke wsprintf,addr buf1,ADDR fmt3, res1, res2, res3, res4,res5
invoke MessageBox,0,ADDR buf1,ADDR titl1,MB_ICONINFORMATION
	entry_point endp
	end
```

Результат выполнения программы представлен на рис. 7.6.

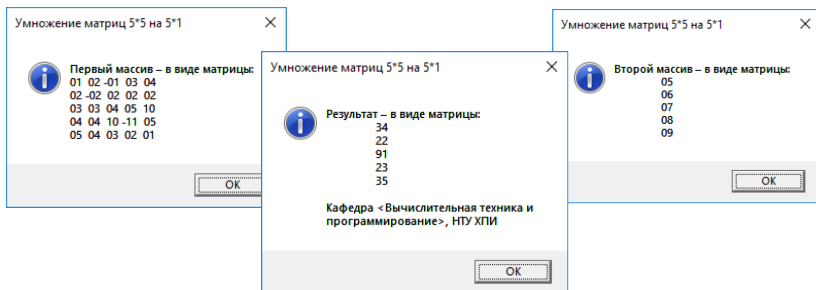


Рис. 7.6. Результат выполнения программы

В программе 7.8 рассмотрен пример использования матрицы как двумерного массива. Смещение, указанное в записи, например mat1[8][32], обозначает, что элемент массива имеет смещение, равное сумме этих смещений: 8 + 32 = 40, т.е. выбирается элемент, который располагается первым во второй строке.

Программа 7.8. Сумма элементов под главной диагональю матрицы:

```
      include \masm64\include64\masm64rt.inc
.data
    mat1   dq 1,2,3,4,5
           dq 6,7,8,9,10
           dq 2,3,4,5,1
           dq 3,4,8,6,3
           dq 6,5,4,7,8
      buf dq ?,0
      titl db "Результат",0
      fmt db "Сумма: %d",0
.code
  entry_point proc
      mov rax,mat1[8][32]
      add rax,mat1[24][64]
      add rax,mat1[40][96]
      add rax,mat1[56][128]
  invoke wsprintf,addr buf,ADDR fmt,rax
  invoke MessageBox,0,addr buf,addr titl,MB_ICONINFORMATION
  invoke ExitProcess,0
entry_point endp
end
```

Рассмотрм пример создания **Windows окно с BMP-файлом**. Для этого будем использовать файлы:

1. largeJpg.asm; исходный код программы пользователя;
2. largeJpg.rc; файл ресурсов пользователя;
3. icon1.ico; иконка пользователя;
4. DSC00388-2.jpg ; фото для окна;
5. makeit.bat с использованием rc.exe.

Рассмотрим последовательность создания **Windows окна с загрузкой BMP-файла:**

1. Создаем новый файл ресурсов (**File – New Project** и вводим имя). В правой верхней панели появляется значок папки и имя файла. Желательно все файлы одного проекта называть одним именем, но с разными расширениями.

Определяемся для себя: будут ли обрабатываться главные кнопки меню или только всплывающие. От этого зависит, надо ли задавать текстовые идентификаторы на главные кнопки.

2. Нажимаем на созданное имя файла правой кнопкой мыши и выбираем «**Add Menu**» (рис. 7.7).

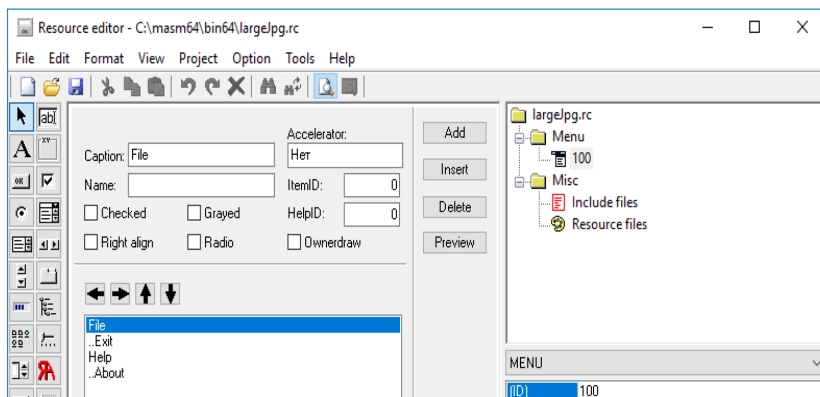


Рис. 7.7. Создание окна в программе ResEd

3. Заполняем свойства кнопки меню:

- в поле Caption вводим название кнопки меню и запоминаем или меняем идентификатор кнопки. Можно поменять свойства пункта меню и в правом нижнем поле.

- в поле Name добавляем текстовый идентификатор, например Name_Prog (если в программе при нажатии на эту кнопку будут происходить какие-нибудь действия).

4. Для ввода новой кнопки нажимаем **Add** и если это будет всплывающая кнопка выполняем действия:

- вводим в поле Caption другое название новой кнопки, которая будет являться дополнительной кнопкой к кнопке «File»;

- в поле Name добавляем текстовый идентификатор, например Exit;

- выбираем стрелку «Вправо». Перед словом «Exit» появляется две точки, что свидетельствует, что создана всплывающая кнопка к предыдущему имени «File».

5. Для создания кнопки меню с именем «Help» нажимаем «**Add**» и выполняем действия:

- вводим в поле Caption название новой кнопки, например «Help», которая будет второй главной кнопкой меню (вместе с кнопкой «File»);
- в поле Name добавляем (если надо) текстовый идентификатор. Стрелку «Вправо» не нажимаем, т.к. это главная кнопка.

Для того, чтобы посмотреть на сделанные изменения необходимо нажать на кнопку «Preview» (рис. 7.8).

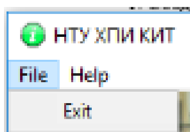


Рис. 7.8. Создание окна в программе ResEd

В окончательном варианте окно программы создания ресурсов представлен на рис. 7.9.

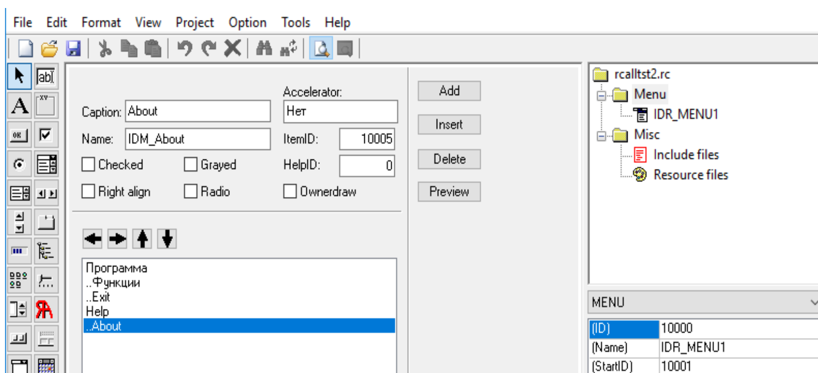


Рис. 7.9. Окончательный вариант окна в программе ResEd

7. Добавляем файл ресурсов.

Добавление файла констант. Для этого, нажимаем правой кнопкой мыши по значку папки, выбираем «Include file», затем «Add». Выбираем в окне программы три точки и вводим путь, например C:\masm64\include64\RESOURCE.H (рис. 7.10).

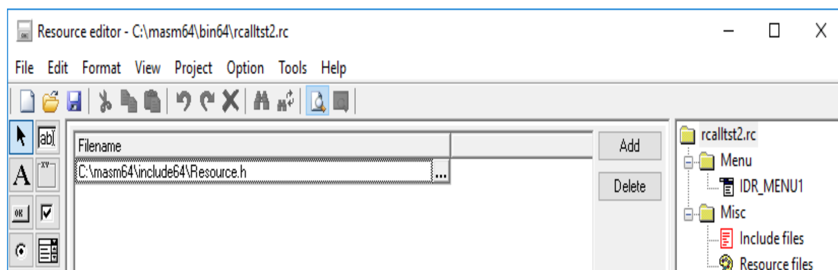


Рис. 7.10. Добавление файла RESOURCE.H в программе ResEd

8. Сохраняем программу

9. Открываем созданный rc-файл и **добавляем строку** с ресурсом курсора:
10 ICON MOVEABLE PURE LOADONCALL DISCARDABLE "icon.ico“

В этой строке число 10 обозначает цифровой идентификатор иконки (ресурс). Он может быть любым, но и в rc- и asm-файлах он должен быть одним и тем же.

10. Добавляем ресурс фотографии

```
20  RCDATA "DSC00388-2.jpg"
```

11. ИЗМЕНЯЕМ ресурс меню на строку

```
100 MENUEX MOVEABLE IMPURE LOADONCALL DISCARDABLE
```

12. Окончательно сохраняем изменения

Программа 7.9. Файл ресурсов rc:

```
#define IDR_MENU1 10000
#define IDM_File 10001
#define IDM_Exit 10002
#define IDM_Help 10003
#define IDM_About 10004
```

```
#include "C:/masm64/include64/Resource.h"
10  ICON MOVEABLE PURE LOADONCALL DISCARDABLE "icon.ico"
20  RCDATA "DSC00388-2.jpg"

100  MENUEX MOVEABLE IMPURE LOADONCALL DISCARDABLE
    BEGIN
      POPUP "File"
      BEGIN
        MENUITEM "Exit",IDM_Exit
      END
      POPUP "Help"
      BEGIN
        MENUITEM "About",IDM_About
      END
    END
```

Программа 7.10. Файл asm:

```
include \masm64\include64\masm64rt.inc
.data?
      hInstance dq ? ; дескриптор програми
      hWnd      dq ? ; дескриптор окна
      hIcon     dq ? ; дескриптор иконки
      hCursor   dq ? ; дескриптор курсора
      sWid      dq ? ; ширина монитора (колич. пикселей по x)
      sHgt      dq ? ; высота монитора (колич. пикселей по y)
     hImage  dq ?
     hStatic dq ?
.data
       classname db "template_class",0
       caption db "НТУ ХПИ КИТ",0

.code
entry_point proc

    GdiPlusBegin       ; initialise GDIPlus
 mov hInstance,rv(GetModuleHandle,0) ; получение и сохранение
дескриптора програми
 mov hIcon,  rv(LoadIcon,hInstance,10) ; загрузка и сохранение
дескриптора иконки
 mov hCursor,rv(LoadCursor,0,IDC_ARROW) ; загрузка курсора и
сохранение
 mov sWid,rv(GetSystemMetrics,SM_CXSCREEN) ; получение кол.
пикселей по x монитора
```

```
 mov sHgt,rv(GetSystemMetrics,SM_CYSCREEN) ; получение кол.
пикселей по у монитора
  mov hImage,rv(ResImageLoad,20) ; макрос загрузки Bitmap
          call main
        GdiPlusEnd        ; GdiPlus cleanup
          invoke ExitProcess,0
          ret
        entry_point endp

        main proc
        LOCAL wc  :WNDCLASSEX ; объявление локальных переменных
        LOCAL lft :QWORD    ;  Лок. переменные содержатся в стеке
        LOCAL top :QWORD     ; и существуют только во время вып. проц.
        LOCAL wid :QWORD
        LOCAL hgt :QWORD
mov wc.cbSize,SIZEOF WNDCLASSEX ; колич. байтов структуры
mov wc.style,CS_BYTEALIGNCLIENT or CS_BYTEALIGNWINDOW ; стиль
окна
mov wc.lpfnWndProc,ptr$(WndProc) ; адрес процедуры WndProc
mov wc.cbClsExtra,0       ; количество байтов для структуры класса
mov wc.cbWndExtra,0       ; количество байтов для структуры окна
mrm wc.hInstance,hInstance ; заполнение поля дескриптора в структуре
mrm wc.hIcon,  hIcon       ; хэндл иконки
mrm wc.hCursor,hCursor     ; хэндл курсора
mrm wc.hbrBackground,0    ; hBrush цвет окна
mov wc.lpszMenuName,0 ; заполнение поля в структуре с именем
ресурса меню
mov wc.lpszClassName,ptr$(classname) ; имя класса
mrm wc.hIconSm,hIcon
invoke RegisterClassEx,ADDR wc ; регистрация класса окна

; расположение окна по середине экрана монитора по координате X
          mov wid,512   ; ширина пользовательского окна в пикселях
          mov hgt,380   ; высота пользовательского окна в пикселях
          mov rax,sWid  ; колич. пикселей монитора по x
          sub rax,wid   ; дельта X = X(монитора) - x(окна пользователя)
          shr rax,1     ; получение середины X
          mov lft,rax   ;

 ; расположение окна по середине экрана монитора по координате Y
          mov rax, sHgt ; колич. пикселей монитора по y
          sub rax, hgt
          shr rax, 1
          mov top, rax
```

```
invoke CreateWindowEx,WS_EX_LEFT or WS_EX_ACCEPTFILES, \
       ADDR classname,ADDR caption, \
       WS_OVERLAPPED or WS_VISIBLE or WS_SYSMENU,\
         lft,top,wid,hgt,0,0,hInstance,0
    mov hWnd,rax ; сохранение дескриптора окна
    call msgloop
    ret
main endp


msgloop proc
    LOCAL msg    :MSG
    LOCAL pmsg   :QWORD
 mov pmsg, ptr$(msg) ; получение адреса структуры сообщения
 jmp gmsg            ; jump directly to GetMessage()
mloop:
 invoke TranslateMessage,pmsg
 invoke DispatchMessage,pmsg
gmsg:
 test rax, rv(GetMessage,pmsg,0,0,0) ; пока GetMessage не вернет ноль
 jnz mloop
 ret
msgloop endp


WndProc proc
hWin:QWORD,uMsg:QWORD,wParam:QWORD,lParam:QWORD
.switch uMsg
    .case WM_COMMAND   ; если выбрано меню
       .switch wParam
          .case 10002    ; если выбрана кнопка Exit
invoke SendMessage,hWin,WM_SYSCOMMAND,SC_CLOSE,0
          .case 10004    ; если выбрана кнопка About
.data
            msgtxt db "masm64",10
            db "Вывод BMP (JPG) файла",0
.code
invoke MsgboxI,hWin,ptr$(msgtxt),"Выбрана кнопка About In
Window",MB_OK,10

rcall SendMessage,hWin,WM_SYSCOMMAND,SC_CLOSE,0 ;
уничтожить окно
       .endsw
    .case WM_CREATE
```

```
invoke CreateWindowEx,WS_EX_LEFT,"STATIC",0,WS_CHILD or
WS_VISIBLE or SS_BITMAP,\
        0,0,0,0,hWin,hInstance,0,0
        mov hStatic,rax
invoke SendMessage,hStatic,STM_SETIMAGE,IMAGE_BITMAP,hImage;
сообщение окну
        invoke LoadMenu,hInstance,100 ; загружает меню из exe-файла
        invoke SetMenu,hWin,rax ; связывает меню с окном
                .return 0
            .case WM_CLOSE ;
                invoke SendMessage,hWin,WM_DESTROY,0,0
            .case WM_DESTROY ;
                invoke PostQuitMessage,NULL
        .endsw
            invoke DefWindowProc,hWin,uMsg,wParam,lParam
            ret
        WndProc endp
            end
```

Окончательный результат окна приведен на рис. 7.11.

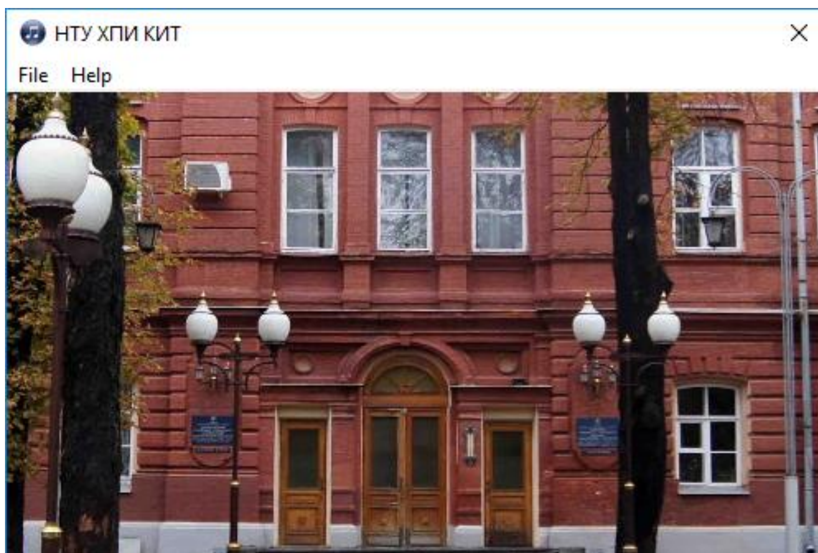


Рис. 7.11. Результат выполнения программы

СПИСОК ЛИТЕРАТУРЫ

1. Касперски Крис. Техника и философия хакерских атак - записки мыщ'а / Касперски Крис. – М.: Соломон-Пресс, 2004. — 272 с.

2. Касперски Крис, Рокко Ева. Искусство дизассемблирования / Касперски Крис, Рокко Ева. – Санкт-Петербург, БХВ-Петербург, 2008 – 896 с.

3. Касперски Крис. Образ мышления – дизассемблер IDA / Касперски Крис. – М.: СОЛОН-Р, 2001. — 480 с.

4. Сикорски М., Хониг Э. Вскрытие покажет! Практический анализ вредоносного ПО. — СПб.: Питер, 2018. — 768 с.: ил. — (Серия «Для профессионалов»).

5. Панов А.С. Реверсинг и защита программ от взлома / А.С. Панов. – Санкт-Петербург, БХВ-Петербург, 2006 – 256 с.

6. Методические указания к выполнению лабораторных работ по курсу «Реверсное программирование», Ч.1 «Внедрение кода» для студентов специальностей: 123 – «Компьютерная инженерия», 125 – «Кибербезопасность» / составитель А.Н. Рысованый. – Х. : «Слово», 2019. – 84 с.

7. Методические указания для выполнения лабораторных работ по курсу «Реверсное программирование», Ч.2 «Крэкинг: практика взлома простых программ. Среда программирования masm64» для студентов специальностей: 123 – «Компьютерная инженерия», 125 – «Кибербезопасность» всех форм обучения/ составитель А.Н. Рысованый. – Х. : «Слово», 2020. – 92 с.

8. Рысованый А.Н. Системное программирование, Ч.1. Программирование в среде masm64 : учеб.-метод. пособие / А.Н. Рысованый. – Харьков : «Слово», 2017. – 108 с.– На рус. яз.

9. Рысованый А.Н. Системное программирование, Ч.2. Расширенные возможности программирования в среде masm64 : учеб.-метод. пособие / А.Н. Рысованый. – Харьков : «Слово», 2017. – 140 с. – На рус. яз.

10. Рисований О. М. Системне програмування : підручник для студентів напряму “Комп’ютерна інженерія” вищих навчальних закладів / О.М. Рисований. – Харків : НТУ “ХПІ”, 2010. – 912 с.

СОДЕРЖАНИЕ

ВСТУПЛЕНИЕ ... 3
1. АНТИОТЛАДОЧНЫЕ ПРИЕМЫ. ОБЩИЕ СВЕДЕНИЯ ... 4
2. ОБРАБОТКА ИСКЛЮЧЕНИЙ ... 5
2.1. Трассировка по флагу TF ... 6
2.2. Трассировка по маске ... 8
3. ТРАССИРОВКА ПО ВРЕМЕНИ ВЫПОЛНЕНИЯ ИНСТРУКЦИЙ ... 10
3.1. Использование команды rdtsc ... 11
3.2. Использование функции GetTickCount ... 15
3.3. Функция TimeGetTime ... 21
3.4. Использование функции QueryPerformanceCounter ... 21
3.5. Функция QueryPerformanceFrequency ... 23
3.6. Использование функции GetSystemTimeAsFileTime ... 23
3.7. Функция GetSystemTimePreciseAsFileTime ... 25
3.8. Функции KeTickCount, KeQueryTimeIncrement ... 25
3.9. Функция KeQueryPerformanceCounter ... 26
3.10. Функция KeQuerySystemTime ... 26
3.11. Функция KeQueryInterruptTime ... 27
3.12. Функция KeQueryTickCount ... 27
3.13. Функция KeQueryUnbiasedInterruptTime ... 28
3.14. Функция GetProcessTimes ... 28
3.15. Функция NtQueryInformationProcess ... 29
3.16. Функция NtQueryInformationThread ... 30
3.17. Функция NtSetInformationThread ... 31
3.18. Лабораторная работа «Антиотладка с использованием функций времени» ... 31
4. ФУНКЦИИ МУЛЬТИМЕДИЙНОГО ТАЙМЕРА ... 39
4.1. Использование функции timeBeginPeriod ... 40
4.2. Функция timeEndPeriod ... 41
4.3. Функция timeGetDevCaps ... 41
4.4. Функция timeGetSystemTime ... 42
4.5. Использование функции timeGetTime ... 42
4.6. Функция timeKillEvent ... 46
4.7. Функция TimeProc ... 47
4.8. Использование функции timeSetEvent ... 47

4.9. Лабораторная работа «Антиотладка с использованием функций мультимедийного таймера» 50
5. API-ФУНКЦИИ ДЛЯ ОБНАРУЖЕНИЯ ОТЛАДЧИКА 56
5.1. Использование функции IsDebuggerPresent 57
5.2. Использование функции CheckRemoteDebuggerPresent 58
5.3. Функция ContinueDebugEvent 60
5.4. Функция DebugActiveProcess 61
5.5. Функция DebugActiveProcessStop 62
5.6. Функция DebugBreak.. 62
5.7. Функция DebugBreakProcess 63
5.8. Функция DebugSetProcessKillOnExit 63
5.9. Функция FatalExit 63
5.10. Функция FlushInstructionCache... 64
5.11. Функция GetThreadContext 64
5.12. Функция GetThreadSelectorEntry.. 65
5.13. Функция OutputDebugString 66
5.14. Функция ReadProcessMemory... 66
5.15. Функция SetThreadContext.. 67
5.16. Функция WaitForDebugEvent.. 68
5.17. Функция Wow64GetThreadContext....................................... 69
5.18. Функция Wow64GetThreadSelectorEntry 69
5.19. Функция Wow64SetThreadContext 70
5.20. Лабораторная работа «Антиотладка с использованием функций обнаружения отладчика» 70
6. ОТЛАДОЧНЫЕ РЕГИСТРЫ... 78
7. ФЛАГИ ОТЛАДКИ, ПАМЯТЬ ПРИЛОЖЕНИЙ................. 80
7.1. Поиск отладчика по названию класса.................................... 80
7.2. Глобальная переменная NtGlobalFlag.................................... 84
7.3. Лабораторная работа «Антиотладка с использованием названия отладчика».. 85

СПИСОК ЛИТЕРАТУРЫ 109

Навчальне видання

РИСОВАНИЙ Олександр Миколайович

«РЕВЕРСНЕ ПРОГРАМУВАННЯ»

Антиналагоджувальні прийоми захисту від реверса.
Середовище програмування masm64

Навчальний посібник

Російською мовою

Роботу до видання рекомендував *М.Й. Заполовський*
Відповідальний за випуск *С.Г. Семенов*

В авторській редакції

План 2020 р., поз. 1.14

Формат 60х84 1/16. Папір офісний.
Riso-друк. Гарнітура Times. Ум. друк. арк. 7. Наклад 50 прим.
Зам. № Ціна договірна.

Видавничий центр НТУ «ХПІ»
Свідоцтво суб'єкта видавничої справи ДК №5478 від 21.08.2017 р.
Вул. Кирпичова, 2, м. Харків, 61002