РЫСОВАНЫЙ
Александр Николаевич –

Родился 22.09.1957.
Закончил с отличием Харьковское высшее военное авиационное командное училище связи в 1977 г., Военно-воздушную инженерную орденов Ленина и Октябрьской революции Краснознаменную академию им. проф. Н.Е. Жуковского в 1984 г., г. Москва.
Специальность "Автоматизированные системы управления", квалификация “военный инженер системотехник”.

Служба в Вооруженных Силах – 31 год: ХВВАКУС; в/ч пп 81524, Wittstock/Dosse; ВВИА; ХВВАУРЭ; ХВУ; ХУВС.
Воинское звание – полковник.

Работает в НТУ “ХПИ” с 2006 г.
Проф. кафедры “Вычислительная техника и программирование” с 2013 г., профессор НТУ ХПИ с 2015 г.
Кандидат технических наук (год защиты – 1995), доцент.
Имеет более 250 научных публикаций, из них 6 учебников, 38 учебных пособий, более 100 изобретений, оформленных в виде авторских свидетельств, патентов, патентов на полезные модели и др.

Руководитель магистров и аспирантов.

Область научных исследований: развитие теории недвоичного сигнатурного анализа для повышения эффективности диагностирования цифровых систем.
e-mail: rysov@rambler.ru

Рассмотрены вопросы использования новой среды программирования masm64 на языке ассемблер ml64, этапы создания программы на masm64, особенности использования отладчика x64Dbg и лабораторные работы с заданиями и примерами выполнения в среде masm64.

Предназначено для студентов специальности 123 – «Компьютерная инженерия», специализаций: 123-01 «Компьютерные системы и сети»; 123-02 «Системное программирование»; 123-03 «Специализированные компьютерные системы».

А.Н. Рысованый

Системное программирование

Часть I
Программирование в среде masm64

Учебно-методическое пособие

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

А.Н. Рысованый

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

Ч.1. «Программирование в среде masm64»

Учебно-методическое пособие
для студентов специальности 123 – «Компьютерная инженерия» специализаций: 123-01 «Компьютерные системы и сети»; 123-02 «Системное программирование»; 123-03 «Специализированные компьютерные системы»

Утверждено
редакционно-издательским
советом университета,
протокол №2 от 23.06.2016

http://blogs.kpi.kharkov.ua/v2/asm/knigi/

Харьков
2017

ББК 32.973-018.1
УДК 004.42
Р54

Рецензенты:

И.А. Фурман, д-р техн. наук, профессор, академик АН Высшей школы Украини, ХНТУСХ;

О.А. Козина, канд. техн. наук, профессор НТУ ХПИ

Розглянуті питання використання нового середовища програмування masm64 на мові асемблер ml64, етапи створення програми на masm64, особливості використання налагоджувача x64dbg та лабораторні роботи із завданнями і прикладами виконання в середовищі masm64.

Призначено для студентів спеціальності 123 – «Комп'ютерна інженерія», спеціалізацій: 123-01 «Комп'ютерні системи і мережі»; 123-02 «Системне програмування»; 123-03 «Спеціалізовані комп'ютерні системи».

Рысованый А.Н.

Р54 Системное программирование, Ч.1. Программирование в среде masm64 : учеб.-метод. пособие / А.Н. Рысованый. – Харьков : «Слово», 2017. – 108 с.– На рус. яз.

Рассмотрены вопросы использования новой среды программирования masm64 на языке ассемблер ml64, этапы создания программы на masm64, особенности использования отладчика x64Dbg и лабораторные работы с заданиями и примерами выполнения в среде masm64.

Предназначено для студентов специальности 123 – «Компьютерная инженерия», специализаций: 123-01 «Компьютерные системы и сети»; 123-02 «Системное программирование»; 123-03 «Специализированные компьютерные системы».

Ил. 39. Библиогр. 11 назв.

УДК 004.42
ББК 32.973-018.1

ВСТУПЛЕНИЕ

Язык Ассемблера – это символическое представление машинного языка [1–3]. Машинный (двоичный) код – это способ представления данных в виде кода, в котором каждый разряд принимает одно из двух возможных значений, обычно обозначаемых цифрами 0 и 1. Машинный код непосредственно использует элементы архитектуры компьютера. ***Системное программирование*** – это программирование с использованием элементов архитектуры компьютера (системных ресурсов) или для изменения возможностей системы. Другими словами, ***системные программирование*** – это программирование с использованием внутренних уровней вычислительной системы или для изменения возможностей операционной системы. ***Системный программист относится к высшей касте программистов****, который кроме программирования обязан знать не только архитектуру компьютера (от архитектуры микропроцессора до архитектуры всех элементов системной платы и всех устройств, которые к ней подключаются), а также операционную систему, которая управляет компьютером.*

Ассемблер позволяет писать короткие и быстрые программы. Язык ассемблера позволяет программисту пользоваться алфавитными мнемоническими кодами операций и по своему усмотрению присваивать символические имена регистрам компьютера и памяти, а также задавать удобные для себя схемы адресации (например, индексную или косвенную). Кроме того, она позволяет использовать разные системы счисления (например, десятичную или шестнадцатиричную) для представления числовых констант и дает возможность помечать строки программы метками с символическими именами, чтобы к ним можно было обращаться (по именам, а не по адресам) из других частей программы (например, для передачи управления).

Перевод программы на языке Ассемблера в выполняемый машинный код (вычисление выражений, раскрытие макрокоманд, замена мнемоники собственно машинных кодов и символьных адресов на абсолютные или относительные адреса) проводится ассемблером – программой-транслятором, которая и дала языку Ассемблера ее название [6–11].

Преимущества Ассемблера:

1. Язык Ассемблер – самый прогрессивный (т.к. использует все возможности компьютера), **постоянно** (иногда каждый год) **развивающийся**, а следовательно, современный язык программирования. В нем постоянно появляются новые наборы команд (новые технологии обработки данных: MMX, SSE1 – SSE4, AVX1, AVX2, AVX3, AVX3-1, AVX3-2, AVX512 (https://ru.wikipedia.org/ wiki/AVX), FMA1-4,…). Фирмы Microsoft и AMD

постоянно занимаются развитием яархитектуры микропроцессора, а использование этих возможностей возможно только с применением языка ассемблер.

2. На сегодняшний день только на языке Ассемблер можно легко (он для этого и предназначен) использовать как «старые» 16-, 32-разрядные, так и новые **64-разрядные регистры под ОС x64**. В фирме Microsoft большая группа программистов занимается усиленной разработкой и поддержкой отладчика x64Dbg для исследования кода программ на ассемблере.

3. Только язык Ассемблер может полно **раскрыть архитектуру** компьютера, т.к. *элементами архитектуры являются* (см. кн.: Преснухин Л.Н. Архитектура и проектирование микро-ЭВМ. Организация вычислительных процессов. Учеб. для ВТУЗов/Под ред. Л.Н. Преснухина. М.: Высш. шк., 1986. – 495 с.):

– типы обрабатываемых данных и средства их представления;
– структуры и системы команд (микрокоманд);
– структуры памяти и средства их адресации;
– структуры микропроцессоров (МП) и их программные модели.

Из перечисленных элементов первые два на 100%, а последние два – на 50% можно раскрыть только применяя Ассемблер. Ассемблер ближе всех приближен к машинному коду, что позволяет раскрыть все связи в архитектуре компьютерной системы. Поэтому, использование Ассемблера при рассмотрении архитектуры является жизненно необходимым условием (а иначе – как увидеть все регистры и управлять ими?!).

4. **Реинжиниринг программ** (реверсное программирование) возможен только на Ассемблере (т.е. там, где не известен исходный код). Ассемблер позволяет не только разобраться в исполняемом (exe) файле, но и изменить его вплоть до полной замены! А это и взлом информации, что в целях защиты государства имеет не только военное значение. И любопытство еще никто не отменял!

5. Внесение в программу кода **с антиотладочными приёмами** на уровне системы команд микропроцессора возможно только на Ассемблере.

6. **Прошивки BIOS** написаны все еще на Ассемблере (хотя настойчивые попытки уже были). Но а если необходимо хотя бы поинтересоваться: как устроен BIOS, … то снова только Ассемблер.

7. **Параллельная обработка** данных возможна только с применением ассемблерных команд. В архитектуре МП регистры YMM имеют размерность по 256 разрядов. А в 2013 году выпущен Manual, где 32 регистра ZMM имеют разрядность 512 бит. Быстродействие программ на ассемблере по обработке массивов данных с использованием этой технологии просто восхищает. Использовать регисры MMX (1997 г.), XMM (1999 г.), YMM (2008 г.), ZMM (2017) наиболее просто на Ассемблере. А «работать» с ними и увидеть их можно в отладчике и тоже на Ассемблере!

8. **Безопасное программирование** (проверка первых трех байтов

(проверка на наличие ассемблерной команды jmp) – для случая с одним из вариантов перехвата функций; подсчет контрольной суммы специально предназначенной для этого ассемблерной командой и др.) возможно только на Ассемблере.

И известные преимущества…

9**. Компактный исполнительный код**. Это видно в любом отладчике кода. На языках высокого уровня код имеет очень большую сегментацию (увеличение длины машинного кода и, как следствие – увеличение задействованной памяти компьютера под эту программу), что тоже видно в отладчике и снова – на языке Ассемблера. И совсем не верно, что, на C++ можно написать программу с одной функцией MessageBox и ее размер будет равен размеру ассемблерной программы. Если программист на ассемблере применил параметры компиляции, то такая программа в 4 раза меньше.

10. **Быстрота выполнения программы**.

В интернете сравнивают быстродействие C++ (http://i-programist.blogspot.ru/2012/05/blog-post.html) и ассемблера для обработки массива чисел. Но при этом используют основные команды ассемблера для работы с целыми числами выпуска 1979 года! А для таких целей надо использовать или строковые команды, или старые команды MMX (1997 г.), или уже AVX (2008 г.) (https://ru.wikipedia.org/wiki/AVX). В последнем случае это сравнение даже не этично. А немного подождать выхода микропроцессора (http://news.online.ua/689465/protsessory-skylake-dlya-pk-ne-budut-podderzhivat-avx-512/) с AVX-512 (команды уже опубликованы в мануале за 2013 г. по ссылке http://www.bagnet. org/news/tech/283376) для одновременной обработки вектора в 512 бит (это 64 байта одновременно) и сравнить. Впечатляет!

В качестве обобщения преимуществ ассемблера можно отметить, что этот язык используется для непосредственного управления операционной системой и для прямого доступа к аппаратуре (системные возможности). Это означает, что на ассемблере пишут оптимальные драйверы и небольшие утилиты, прошивки BIOS, загрузчики ядра ОС, “движки” игрушек, вирусы, компиляторы и многое другое. Ассемблер служит инструментом для связи с нетрадиционными внешними устройствами. Также он необходим при оптимизации критических блоков в прикладных программах с целью повышения их быстродействия.

Из приведенных преимуществ языка Ассемблера перед другими языками программирования можно сделать однозначный вывод: ***ассемблер – самый основной язык системного программирования****. Не знание Ассемблера свидетельствует только об ущербности программиста, который не может использовать богатейшие возможности этого языка. Использование других языков в системном программировании – от ленности ума и коммерческой целесообразности (так легче жить).*

Профессиональный системный программист, который не владеет Ассемблером похож на инвалида, который программирует, но не полностью используя возможности системы. Тогда это программист, но не полностью системный, а следовательно, непрофессиональный!..

1. ЭТАПЫ СОЗДАНИЯ ПРОГРАММЫ НА ЯЗЫКЕ АССЕМБЛЕРА

Ассемблер может создавать листинг программы с номерами строк, адресами переменных, операторами и таблицей перекрестных ссылок символов и переменных. Наряду с ассемблером используется программа, называется компоновщиком (linker), которая объединяет отдельные файлы, созданные ассемблером, в единую исполняемую программу.

При изучении языка ассемблера и разработке реальных программ на этом языке под операционную систему (ОС) Windows процесс распределения ячеек памяти для сохранения кодов команд и чисел выполняется автоматизировано [4–5].

Этапы создания программы на языке ассемблера такие:

- подготовка (или внесения изменений) исходного текста программы;
- ассемблирование программы (получение объектного кода);
- компоновка программы (получение исполнительного файла программы);
- отладка программы (исправление ошибок).

Обычно эти этапы циклически повторяются, поскольку при обнаружении ошибок на всех этапах приходится возвращаться к первому этапу и вносить изменения в текст программы для исправления ошибок.

Ассемблеры бывают двух типов: однофазные и двухфазные.

Однофазные ассемблеры могут обрабатывать только такие программы, в которых символы появляются в поле названия до того, как на них дается ссылка в поле операндов.

Трансляция программы двухфазным ассемблером проводится в два этапа: в первой фазе трансляции ассемблер последовательно считает каждое предложение начальной программы, частично ее транслирует и строит полную таблицу символов; во второй фазе ассемблер заканчивает трансляцию программы, используя таблицу символов первой фазы как входную информацию.

Для обоих типов ассемблеров символ, записанный в поле операндов, обязательно должен быть определен в поле названия, иначе будет сообщение об ошибке.

1.1. Подготовка текста программы на masm64

Текст программы на языке ассемблера записывается в один или несколько текстовых файлов. Имена файлов и их расширения могут быть любые, но принято использовать расширение *.asm. Эти файлы являются текстовыми, их можно подготовить с помощью любого текстового редактора, например блокнота или Notepad++, и хранить в виде обычных файлов в формате ASCII. Использование Word нежелательно, потому что такой текст содержит большое количество служебных символов, которые являются невидимыми и приводят к ошибкам трансляции.

В связи с тем, что Notepad++ имеет большие возможности по подготовке текста программ, например нумерация строк, подсветка синтексиса, то пользование этой средой более предпочтительно.

Структура программы под Win64 может быть такой:

```
    title первая программа на masm64
    OPTION DOTNAME ; включение и отключение функций ассемблера
.DATA               ; директива начала сегмента данных
…
.CODE           ; директива начала сегмента команд
    WinMain proc ;
    …   ; код программы
    ret             ; выход из ОС Windows
    WinMain endp
    end
```

Директива TITLE – это заголовок (титул) программы. Его часто может и не быть.

Опция OPTION DOTNAME – включение и отключение API-функций ассемблера, если они применяются в программе.

WinMain proc – имя программы. Это имя может быть любым, но оно должно повторяться в пакетном (командном) bat-файле при линковании программы как точка входа в программу.

Наиболее часто используемые варианты ассемблирования программы:

– через bat-файл;

– через среду masm64.

Достоинство запуска через пакетный (командный) bat-файл:
– можно минимизировать размер файла еще на этапе компиляции;
– применяют, если программа состоит из нескольких файлов.

Структура пакетного **bat-файла** программы с именем 1.asm в самом простом случае может быть такой:

```
Masm64\Bin\ml64 /c 1.asm
Masm64\Bin\link /subsystem:windows /entry:WinMain 1.obj
pause
del 1.obj
start 1.exe
```

При такой записи Masm64 должен быть расположен в корневом каталоге диска C:.

1.2. Асемблирование програми на masm64

Подготовленные тексты программы с расширением *.asm являются входными данными для программ, которые называются ассемблерами (например, программы Masm, Wasm, Fasm, Nasm). Задача программы ассемблера – превратить текст программы в форму двоичных команд, которые может выполнить МП. После ассемблирования получают файлы объектных модулей, имеющих расширение *.obj. Процесс ассемблирования чаще называют ***трансляцией.***

На момент написания этих методических указаний главный разработчик (как считаю я) среды masm64 **Стивен Хатчессон** еще не выпустил ни одной окончательной версии, хотя видно на сайте (www.masm32.com) в разделе «Форум», что работы усиленно ведутся. Но студенты кафедры вычислительной техники и программирования НТУ ХПИ программируют на ассемблере под x64 фирмы Intel. Для этой цели используется или непосредственная среда Visual Studio v10 и старше, или среда masm64, которую разработал Василий Сотников, но усовершенствовал и предоставил ее нам nickname Mikl (ссылка для скачивания masm64: http://blogs.kpi.kharkov.ua/v2/asm/soft/). Убежден, что в самое ближайшее время можно будет перейти на сборку Стива Хатчессона (Сидней, Австралия) в силу того, что у него существует поддержка обширных библиотек функций и будет дальнейшее развитие. Приятно, что основу библиотеки макросов у него составляет сборка Василия Сотникова.

Последние усовершенствования masm64, опубликованного на сайте http://blogs.kpi.kharkov.ua/v2/asm/soft/ поддерживают команды AVX. Об их использовании можно ознакомиться во 2-й части методических указаний.

Для простоты использования ассемблера Masm64 (www.masm32.com) его необходимо установить на системный диск C:/. Если поставить ассемблер Masm64 не в системный диск, а на другой логический диск, то в таком случае необходимо конкретно прописывать пути подключения библиотек расположения API-функций.

Среду Masm64 необходимо поставить в корневой каталог диска c:\. Это необходимо для того, чтобы в каждой программе не указывать пути до подключаемых библиотек.

Внешний вид редактора среды masm64 представлен на рис. 1.1, который открывается после вызова файла qeditor.exe, и располагается на c:\Masm64\.

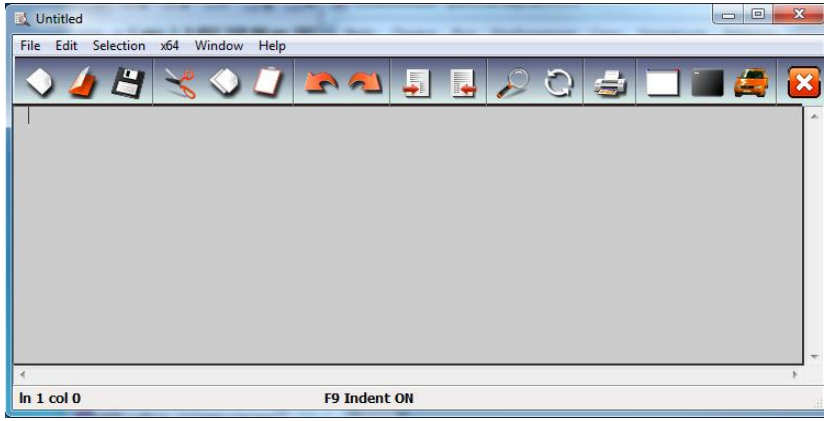

Рис. 1.1. Внешний вид редактора среды masm64

В этой среде пишется ассемблерная программа и сохраняется путем выбора кнопок меню File/ Save as. Необходимо указать имя латинскими буквами и обязательно расширение .asm.

Для получения exe-файла необходимо выбрать кнопку меню x64, а затем Compile&Link. После чего исполняющий exe-файл будет расположен в c:\Masm64\bin.

Программа ml64.exe с Masm64 выполняет трансляцию исходного текста программы в промежуточный объектный файл. Программа

link.exe выполняет компоновку объектного (объектных) файла (файлов) и библиотек в единую исполняемую программу.

Трансляцию файлов с расширением **.asm** можно выполнить из **командной строки** (если программа состоит из одного файла):

ml64 /c 1.asm

Параметр / C означает, что выполняется только ассемблирование. Вызов линкера link.exe пока не выполняется, так как его вызываем вручную позже.

В процессе трансляции текста программы выполняются такие действия:

1. Анализируются директивы условного ассемблирования, и в случае истинности указанных у них условий выполняются те или другие шаги.
2. Разворачиваются макросы.
3. Вычисляются константные выражения, при этом они замещаются вычисленными значениями.
4. Декодируются команды и операнды, которые не находятся в памяти. Например, декодируется команда mov eax, 2017, поскольку она не имеет операндов, расположенных в памяти.
5. Сохраняются сдвиги переменных в памяти относительно смещения в сегментах.
6. Сегменты и их атрибуты размещаются в объектном файле.
7. В объектном файле сохраняются перемещаемые адреса (relocatable addresses).
8. При необходимости создается файл листинга.
9. Непосредственно программе link.exe передаются некоторые директивы (например, INCLUDELIB і DOSSEG).

Опции компиляции ML.EXE:

ML [/ опции] список_файлов [/link опции _линкера]

/Bl<linker> – использовать альтернативный линкер;
/c – ассемблировать без линкования (использование внешнего линкера (например link.exe) для компоновки файлов);
/Cp – сохранить регистр идентификаторов пользователя;
/Cu – перевести все идентификаторы в верхний регистр;
/Cx – сохранить регистр publics, externs;
/D<name>[=text] – определить текст макроса;

/EP – выводить листинг препроцессора в стандартный поток выведения;
/F <hex> – установить размер стека (в байтах);
/Fe<file> – установить имя выполняемого файла;
/Fl[file] – генерировать листинг;
/Fm[file] – генерировать карту памяти;
/Fo<file> – имя объектного файла;
/Fr[file] – установить ограниченную информацию;
/FR[file] – установить полную информацию;
/I<name> – прибавить include-путь;
/link < опции линкеру и библиотеки >;
/nologo – запретить сообщение ML.EXE об авторском праве;
/Sa – максимизировать листинг;
/Sc – выдавать тайминги в листинг;
/Sf – выдавать листинг первого прохождения ассемблера;
/Sl<width> – установить ширину строки;
/Sn – подавить листинг таблицы символов;
/Sp<length> – установить длину страницы;
/Ss<string> – установить подзаглавие;
/St<string> – установить заглавие;
/Sx – составить список условных выражений ошибок;
/Ta<file> – ассемблировать не.ASM файлы;
/w – аналогично /W0 /WX;
/WX – расценивать предупреждение как ошибку;
/W<number> – установить уровень предупреждений;
/X – проигнорировать INCLUDE путь окружения;
/Zd – добавить номера строк в наладочную информацию;
/Zf – сделать все символы общедоступными;
/Zi – добавить символическую отладочную информацию;
/Zm – обеспечить совместимость с MASM 5.10;
/Zp[n] – установить выравнивание структур;
/Zs – выполнить только проверку синтаксиса.

Для трансляции и линкования *простых* программ **удобно** использовать среду **masm64**. Для этого необходимо запустить программу masm64 Editor, вставить или набрать свою программу, сохранить ее с расширением .asm и выбрать путь меню Project / Assemble & Link. Если ошибок не будет, то будет создан exe-файл.

1.3. Компоновка программы

После успешной трансляции текста ассемблерной программы результат в виде объектного файла передается компоновщику link.exe. Он выполняет объединение объектных модулей в один файл. Сегменты, определенные в программе, группируются в соответствии с инструкциями, которые содержатся в объектном файле. Вся информация о размещении сегментов записывается в заглавие выполняемого файла.

Результатом компоновки являются исполнительные файлы, которые имеют расширение *.exe.

В ассемблере masm64 для генерирования 64-разрядных EXE-файлов может быть использован одна из командных строк:

link /subsystem:windows /entry:WinMain 1.obj

В **большинстве случаев** для выполнения программ удобно создать пакетный (командный) bat-файл, например, с содержимым:

```
Masm64\Bin\ml64 /c 1.asm
Masm64\Bin\link /subsystem:windows /entry:WinMain 1.obj
pause
del 1.obj
start 1.exe
```

После того, как такой командный bat-файл создан, необходимо его выполнить (дважды нажать на него).

Структура программы (не только на ассемблере) определяется несколькими факторами:

– архитектурой микропроцессора;

– особенностями той операционной системы, под управлением которой эта программа будет выполняться;

– правилами работы выбранного компилятора – разные компиляторы ставят разные требования к начальному тексту программы.

MASM поддерживают ***упрощенную сегментацию***. Суть такой сегментации в том, что директивы .CODE и .DATA могут появляться в тексте программы несколько раз. Транслятор потом собирает код и данные вместе. Основной целью такого подхода является возможность приблизить в тексте программы данные к тем строкам, где они используются.

Параметры компоновщика приведены по ссылке https://msdn.microsoft.com/ru-ru/library/y0zzbyt4.aspx

1.4. Отладка программы для ml64.exe

Любая программа нуждается в отладке (исправление ошибок). На сегодняшний день для отладки программ на ассемблере, написанных для архитектуры x64 применяется программа-отладчик x64Dbg, которая загружает исполняющие файлы с расширением exe.

С помощью отладчика можно, например, пересматривать содержимое разных участков памяти, выполнять программу шаг за шагом, изменять программный код, сохранять изменения в exe-файле и др.

При написании программ **обязательно** использовать справочник MSDN (http://msdn.microsoft.com/ru-ru/library).

1.5. Редакторы текста

В качестве редактора текста при написании и отладки программ проще всего используют блокнот Notepad++. Этот редактор, как и некоторые другие, поддерживают подсветку синтаксиса, возможность запуска внешних приложений.

На начальном уровне можно использовать именно среду **masm64 (файл editor.exe**) или командную строку файлового менеджера FAR.

Но наиболее полными возможностями по ассемблированию программ с использованием ml64.exe владеет среда Visual Studio последних версий, потому что поддерживает все новые API-функции и новые ассемблерные команды.

При самостоятельной проработке учебного материала можно использовать программы с сайта ресурса **http://wasm.in/forum/**

Контрольные вопросы

1. Какие преимущества имеет ассемблер ml64 над другими языками программирования?
2. Какие этапы создания программы на языке ассемблеру?
3. Какие действия выполняются при трансляции программы?
4. Для чего выполняется компоновка программы и какие опции компоновки чаще всего используются?

2. ОТЛАДЧИК X64DBG

Основные возможности x64dbg

– открытый исходный код;
– интуитивный и знакомый пользовательский интерфейс;
– Си-подобный анализатор выражений;
– отладчик DLL и EXE файлов;
– IDA-подобная боковая панель с стрелками перескакиваний;
– IDA-подобное подсвечивание токенов, регистров и т.д.;
– символьный вид;
– контекстно-чувствительный просмотр регистров;
– настраиваемые цветовые схемы;
– динамическое распознавание модулей и строк;
– дамп памяти нескольких типов данных;
– динамический просмотр стека.

Документацию по использованию отладчика x64Dbg можно открыть по нажатию на клавишу F1 или выбрать Справка/Руководство.

Для отладки exe-файла необходимо загрузить файл в отладчик. При первоначальной загрузке файла необходимо выполнить: Файл/Открыть. При повторном открытии лучше использовать: Файл/Последние файлы.

Для исправления команды необходимо дважды нажать на левую клавишу мышки и в открывшемся окне осуществить необходимые действия. После чего нажать правую клавишу мышки и выбрать «Ассемблирование»

Для получения алгоритма (графа) выполнения программы необходимо нажать букву «G» или нажать на правую клавишу мышки и выбрать «Граф».

Обычные точки остановки

Для того, чтобы поставить *обычные точки остановки* (ordinary breakpoints), необходимо в окне дизассемблера использовать клавишу <F2> или двойное щелкание мыши во второй колонке окна кода. В этом случае команда окрашивающаяся в красный цвет. Этот вид остановки необходим для того, чтобы в точке остановки можно было бы проверить состояние регистров, переменных, стека. Вторичное нажатие на

клавишу <F2> в точке остановки или двойное щелкание мыши удаляют точку остановки.

Дамп памяти

Для вызова интересующего адреса памяти необходимо нажать на правую клавишу мышки и выбрать Перейти/Выражение/ и «Ввести адрес» или нажать правую клавишу мышки и выбрать «Перейти к адресу».

Контрольные вопросы

1. В чем состоят основные возможности x64dbg?
2. Как вызвать справку в отладчике?
3. Как осуществляется исправление команды в отладчике?
4. Как получить граф программы?
5. Как управлять контрольными точками в отладчике?

3. ПАКЕТНЫЙ ФАЙЛ

В операционных системах семейства Microsoft Windows простейшим средством автоматизации обработки файлов (и каталогов) служат так называемые командные (пакетные) файлы. В состав многих операционных систем входит командный процессор. Это программа, которая инициирует выполнение действий в ответ на команды, вводимые пользователем с клавиатуры (http://www.xmarks.com/s/site/www.rsdn.ru/article/winshell/ batanyca.xml).

BAT-файл (также известный как пакетный файл или батник) это текстовый документ с расширением .bat, в котором записаны команды, которые нужно выполнить с помощью командной строки. При запуске такого файла запускается программа CMD, которая считывает команды с данного файла и последовательно их выполняет (http://www.celitel.info/klad/nhelp/helpbat.php).

Основная область применения – автоматизация наиболее рутинных операций, которые регулярно приходится совершать пользователю компьютера: например, копирование, перемещение, переименование, удаление файлов; работа с папками; архивация и т.п.

Вирусы могут быть написаны в виде пакетного файла, известны также генераторы вирусов, являющиеся пакетными файлами.

С помощью BAT файла можно выполнять запуск программ, резервное копирование файлов, архивацию данных и многое другое.

Пример командного файла для запуска калькулятора и среды программирования masm64:

```
start calc
start Masm64/qeditor.exe
```

При программировании в среде masm64 bat-файл необходим, когда проект состоит из нескольких файлов.

Командная строка и команды

Для того чтобы запустить командный процессор необходимо:

1. Нажать на кнопку **Пуск**. На экран будет выведено главное меню.
2. Выбрать в главном меню пункт «Выполнить». На экран будет выведено диалоговое окно **Запуск программы**.
3. В поле Открыть ввести строку **cmd**.
4. Нажать на кнопку **OK**. На экран будет выведено окно командного процессора.

Для доступа к перечню команд командного процессора в открывшемся окне необходимо ввести команду **help**. Этот перечень не вмещается на один экран. Та же проблема возникает с текстом описания и других команд. Поэтому выдачу этого списка можно перенаправить в файл:

help > commands.txt

В этом случае, как один из вариантов, он запишется в c:\Users\...\

Для того чтобы получить описание конкретной команды, в качестве параметра команда **help** следует указать ее имя. Например, командная строка выводит на экран описание команды **for:**

help for

Для того чтобы сформировать текстовый файл с описанием команды **for**, надо ввести команду:

help for > for.txt

Список команд при выполнении команды help

Для получения сведений об определенной команде наберите HELP <имя команды>
ASSOC – вывод либо изменение сопоставлений по расширениям имен файлов;
ATTRIB – отображение и изменение атрибутов файлов;
BREAK – включение и выключение режима обработки комбинации клавиш CTRL+C;
BCDEDIT – задание свойств в базе данных загрузки для управления начальной загрузкой;
CACLS – отображение и редактирование списков управления доступом (ACL) к файлам;
CALL – вызов одного пакетного файла из другого;
CD – вывод имени либо смена текущей папки;
CHCP – вывод либо установка активной кодовой страницы;
CHDIR – вывод имени либо смена текущей папки;
CHKDSK – проверка диска и вывод статистики;
CHKNTFS – отображение или изменение выполнения проверки диска во время загрузки;
CLS – очистка экрана;
CMD – запуск еще одного интерпретатора командных строк Windows;
COLOR– установка цветов переднего плана и фона, используемых по умолчанию;
COMP – сравнение содержимого двух файлов или двух наборов файлов;
COMPACT – отображение и изменение сжатия файлов в разделах NTFS;
CONVERT – преобразование дисковых томов FAT в NTFS. Нельзя выполнить преобразование текущего активного диска;

COPY – копирование одного или нескольких файлов в другое место;
DATE – вывод либо установка текущей даты;
DEL – удаление одного или нескольких файлов;
DIR – вывод списка файлов и подпапок из указанной папки;
DISKCOMP – сравнение содержимого двух гибких дисков;
DISKCOPY – копирование содержимого одного гибкого диска на другой;
DISKPART – отображение и настройка свойств раздела диска;
DOSKEY – редактирование и повторный вызов командных строк создание макросов;
DRIVERQUERY – отображение текущего состояния и свойств драйвера устройства;
ECHO – вывод сообщений и переключение режима отображения команд на экране;
ENDLOCAL – конец локальных изменений среды для пакетного файла;
ERASE – удаление одного или нескольких файлов;
EXIT – завершение работы программы CMD.EXE (интерпретатора командных строк);
FC – сравнение двух файлов или двух наборов файлов и вывод различий между ними;
FIND – поиск текстовой строки в одном или нескольких файлах.
FINDSTR – поиск строк в файлах;
FOR – запуск указанной команды для каждого из файлов в наборе;
FORMAT – форматирование диска для работы с Windows;
FSUTIL – отображение и настройка свойств файловой системы;
FTYPE – вывод либо изменение типов файлов, используемых при сопоставлении по расширениям имен файлов;
GOTO – передача управления в отмеченную строку пакетного файла;
GPRESULT – отображение информации о групповой политике для компьютера или пользователя;
GRAFTABL – разрешение Windows отображать расширенный набор символов в графическом режиме;
HELP – вывод справочной информации о командах Windows;
ICACLS – отображение, изменение, архивация или восстановление списков ACL для файлов и каталогов;
IF – оператор условного выполнения команд в пакетном файле;
LABEL – создание, изменение и удаление меток тома для дисков;
MD – создание папки;
MKDIR – создание папки;
MKLINK – создание ссылок;
MODE –конфигурирование системных устройств;
MORE – последовательный вывод данных по частям размером в один экран;
MOVE – перемещение одного или нескольких файлов из одной папки в другую;

OPENFILES – отображение файлов, открытых удаленным пользователем в общей папке;
PATH – отображает или устанавливает путь поиска исполняемых файлов;
PAUSE – приостанавливает выполнение пакетного файла и выводит сообщение;
POPD – восстановление предыдущего значения активной папки, сохраненное с помощью команды PUSHD;
PRINT – вывод на печать содержимого текстового файла;
PROMPT – изменение приглашения в командной строке Windows;
PUSHD – сохранение значения активной папки и переход к другой папке;
RD – удаление папки;
RECOVER – восстановление данных, которые можно прочитать, с плохого или поврежденного диска;
REM – помещение комментариев в пакетные файлы и файл CONFIG.SYS;
REN – переименование файлов или папок;
RENAME – переименование файлов или папок;
REPLACE – замещение файлов;
RMDIR – удаление папки;
ROBOCOPY – улучшенное средство копирования файлов и деревьев каталогов;
SET – показывает, устанавливает и удаляет переменные среды Windows;
SETLOCAL – локализация изменений среды в пакетном файле;
SC – отображение и настройка службы (фоновые процессы);
SCHTASKS – выполнение команды и запуск программы по расписанию;
SHIFT – изменение положения (сдвиг) подставляемых параметров для пакетного файла;
SHUTDOWN – локальное или удаленное выключение компьютера.
SORT – сортировка ввода;
START – выполнение программы или команды в отдельном окне;
SUBST – назначение заданному пути имени диска;
SYSTEMINFO – вывод сведений о системе и конфигурации компьютера;
TASKLIST – отображение всех выполняемых задач, включая службы;
TASKKILL – прекращение или остановка процесса или приложения;
TIME – вывод и установка системного времени;
TITLE – назначение заголовка окна для текущего сеанса интерпретатора командных строк CMD.EXE;
TREE – графическое отображение структуры каталогов диска или папки;

TYPE – вывод на экран содержимого текстовых файлов;
VER – вывод сведений о версии Windows;

VERIFY – установка режима проверки правильности записи файлов на диск;

VOL – вывод метки и серийного номера тома для диска;
XCOPY – копирование файлов и деревьев каталогов;
WMIC – вывод сведений WMI в интерактивной среде.

Дополнительные сведения о программах приведены в описании программ командной строки в справке.

Синтаксис команд командного файла

В командном файле каждая команда занимает одну строку. Например, команда

attrib -S -H /D /S,

отображает скрытые файлы.

Однако, существует способ расположить одну команду на нескольких подряд идущих строк. Для этого в конце переноса строки следует поставить символ «крышка» «^» (http://philosoft-services.com/batniki.zhtml).

Правило синтаксиса команд командного файла состоит в том, что при указании параметров сначала указывается источник, а котом результат. Например, если необходимо переместить файл **masm64-1.asm** из каталога **x64-1** в каталог **x64-2**, необходимо ввести команду:

move x64-1\ masm64-1.asm x64-2

Сначала указывается то, что переименовать, а затем, во что переименовать.

А если необходимо переименовать файл **masm64-1.asm** в файл **masm64-2.asm**, то команда должна быть записана так:

ren masm64-1.asm masm64-2.asm

Пример запуска калькулятора и выполнение им команд:

```
:start                    1. Метка, создающая цикличность программы.
@echo off                 2. Вывод выполняющихся строк на экран
Cls                       3. Очистка экрана
title   Calc              4. Изменение заголовка на строку «Calc».
color 71   5. Изменение цвета шрифта и фона (тёмно-синий на светло-сером
echo  Enter the expression:   6. Вывод строки «Enter the expression».

set /p Exp=               7. Создание переменной Exp для хранения ввода
set /a Result=%Exp%  8. Вычисление результата
cls                       9. Очистка экрана
echo It is calculated    10. Вывод строки «It is calculated»
```

```
echo Your expression:%Exp%  11. Вывод «Your expression» и
значения Exp
echo Result:%Result%  12. Вывод строки «Result» и переменной Result
echo.                 13.
echo Press any key ... 14. Вывод строки «Press any key . . .»
pause > nul           15. Остановка выполнения до нажатия клавиши
goto start
```

Текущий каталог. Абсолютные и относительные пути

При работе с файловыми командами имеет большое значение понятие текущего каталога. Пример полного каталога:

c:\Masm64\bin\AVX64-1.asm.

Какой бы каталог ни оказался текущим в момент ввода команды, полный путь будет соответствовать одному и тому же файлу. Для относительного пути текущий каталог служит отправной точкой. Простейший случай относительного пути – имя файла. Это, что файл с таким именем, расположенный в текущем каталоге.

Для записи относительного пути к текущему каталогу существует условная запись «**.**» (точка). Для записи относительного пути к каталогу, в котором содержится текущий каталог, существует условная запись «**..**» (две точки). Команда

```
copy *.* .\mainBox
```

копирует все файлы из текущего каталога в каталог mainBox, расположенный рядом с ним.

Комментирование командного файла и его выдачи. Команды echo и rem

Для оформления комментариев используется команда **rem**. Это фиктивная команда, которая не предполагает выполнения каких бы то ни было действий.

Пример оформления командного файла:

```
rem ******************************************************
rem Формирование файлов справки по командам copy и move
rem ******************************************************
  rem Формируем файлы справки
  help copy > copy.help
  help move > move.help
rem Создаем каталог для хранения файлов справки
md msdos-help
    rem Перемещаем файлы справки в подготовленный каталог
    move *.help msdos-help
```

Пустые строки позволяют сделать командный файл более читабельным.

Для вывода сообщений применяется команда echo.

Выдачу команд на экран можно отключить с помощью команды **@echo off**.

Передача командному файлу параметров

Механизм обработки параметров заключается в следующем. Если при запуске командного файла пользователь указал несколько параметров, то в тексте командного файла первый из них обозначается записью **%1**, второй записью **%2**, третий записью **%3** и т.д.

В качестве примера можно создать командный файл, который сначала формирует справку с описанием заданной пользователем команды, а потом загружает его для просмотра в блокнот. При очередном запуске командного файла он сообщает, какая именно команда нас интересует на этот раз.

Пример оформления командного файла с именем **help1.bat**:

```
@echo off
rem Формируем файл с описанием команды,
rem имя которой передано параметром
help %1 > help.tmp
  rem Загружаем файл описания в редактор Блокнот
  notepad help.tmp
```

Теперь, чтобы загрузить в блокнот описание команды, например, **dir**, необходимо ввести команду следующим образом.

```
show-help.bat dir
```

Проверка условий и выбор вариантов. Команды if и goto

Команда if позволяет выделять в командном файле группы команд, которые выполняются или не выполняются в зависимости от определенных условий.

Проверка условия – необходимая мера при создании командных файлов, использующих параметры. Перед тем, как начинать работу, командный файл, вообще говоря, должен удостовериться в том, что ему передан корректный набор параметров. Кроме того, если командный файл удаляет, перемещает или перезаписывает какие-либо данные, то при некорректных параметрах он может нанести ущерб.

Для проверки существования файла в языке команд MS-DOS предусмотрено специальное слово exist:

```
if exist %filename%.exe del %filename%.exe
```

4. ЛАБОРАТОРНЫЕ РАБОТЫ

ЛАБОРАТОРНАЯ РАБОТА 1
ПРОГРАММИРОВАНИЕ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ

Цель занятия: приобрести практические навыки составления, налаживания и выполнения программ, написанных языком ассемблера для программирования арифметических операций под МП платформ x86/64.

В отчете представить:

– номер и название лабораторной работы;

– задание;

– текст программы с комментариями к каждой строке программы;

– результат выполнения программы (скриншоты упрощенных окон (MessageBox) и отладчиков);

– алгоритм программы (клавиша «G» в окне отладчика);

– особенности выполнения, где описать общий алгоритм (последовательность) выполнения и особенности программы (на что необходимо с вашей точки зрения уделить внимание).

Постановка задачи

Выполнить задание в двух вариантах:

1-й вариант. Выполнить задание под Win32.

2-й вариант. Выполнить задание под Win64 **(**из материалов лекции**).**

Вывести результат через функцию MessageBox.

Задание

Согласно номеру студента в группе выбрать вариант задания и написать на ассемблере программу вычисления одного из выражений:

1. $2d/c - ad$;	8. $2ab - 8c/b$;	15. $d/3c - 15ac$;	22. $2b - e/22b$;
2. $2a - e/2c$;	9. $8d - 9d/c$;	16. $2d/3c - 16cd$;	23. $ab/3a - 23c$;
3. $b/a + c/a$;	10. $3e/b + 10c$;	17. $2a/7 - d/17e$;	24. $d/4a + 24d/c$;
4. $d/b - d/4c$;	11. $2d/3b - 11c$;	18. $2ab - 18c/d$;	25. $e/8b + 25ac$;
5. $e/3c + ac$;	12. $3b - 12c/d$.	19. $8d/b - 19d/c$;	26. $4d/4a - 26cd$;
6. $2d/c - ad$;	13. $cb/3a - 13a$;	20. $3d/b + 20e$;	27. $8a/7 - d/27b$;
7. $2a/7 - c/7e$;	14. $e/4b - d/14c$;	21. $2d/a - 21bd$;	28. $4ac - 28c/b$,

где a, b, c, d, e, f – числа.

Примеры решения

Решение на **masm32**

Программа 4.1.1. Решение уравнения ab/c на masm32:

```
; Решение уравнения ab/c на masm32
.686
.model flat, stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
count PROTO arg_c:DWORD, arg_a:DWORD, arg_e:DWORD
.data
_a1 dd 7
_b1 dd 96
_c1 dd 19
_res1 dd ?
_title db "Лабораторная работа №1. Арифметические операции",0
strbuf dw ?,0
_text db "masm32. Вывод результата ab/c через
MessageBox:",0ah,"Результат: %d - целая часть",0ah, 0ah,
"Автор: Рысованый А.Н.,  каф. ВТП,НТУ ХПИ",0
.code
count proc arg_a:DWORD, arg_b:DWORD, arg_c:DWORD
mov eax,arg_a
mul arg_b   ; edx,eax
div arg_c
mov _res1,eax
ret
count endp

start:
invoke count, _a1, _b1, _c1
invoke wsprintf, ADDR strbuf, ADDR _text, _res1
invoke MessageBox, NULL, addr strbuf, addr _title, MB_ICONINFORMATION
invoke ExitProcess, 0
end start
```

В приведенной программе не обязательно в именах переменных ставить знак подчеркивания. Такие подчеркивания необходимы, если общая программы состоит из нескольких файлов.

Результат выполнения программы приведен на рис. 4.1.1.

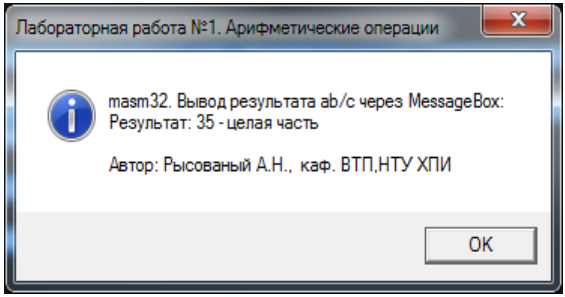

Рис. 4.1.1. Результат выполнения программы на masm32

Программа 4.1.2. Решение уравнения 8d – 9d/c на masm32:

```
; Решение уравнения 8d – 9d/c   на masm32
.686
.model flat, stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
firstfunc PROTO _const1:DWORD,_d1: DWORD,_const2:DWORD,
_d2:DWORD, _c1:DWORD
.data   ;8d – 9d/c
const1 dd 8
d1 dd 2
const2 dd 9
d2 dd 1
c1 dd 3
_temp1 dd ?,0
_title db "Лабораторная работа №1. Арифм. операции",0
strbuf dw ?,0
_text db "masm32. Вывод результата ab/c через
MessageBox:",0ah,"Результат: %d — целая часть",0ah, 0ah,
"Автор: Рысованый А.Н., каф. ВТП,НТУ ХПИ",0

.code
start:
invoke firstfunc, const1,d1,const2,d2,c1
invoke wsprintf, ADDR strbuf, ADDR _text, _temp1
```

```
invoke MessageBox, NULL, addr strbuf, addr _title,
MB_ICONINFORMATION
        invoke ExitProcess, 0

firstfunc proc _const1:DWORD,_d1:DWORD,_const2:DWORD,_d2:DWORD,
_c1:DWORD
        mov eax, _const1   ; загрузка числа const1 dd 8
        mul _d1            ; 8d
        mov _temp1, eax    ;
        mov eax, _const2   ;
        mul _d2            ;
        div _c1
        sub _temp1, eax
        ret
        firstfunc endp

        END start
```

Результат выполнения программы приведен на рис. 4.1.2.

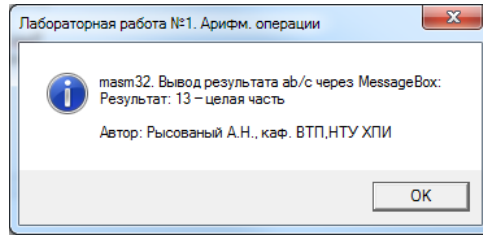

Рис. 4.1.2. Результат выполнения программы

Решение на **masm64**

Вариант 1

Программа 4.1.3. Решение уравнения ab/c на masm64:

```
; Решение уравнения ab/c на masm64
OPTION DOTNAME ; включение и отключение функций ассемблера
include temphls.inc   ; invoke и другие высокоуровневые инструкции
include win64.inc     ; константы
include kernel32.inc  ; прототипы функций из библиотеки kernel32.dll
includelib kernel32.lib
include user32.inc
includelib user32.lib
OPTION PROLOGUE:none  ; пролог функций
```

```
        OPTION EPILOGUE:none    ; эпилог функций
        count PROTO arg_a:QWORD, arg_b:QWORD, arg_c:QWORD
.data
        _a1 dq 7
        _b1 dq 96
        _c1 dq 19
        _res1 dq 0
_title db "Лабораторная работа №1. Выполнение арифметических операций",0
        strbuf dq ?,0
_text db "Вывод результата ab/c через MessageBox:",0ah,
 "Результат: %d - целая часть",0ah,0ah,
"Автор: Рысованый А.Н., каф. ВТП, НТУ ХПИ",10,
9,"Сайт:  http://blogs.kpi.kharkov.ua/v2/asm/",0

.code
        count proc arg_a:QWORD, arg_b:QWORD, arg_c:QWORD
        mov rax,rcx  ; arg_a
      mul rdx        ; rdx:rax
      div r8
        ;leave ; Восстановление кадра стека.
        ret
        count endp
WinMain proc
sub rsp,28h; выравнивание стека 28h=32d+8;  8 - возврат
mov rbp,rsp ; сохранение выравненного значения стека
        invoke count,_a1,_b1,_c1
        invoke wsprintf, ADDR strbuf, ADDR _text, rax
invoke MessageBox, NULL, addr strbuf, addr _title, MB_ICONINFORMATION
        invoke ExitProcess,0
        WinMain endp
        end
```

Для получения исполняющего файла с расширением .exe простых программ можно воспользоваться текстовым блокнотом (редактором) среды masm64. Для этого запускаем файл qeditor.exe, который должен находится в c:\Masm64\.

В этой среде пишется ассемблерная программа и сохраняется путем выбора кнопок меню File/ Save as. Необходимо указать имя латинскими буквами и обязательно расширение .asm.

Для получения exe-файла необходимо выбрать кнопку меню x64, а затем Compile&Link. После чего исполняющий exe-файл будет расположен в c:\Masm64\bin.

Результат выполнения программы приведен на рис. 4.1.3.

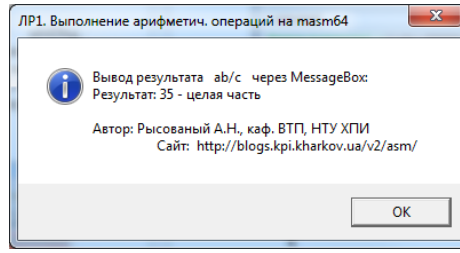

Рис. 4.1.3. Результат выполнения программы на masm64

Для отладки exe-файла запускается отладчик x64Dbg, расположенный по адресу c:\x64dbg\release\x64\ и загружается исследуемая программа с расширением .exe. Если необходимо выполнить эту программу, то следует нажать F9 или нажать на кнопку Отладка и выбрать Выполнить/F9. Внешний вид отладчика x64Dbg с загруженной программой представлен на рис. 4.1.4.

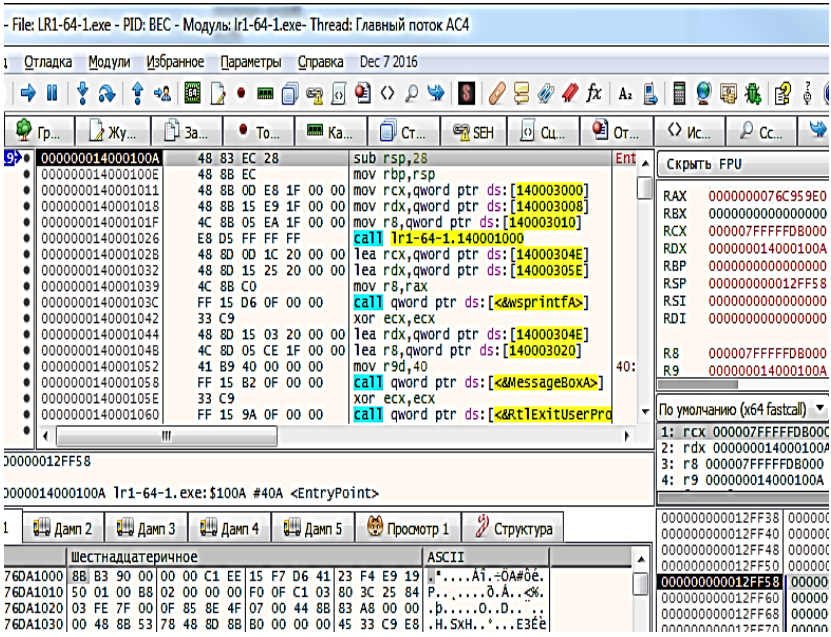

Рис. 4.1.4. Внешний вид отладчика x64Dbg

Первые две команды в окне CPU выравнивают стек.

Следующая группа команд

```
mov rcx,qword ptr ds:[140003000]
mov rdx,qword ptr ds:[140003008]
mov r8,qword ptr ds:[140003010]
```

относится к нашей подпрограмме (функции) с именем count, которая вызывается строкой кода

```
invoke count,_a1,_b1,_c1
```

Так как в подпрограмме (функции) count передаются три параметра, то первый из них передается в регистре rcx, второй – в rdx, третий – в r8. Чтобы увидеть в окне дампа памяти содержимое выделенных ассемблером нашей программе ячеек памяти необходимо в окне программы нажать правую клавишу мышки и выбрать Перейти/Выражение (рис. 4.1.5).

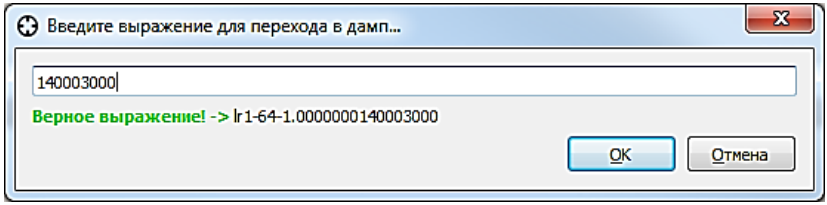

Рис. 4.1.5. Окно отладчика x64Dbg

В это окно необходимо ввести адрес ячеек памяти, начальный номер которых в исследуемой программе встречается первым. Можно посмотреть любой адрес ячейки памяти, но, как правило, нас интересуют те ячейки памяти, которые задействует исследуемая программа.

Для того, чтобы пошагово отрассировать команды последовательно нажимаем на кнопку F7 (рис. 4.1.6).

При наведении курсора мышки на команду call появляется окно с раскрытием этой функции. А для того, чтобы попасть во внутрь этой функции необходимо нажать на кнопку F7. В случае если нажать на кнопку F8, то функция будет сразу выполнена и во внутрь функции мы не попадем.

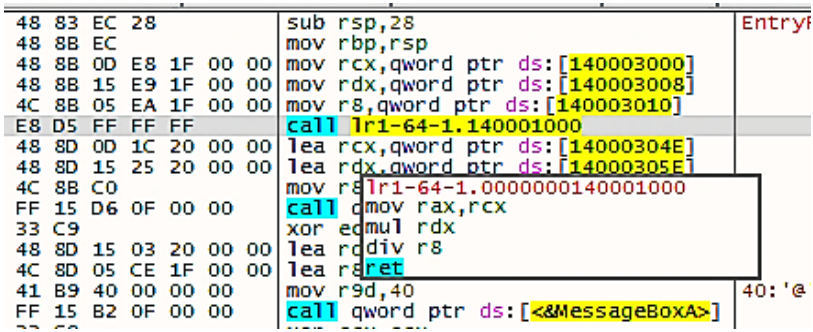

Рис. 4.1.6. Окно отладчика x64Dbg с кодом программы

Следовательно, для пошагового выполнения каждой строки необходимо использовать или F7 (для выполнения строки кода и вхождения в процедуру), или F8 (для выполнения строки кода и не вхождения в процедуру).

В рассматриваемой программе обрабатываются целые числа размерностью DQ и параметры передаются в процедуре. Следовательно, первый параметр передается в регистре rcx (arg_a), второй – rdx(arg_b), третий – r8 (arg_c). Если был бы четвертый параметр, то он бы передавался через регистр r9.

Операции умножения и деления используют регистры rdx:rax.

Команда div r8 сохраняет результат в регистре rax, который функцией

invoke wsprintf, ADDR strbuf, ADDR _text, rax

преобразуется в символы и сохраняется в переменной strbuf.

Не следует забывать, что при отладке приходится часто перезагружать в отладчике исполняемый файл программы. В этом случае, чтобы снова не отслеживать все просмотренные строки кода кнопкой «F7» (или «F8»), можно поставить клавишей «F2» контрольную точку на последней проверенной строке, затем перегрузить программу и нажать кнопку «F9».

Вариант 2

```
OPTION DOTNAME ; включение и отключение функций ассемблера
include temphls.inc   ; invoke и другие высокоуровневые инструкции
include win64.inc     ; константы
include kernel32.inc  ; прототипы функций из библиотеки kernel32.dll
```

```
        includelib kernel32.lib
        include user32.inc
        includelib user32.lib
        OPTION PROLOGUE:none  ; пролог функций
        OPTION EPILOGUE:none    ; эпилог функций
.data
        _a1 dq 7
        _b1 dq 96
        _c1 dq 19
        _res1 dq 0
        _title db "ЛР1. Выполнение арифметич. операций на masm64",0
        strbuf dq ?,0
_text db "Вывод результата  ab/c  через MessageBox:",0ah,
   "Результат: %d - целая часть",0ah,0ah, "Автор:              каф. ВТП, НТУ
ХПИ",0

.code

   count proc arg_a:QWORD, arg_b:QWORD, arg_c:QWORD
   push rbp
   mov rbp,rsp ; сохранение выравненного значения стека
   mov arg_a,rcx
   mov arg_b,rdx
   mov arg_c,r8

      mov rax,arg_a
      mul arg_b   ; rdx:rax
      div arg_c
      mov _res1,rax
        leave
        retn
        count endp

   WinMain proc
sub rsp,28h; выравнивание стека 28h=32d+8;  8 - возврат
mov rbp,rsp ; сохранение выравненного значения стека

        invoke count,_a1,_b1,_c1
invoke wsprintf,ADDR strbuf,ADDR _text,_res1
invoke MessageBox, NULL, addr strbuf, addr _title, MB_ICONINFORMATION
        invoke ExitProcess, 0
        WinMain endp
        end
```

ЛАБОРАТОРНА РОБОТА 2
ПРОЦЕДУРЫ С ПАРАМЕТРАМИ

Цель занятия:

- углубить и закрепить знание по архитектуре МП платформ x86/64 и навыки его программирования;

- приобрести практические навыки составления, налаживания и выполнения программ, написанных языком ассемблера и оформленных в виде процедур с параметрами под МП платформ x86/64 в средах программирования masm32 и masm64.

В отчете представить:

– номер и название лабораторной работы;

– задание;

– текст программы с комментариями к каждой строке программы;

– результат выполнения программы (скриншоты упрощенных окон (MessageBox) и отладчиков);

– алгоритм программы (клавиша «G» в окне отладчика);

– особенности выполнения, где описать общий алгоритм (последовательность) выполнения и особенности программы (на что необходимо с вашей точки зрения уделить внимание).

Постановка задачи

Выполнить задание в двух вариантах:

1-й вариант. Выполнить задание под **Win32.**

2-й вариант. Выполнить задание под **Win64.**

Вывести результат через функцию MessageBox.

Методические указания

При выполнении задания 2 под masm64 можно записать массив целочисленных чисел как параметры процедуры. Но в этом случае нельзя использовать приращение адреса, т.к. параметры передаются сначала через целочисленные регистры rcx, rdx, r8, r9, а потом ячейки стека. И не получается обрабатывать числа массива в цикле — теряется смысл массива. Целесообразнее использовать только другие числовые параметры, например, если есть число ограничения выбора или количество циклов обработки и т.д.

Задание №1

Согласно номеру студента в группе выбрать вариант задания и написать на ассемблере программу вычисления одного из выражений:

1. ac + b/d + f/e;
2. f/e - b/d – a/c;
3. b/d + f/e - ca;
4. db – hc +a/e+f;
5. a/c + bd - efg;
6. a/b/c- d/e + f;
7. a/d + c/b +efg;
8. ef + d/c - ab;
9. a/b-cd/e+f/g;
10. a + b/c/d + efg;
11. abc - de/fg;
12. a/b + cd + e/fg;
13. ab + cd/ef - g;
14. abcd - ef/g;
15. g/f + e/d - cb/a;
16. gf/e + dc/b - a;
17. gf + e/d/c + ba;
18. g + fe/d/cb - a;
19. gf/e+dc/b + a;
20. g/f + (ed)/c+b/a;
21. 2d/a – 21bc+f;
22. 4b – a/22c + de;
23. ab/4e – 16cde;
24. d/ba – cd/e/f;
25. e/8b + acdef;
26. 4d/a – cd+ bef;
27. 8a/b + cd – (e/f)g;
28. 4abc – e/f - d,

где *a, b, c, d, e, f, g* – числа

Задание №2

1. Заданы натуральные числа $a, a_2...a_n$. Указать те из них, в которых остаток от деления на M будет равняться L ($0 <= L <= M - 1$)

2. Задан масив A с числом элементов больше 7. Написать программу определения количества элементов массива A, которые удовлетворяют условию $L \geq A_i \geq M$, де $L = 9$ та $M = 30$.

3. Задан масив A с числом элементов больше 7. Написать программу формирования массива B из последних 13 элементов массива A, которые равняются нулю.

4. Задан масив A с числом элементов больше 8. Написать программу формирования массива B из элементов массива A, которые удовлетворяют условие $A_i \leq E$ при $E = 12$.

5. Задан масив A с числом элементов больше 8. Написать программу определения суммы и количества элементов массива A, которые удовлетворяют условие $A_i \leq E$ при $E = -13$.

6. Написать программу определения количества пар элементов, которые удовлетворяют условие $A_i \leq B_i$.

7. Заданы массивы A и B с числом элементов больше 7. Написать программу формирования массива C по такому правилу: если $A_i - B_i \leq 0$, то $C_j = A_i$.

8. Задан масив A с числом элементов больше 8. Написать программу определения максимального из отрицательных элементов массива A.

9. Задан масив A с числом элементов больше 9. Структура массива A такая: $X1$, $Y1$; $X2$, $Y2$; … . Написать программу определения количества пар, для которых выполняется условие $X_i < Y_i$.

10. Задан масив A с числом элементов больше 7. Написать программу формирования массива B из элементов массива A, в которых биты 0, 2 и 5 имеют нули.

11. Задан массив A с числом элементов больше 8. Написать программу определения суммы элементов массива A, для которых биты 2 и 10 совпадают.

12. Задан массив A с числом элементов больше 8. Структура массива A такая: $X1$, $Y1$; $X2$, $Y2$; … . Написать программу определения минимального X_i .

13. Задан массив A с числом элементов больше 8. Написать программу определения минимального из положительных элементов массива A.

14. Заданы массивы A и B. Написать программу формирования массива C по такому правилу: если $A_i + B_i > 0$, то $C_j = B_i$.

15. Заданы массивы A и B. Написать программу определения количества пар элементов, которые удовлетворяют условию $A_i > B_i$.

16. Задан массив A с числом элементов больше 8. Написать программу определения суммы и количества элементов массива A, которые удовлетворяют условию $A_i > E$ при $E = -10$.

17. Задан массив A с числом элементов больше 8. Написать программу формирования массива B из элементов массива A, которые удовлетворяют условие $A_i > E$ при $E = 5$.

18. Задан массив A с числом элементов больше 10. Написать программу формирования массива B из первых 8 положительных элементов массива A.

19. Задан массив A с числом элементов больше 10. Написать программу определения количества элементов массива A, которые удовлетворяют условию $L < A_i \le M$, де $L = 2$ та $M = 10$.

20. Задан массив A с числом элементов больше 8. Написать программу формирования массива C по такому правилу: если $A_i > B_i$, то $C_i = A_i + B_i$; иначе $C_i = A_i - B_i$.

Примеры решения

Решение на **masm32**

Программа 4.2.1. Решение уравнения ab + c/d + e - f на masm32:

```
; Решение уравнения ab + c/d + e - f на masm32. Процедура с параметрами
```

```
    .686
    .model flat, stdcall
    option casemap:none
    include \masm32\include\windows.inc
    include \masm32\include\kernel32.inc
    include \masm32\include\user32.inc
    includelib \masm32\lib\user32.lib
    includelib \masm32\lib\kernel32.lib
count PROTO arg_a:DWORD,arg_b:DWORD,arg_c:DWORD,
arg_d:DWORD, arg_e:DWORD, arg_f:DWORD

.data
        _a dd 2
        _b dd 3
        _c dd 21
        _d dd 5
        _e dd 10
        _f dd 4
        _res dd ?,0
        strbuf dd ?,0
_text db "Уравнение  ab + c/d + e - f",0ah,"Результат: %d",0ah,
 "Адрес переменной в памяти: %p",0ah,0ah, "Автор:  каф. ВТП, НТУ
ХПИ",0
        _title db "Лабораторная работа №2 (masm32)",0

.code
count proc arg_a:DWORD,arg_b:DWORD,arg_c:DWORD,arg_d:DWORD,
arg_e:DWORD, arg_f:DWORD
      mov eax,arg_a
      mul arg_b     ; edx,eax
      mov esi,eax   ; сохранение промежуточного результата
      mov eax,arg_c ;
      div arg_d   ;
      add eax,esi   ;
      add eax,arg_e  ;
      sub eax,arg_f  ;
      mov _res,eax
      ret
      count endp
start:
        invoke count,_a,_b,_c,_d,_e,_f
invoke wsprintf, ADDR strbuf, ADDR _text, _res, ADDR _res
invoke MessageBox, NULL, addr strbuf, addr _title,
MB_ICONINFORMATION
        invoke ExitProcess, 0
```

```
END start
```

Текст программы можно уменьшить, если повторяющийся фрагмент в начале программы записать в файл, например, win32main.inc и разместить его по адресу c:\masm32\include\. Содержимое файла такое:

```
.686
.model flat, stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
```

В результате текст программы будет следующим:

```
include c:\masm32\include\win32main.inc
count PROTO arg_a:DWORD,arg_b:DWORD, arg_c:DWORD,
arg_d:DWORD, arg_e:DWORD, arg_f:DWORD

.data
        _a dd 2
        _b dd 3
        _c dd 21
        _d dd 5
        _e dd 10
        _f dd 4
        _res dd ?,0
        strbuf dd ?,0
_text db "Уравнение  ab + c/d + e - f",0ah,"Результат: %d",0ah,
"Адрес переменной в памяти: %p",0ah,0ah, "Автор:      каф. ВТП, НТУ ХПИ",0
        _title db "Лабораторная работа №2 (masm32)",0

.code

count proc arg_a:DWORD,arg_b:DWORD,arg_c:DWORD,arg_d:DWORD,
arg_e:DWORD, arg_f:DWORD
        mov eax,arg_a
        mul arg_b   ; edx,eax
        mov esi,eax ; сохранение промежуточного результата
        mov eax,arg_c ;
        div arg_d   ;
        add eax,esi   ;
        add eax,arg_e  ;
        sub eax,arg_f  ;
```

```
        mov _res,eax
        ret
        count endp
start:
        invoke count,_a,_b,_c,_d,_e,_f
invoke wsprintf, ADDR strbuf, ADDR _text, _res, ADDR _res
invoke MessageBox, NULL, addr strbuf, addr _title, MB_ICONINFORMATION
        invoke ExitProcess, 0
        END start
```

Результат выполнения программы приведен на рис. 4.2.1

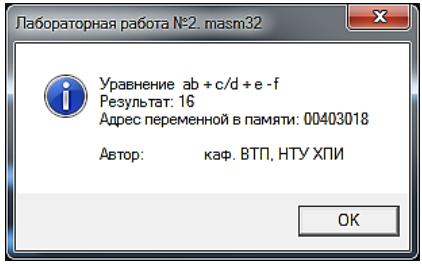

Рис. 4.2.1. Результат выполнения программы на masm32

Решение на **masm64**

Особенностью условия задания для masm64 является то, что число переменных превышает четыре – количество параметров, передаваемых в функции через регистры. Остальные параметры в виде целых чисел передаются через стек. Для действительных чисел первые четыре параметра передаются через XMM0L, XMM1L, XMM2L и XMM3L, остальное — через стек.

Вариант 1

Программа 4.2.2. Решение уравнения ab/c на masm64:

```
  ; Решение уравнения ab/c на masm64. Процедура с параметрами
  include win64a.inc  ; подключаемые библиотеки
count PROTO arg_a:QWORD,arg_b:QWORD,arg_c:QWORD,
arg_d:QWORD, arg_e:QWORD, arg_f:QWORD
.data
    _a1 dq 2
```

```
    _b1 dq 3
    _c1 dq 19
    _d1 dq 2
    _e1 dq 3
    _f1 dq 4
        _res1 dq 0
        _title db "ЛР2. Процедуры. masm64",0
        strbuf dq ?,0
_text db "Уравнение  ab + c/d + e - f",0ah,"Результат: %d",0ah,
"Адрес переменной в памяти: %p",0ah,0ah,
"Автор: Рысованый А.Н., каф. ВТП, НТУ ХПИ",10,
9,"Сайт:  http://blogs.kpi.kharkov.ua/v2/asm/",0

.code

count proc arg_a:QWORD, arg_b:QWORD, arg_c:QWORD,arg_d:QWORD,
arg_e:QWORD, arg_f:QWORD
        mov rax,rcx      ; arg_a
        mul rdx          ; rdx:rax  (ab)
        mov rsi,rax   ; сохранение результата предыдущего умножения
        mov rax,r8    ; занесение arg_c
        div r9           ; rdx: rax <- rax/r9
        add rax,rsi      ;
        add rax,[rbp+20h]  ; сложение  c  arg_e
        sub rax,[rbp+28h]  ; вычитание arg_f
        mov _res1,rax      ;
        ;leave ; Восстановление кадра стека.
        ret
        count endp

WinMain proc
sub rsp,28h; выравнивание стека 28h=32d+8; 32d x 8 = 256; 8 - возврат
    mov rbp,rsp ; сохранение выравненного значения стека

invoke count,_a1,_b1,_c1,_d1,_e1,_f1
invoke wsprintf, ADDR strbuf, ADDR _text, _res1, ADDR _res1
invoke MessageBox, NULL, addr strbuf, addr _title,
MB_ICONINFORMATION
        invoke ExitProcess,0
        WinMain endp
        end
```

Первой строкой кода есть строка

WinMain proc,

которая показывает, что программа оформлена ввиде процедуры с именем WinMain.

Затем выполняется выравнивание стека и сохранение выравненного значения стека:

```
sub rsp,28h; выравнивание стека 28h=32d+8;  8 - возврат
    mov rbp,rsp ; сохранение выравненного значения стека
```

Затем вызывается функция count с параметрами:

```
invoke count,_a1,_b1,_c1,_d1,_e1,_f1
```

Эта функция (процедура) в этой программе размещена в начале кода, в которой приведено название функции (процедуры), директива принадлежности – proc и размерность используемых аргументов. Особенностью процедур с параметрами является то, что целочисленные параметры передаются последовательно через регистры: rcx, rdx, r8, r9. Если количество параметров превышает шесть, то оставшиеся параметры передаются через стек. При этом, как и раньше следует помнить о выравнивании стека по границе, кратной 16 байтам. Следовательно, пятый параметр передается через [rbp+20h], шестой – через [rbp+28h], а следующие – с приращением 8h.

Поэтому, в регистре rcx передается параметр arg_a (строка mov rax,rcx).

В строке mul rdx происходит перемножение регистра rax и arg_b:QWORD.

Третий параметр передается через регистр r8.

Четвертый параметр передается через регистр r9.

Результат выполнения программы приведен на рис. 4.2.2.

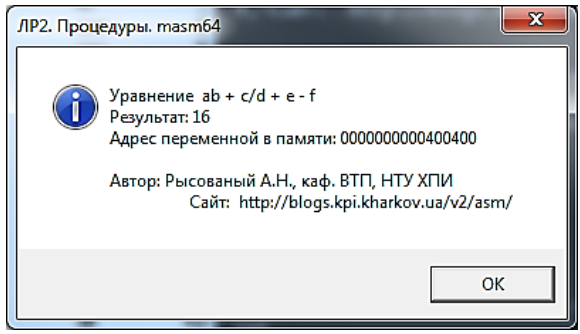

Рис. 4.2.2. Результат выполнения программы

В программе выводится два значения: результат – переменная_res1 и адрес ячейки памяти, где хранится этот результат – ADDR _res1. Но для вывода последнего используются символы форматирования %p.

Окно CPU отладчика x64Dbg перед трассировкой программы приведено на рис. 4.2.3.

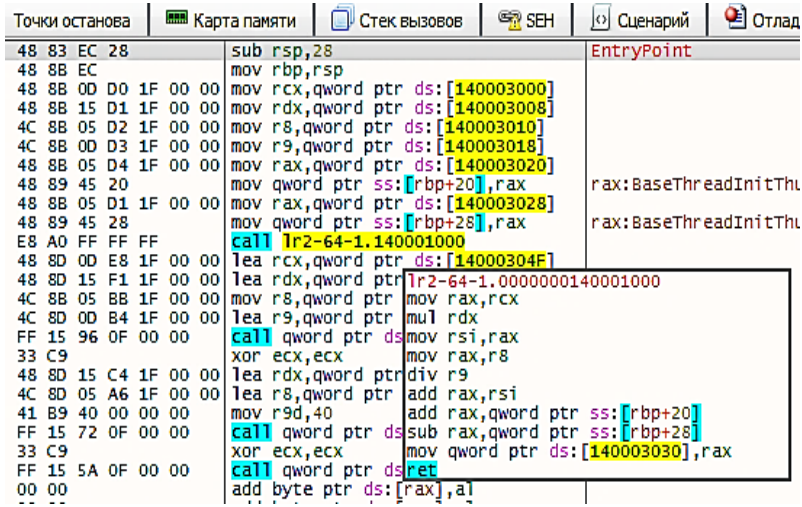

Рис. 4.2.3. Окно CPU отладчика x64Dbg перед трассировкой программы

В этом окне отладчика еще до трассировки уже видны номера используемых в программе ячеек памяти, регистры, в которые передаются параметры процедуры и последовательность их использования. При наведении мышки на строку кода с вызовом функции в отдельном окне показывается раскрытие этой функции.

Задание 2 – массив. Решение на **masm32.**

Массив состоит из чисел 15-25. Необходимо найти сумму чисел, 2-й и 7-й бит которых одинаковые.

Программа 4.2.3. Массив на masm32:

```
.686
.model flat,stdcall
option casemap:none ; отличие строчных и больших букв
```

```
        include \masm32\include\windows.inc ; файлы структур, констант .
        include \masm32\include\user32.inc ; файлы интерфейса .
        include \masm32\include\kernel32.inc ; системные функции.
        includelib \masm32\lib\user32.lib
        includelib \masm32\lib\kernel32.lib
.data
        _title db "ЛР2-2. Массивы. masm32", 0
info db "Массив с чисел 15 - 25.",0Ah,0Dh,
"Нахождение суммы чисел, 2-й и 7-й бит которых одинаковые", 0Ah,0Dh,
 "Сумма чисел : %d",0
        x dd 15,16,17,18,19,20,21,22,23,24,25
        N  equ ($-x)/4  ; N dd 11
        res dd 0
        buf db 8 dup(?)

.code
start:
        mov edx, 0
        mov ecx, N    ; счетчик числа элементов массива
        mov ebx, 0    ; начальный адрес первого элемента в массиве
a0:  mov eax, x[ebx] ; запись элемента массива в eax
bt eax, 2   ; выделяем 2-ой бит из eax (значение записывается во флаг
CF)
        jc a1             ; если CF=1, то перейти на метку a1
        jnc a2             ; если CF=0, то перейти на метку a2
a1: bt eax, 7; выделяем 7-ой бит из eax (значение записывается во флаг CF)
        jc a3                   ; если CF=1, то перейти на метку a3
        jnc K                   ; если CF=0, то перейти на метку K
a2: bt eax, 7; выделяем 7-ой бит из eax (значение записывается во флаг CF)
        jnc a3                        ; если CF=0, то перейти на метку a3
        jc K                               ; если CF=1, то перейти на метку K
a3: add res, eax     ; считаем сумму подходящих элементов массива
K: add ebx, type x   ; повторение цикла, пока не закончатся элементы
        dec ecx
        jnz a0
        invoke wsprintf,ADDR buf,ADDR info,res;
        invoke MessageBox, 0, addr buf, addr _title, MB_ICONINFORMATION
        invoke ExitProcess, 0
        end start
```

Результат выполнения программы приведен на рис. 4.2.4.

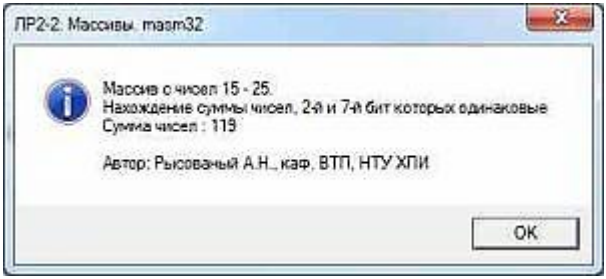

Рис. 4.2.4. Результат выполнения программы

Задание 2 – массив. Решение на **masm64.**

Программа 4.2.4. Массив на masm64 с передачей параметра в процедуре:

```
        title   массив на masm64 с передачей параметра
include win64a.inc  ; подключаемые библиотеки
        count  PROTO  arg_a:QWORD
.data
        _title db "ЛР2-2. Массивы. masm64", 0
info db "Массив из чисел 15 - 25.",0Ah,0Dh,
"Нахождение суммы чисел, 2-й и 7-й бит которых одинаковые.",
0Ah,0Ah,"Сумма чисел : %d",0Ah,0Ah,
    "Автор: Рысованый А.Н., каф. ВТП, НТУ ХПИ",10,
    9,"Сайт:  http://blogs.kpi.kharkov.ua/v2/asm/",0
    x1     dq 15,16,17,18,19,20,21,22,23,24,25
    len1   equ ($-x1)/type x1       ; len1 dd 11
    res dq 0
    buf dq 0,0

.code
    count  proc  arg_a:QWORD
      lea rbx,x1    ; начальный адрес первого элемента в массиве
a0:
      mov rax,[rbx] ; запись элемента массива в rax
bt rax,2 ; выделяем 2-ой бит из rax (значение записывается во флаг CF)
      jc a1         ; если CF=1, то перейти на метку a1
      jnc a2        ; если CF=0, то перейти на метку a2
a1:
bt rax,7; выделяем 7-ой бит из eax (значение записывается во флаг CF)
      jc a3           ; если CF=1, то перейти на метку a3
      jnc K           ; если CF=0, то перейти на метку K
a2:
```

```
bt rax,7; выделяем 7-ой бит из eax (значение записывается во флаг CF)
        jnc a3          ; если CF=0, то перейти на метку a3
        jc K            ; если CF=1, то перейти на метку K
a3: add res,rax   ; считаем сумму подходящих элементов массива
K: add rbx,type x1; повторение цикла, пока не закончатся элементы
        dec rcx
        jnz a0
        ret
        count endp

        WinMain proc
sub rsp,28h; выравнивание стека 28h=32d+8;  8 - возврат
    mov rbp,rsp ; сохранение выравненного значения стека
invoke count,len1
invoke wsprintf,ADDR buf,ADDR info,res;
invoke MessageBox,0,addr buf,addr _title,MB_ICONINFORMATION
        invoke ExitProcess, 0
        WinMain endp
        end
```

Результат выполнения программы приведен на рис. 4.2.5.

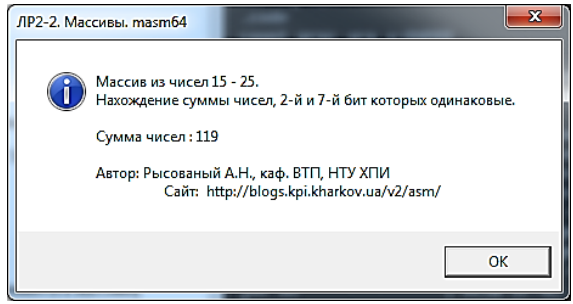

Рис. 4.2.5. Результат выполнения программы

В программе используется собственная функция с именем count и одним параметром len1. Этот параметр расчитывается в сегменте данных (.data) оператором equ $- . Оператор equ вычисляет разницу в байтах между текущим значением $ и началом расположения применяемой переменной.

А прототип внутренней функция count приводится в начале программы после подключаемых библиотек и указывает размерность этого параметра:

```
count  PROTO  arg_a:QWORD
```

Эта функция раскрыта в начале сегмента кода (.code). Однако она может быть раскрыта в различных местах:

– в начале программы;

– в конце программы до команды end;

– в отдельном файле.

Переменнная len1 определяет количество чисел в массиве x1. В программе не используется передача чисел массива в параметрах, т.к. при большом количестве такой подход не является разумным.

Окно с кодом программы отладчика x86Dbg на момент начала отладки приведено на рис. 4.2.6.

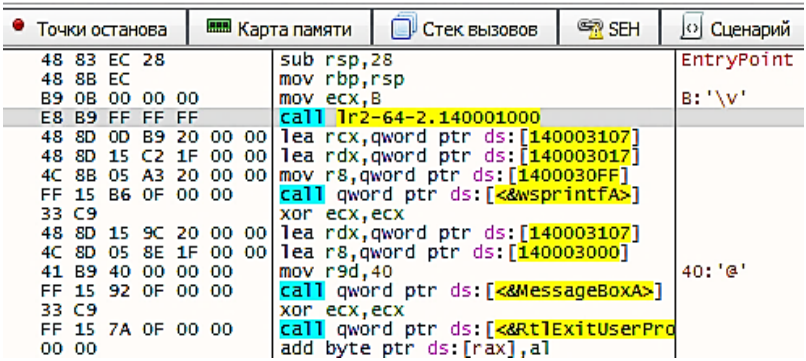

Рис. 4.2.6 Окно с кодом программы отладчика x86Dbg на момент начала отладки

Строки sub rsp,28h

mov rbp,rsp ; сохранение выравненного значения стека

производят выравнивание стека и сохранение результата выравнивания.

Строка кода mov ecx,8 в отладчике непосредственно в тексте программы отсутствует, но она является первым параметром функции count и показывает количество чисел в массиве.

Окно отладчика x86Dbg с кодом раскрытой процедуры count с параметром len1 приведено на рис. 4.2.7.

Напоминаю, что при отладке приходится часто перезагружать в отладчике исполняемый файл программы. В этом случае, чтобы снова не отслеживать все просмотренные строки кода кнопкой «F7» (или «F8»), можно поставить клавишей «F2» контрольную точку на последней проверенной строке, затем перегрузить программу и нажать

кнопку «F9». В этом случае произойдет выполнение программы до команды, на которую «поставили» контрольную точку.

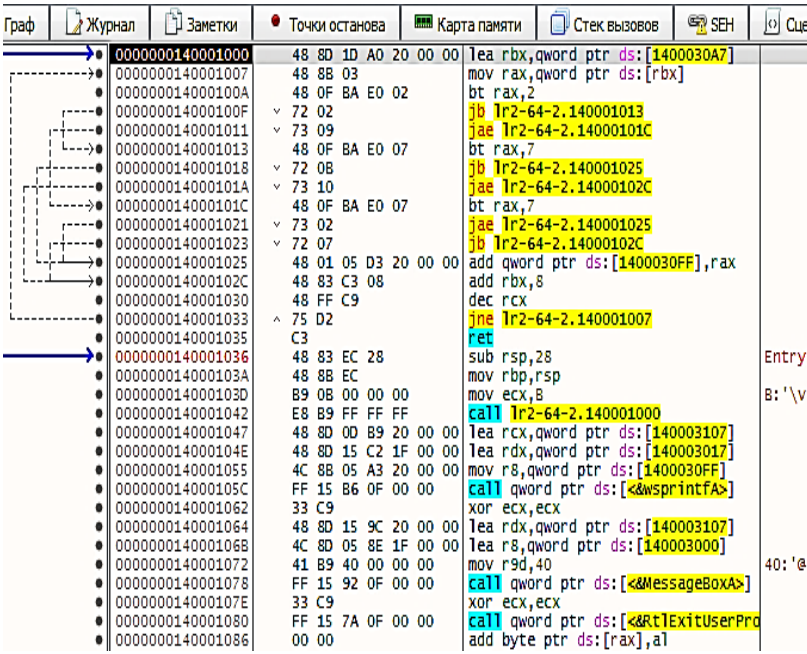

Рис. 4.2.7. Окно отладчика x86Dbg с кодом раскрытой процедуры count

В этом окне вместо имен меток приводятся адреса ячеек, на которые происходит передача управления, в случае, если выполняется соответствующее условие.

Содержимое ячеек памяти (дамп памяти) приведен на рис. 4.2.8.

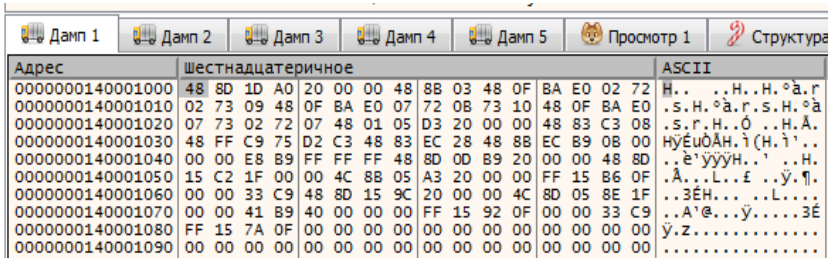

Рис. 4.2.8. Содержимое ячеек памяти

По содержимому ячеек памяти анализируется соответствующая команда.

После выполнения функции invoke count,len1 происходит преобразование ячейки памяти res с целочисленным содержимым для последующего вывода функцией MessageBox.

Программа 4.2.5. Формирование массива из чисел, которые меньше числа 12 с использованием собственной процедуры (на masm64):

```
        ; masm64. Формирование массива из чисел. Проц. с параметром
        title анализ чисел массива; masm64
        ; сформировать массив из чисел, которые меньше числа 12
include win64a.inc ; библиотеки для подключения
    LR2_2  PROTO  arg_a:QWORD
.data
    mas1 dq 0,17,7,8,12,15,16
    len1 equ ($-mas1)/type mas1
    mas2 dq len1 dup(0h)
    const1 dq 12
_title db " Анализ чисел массива. masm64",0
          strbuf dq ?,0
_text db 9,"Reusult:",0ah,0ah,"mas2[0] = %d",10,"mas2[1] = %d",10,
  "mas2[2] = %d",10,"mas2[3] = %d",0ah,"mas2[4] = %d",10,
  "mas2[5] = %d",10,"mas2[6] = %d",10,10,
  "Автор: Рысованый А.Н., каф. ВТП, НТУ ХПИ",0

.code
    LR2_2  proc  arg_a:QWORD
      mov r15,len1 ;
      lea rsi,mas1 ; занесение адреса начала элементов массива mas1
```

```
        lea rdi,mas2 ; занесение адреса массива результата массива mas2
@1: mov rax,[rsi]
            cmp rax,rcx;  arg_a
            js less
            jz equal
        ; if mas1[i] > 12
            add rsi,type mas1
            ;add rdi,type mas2
            dec r15
            jnz @1
            jz equalEnd
less:                       ; if mas1[i] < 12
            mov [rdi],rax         ; запись в новый массив
            add rsi,type mas1 ; увеличение адреса на размерность чисел
            add rdi,type mas2 ;
            dec r15
            jnz @1
            jmp equalEnd
        equal:              ; if mas1[i] = 12
            mov [rdi],rax
            add rsi,type mas1
            add rdi,type mas2
            dec r15
            jnz @1
equalEnd:
         ret
        LR2_2 endp

WinMain proc
sub rsp,28h; выравнивание стека 28h=32d+8;  8 - возврат
        mov rbp,rsp ; сохранение выравненного значения стека

invoke LR2_2,const1
invoke wsprintf,ADDR strbuf,ADDR _text,mas2,mas2[8],mas2[16],mas2[24],
mas2[32],mas2[40],mas2[48];
        invoke MessageBox,0,addr strbuf,addr _title,MB_OK
        invoke ExitProcess, 0
        WinMain endp
        end
```

Программа с именем WinMain (точкой входа) вызывает процедуру LR2_2 с параметром const1, в качестве которого передается число 12.

Окно отладчика x86Dbg с кодом раскрытой процедуры

LR2_2 proc arg_a:QWORD

приведено на рис. 4.2.9.

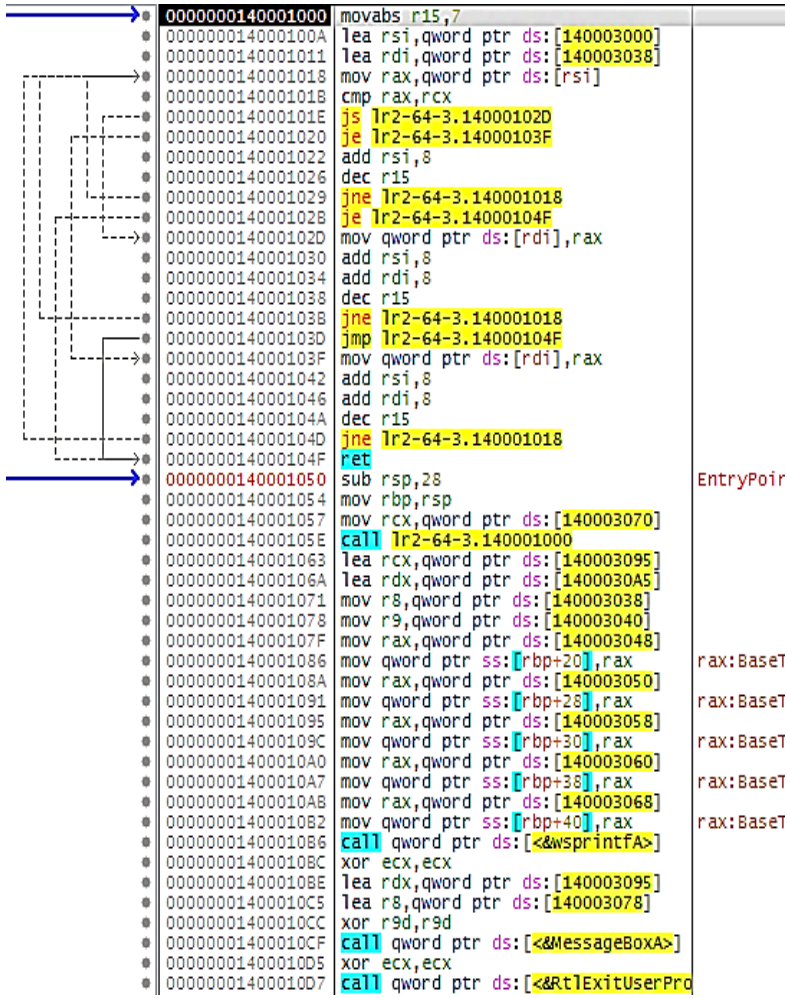

Рис. 4.2.9. Результат выполнения программы

Строки кода

lea rsi,mas1 ; занесение адреса начала элементов массива mas1

lea rdi,mas2 ; занесение адреса массива результата массива mas2

записывают адреса массивов в соответствующие ячейки.

Строка кода cmp rax,rcx; arg_a
осуществляет сравнение числа 12, которое передается в rcx и числа с массива mas1.

Команда js less осуществляет переход на метку less, если результат сравнения (вычитания) отрицательный.

Строки mov [rdi],rax ; запись в новый массив
add rsi,type mas1 ; увеличение адреса на размерность числа
add rdi,type mas2 ;
осуществляют запись числа, которое меньше 12, в новый массив, а оставшиеся две команды производят увеличение адресов на размерность чисел массива.

А т.к. количество элементов массива записано командой mov r15,len1, то строками кода
dec r15
jnz @1
осуществляется вычитание 1 из r15 и дальнейший анализ результата декрементирования на не ноль. Если в r15 не ноль, то происходит передача упраления на метку @1. В противном случае – будет выполняться следующая команда jmp equalEnd, по которой происходит безусловный переход на метку с именем equalEnd.
В отладчике отображаются загружаемые ячейки памяти и адреса меток, на которые происходит передача управления.

Результат выполнения программы приведен на рис. 4.2.10.

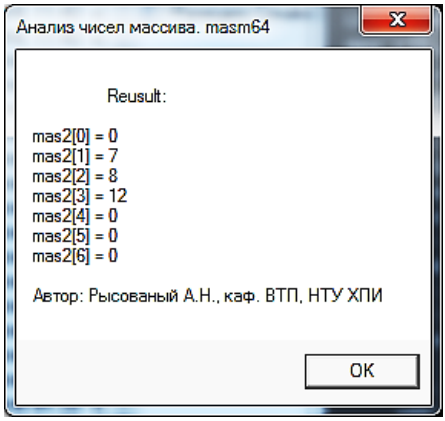

Рис. 4.2.10. Результат выполнения программы

Программа 4.2.6.

```
        Title                   ; masm64
; Задан массив ввиде структуры A: X1,Y1; X2,Y2; ... ..",10,
;  Определить количество пар, для которых Xi <= Yi."
        include win64a.inc ; библиотеки для подключения
        count PROTO arg_a:QWORD
   array struct
    Xi dq ?
    Yi dq ?
   array ends
.data
  szBuf dq 1 dup(?),0
  fmt  db "Задание: задан массив ввиде структуры A: X1,Y1; X2,Y2; ...
..",10,
  "Определить количество пар, для которых Xi <= Yi.",10,10,
  "Счетчик условия = %d",10,10, "Автор: Рысованый А.Н., КИТ, НТУ
ХПИ",0
titl1 db "masm64. Массив ввиде структуры",0
        A1  array
<5,3>,<3,5>,<2.5,2.6>,<8.9,4.6>,<3.7,8.3>,<4.2,2.1>,<2.1,3.9>
        len1  equ ($-A1)/type A1

.code
        count proc arg_a:QWORD
          lea rdi,A1  ; загрузка адреса массива
          xor rsi,rsi ; очистка счетчика совпадения условий
          mov rcx, len1 ; количество пар чисел

@@1:
        mov rax,[rdi]     ; извлечение первого числа пары
        mov rdx,[rdi+8] ; извлечение второго числа пары
        cmp rax,rdx       ; сравнение чисел

          js @1         ; перейти по знаку результата
          jmp @2      ; в противном случае

@1: inc rsi             ; увеличение на 1 счетчика условий
@2: add rdi,16      ; величина отличия в парах чисел
        loop @@1
        ret
        count endp

WinMain proc
sub rsp,28h; выравнивание стека 28h=32d+8;  8 - возврат
```

```
mov rbp,rsp ; сохранение выравненного значения стека
    invoke count,len1
    invoke wsprintf,ADDR szBuf,ADDR fmt,rsi
    invoke MessageBox,0,addr szBuf,addr titl1,MB_ICONINFORMATION
    invoke ExitProcess,0
  WinMain endp
  end
```

Результат выполнения программы приведен на рис. 4.2.11.

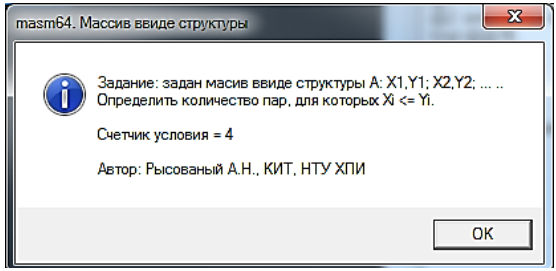

Рис. 4.2.11. Результат выполнения программы

ЛАБОРАТОРНАЯ РАБОТА 3
КОМАНДЫ СДВИГА

Цель занятия: приобрести практические навыки составления, отладки и выполнения программ, написанных языком ассемблера для вычисления математического уравнения при помощи арифметических команд и команд сдвига, сравнение времени их выполнения под МП платформ x86/64.

В отчете представить:

– номер и название лабораторной работы;

– задание;

– текст программы с комментариями к каждой строке программы;

– результат выполнения программы (скриншоты упрощенных окон (MessageBox) и отладчиков);

– алгоритм программы (клавиша «G» в окне отладчика);

– особенности выполнения, где описать общий алгоритм (последовательность) выполнения и особенности программы (на что необходимо с вашей точки зрения уделить внимание).

Постановка задачи

Выполнить задание в двух вариантах:

1-й вариант. Выполнить задание под Win32.

2-й вариант. Выполнить задание под Win64 (из материалов лекции**).**

Задание

Заданы числа: *a, b, c, d, e, f, g,* которые являются показателями степени 2. В одной программе вычислить выражение двумя способами: при помощи арифметических команд и команд сдвига, которые заменяют арифметические команды. Вывести через одну функцию MessageBox результаты выполнения и количество тактов, которые использованы для их вычислений.

Согласно номеру студента в группе выбрать вариант задания и написать на ассемблере программу вычисления одного из выражений:

1. a/c - e/f – ad;
2. a/b + c/d – eg;
3. a/b + cd - f/(ge);
4. d/a – cd+ bef;
5. g/f + (ed)/c+b/a;
6. e/b + a/c – def;
7. ab + cd/e + f - g;
8. a/b + cd + e/f + g;
9. d/b/a – cd/e/f;
10. abc - de/f;
11. ab/e – cde;
12. b – a/c + de;
13. d/a – bc + f;
14. a + b/c/d + efg;
15. ac + b/d + f/e;
16. a/b-cd/e+fg;
17. fe - b/d – a/c;
18. g/f + e/dc + ba;
19. g/e+d/b + a;
20. g + f/d + cb - a;
21. g/f/e + dc/b - a;
22. g/f + e/d - c/a;
23. a/d + f/e - ca;
24. ab – gc +a/e + f;
25. a/c + b/d - efg;
26. a/b/c- de + f;
27. a/d + c/b +e+ fg;
28. af + d/c - ab;
29. a/b + c/d + ef;
30 abc + def,

где *a, b, c, d, e, f, g* – числа – показатели степени 2.

Примеры решения

Решение на **masm32**

Программа 4.3.1. Решение уравнения a/b + cd на masm32:

```
; Решение уравнения a/b + cd на masm32
.686                    ; определение типа микропроцессора
.model flat, stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

count PROTO
arg_a:DWORD,arg_b:DWORD,arg_c:DWORD,arg_d:DWORD,
arg_e:DWORD, arg_f:DWORD

.data
_a dd 16
_b dd 4
_c dd 64
_d dd 32
_e dd 128
_f dd 256
_res dd ?,0
_res1 dd ?,0
```

```
_res2 dd ?,0
_res3 dd ?,0
_title db "Лабораторная работа №3. Команды сдвига. (masm32)",0
_text db "Уравнение  a/b + cd",0ah,"Результат выполнения арифм.
команд:  %d",0ah,"Число тактов: %d",0ah,0ah,
"Результат выполнения команд сдвига: %d",0ah,"Число тактов:
%d",0ah,0ah,
"Автор: Рысованый А.Н., каф. ВТП, НТУ ХПИ",0
strbuf dd ?,0

.code

count proc arg_a:DWORD,arg_b:DWORD,arg_c:DWORD,arg_d:DWORD,
arg_e:DWORD,arg_f:DWORD

 rdtsc          ; rdx,rax - получение числа тактов

  xchg edi,eax ;обмен значениями регистров
mov eax,arg_a ;
xor edx,edx   ; подготовка к делению
div arg_b     ; edx,eax
mov esi,eax   ; сохранение промежуточного результата
mov eax,arg_c ;
mul arg_d     ;
add eax,esi   ;
mov _res,eax

 rdtsc  ; получение числа тактов

sub eax, edi ; вычитание из последнего числа тактов предыдущего числа
  mov _res1,eax
ret
count endp

count2 proc arg_a:DWORD,arg_b:DWORD,arg_c:DWORD,arg_d:DWORD,
arg_e:DWORD, arg_f:DWORD

 rdtsc
  xchg edi,eax

mov eax,arg_a ;
sar eax,2     ;
mov esi,eax   ; сохранение промежуточного результата
mov eax,arg_c ;
sal eax,5     ;
```

```
add eax,esi   ;
mov _res2,eax

 rdtsc
  sub eax, edi
  mov _res3,eax

ret
count2 endp

start:
     invoke count,_a,_b,_c,_d,_e,_f
     invoke count2,_a,_b,_c,_d,_e,_f
invoke wsprintf, ADDR strbuf, ADDR _text, _res, _res1,_res2,_res3

invoke MessageBox, NULL, addr strbuf, addr _title,
MB_ICONINFORMATION

invoke ExitProcess, 0
END start
```

Результат выполнения программы 4.3.1 приведен на рис. 4.3.1.

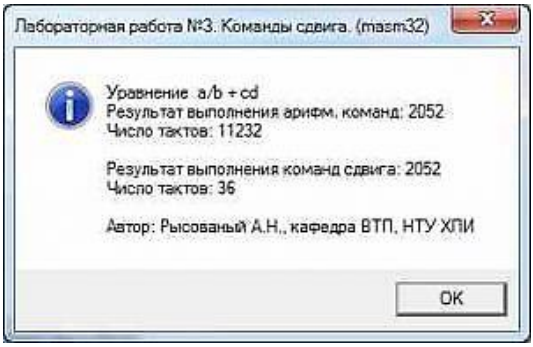

Рис. 4.3.1. Результат выполнения программы на masm32

Программа 4.3.2. Решение уравнения a/b + cd на masm64:

```
; уравнения a/b + cd на masm64. Проц. с параметрами

      include win64a.inc  ; подключаемые библиотеки
```

```
count PROTO arg_a:QWORD,arg_b:QWORD,arg_c:QWORD,
arg_d:QWORD, arg_e:QWORD, arg_f:QWORD

.data
        _a dq 16
        _b dq 4
        _c dq 64
        _d dq 32
        _e dq 128
        _f dq 256
        _res dq ?,0
        _res1 dq ?,0
        _res2 dq ?,0
        _res3 dq ?,0
_title db "Лабораторная работа №3. Команды сдвига. (masm64)",0
_text db "Уравнение a/b + cd",0ah,"Результат выполнения арифм. команд:
%d",0ah,"Число тактов: %d",0ah,0ah,
"Результат выполнения команд сдвига: %d",0ah,"Число тактов:
%d",0ah,0ah,
"Автор: Рысованый А.Н., каф. ВТП, НТУ ХПИ",10,
9,"Сайт:  http://blogs.kpi.kharkov.ua/v2/asm/",0

        strbuf dq ?,0

.code
count proc arg_a:QWORD,arg_b:QWORD,arg_c:QWORD,arg_d:QWORD,
arg_e:QWORD,arg_f:QWORD

      mov r10,rdx ; сохранение второго параметра

      rdtsc          ; rdx,rax - получение числа тактов
        xchg rdi,rax ; обмен значениями регистров

        mov rax,rcx ; arg_a
        xor rdx,rdx
        div r10        ; edx,eax
        mov rsi,rax   ; сохранение промежуточного результата
        mov rax,r8 ;
        mul r9       ;
        add rax,rsi      ;
        mov _res,rax

      rdtsc            ; получение числа тактов
sub rax,rdi   ; вычитание из последнего числа тактов предыдущего числа
        mov _res1,rax
```

```
        ret

        count endp

count2 proc arg_a:QWORD,arg_b:QWORD,arg_c:QWORD,arg_d:QWORD,
arg_e:QWORD, arg_f:QWORD
        rdtsc
        xchg rdi,rax
        sar rcx,2 ;
        mov rsi,rcx ; сохранение промежуточного результата
        sal r8,5 ;
        add r8,rsi ;
        mov _res2,r8
        rdtsc
        sub rax, rdi
        mov _res3,rax
        ret
        count2 endp

WinMain proc
sub rsp,28h; выравнивание стека 28h=32d+8;  8 - возврат
mov rbp,rsp ; сохранение выравненного значения стека

        invoke count,_a,_b,_c,_d,_e,_f
        invoke count2,_a,_b,_c,_d,_e,_f
        invoke wsprintf, ADDR strbuf, ADDR _text, _res, _res1,_res2,_res3
invoke MessageBox, NULL, addr strbuf, addr _title, MB_ICONINFORMATION
        invoke ExitProcess, 0
        WinMain endp
        end
```

Программа с WinMain после выравнивания и сохранения стека вызывает функцию с именем count. Эта функция (процедура или подпрограмма) выполняет уравнение a/b + cd с использованием команд умножения и деления. В начале функции count proc выполняется команда mov r10,rdx. Эта команда необходима для того, чтобы сохранить параметр, который передается в процедуре вторым по счету – arg_b. Этот параметр передается в регистре rdx. А следующая команда rdtsc сохраняет значения тиков в регистрах rdx,rax. Если не пересохранить командой mov r10,rdx значение rdx, то после выполнения следующей команды будет потеряно значение _b dq 4.

Для отладки программы надо вызвать отладчик x64Dbg и загрузить в него исследуемый exe-файл (рис. 4.3.2).

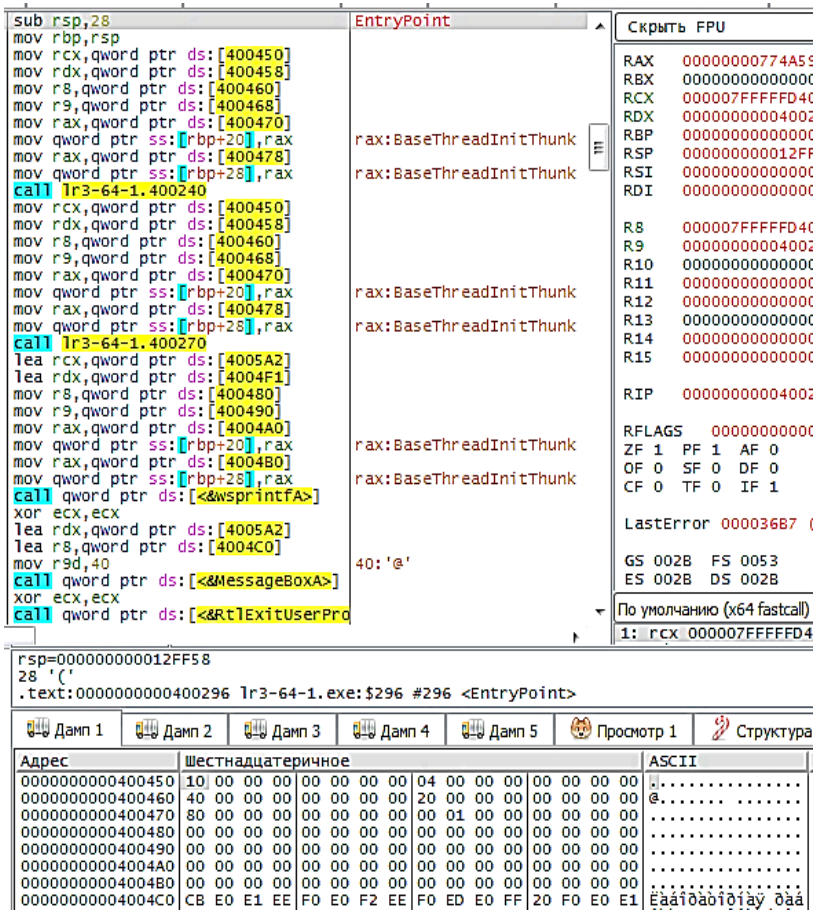

Рис. 4.3.2. Окна отладчика x64Dbg с exe-файлом

На этом рисунке видно, что в функции count параметры передаются через регистры rcx, rdx, r8 и r9, а оставшиеся два параметра – через стек. Причем, каждый из них сначала из ячейки памяти передается в регистр rax, а затем стек. Например, параметр _e передается такой последовательностью команд:

```
mov rax,qword ptr ds:[400470]
mov qword ptr ss:[rbp+20],rax
```

Следующий параметр передается последовательностью:

```
mov rax,qword ptr ds:[400478]
mov qword ptr ss:[rbp+28],rax
```

Следующая функция (процедура или подпрограмма) выполняет уравнение a/b + cd с использованием команд сдвига вместо использования команд умножения и деления.

Команда умножения заменяется сдвигом влево, а команда деления – сдвигом вправо.

Для того, чтобы последовательно отслеживать содержимое регистров и ячеек памяти следует нажимать кнопку F7 для «вхождения» в процедуру. Это действие актуально при отлпадке собственнных процедур, в нашем случае– это процедуры с именами count и count2 (рис. 4.3.3).

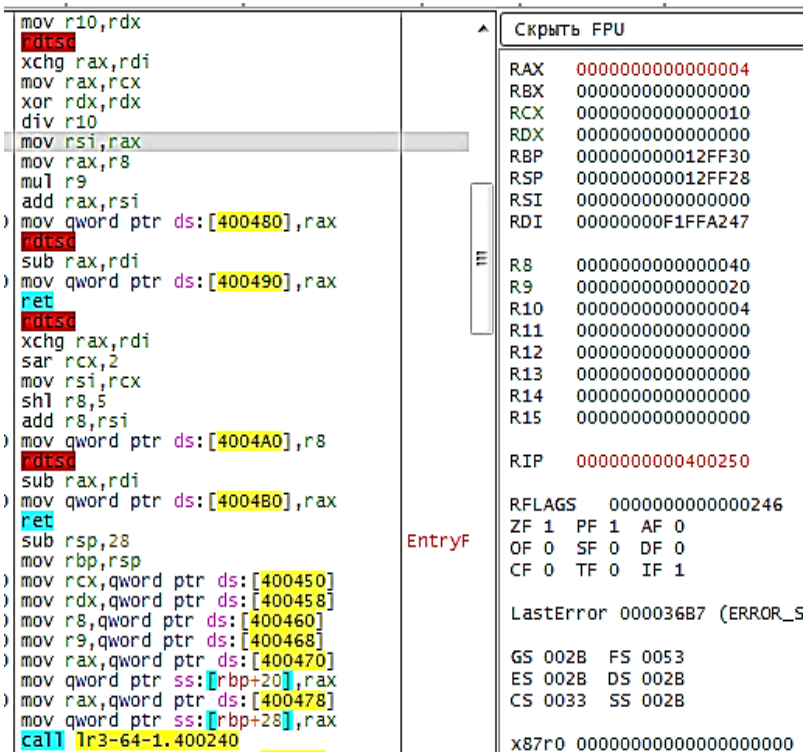

Рис. 4.3.3. Раскрытие процедур count и count2 в отладчике x64Dbg

Результат выполнения программы 4.3.2 приведен на рис. 4.3.4.

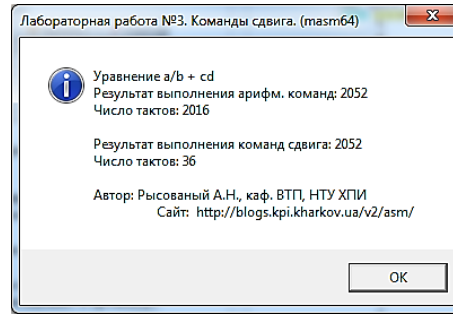

Рис. 4.3.4. Результат выполнения программы на masm64

Результат решения уравнения с использованием команд умножения деления, и команд сдвига – одинаков. Однако время выполнения уравнения с использованием команд сдвига вместо существенно меньше. Но команды сдвига можно использовать не всегда – тогда, когда числа являются показателями степени 2. Например, умножение на 2 можно заменить сдвигом влево на 1 разряд, а деление на 2 – сдвигом вправо на 1 разряд и т.п.

ЛАБОРАТОРНАЯ РАБОТА 4
ТЕСТИРОВАНИЕ БИТОВ

Цель занятия: приобрести практические навыки составления, отладки и выполнения программ, написанных языком ассемблера для вычисления математического уравнения при помощи арифметических команд и команд сдвига, сравнение времени их выполнения под МП платформ x86/64.

В отчете представить:

– номер и название лабораторной работы;

– задание;

– текст программы с комментариями к каждой строке программы;

– результат выполнения программы (скриншоты упрощенных окон (MessageBox) и отладчиков);

– алгоритм программы (клавиша «G» в окне отладчика);

– особенности выполнения, где описать общий словесный алгоритм (последовательность) выполнения и особенности программы (на что необходимо с вашей точки зрения уделить внимание).

Постановка задачи

Выполнить задание под **Win64 в среде masm64.**

Задание

1. Задан массив A из N = 25 элементов. Привести программу определения суммы элементов массива A, для которых биты 0,1 и 4 совпадают.

2. Задан массив A из N = 30 элементов. Привести программу определения суммы элементов массива A, для которых биты 2 и 7 совпадают.

3. Заданы массивы A и B из N = 25 элементов. Привести программу формирования массива C по такому правилу: если у элементов массивов Ai и Bi биты 0, 1 и 2 совпадают, то $Ci = Ai – Bi$.

4. Задан массив A из N = 40 элементов. Привести программу формирования массива B из элементов массива A, y которых 0, 2 и 5 биты имеют нули.

5. Заданы массивы *A u B* из $N = 50$ элементов. Привести программу формирования массива *C* по такому правилу: если у элементов *Ai* и *Bi* биты 0, 1 и 2 совпадают то $Ci = Ai + Bi$.

6. Заданы массивы *A* и *B* из $N = 40$ элементов. Привести программу формирования массива *C* по такому правилу: если $Ai + Bi \le 0$, то $Cj = Bi$.

7. Задан массив *A* из $N = 25$ элементов. Привести программу определения количества элементов массива *A,* которые удовлетворяют условию $L \le Ai < M$, где $L = -3$ и $M = 15$.

8. Задан массив *A* из $N = 60$ элементов. Привести программу определения количества элементов массива *A,* которые удовлетворяют условию $L < Ai < M$, где $L = 4$ и $M = 18$.

9. Задан массив *A* из $N = 50$ элементов. Привести программу определения количества элементов массива *A,* которые удовлетворяют условию $L \le \mathrm{Ai} \le M$, где $L = -5$ и $M = 19$.

10. Задан массив *A* из $N = 40$ элементов. Привести программу определения количества элементов массива *A,* которые удовлетворяют условию $L \ge Ai < M$, где $L = 6$ и $M = 20$.

Примеры решения

Пример 4.4.1. Задан массив из 50 элементов. Подсчитать сумму элементов массива, для которых биты 0 и 5 совпадают.

Программа 4.4.1.

```
        ; masm64. Подсчет суммы элементов массива, для которых
        ; биты 0 и 5 совпадают
include win64a.inc ; вызов программы с подключаемыми библиотеками

        err1 PROTO arg_a:QWORD  ; !!! определение прототипа

.data
        buf dq ?,0      ;
ifmt db "Задано массив из 50 элементов:",0dh,0ah,\
"10, 12, 13, 14, 8",10,"dw 20 dup(0)",10,"dw 25 dup(1)",10,10,
"Сумма элементов массива, для которых биты 0 и 5 совпадают: =
%d",10,10,\
"Автор программы:  Рысованый А.Н., г. Харьков, фак. КИТ, НТУ ХПИ",0

        titl1 db "masm64.  Исследование команды bt",0 ; название окна

        mas1 dw 10,12,13,14,8
```

```
          dw 20 dup(0)
          dw 25 dup(1)      ; массив mas1 слов
       len1 equ ($-mas1)/type mas1 ; вычисление количества слов в mas1
       sum dw 0            ; ячейка для результата
.code
       WinMain proc
sub rsp,28h; выравнивание стека 28h=32d+8;  8 - возврат
  mov rbp,rsp ; сохранение выравненного значения стека

        invoke err1,len1
 invoke wsprintf,ADDR buf,ADDR ifmt,r15 ; функция преобразования r15
 invoke MessageBox,0,ADDR buf,ADDR titl1,MB_ICONINFORMATION;
        invoke ExitProcess,0
       WinMain endp

  err1 proc arg_a:QWORD
  ; arg_a передается в rcx
       lea rsi,mas1  ; начальный адрес массива mas1
m1:   mov ax,[rsi] ; в ax заносится элемент массива
       bt ax,0        ; выбор нулевого бита
       setc bh        ; если cf=1, то установление 1 в bh
       bt ax,5        ; выбор пятого бита
       setc bl        ; если cf = 1, то установление 1 в bl
       cmp bh,bl    ; сравнение битов
       jne m2         ; если не равняется, то перейти на m2
       add sum,ax     ; добавление выбранных элементов массиву
m2:  add rsi,2 ; увеличение адреса mas1 для выборки нового числа
       dec ecx          ; уменьшение счетчика чисел в массиве mas1
       jnz m1         ; перейти на метку m1, если не нуль
    movzx r15,sum  ; сохранение результата с расширением разрядности
       ret
       err1 endp
       end
```

При отладке программы удобно использовать небольшой массив с конкретными данными. После отладки программы массив может быть определен несколькими вариантами.

Вариант1:

```
  mas1 dq 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
    21,22,23,24,25,26,27,28,29,30,  и так далее  до 99.
```

Вариант2:

```
  buf dw 10 dup (1)
       dw 20 dup (0)
```

```
dw 70 dup (255),0          ; буфер вывода сообщения
```

Размер чисел массивов должен совпадать требованию к программе.

Результат выполнения программы 4.4.1 приведен на рис. 4.4.1.

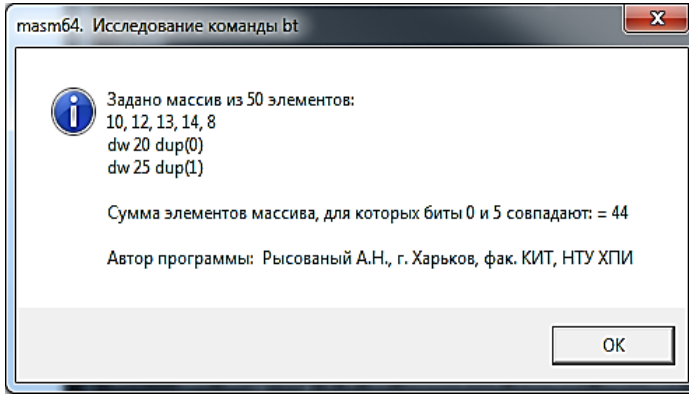

Рис. 4.4.1. Результат выполнения программы

В связи с тем, что по условию задано достаточно большой массив, а для отладки достаточно небольшого количества чисел массива, то в программе проверяется только первые 5 чисел, а для остальных применены операторы дублирования:

```
mas1 dw 10,12,13,14,8
     dw 20 dup(0)
     dw 25 dup(1)      ; массив mas1 слов
```

Вычисление чисел массива определяется строкой кода

```
len1  equ ($-mas1)/type mas1 ; вычисление количества чисел в mas1
```

Для masm64 программа должна иметь имя. Если не менять командный файл asm2.bat, то имя это задано по умолчанию как WinMain и, по этому, оформлено ввиде процедуры.

Программа, непосредственно выполняющая условия задания, оформлена ввиде процедуры с именем err1. А так как применена собственная процедура err1, то обязательно необходимо указать количество и размер параметров этой процедуры – так называемый прототип функции. Этот прототип

```
err1 PROTO arg_a:QWORD
```

указывается в начале программы перед секцией данных.

Для отладки программы надо вызвать отладчик x64Dbg и загрузить в него исследуемый exe-файл (рис. 4.4.2).

Рис. 4.4.2. Окна отладчика x64Dbg

В отладчике необходимо вызвать начальное значение дампа памяти с адреса, который вас интересут или встречается первым. В этой программе таким адресом есть 4003B0. Для вызова интересующего адреса необходимо нажать на правую клавишу мышки и выбрать Перейти/Выражение/ и «Ввести адрес».

В собственной функции err1 используется один параметр len1:

invoke err1,len1

А как известно, первые четыре целочисленных параметра передаются через регистры. Первый параметр передается через регистр rcx. Поэтому, в программе отпадает необходимость самостоятельно загружать счетчик. А в строках

```
dec ecx          ; уменьшение счетчика чисел в массиве mas1
jnz m1           ; перейти на метку m1, если не нуль
```

происходит уменьшение счетчика на единицу и переход на метку, если в этом счетчике не ноль.

Выбор необходимого по условию бита осуществляется последовательным использованием команды **bt.**

Алгоритм (граф) выполнения программы можно получить в окне отладчика x64Dbg, если нажать правую кнопку мышки и выбрать – «Граф» (рис. 4.4.3).

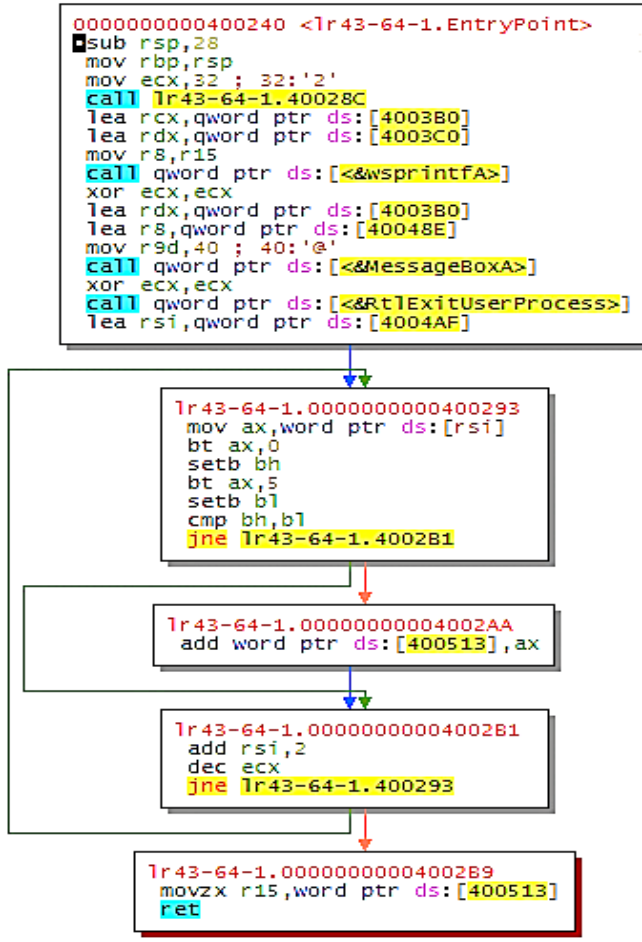

Рис. 4.4.3. Алгоритм (граф) программы в окне отладчика x64Dbg

ЛАБОРАТОРНАЯ РАБОТА 5
ПЕРЕДАЧА ПАРАМЕТРОВ ЧЕРЕЗ ТАБЛИЦУ АДРЕСОВ

Цель занятия: приобрести практические навыки составления, отладки и выполнения программ, написанных языком ассемблера при передачи параметров через таблицу адресов для МП платформ x86/64.

В отчете представить:

– номер и название лабораторной работы;

– задание;

– текст программы с комментариями к каждой строке программы;

– результат выполнения программы (скриншоты упрощенных окон (MessageBox) и отладчиков);

– алгоритм программы (клавиша «G» в окне отладчика);

– особенности выполнения, где описать общий алгоритм (последовательность) выполнения и особенности программы (на что необходимо с вашей точки зрения уделить внимание).

Постановка задачи

Выбрать вариант задания и написать на ассемблере программу вычисления выражения.

1. Заданы массивы *A* и *B по N* = 15 элементов. Написать программу формирования массива *C* по следующему правилу: если $Ai < Bi$, то $Ci = Ai \times Bi$; иначе – $Ci = Ai / Bi$.

2. Заданы массивы *A* и *B по N* = 20 элементов. Написать программу формирования массива *C* по следующему правилу: если $Ai \times Bi > L$, то $Ci = Ai$; иначе – $Ci = Bi$.

3. Заданы массивы *A* и *B по N* = 25 элементов. Написать программу формирования массива *C* по следующему правилу: если $Ai / Bi < L$, то $Ci = Ai$; иначе – $Ci = Bi$.

4. Задан массив *A из N* = 55 элементов. Написать программу формирования массива *B* из элементов массива *A,* которые удовлетворяют условию $Ai \le E$ при $E = 12$.

5. Задан массив *A из* N = 70 элементов. Написать программу определения суммы и количества элементов массива *A,* которые удовлетворяют условию $Ai \le E$ при $E = -13$.

6. Заданы массивы *A* и *B по N* = 50 элементов. Написать программу определения количества пар элементов, которые удовлетворяют условию $Ai \le Bi$.

7. Заданы массивы *A* и *B по N* = 65 элементов. Написать программу формирования массива *C* по следующему правилу: если $Ai - Bi \le 0$, то $Cj = Ai$.

8. Задан массив *A из* N = 50 элементов. Написать программу определения максимального из отрицательных элементов массива *A*.

9. Задан массив *A из* N = 50 элементов. Структура массива *A* такая*:* *X*1, *Y*1; *X*2, *Y*2; … Написать программу определения количества пар, для которых выполняется условие $Xi < Yi$.

10. Задан массив *A из N* = 28 элементов. Написать программу формирования массива *B* из элементов массива *A,* в которых биты 0, 2 и 5 имеют нули.

11. Задан массив *A из N* = 40 элементов. Написать программу определения суммы элементов массива *A,* для которых биты 2 и 10 совпадают.

12. Задан массив *A из N* = 80 элементов. Структура массива *A* такая*:* *X*1, *Y*1; *X*2, *Y*2; .... Написать программу определения количества пар, для которых выполняется условие $Xi > Yi$.

13. Задан массив *A из N* = 20 элементов. Написать программу определения минимального из положительных элементов массива *A*.

14. Заданы массивы *A* и *B по N* = 30 элементов. Написать программу формирования массива *C* по следующему правилу: если $Ai + Bi > 0$, то $Cj = Bi$.

15. Заданы массивы *A* и *B по* N = 20 элементов. Написать программу определения количества пар элементов, которые удовлетворяют условию $Ai > Bi$.

16. Задан массив *A из N* = 40 элементов. Написать программу определения суммы и количества элементов массива *A,* которые удовлетворяют условию $Ai > E$ при $E = -10$.

17. Задан массив *A из N* = 22 элементов. Написать программу формирования массива *B* из элементов массива *A,* которые удовлетворяют условию $Ai > E$ при $E = 5$.

18. Задан массив *A из N* = 30 элементов. Написать программу формирования массива *B* с первых 8 положительных элементов массива *A*.

19. Задан массив *A из* N = 20 элементов. Написать программу определения количества элементов массива *A,* которые удовлетворяют условию $L < Ai \le M$, де $L = 2$ та $M = 10$.

20. Заданы массивы *A* и *B по N* = 11 элементов. Написать программу формирования массива *C* по следующему правилу: если *Ai* > *Bi,* то *Ci* = *Ai* + *Bi*; иначе – *Ci* = *Ai* – *Bi.*

21. Задан массив *A из N* = 50 элементов. Написать программу определения количества элементов массива *A,* которые удовлетворяют условию *L* > *Ai* > *M*, де *L* = 8 та *M* = 28.

22. Заданы массивы *A* и *B по N* = 10 элементов. Написать программу формирования массива *C* по следующему правилу: если *Ai* < *Bi*, то *Ci* = *Bi* – *Ai*; иначе – *Ci* = *Ai* + *Bi.*

23. Заданы массивы *A* и *B по N* = 12 элементов. Написать программу формирования массива *C* по следующему правилу: если $A_i \le B_i$, то $C_i = B_i - A_i$, иначе – $C_i = A_i + Bi$.

24. Задан массив *A из N* = 55 элементов. Написать программу определения количества элементов массива *A,* которые удовлетворяют условию *L* ≥ *Ai* ≥ *M*, где *L* = 9 та *M* = 30.

25. Задан массив *A из N* = 45 элементов. Написать программу формирования массива *B* из последних 13 элементов массива *A,* которые равны нулю.

Примеры решения

Решение на **masm32**

Проанализировать массивы *A* и *B*, в каждом из которых по 10 элементов типа DWORD. Необходимо определить количество элементов массива Ai, которые удовлетворяют условию $A_i \le B_i$, и использовать передачу параметров через таблицу адресов (программа 4.5.1).

Программа 4.5.1. Передача параметров через таблицу адресов на masm32:

```
        ; Передача параметров через таблицу адресов
        include c:\masm32\include\win32main.inc
.data               ; директива определения данных
 A1 DD 0,1,2,3,4,5,6,7,8,9 ; сохранение в массиве A1 40 байтов
 B1 DD 9,8,7,6,5,4,3,2,1,0
 N1 DD 10   ; число чисел в массиве
 TA dd 21 DUP (0),0
        titl1 db "ЛР-4. Таблица адресов (masm32)",0
_info db "Определить количество элементов массива Ai, для которых Ai <=
Bi: ",0ah,
"mas A1 = 0,1,2,3,4,5,6,7,8,9",0ah,
"mas B1 = 9,8,7,6,5,4,3,2,1,0",0ah,
```

```
"Результат: %d",0ah,0ah,"Автор:  Рысованый А.Н., каф. ВТП, НТУ
ХПИ",0ah,0
        zBuf dd ?,0

.code                   ; директива начала кода программы
_start:     ; метка начала программы с именем _start
        mov TA,offset A1        ; загрузить адрес массива A1
        mov TA+40,offset B1   ; загрузить адрес массива B1
        mov TA+81, offset N1  ; загрузить адрес счетчика N1
        mov ebx, [TA] ; загрузить адрес начала таблицы адресов
        call prog1  ;
invoke wsprintf,ADDR zBuf,ADDR _info,edx;
invoke MessageBox,0,addr zBuf,addr titl1,MB_OK
        invoke ExitProcess, 0

prog1 proc
        xor edx,edx             ; обнуление счетчика чисел массива
        mov ecx,[ebx+80]   ; счетчик = 0Ah (10)
m1: mov esi,[ebx] ; выбрать i-й элемент массива A1 относительно TA (+1)
     mov edi,[ebx+40] ; выбрать i-й элемент массива B1 относительно TA (+1)
        cmp esi,edi  ; сравнение элементов массивов
        js  m2          ; перейти, если меньше
        jmp m3          ; перейти в противоположном случае
m2:  inc edx          ; подсчитать количество элементов
m3:  add ebx,4
            ;add esi,4 ; подготовка следующего элемента массива A1
            ;add edi,4 ; подготовка следующего элемента массива B1
            loop m1   ; перейти на m1, если CX /= 0
        ret             ; возвращение из процедуры
prog1 endp          ; окончание процедуры
        end _start  ; окончание программы с именем  _start
```

При составлении такой программы особое внимание необходимо уделить размерности таблицы адресов _TA. Эта размерность состоит из размерностей чисел, которые расположены в секции data перед таблицей адресов _TA: массив A1 (10 чисел размерностью DD), массив B1 (10 чисел размерностью DD), ячейка N1 (одно число размерности DD). Всего 21 число размерностью DD. Следовательно, размерность _TA должна равняться 21 числу и заканчиваться символом окончания строки, т.е. нулем. Если нуль не поставить, то необходимо уже отвести 22 DD для хранения следующей строки (для titl1).

Команда mov _TA,offset A1 загружает адрес массива A1, а уже относительно этой занятой ячейки следующая команда mov TA+40,offset

B1 загружает 40 ячеек (0-39) адреса массива B1, но уже со сдвигом на занятую предыдущей командой ячейку.

Команда mov _TA,offset A1 загружает в таблицу TA начальный адрес расположения 40-байтного массива A1 (т.е. 0 – 39).

Команда mov _TA+40,offset B1 загружает начиная с 40-й ячейки адрес 40-байтного массива B1.

По количеству чисел массивы одинаковы. Поэтому строка N1 DD 10 присваивает переменной N1 значение 10 – количество элементов в массиве, размерностью DD. Особенностью этой записи есть то, что указывается размерность числа (DD). Ведь напрашивается решение использовать оператор “N1 EQU ($-B1)/type B1“, но в этом случае этот вариант использовать не рекомендуется.

Результат выполнения программы 4.5.1 приведен на рис. 4.5.1.

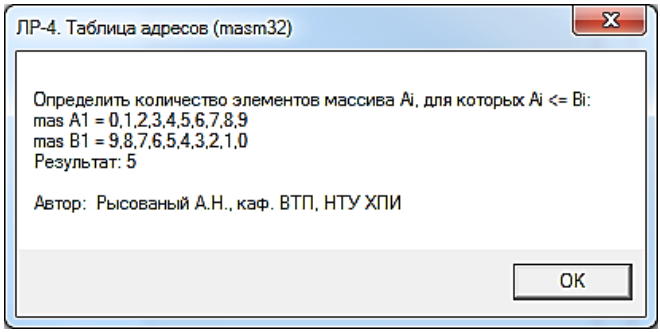

Рис. 4.5.1. Результат выполнения программы на masm32

Решение на **masm64.**

Программа 4.5.2. Передача параметров через таблицу адресов на masm64:

```
      ; masm64. Передача параметров через таблицу адресов
      A1 dq 0,1,2,3,4,5,6,7,8,9 ; сохранение в массиве памяти с именем A1
      ; чисел, каждое в 32-разрядную ячейку
   B1 dq 9,8,7,6,5,4,3,2,1,0
   N1 dq 10
   _TA dq 21 DUP (0) ; 22
zBuf dq ?,0

_info db "Определить количество элементов массива Ai, для которых Ai
<= Bi: ",0ah,
"mas A1 = 0,1,2,3,4,5,6,7,8,9",0ah,
```

```
"mas B1 = 9,8,7,6,5,4,3,2,1,0",0ah,
"Результат: %d",0ah,0ah,"Автор:  Рысованый А.

.code       ; директива начала кода программы
  WinMain proc
sub rsp,28h; выравнив. стека 28h=32d+8; 32d x 8 = 256/16=16 байт; 8 - возврат
     mov rbp,rsp ; сохранение выравненного значения стека
          lea rax,A1              ; загрузить адрес массива A1
          mov _TA,rax
          lea rax,B1              ; загрузить адрес массива B1
          mov _TA+80,rax
          lea rax,N1              ; загрузить адрес массива N1
          mov _TA+161,rax ; 160
          mov rbx, [_TA] ; загрузить адрес начала таблицы адресов

          call prog1  ;
     invoke wsprintf,ADDR zBuf,ADDR _info,r10;
     invoke MessageBox,0,addr zBuf,addr titl1,MB_OK
     invoke ExitProcess,0
          WinMain endp

prog1 proc
          xor r10,r10             ; обнуление счетчика чисел массива
          mov rcx,[rbx+160]   ; счетчик = 0Ah (10)
m1: mov rsi,[rbx]  ; выбрать i-й элемент массива A1
          mov rdi,[rbx+80]   ; выбрать i-й элемент массива B1
          cmp rsi,rdi           ; сравнение элементов массивов
          jle m2       ; перейти, если меньше
          jmp m3     ; перейти в противоположном случае
m2:  inc r10         ; подсчитать количество элементов
m3:  add rbx,8
          loop m1   ; перейти на m1, если CX /= 0
          ret                     ; возвращение из процедуры
          prog1 endp          ; окончание процедуры
          end
```

При переделывании программы с masm32 на masm64 увеличиваются на величину чисел в массиве (увеличиваются на 40) значения смещения в таблице адресов _TA. Кроме того, счетчик чисел массива изменен с edx на r10. Это сделано из-за того, что при преобразовании счетчика функцией wsprintf первый параметр (zBuf) передается в rcx, а второй – в rdx и тем самым меняется первоначальное нужное нам значение rdx. Можно было бы и оставить в качестве счетчика и rdx, но тогда надо было произвести перед выполнением

функции wsprintf пересохранение этого счетчика в неиспользуемый регистр или ячейку памяти (хуже по времени выполнения).

Алгоритм (граф) выполнения программы можно получить в окне отладчика x64Dbg, если нажать правую кнопку мышки и выбрать – «Граф» (рис. 4.5.2).

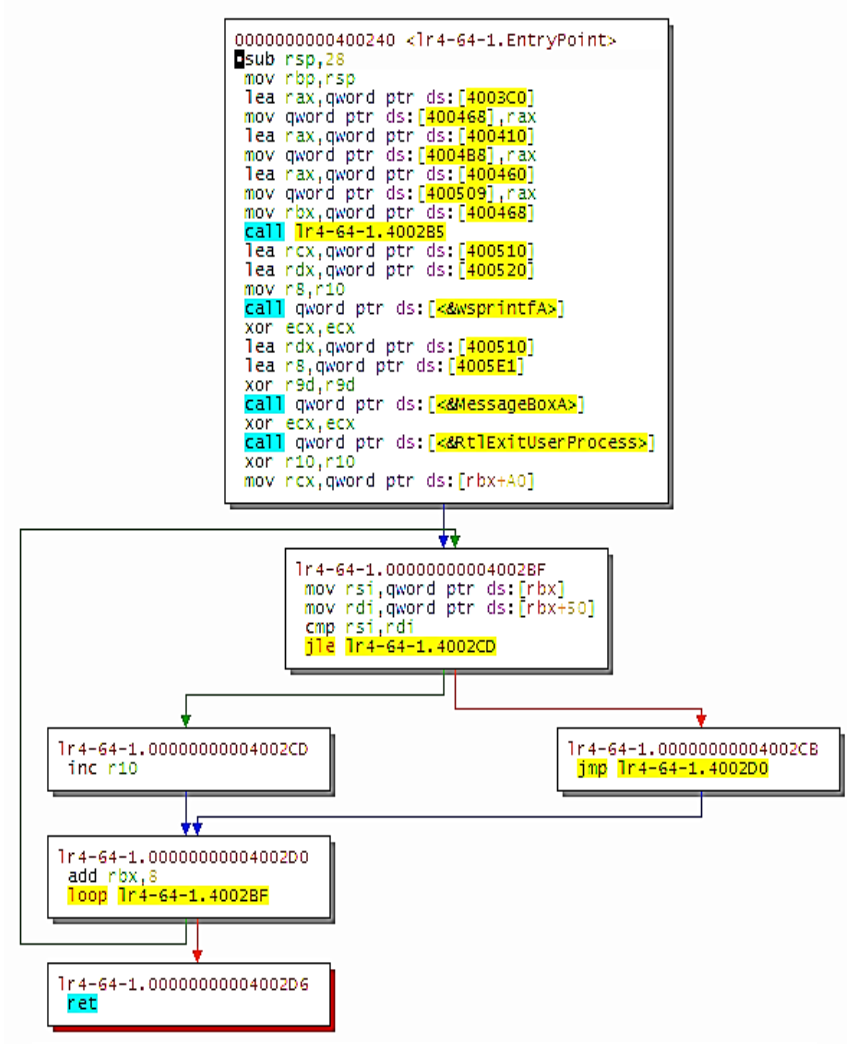

Рис. 4.5.2. Алгоритм (граф) программы в окне отладчика x64Dbg

Алгоритм – это набор инструкций (команд), описывающих порядок действий программы для достижения некоторого результата.

Результат выполнения программы 4.5.2 приведен на рис. 4.5.3.

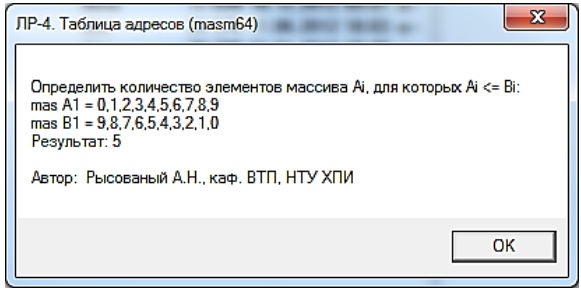

Рис. 4.5.3. Результат выполнения программы на masm64

ЛАБОРАТОРНАЯ РАБОТА 6
ФАЙЛЫ

Цель занятия: приобрести практические навыки составления, отладки и выполнения программ, написанных языком ассемблера с созданием файлов и записи результатов в них для МП платформы x64 среды masm64.

В отчете представить:

– номер и название лабораторной работы;

– задание;

– текст программы с комментариями к каждой строке программы;

– результат выполнения программы (скриншоты упрощенных окон (MessageBox) и отладчиков);

– алгоритм программы (клавиша «G» в окне отладчика) или ее основная часть, если алгоритм достаточно большой;

– особенности выполнения, где описать общий алгоритм (последовательность) выполнения и особенности программы (на что необходимо с вашей точки зрения уделить внимание).

Постановка задачи

Выполнить задание в 2-х вариантах: на masm32 и на masm64.

Создать файл и в него записать полученные результаты.

Запустить внешний файл с информацией об авторе и кратким словесным алгоритмом.

Использовать отдельный файл с подключаемыми библиотеками.

Согласно номеру студента в группе выбрать вариант задания и написать на ассемблере программу вычисления одного из выражений.

Задание 1

1. Проанализировать массив данных из 15 элементов. Подсчитать и сохранить количество элементов массива, если их значение меньше или больше 132 и количество элементов массива, значения которых равняются 132. Вывести соответствующие сообщения.

2. Проанализировать массив данных из 10 элементов. Складывать элементы массива до тех пор, пока значение суммы не превысит 512. Сохранить номер элемента, на котором состоялось превышение

значения. Если сумма элементов не достигла значения 512, то выдать соответствующее сообщение.

3. Проанализировать массив данных из 16 элементов. Элементами массива является и числа 5, 32, 64, 96 и 128. Подсчитать и вывести на экран количество повторений каждого элемента.

4. Проанализировать массив данных из 12 элементов. Создать массив из элементов первого массива, которые равняются 128. Прервать выполнение программы, если будет найдено 5 элементов со значением 128. Вывести соответствующие сообщения.

5. Проанализировать 2 массива, которые состоят из 15 элементов каждый. Подсчитать количество элементов первого массива, которые имеют равные значения во втором массиве. Вывести соответствующие сообщения.

6. Проанализировать массив данных из 14 элементов. Подсчитать количество элементов, значение которых равняется 55h. Счет прервать, если количество элементов превысит 3. Вывести соответствующие сообщения.

7. Проанализировать массив данных из 15 элементов. Элементами массива является также числа 10, 20, 30. Подсчитать количество повторений каждого элемента. Вывести соответствующие сообщения.

8. Для функции $Y = 40X + 65$ найти первое значение аргумента, при котором значение функции превысит 1024. Начальное значение аргумента $X = 20$. Вывести соответствующие сообщения.

9. Для функции $Y = 40X + 10$ получить первое значение, которое превышает 512, начиная с $X = 1$. Вывести значение аргумента и функции.

10. Для функции $7X + 85$ найти первое значение аргумента, при котором младшие цифры результата выполнения функции равняются 155. Вывести результат выполнения функции и его аргумент.

11. Задан массив *A из* $N = 45$ элементов. Написать программу формирования массива *B* из последних 13 элементов массива *A,* которые равны нулю.

12. Задан массив *A из* $N = 55$ элементов. Написать программу определения количества элементов массива *A,* которые удовлетворяют условию $L \geq Ai \geq M$, где $L = 9$ та $M = 30$.

13. Задан массив *A из* $N = 15$ элементов. Написать программу определения минимального из положительных элементов массива *A.*

14. Задан массив *A из* N = 20 элементов. Написать программу определения максимального из отрицательных элементов массива *A.*

15. Задан массив *A из N* = 25 элементов. Написать программу формирования массива *B* с первых 10 положительных элементов массива *A*.

Задание 2

Согласно последней цифре номера студента в группе выбрать вариант задания и написать программу:

1. Определение версии программы.
2. Определение даты последнего изменения файла.
3. Определение даты последнего доступа к файлу.
4. Определение даты создания файла.
5. Определение типа файла.
6. Определение свойств файла.
7. Удаление каталога с подкаталогами.
8. Определение файлов в определенной директории.
9. Определение пути к личной программе.
10. Копирование директории.

При написании программы можно использовать ссылку на сайт http://www.desksoft.ru/ с выбором иконки сайта с подписью DRKB.

Примеры решения

Решение на **masm32**

Пример 4.6.1. Проанализировать массив данных из 15 элементов. Элементами массива является и числа 10, 20, 30 и 128. Подсчитать и вывести на экран количество повторений каждого элемента (программа 4.6.1). Пример выполнен без запуска внешнего файла и без файла с подключаемыми библиотеками.

Программа 4.6.1. Работа с файлами на masm32:

```
; Проанализировать массив данных из 15 элементов.
; Вывести на экран количество повторений каждого элемента.
.686
.model flat, stdcall
option casemap :none
include \masm32\include\windows.inc
include \masm32\macros\macros.asm
uselib kernel32,user32
BSIZE equ 13 ;
.data
mas_a db 10,15,30,20,180,180,20,20,30,30,10,180,30,180,5
```

```
        mas_len equ $-mas_a
        count_10 dd 0
        count_20 dd 0
        count_30 dd 0
        count_180 dd 0

fName   BYTE "LR5-32-1.txt",0
        fHandle DD ?  ;
        cWritten DD ? ;
        fmt2 db "%d",0
        buf2 db BSIZE dup(?);
        hFile dd ?,0

fmt db "Дан массив данных из 15 элементов. Элементами массива
являются числа 10, 20, 30 і 180.",
"Подсчитать количество повторений каждого элемента.",10,10, "Число
повторений:",0ah,
   "10 - %d",0ah,"20 - %d",0ah,"30 - %d",0ah,"180 - %d",0ah,"Показать
автора: ОК",0
   titl2 db "Автор",0
   inf2 db "Автор: Рысованый А.Н.,НТУ ХПИ",10,10,
   "Алгоритм: сначала выводятся сообщения оператору, а после их
закрытия создается файл с информацией.",0

   titl1 db " Файлы",0
   buf db ?,0
.code
_st:
        lea edi,mas_a
        mov ecx,mas_len
 cycle:
        movzx eax,byte ptr [edi]
        cmp eax,10
        jz inc10
        cmp eax,20
        je inc20
        cmp eax,30
        je inc30
        cmp eax,180
        je inc180
         jmp skip
inc10:
        inc count_10
        jmp skip
inc20:
```

```
        inc count_20
        jmp skip
inc30:
        inc count_30
        jmp skip
inc180:
        inc count_180
       skip:
        inc edi
        loop cycle
invoke wsprintf,addr buf,addr fmt,count_10,count_20,count_30,count_180
invoke MessageBox,0,addr buf,addr titl1,
MB_OKCANCEL+MB_ICONQUESTION
       .IF eax == IDOK
invoke MessageBox,0,addr inf2,addr titl2,MB_ICONINFORMATION
       .ENDIF
        xor edi,edi
        push edi
invoke wsprintf,addr buf2[edi],addr fmt2,count_10;
        pop edi
        add edi,4
        push edi
invoke wsprintf,addr buf2[edi],addr fmt2,count_20;
        pop edi
        add edi,4
        push edi
invoke wsprintf,addr buf2[edi],addr fmt2,count_30;
        pop edi
        add edi,4
invoke wsprintf,addr buf2[edi],addr fmt2,count_180;
invoke CreateFile,ADDR fName,GENERIC_WRITE,0,0,
CREATE_ALWAYS,FILE_ATTRIBUTE_ARCHIVE,0
       mov hFile, eax              ;
       invoke WriteFile,hFile,ADDR buf2,BSIZE,ADDR cWritten,0

       invoke CloseHandle,hFile
       invoke ExitProcess,0
       end _st
```

Особенность: обратите внимание на формирование буфера для вывода чисел в файл. Он формируется отдельно от буфера для вывода чисел через функцию MessageBox.

Результат выполнения программы 4.6.1 приведен на рис. 4.6.1 – 4.6.2.

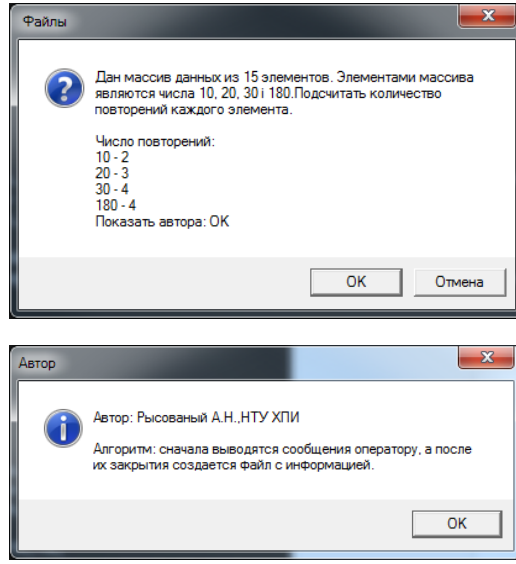

Рис. 4.6.1. Сообщения программы на masm32

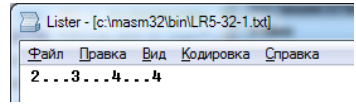

Рис. 4.6.2. Результат выполнения программы на masm32

Решение на **masm64**

При написании программы необходимо использовать вызов внешнего файла через

invoke WinExec,addr szFileName,SW_SHOW

до первого вызова функции MessageBox.

Можно использовать для запуска :

invoke WinExec,ADDR szRunDLL,SW_HIDE

или invoke CreateProcess,ADDR programname,0,0,0,FALSE,\
NORMAL_PRIORITY_CLASS, 0,0,ADDR startInfo,ADDR processInfo
или invoke ShellExecute, 0, chr$(«open»), chr$(«hello_world.exe»), 0, 0, SW_SHOWNORMAL

и много еще чего… Но не забывайте подключать необходимые библиотеки.

В приведенной ниже программе рассматривается процесс обработки каждой ячейки отдельно и, поэтому, содержит избыточный код. Но этот недостаток исключен при дальнейшем рассмотрении этого материала.

Программа 4.6.2. Передача параметров через таблицу адресов на masm64:

```
        title создание и запись в файл; masm64
        include win64a.inc ; библиотеки для подключени¤
        BSIZE equ 32
.data
        mas_a db 10,15,30,20,180,180,20,20,30,30,10,180,30,180,5
        mas_len equ $-mas_a
        count_10 dq 0
        count_20 dq 0
        count_30 dq 0
        count_180 dq 0
          fName  BYTE "LR5-64-1.txt",0
          fHandle dq ?  ;
          cWritten dq ? ;
          fmt2 db "%d",0
          buf2 dq 4 dup(0);  b BSIZE dup(?);
          hFile dq ?,0
fmt db "Дан массив данных из 15 элементов. Элементами массива
являются числа 10, 20, 30 i 180.",
"Подсчитать количество повторений каждого элемента.",10,10, "Число
повторений:",0ah,
"10 - %d",0ah,"20 - %d",0ah,"30 - %d",0ah,"180 - %d",0ah,"Показать
автора: ОК",0
          titl2 db "Автор",0
inf2 db "Автор: Рысованый А.Н.,НТУ ХПИ",10,10,
"Алгоритм: сначала выводятся сообщения оператору, а после их
закрытия создается файл с информацией.",0
          titl1 db " Файлы",0
          buf dq ?,0
          szFileName db "c:\Masm64\bin\LR5-64-1-sms.exe",0

.code
WinMain proc
sub rsp,28h; выравнив. стека 28h=32d+8;  8 - возврат
  mov rbp,rsp ; сохранение выравненного значения стека
           lea rdi,mas_a
```

```
          mov rcx,mas_len
 cycle:
          movzx rax,byte ptr [rdi]
          cmp rax,10
          jz inc10
          cmp rax,20
          je inc20
          cmp rax,30
          je inc30
          cmp rax,180
          je inc180
           jmp skip
inc10:    inc count_10
          jmp skip
inc20:     inc count_20
          jmp skip
inc30:    inc count_30
          jmp skip
inc180:    inc count_180
skip:        inc rdi
          loop cycle
          invoke WinExec,addr szFileName,SW_SHOW
invoke wsprintf,addr buf,addr fmt,count_10,count_20,count_30, count_180
invoke MessageBox,0,addr buf,addr titl1,
MB_OKCANCEL+MB_ICONQUESTION

          .if rax == IDOK
invoke MessageBox,0,addr inf2,addr titl2,MB_ICONINFORMATION
          .endif

           lea rdi,buf2
            push rdi
invoke wsprintf,addr [rdi],addr fmt2,count_10;
            pop rdi
            add rdi,type buf2  ;8
            push rdi
 invoke wsprintf,addr [rdi],addr fmt2,count_20;
          pop rdi
          add rdi,8
          push rdi
invoke wsprintf,addr [rdi],addr fmt2,count_30;
          pop rdi
          add rdi,8
invoke wsprintf,addr [rdi],addr fmt2,count_180;
```

```
invoke CreateFile,ADDR fName,GENERIC_WRITE,0,0,CREATE_ALWAYS,
FILE_ATTRIBUTE_ARCHIVE,0
        mov hFile, rax              ;
        invoke WriteFile,hFile,ADDR buf2,BSIZE,ADDR cWritten,0
        invoke CloseHandle,hFile
        invoke ExitProcess,0
        WinMain endp
        end
```

Программа LR5-64-1-sms.asm вызова и выполнения внешнего файла компилируется до компиляции основной программы. Ее текст следующий:

```
        title Файл. Запуск внешнего файла в masm64
include win64a.inc ; библиотеки, см
http://dsmhelp.narod.ru/environment.htm
        .data
titl db "Сообщение о завершении программы",0; титул упрощенного окна
inf1 db "Запущена внешняя программа через", 10,
"invoke WinExec,addr szFileName,SW_SHOW",10,"в masm64",10,10,
"Автор программы: Рысованый А.Н., каф. ВТП, НТУ ХПИ",0
  .code
WinMain proc
sub rsp,28h; выравнив. стека 28h=40d=32d+8;  8 - возврат
mov rbp,rsp ; сохранение выравненного значения стека
        invoke MessageBox,0,addr inf1,addr titl, MB_ICONINFORMATION;
        invoke ExitProcess,0
        WinMain endp
        end
```

Результат выполнения программы LR5-64-1-sms.exe приведен на рис. 4.6.3.

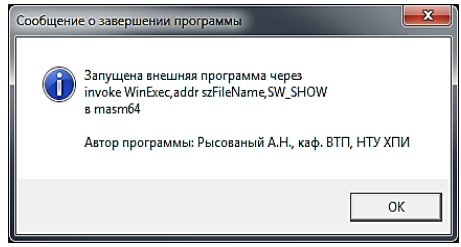

Рис.4.6.3. Результат выполнения программы на masm64

Результат выполнения программы 4.6.2 приведен на рис. 4.6.4.

При нажатии на кнопку «OK» выводится упрощенное окно (рис. 4.6.5).

После выполнения программы создается текстовый файл с именем LR5-32-1.txt (рис. 4.6.6).

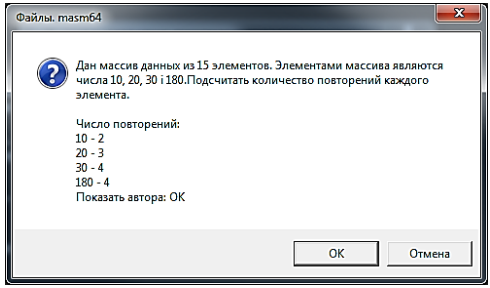

Рис. 4.6.4. Результат выполнения программы на masm64

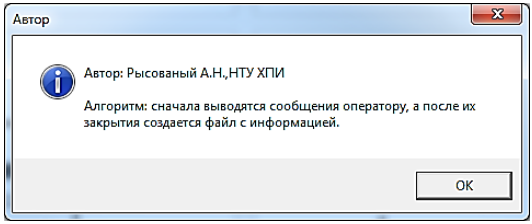

Рис. 4.6.5. Результат выполнения программы

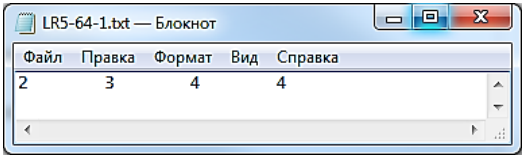

Рис.4.6.6. Содержимое текстового файла

Алгоритм (граф) выполнения программы можно получить в окне отладчика x64Dbg, если нажать правую кнопку мышки и выбрать – «Граф» (рис. 4.6.7).

Однако, программа будет более удобночитаемой, если использовать упрощенную сегментацию (с несколькими секциями data) и, самое главное, преобразованием в одной функции

invoke wsprintf,addr [rdi],addr fmt2,count_10,count_20,count_30,count_180;

В этом случае фрагмент кода будет таким:

```
.data
        fmt2 db 4 dup("%x "),0    ; строка форматирования
.code
         lea rdi,buf2
invoke wsprintf,addr [rdi],addr fmt2,count_10,count_20,count_30,count_180;
```

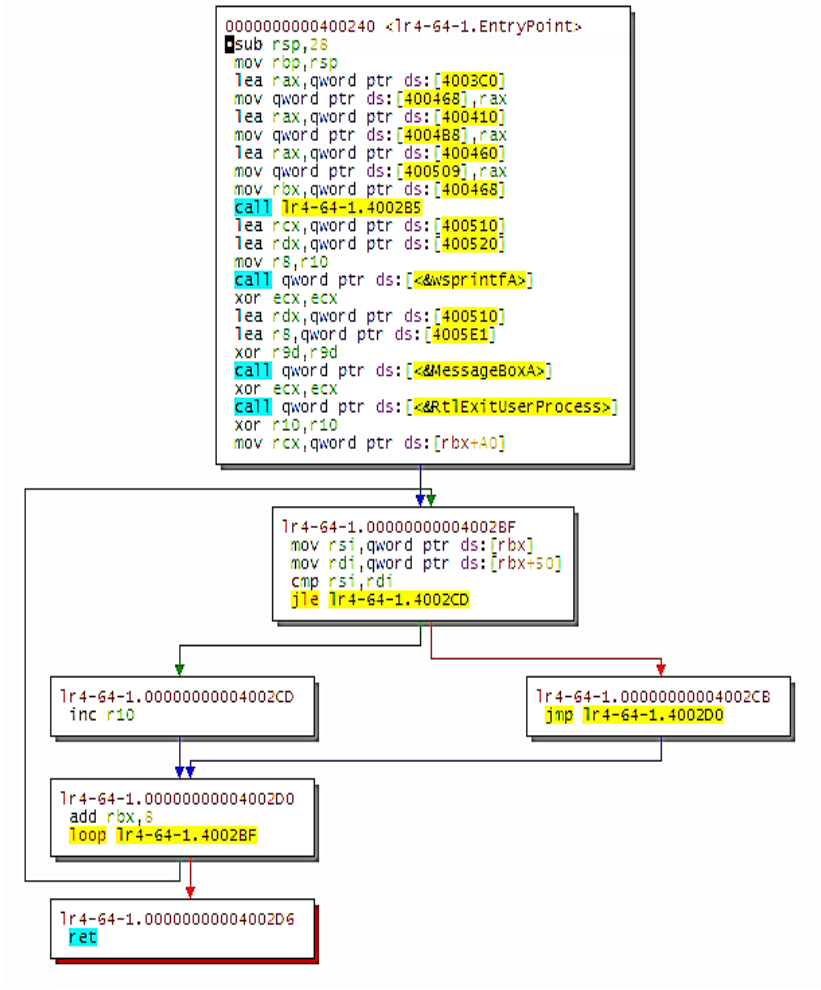

Рис. 4.6.7. Алгоритм (граф) программы в окне отладчика x64Dbg

ЛАБОРАТОРНАЯ РАБОТА 7
СОПРОЦЕССОР

Цель занятия: приобрести практические навыки составления, отладки и выполнения программ, написанных языком ассемблера для сопроцессора платформ x86/x64 в средах masm32 и masm64.

В отчете представить:

– номер и название лабораторной работы;

– задание;

– текст программы с комментариями к каждой строке программы;

– результат выполнения программы (скриншоты упрощенных окон (MessageBox) и отладчиков);

– алгоритм программы (клавиша «G» в окне отладчика);

– особенности выполнения, где описать общий алгоритм (последовательность) выполнения и особенности программы (на что необходимо с вашей точки зрения уделить внимание).

Постановка задачи

Выполнить задание в 2-х вариантах: на masm32 и на masm64.

На masm32 выполнить задание в двух вариантах: при помощи команд сопроцессора и их же при помощи встроенных masm32-функций (2 программы).

На masm64 выполнить задания при помощи команд сопроцессора (встроенные функции преобразования для последующего вывода на сегодня отсутствуют)

Согласно номеру студента в группе выбрать вариант задания и написать на ассемблере программу вычисления одного из выражений.

Методические рекомендации

Для отладке окончательного варианта программы сначала необходимо наладить часть программы с получением одного результата и его выводом. На втором этапе – получение массива результатов. На третьем – вывод всего массива.

Задание 1

Исследовать выполнение арифметических операций сопроцессором. Результат вывести на экран функцией MessageBox.

1. Вычислить 6 значений функции $Y_n = 25x^3 - 2{,}1$ (x изменяется с шагом 0.2).

2. Вычислить 5 значений функции $Y_n = 7x^3/(2x^2 + 1{,}6)$ (x изменяется от 1 с шагом 4).

3. Вычислить 4 значения функции $Y_n = 37/(2x^2 + 7{,}3)$ (x изменяется от 1 с шагом 2).

4. Вычислить 5 значений функции $Y_n = 5{,}1x^2 + 5{,}3$ (x изменяется от 4,7 с шагом 3). Результат округлить к ближайшему целому числу и вывести.

5. Вычислить 6 значений функции $Y_n = 4x/(x + 5)$ (x изменяется от 3 с шагом 1,25). Результат округлить к целому числу.

6. Вычислить 4 значения функции: $Y_n = 125/(3x^2 - 1{,}1)$ (x изменяется от 3 с шагом 1,5). Результат округлить в меньшую сторону.

7. Вычислить 4 значения функции $Y_n = 2{,}5x^2 - 3{,}2$ (x изменяется от 4 с шагом 1).

8. Вычислить 3 значения функции $Y_n = 25x^2 + 2{,}1$ (x изменяется от 3 с шагом 2,5).

9. Вычислить 4 значения функции $Y_n = 150/(x^2 - 7)$ (x изменяется от 2 с шагом 3,1).

10. Вычислить 6 значений функции $Y_n = 256/(3x^2 + 31)$ (x изменяется от 2 с шагом 3).

11. Найти первое значение аргумента функции $Y = 7(x + 0{,}3)$, при котором младшие целые цифры результата выполнения функции будут равны 15 (x изменяется от 2 с шагом 3,5).

12. Найти целое значение аргумента, при котором функция $Y = 10/(x^3 + 1{,}7)$ станет меньше 0,3 (x изменяется от 2 с шагом 2,5).

13. Найти значение x, при котором функция $Y = 3^x/4 - 6$ будет равна 12,5.

14. Найти значение x, при котором выполняется функция $8 \operatorname{arctg} 0{,}1 + \operatorname{arctg} x = \pi/4$.

15. Определить номер (x) элемента функции $x_n = 2^x + 5$, при котором сумма элементов превысит 12 000.

16. Найти значение x, при котором функция $\lg(2^x + 3)$ будет больше 2,5.

17. Найти целое значение аргумента, при котором функция $Y = 5{,}6^x/3x^2$ будет больше 125.

18. Найти целое значение аргумента, при котором функция $Y = 2^x + 30$ будет больше 100.

19. Найти первое значение аргумента функции $Y = 9(x^2 + 0{,}6)$, при котором младшие целые цифры результата выполнения функции будут равны 12 (x изменяется от 3 с шагом 5,5).

20. Найти целое значение аргумента, при котором функция $Y = 2^x/5 + 4$ превысит 400.

21. Вычислить 5 значений функции $Y = 5{,}2 * \ln(\sin x)$ (x изменяется в градусах с шагом 1,5).

22. Вычислить 4 значения функции $Y = 5^x + \cos x$ (x изменяется от 0,4 с шагом 0,15).

23. Вычислить 6 значений функции $Y = 4{,}5 \lg(\operatorname{tg} x)$ (x изменяется в градусах с шагом 10).

24. Вычислить 6 значений функции $Y = 12^x - \sin x$ (x изменяется с шагом 0,05).

25. Вычислить 8 значений функции $Y = 3{,}3 * \log_2(x^2 + 1)$ (x изменяется с шагом 0,3).

26. Вычислить 4 значения функции $Y = 5{,}1(\sin x)$ (x изменяется в градусах с шагом 8,5).

27. Вычислить 6 значений функции $Y = \lg(x^2 + \sqrt{x})$ (x изменяется с шагом 0,3).

28. Вычислить 5 значений функции $Y = 4{,}3(x^2 + 1)$ (x изменяется с шагом 1,7).

29. Вычислить 6 значений функции $Y = 6{,}2 \lg(\cos x)$ (x изменяется в градусах с шагом 1,5).

30. Вычислить 7 значений функции $Y = 3{,}1 + 2/\cos x$ (x изменяется в градусах от 10 с шагом 8).

Примеры решения

Решение на **masm32**

Пример 4.7.1. masm32. Вычислить 5 значений функции Y = 4,3(x^2 + 1) (x изменяется с шагом 1,7)

Программа 4.7.1:

```
; masm32. Вычислить 5 значений функции Y = 4,3(x^2 + 1)
; (x изменяется с шагом 1,7)
      .686
      .model flat, stdcall
      option casemap:none
  include C:\masm32\include\windows.inc
  include C:\masm32\include\kernel32.inc
  include C:\masm32\include\user32.inc
```

```
  include C:\masm32\include\fpu.inc
  includelib C:\masm32\lib\kernel32.lib
  includelib C:\masm32\lib\user32.lib
  includelib C:\masm32\lib\fpu.lib
.data
        CrLf equ 0A0Dh
        _y1 dt 0.0
        _y2 dt 0.0
        _y3 dt 0.0
        _y4 dt 0.0
        _y5 dt 0.0
        _x DWORD 3.0
        _op1 DWORD 4.3
        _op2 DWORD 1.0
        _zero DWORD 0.0
        _step DWORD 1.7
info db "Автор программы: Рысованый А.Н., каф. ВТП, НТУ ХПИ.",10,10,
"Y = 4,3(x^2 + 1), где x изменяется с  шагом 1.7",10,10,
         "y1 = "
        _res1 db 14 DUP(0),10,13
        db "y2 = "
        _res2 db 14 DUP(0),10,13
        db "y3 = "
        _res3 db 14 DUP(0),10,13
        db "y4 = "
        _res4 db 14 DUP(0),10,13
        db "y5 = "
        _res5 db 15 DUP(0),10,13
        ttl db "Обработка чисел на сопроцессоре в цикле",0

.code
_start:
    finit
    mov ecx, 5
m1:     fld _x
        fmul _x
        fadd _op2
        fmul _op1
        fld _x
        fadd _step
        fstp _x
    loop m1
        fstp _y5
        fstp _y4
        fstp _y3
```

```
        fstp _y2
        fstp _y1
invoke FpuFLtoA,offset _y1,10,offset _res1,SRC1_REAL or SRC2_DIMM
        mov word ptr _res1 + 14, CrLf
invoke FpuFLtoA,offset _y2,10,offset _res2,SRC1_REAL or SRC2_DIMM
        mov word ptr _res2 + 14, CrLf
invoke FpuFLtoA,offset _y3,10,offset _res3,SRC1_REAL or SRC2_DIMM
        mov word ptr _res3 + 14, CrLf
invoke FpuFLtoA,offset _y4,10,offset _res4,SRC1_REAL or SRC2_DIMM
        mov word ptr _res4 + 14, CrLf
invoke FpuFLtoA,offset _y5,10,offset _res5,SRC1_REAL or SRC2_DIMM
invoke MessageBox, 0, offset info, offset ttl, MB_ICONINFORMATION
        invoke  ExitProcess, 0
    end _start
```

Результат выполнения программы представлен на рис. 4..7.1.

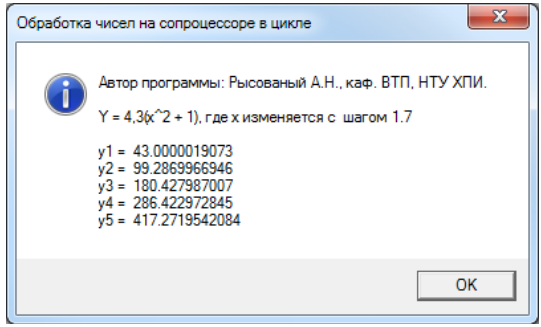

Рис. 4.7.1. Результат выполнения программы

Пример 4.7.2. Программирование уравнения Y = lg(x^2 + sqrt(x)) -- 6 results with x - 0.3 each с использованием встроенных masm32-функций сопроцессора.

Программа 4.7.2.

```
; masm32. Y = lg(x^2 + sqrt(x)) – 6
; results with x - 0.3 each  с использованием
; встроенных masm32-функций сопроцессора:
    .686
    .model flat, stdcall
    option casemap :none
  include \masm32\include\windows.inc
  include \masm32\macros\macros.asm
  uselib kernel32,user32,fpu,masm32
```

```
.data
        x real10 1.0
        incr real10 0.3
        szfmt db "y%d = ",0
        result db 260 dup(?)
        buf db 100 dup(?)
        titl db "Встроенные masm32-функции сопроцессора",0
        count dd 1
        uravnenie db "Y = lg(x^2 +  sqrt(x)) -- 6 results with x - 0.3 each",0
        buf2 db 100 dup(?)
        newLine db 0ah,0
        avt db "Автор: Рысованый А.Н., каф. ВТП, НТУ ХПИ"
.code ; Y = lg(x^2 +  sqrt(x)) -- 6 results with x +- 0.3 each
st1:
        mov edi, 6      ; количество циклов получения результатов
        invoke szCatStr,addr result,addr uravnenie
        invoke szCatStr,addr result,addr newLine
        invoke szCatStr,addr result,addr newLine
finit
@1:
invoke FpuMul,addr x,addr x,0,SRC1_REAL or SRC2_REAL or DEST_FPU
        invoke FpuSqrt,addr x,1,SRC1_REAL or DEST_FPU
        invoke FpuAdd,0,1,0,SRC1_FPU or SRC2_FPU or DEST_FPU
        invoke FpuLnx,0,0,SRC1_FPU or DEST_FPU
        invoke wsprintf,addr buf2,addr szfmt,count
invoke FpuFLtoA,0,5,addr buf,SRC1_FPU or SRC2_DIMM
     invoke szCatStr,addr result,addr buf2
     invoke szCatStr,addr result,addr buf
     invoke szCatStr,addr result,addr newLine
invoke FpuAdd,addr x,addr incr,addr x,SRC1_REAL or SRC2_REAL or
DEST_MEM
        inc count
        dec edi
jnz @1
     invoke szCatStr,addr result,addr newLine
     invoke szCatStr,addr result,addr avt
       invoke MessageBox,0,addr result,addr titl,MB_ICONINFORMATION
       invoke ExitProcess, 0
       end st1
```

После задания количества циклов вычисления результатов командой

```
        mov edi, 6
```

начинает сразу формироваться буфер для вывода. Для этого используется функция добавление в строку буфера (функция szCatStr).

При первом вызове функции

```
invoke szCatStr,addr result,addr uravnenie
```

в буфер записывается выражение уравнения. При следующих двух вызовах

```
invoke szCatStr,addr result,addr newLine
invoke szCatStr,addr result,addr newLine
```

в буфер заносятся 2 симола перевода строки.

После них в цикле для разных значений шага расчета производится вычисление. Каждый полученный результат преобразуется из числа в символ функцией

```
invoke FpuFLtoA,0,5,addr buf,SRC1_FPU or SRC2_DIMM
```

и записывается в конец буфера.

Заканчивается процесс формирования буфера строками программы

```
invoke szCatStr,addr result,addr newLine
invoke szCatStr,addr result,addr avt
```

Первая строка для наглядности вывода записывает в буфер символ перевода коретки, а вторая вставляет в буфер информацию об авторе программы.

Результат выполнения программы представлен на рис. 4.7.2.

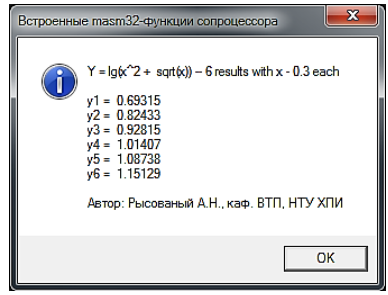

Рис. 4.7.2. Результат выполнения программы

Решение на **masm64.**

Программа 4.7.3:

```
; Вычислить 5 значений функции Y = 4,3(x^2 + 1)
; (x изменяется с шагом 1,7).
include win64a.inc ; библиотеки, см
http://dsmhelp.narod.ru/environment.htm
.data
    _x dq 3.0
```

```
    _op1 dq 4.3
    _op2 dq 1.0
    _zero dq 0.0
    _step dq 1.7
    res1 dq 0
    res2 dq 0
    res3 dq 0
    res4 dq 0
    res5 dq 0
    tit1 db "masm64. Выполнение уравнения на сопроцессоре.",0
    buf dq 5 dup(0)
ifmt db "Вывод результата Y = 4,3(x^2 + 1) через
MessageBox:",10,10,"Результат: %d, %d, %d, %d, %d",0ah,0ah,
    "Автор: Рысованый А.Н., каф. ВТП, НТУ ХПИ",0
.code
WinMain proc
sub rsp,28h; стек 28h=40d=32d+8;  8 - возврат
    mov rbp,rsp ; сохранение выравненного значения стека
finit
    mov ecx, 5
m1:  fld _x
       fmul _x
       fadd _op2
       fmul _op1
       fld _x
       fadd _step
       fstp _x
     loop m1
fisttp res1 ; сохранение в 64-разрядных ячейках памяти с округлением
    fisttp res2
    fisttp res3
    fisttp res4
    fisttp res5

    invoke wsprintf,ADDR buf,ADDR ifmt, res5,res4,res3,res2,res1
    invoke MessageBox,0,addr buf,addr tit1,MB_ICONINFORMATION;
      invoke  ExitProcess,0
      WinMain endp
      end
```

Программа с именем WinMain оформлена в виде прогцедуры без параметров. После выравнивания стека и сохранения этого значения командой finit производится инициализация сопроцессора.

По условию задания необходимо посчитать 5 значений уравнения. Поэтому число циклов расчета заносится строкой кода mov ecx, 5 в регистр ecx.

Основная часть программы выполняется в цикле m1. Шаг изменения хначения x хранится в переменной _step.

Команда loop состоит из двух последовательно выполняемых команд: первая производит декрементирование регистра ecx, а вторая производит перевод ветви выполнения кода на метку m1, если в регистре ecx не ноль.

Особенностью вывода через функцию MessageBox является применение команд технологии SSE3 – команды fisttp. Команда **FISTTP** выполняет преобразование вещественного числа в целое с сохранением целочисленного значения и округлением в сторону нуля. В связи с тем, что при расчете 5-ти чисел они сохраняются в стеке сопроцессора (стек не переполняется), то блок этих команд вынесен за этот цикл:

```
fisttp res1
fisttp res2
fisttp res3
fisttp res4
fisttp res5
```

Результат выполнения программы представлен на рис. 4.7.3.

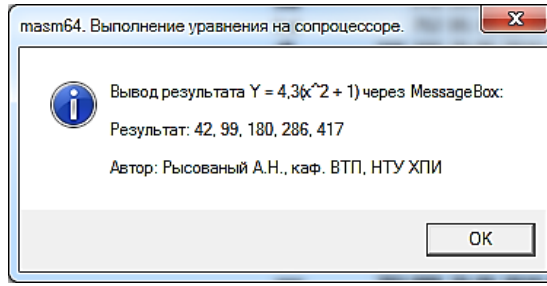

Рис. 4.7.3. Результат выполнения программы

В связи с тем, что в программе применены команды fisttp сохранения в 64-разрядную ячейку памяти с округлением, то результат немного не совпадает с истинным значением.

Алгоритм (граф) выполнения программы можно получить в окне отладчика x64Dbg, если нажать правую кнопку мышки и выбрать – «Граф» (рис. 4.7.4).

Рис. 4.7.4. Алгоритм (граф) программы в окне отладчика x64Dbg

ПРИЛОЖЕНИЯ

Приложение 1

СПИСОК НАБОРОВ КОМАНД

Год появления	Тип процессора	Число команд набора	Общее число команд	Назначение
1979	i8086	170		Начальный набор команд x86
1985	i386	50	220	Переход к архитектуре IA-32
1997	Pentium/MMX	57	277	MMX-команды
1999	Pentium III	70	347	SSE-команды
2000	Pentium 4 Northwood	144	491	SSE2-команды
2004	Pentium 4 Prescott	13	504	SSE3-команды
		10	514	Intel VT-x
2006	Core2 Duo (65 нм)	32	546	SSSE3-команды
2007	Penryn (45 нм)	47	593	SSE4.1
2008	Core i7 (45нм)	7	600	SSE4.2
2008	Sandy Bridge (2011)	683/53	1283	AVX
2009	Core i5 (32 нм)	6	1289	AES-NI
2011	Haswell (22 нм, 2013)			AVX2
2013	Xeon Phi; Cannon Lake (10/14 нм, 2018)			AVX-512 (AVX3.1)

В первоначальный набор AVX-512 входит восемь групп инструкций:

AVX-512 Conflict Detection Instructions (CDI);
AVX-512 Exponential and Reciprocal Instructions (ERI);
AVX-512 Prefetch Instructions (PFI);
AVX-512 Vector Length Extensions (VL);
AVX-512 Byte and Word Instructions (BW);
AVX-512 Doubleword and Quadword Instructions (DQ);
AVX-512 Integer Fused Multiply Add (IFMA);
AVX-512 Vector Byte Manipulation Instructions (VBMI).

Для глубинного обучения (англ. Deep learning) предназначены группы: AVX512_4VNNIW и AVX512_4FMAPS) https://geektimes.ru/post/281518/).

Приложение 2

КОМАНДЫ ОБЩЕГО НАЗНАЧЕНИЯ

Команды передачи данные
MOV – пересылка;
CMOVxx – условная пересылка;
XCHG – обмен значений между регистрами;
BSWAP – перестановка байтов;
XADD – обмен и сложение;
CMPXCHG – сравнение и обмен;
CMPXCHG8B – сравнение и обмен 8 байтов;
PUSH – поместить значение в стек;
POP – взять значение из стека;
PUSHA/PUSHAD – поместить в стек значение регистров общего назначения;
POPA/POPAD – взять значение регистров общего назначения из стека;
CWD – преобразовать Word в Dword;
CDQ – преобразовать Dword в Qword;
CBW – преобразовать Byte в Word;
CWDE – превратить Word в Dword в регистре rax;
MOVSX – расширить с учетом знака;
MOVZX –расширить нулевым значением.

Двоичные арифметические команды
ADD – сложение;
ADC – сложение с учетом переноса;
SUB – вычитание;
SBB – вычитание с заемом;
IMUL – знаковое умножение;
MUL – беззнаковое умножение;
IDIV – знаковое деление;
DIV – беззнаковое деление;
INC – инкремент;
DEC – декремент;
NEG – изменение знака;
CMP – сравнение.

Логические команды
AND – побитовое логическое И;

OR – побитовое логическое ИЛИ;
XOR – побитовое логическое “Исключающее ИЛИ ”;
NOT – побитовое логическое НЕ.

Команды побитового сдвига и вращения (циклического сдвига)
SAR – арифметический сдвиг вправо;
SHR – логический сдвиг вправо;
SAL/SHL – арифметический/ логический сдвиг влево;
SHRD – двойной сдвиг вправо;
SHLD – двойной сдвиг влево;
ROR – сдвиг вправо;
ROL – сдвиг влево;
RCR – сдвиг вправо через флажок переноса;
RCL – сдвиг влево через флажок переноса.

Команды работы с битами и байтами
BT – проверка бита;
BTS – проверка и установка бита;
BTR – проверка и сброс бита;
BTC – проверка и инверсия бита;
BSF – проверка бита в прямом направлении;
BSR – проверка бита в обратном направлении;
SETxx – установить значение байта в зависимости от флажка;
TEST – логическое сравнение.

Команды передачи управления
JMP – безусловный переход;
Jxx – условен переход;
JCXZ/JECXZ – переход, если cx/rcx равняется 0;
LOOP – цикл со счетчиком в ecx;
LOOPZ/LOOPE – цикл со счетчиком в ecx и выходе при нуле/равенстве;
LOOPNZ/LOOPNE – цикл со счетчиком в ecx и выход при не нуле/неравенстве;
CALL – вызов подпрограммы;
RET – возвращение из подпрограммы;
IRET – возвращение из прерывания;
INT – вызов программного прерывания;
INTO – вызов прерывания по переполнению;
BOUND – переход при выходе значения за заданные рамки;
ENTER – высокоуровневый вход в процедуру;

LEAVE – высокоуровневый выход из процедуры.

Команды работы со строками (последовательностями)
MOVS – пересылка сроки;
MOVSB –- пересылка строки байтов;
MOVSW – пересылка строки слов;
MOVSD – пересылка строки двойных слов;
CMPS – сравнение строки;
CMPSB – сравнение строки байтов;
CMPSW – сравнение строки слов;
CMPSD – сравнение строки двойных слов;
SCAS – проверка строки;
SCASB –- проверка строки байтов;
SCASW – проверка строки слов;
SCASD – проверка строки двойных слов;
LODS – чтение строки;
LODSB – чтение строки байтов;
LODSW – чтение строки слов;
LODSD – чтение строки двойных слов;
STOS – запись строки;
STOSB – запись строки байтов;
STOSW – запись строки слов;
STOSD – запись строки двойных слов;
REP – префикс повторения: пока ecx не равняется 0;
REPE/REPZ – префикс повторения: пока равно/нуль;
REPNE/REPNZ – префикс повторения: пока не равно/не нуль;
INS – чтение из порта строки;
INSB – чтение из порта строки байтов;
INSW – чтение из порта строки слов;
INSD – чтение из порта строки двойных слов;
OUT – запись в порт строки;
OUTB – запись в порт строки байтов;
OUTW – запись в порт строки слов;
OUTD – запись в порт строки двойных слов.

Команды управления флажками
STC – установить флажок переноса;
CLC – очистить флажок переноса;
CMC – инвертировать флажок переноса;
CLD – очистить флажок направления;
STD – установить флажок направления;

LAHF – загрузить признак в ah;
SAHF – записать ah в признаки;
PUSHFD – поместить EFLAGS в стек;
POPF/POPFD – взять EFLAGS из стека;
STI – установить признак прерывания;
CLI – очистить признак прерывания.

Команды работы с сегментными регистрами
LDS – загрузить дальний указатель, используя ds;
LES – загрузить дальний указатель, используя es;
LFS – загрузить дальний указатель, используя fs;
LGS – загрузить дальний указатель, используя gs;
LSS – загрузить дальний указатель, используя ss.

Дополнительные команды
LEA – загрузить эффективный адрес;
NOP – нет операции;
Ud2 – неопределенная инструкция;
XLAT/XLATB – табличная трансляция;
CPUID – идентификация процессора.

Условия в разных условных командах (Cmovxx, Setxx, Jxx)
A/NBE (CF = 0 та ZF = 0) –"выше" или "не ниже или равняется";
AE/NB/NC (CF = 0) – "выше или равняется" или "не ниже" или нет переноса";
B/NAE/C (CF = 1) – "ниже" или "не выше или равняется" или "есть перенос";
BE/NA (CF = 1 або ZF = 1) – "ниже или равняется" или "не выше";
E/Z (ZF = 1) – " равняется" или "нуль";
NE/NZ (ZF = 0) – " не равняется" или "не нуль";
G/NLE (ZF = 0 та SF = OF) – "больше" или "не меньше или равно";
GE/NL (SF = OF) – " больше или равняется" или "не меньше";
L/NGE (SF != OF) – " меньше" или "не больше или равно";
LE/NG (ZF = 0 АБО SF != OF) – "меньше или равно" или "не больше";
NO (OF = 0) – "нет переполнения";
O (OF = 1) – "есть переполнение";
NP/PO (PF = 0) – "нет четности" или "непарное";
P/PE (PF = 1) – "есть четность" или "парное";
NS (SF = 0) – "нет знака";
S (SF = 1) – "есть знак".

Системные команды
LGDT – загрузить регистр глобальной таблицы дескрипторов (GDT);
SGDT – сохранить регистр глобальной таблицы дескрипторов (GDT);
LLDT – загрузить регистр локальной таблицы дескрипторов (GDT);
SLDT – сохранить регистр локальной таблицы дескрипторов (GDT);
LTR – загрузить регистр задания;
STR – сохранить регистр задания;
LIDT – загрузить регистр таблицы дескрипторов прерываний (IDT);
SIDT – сохранить регистр таблицы дескрипторов прерываний (IDT);
MOV – загрузка и сохранение регистров;
LMSW – загрузить слово состояния машины;
SMSW – сохранить слово состояния машины;
CLTS – очистить флаг переключения задания;
ARPL – согласовывать спросивший уровень привилегий;
LAR – загрузить права доступа;
LSL – загрузить границу сегмента;
VERR – проверить сегмент на возможность чтения;
VERW – проверить сегмент на возможность записи;
INVD – очистить кэш без обратной записи;
WBINVD – очистить кэш с обратной записью;
INVLPG – очистить элемент TLB;
LOCK – префикс: заблокировать шину;
HLT – остановить процессор;
RSM – вернуться из режима системного управления;
RDMSR – прочитать регистр, специфический для модели;
WRMSR – записать в регистр, специфический для модели;
RDPMC – прочитать счетчик производительности;
RDTSC – прочитать счетчик меток реального времени;
SYSENTER – быстрый переход к точке входа в кода на уровне привилегий 0;
SYSEXIT – быстрое возвращение из кода на уровне привилегий 0 к коду на уровне 3.

Приложение 3

КОМАНДЫ СОПРОЦЕССОРА (X87 FPU)

Команды передачи данных
FLD – загрузить вещественное число;
FST – записать вещественное число;
FSTP – записать и выгрузить вещественное число;
FILD – загрузить целое число;
FIST – записать целое число;
FISTP – записать и выгрузить целое число;
FBLD – загрузить число BCD;
FBSTP – записать и выгрузить число BCD;
FXCH – поменять регистры местами;
FCMOVE – действительное условное присвоение, если равно;
FCMOVNE – действительное условное присвоение, если не равно;
FCMOVB – действительное условное присвоение, если меньше;
FCMOVBE – действительное условное присвоение, если меньше или равно;
FCMOVNB – действительное условное присвоение, если не меньше;
FCMOVNBE – действительное условное присвоение, если не меньше или равно;
FCMOVU – действительное условное присвоение, если несравнимые;
FCMOVNU – действительное условное присвоение, если не несравнимые.

Базовые арифметические команды
FADD – сложение с вещественным числом;
FADDP – сложение с вещественным числом с выталкиванием;
FIADD – сложение с целым числом;
FSUB – вычитание вещественного числа;
FSUBP – вычитание вещественного числа с выталкиванием;
FISUB – вычитание целого числа;
FSUBR – обратное вычитание вещественного числа;
FSUBRP – обратное вычитание вещественного числа с выталкиванием;
FISUBR – обратное вычитание целого числа;
FMUL – умножение вещественного числа;
FMULP – умножение вещественного числа с выталкиванием;
FIMUL – умножение целого числа;

FDIV – деление вещественного числа;
FDIVP – деление вещественного числа с выталкиванием;
FIDIV – деление целого числа;
FDIVR – обратное деление вещественного числа;
FDIVRP – обратное деление вещественного числа с выталкиванием;
FIDIVR – обратное деление целого числа;
FPREM – вычисление остатка от деления;
FPREM1 – вычисление остатка от деления (по стандарту IEEE);
FABS – модуль числа;
FCHS – изменение знака числа;
FRNDINT – округление к целому;
FSCALE – умножение на степень двойки;
FSQRT – квадратный корень;
FXTRACT – разложение вещественного числа на мантиссу и экспоненту.

Команды сравнения
FCOM – действительное сравнение;
FCOMP – действительное сравнение с выталкиванием;
FCOMPP – действительное сравнение с двойным выталкиванием;
FUCOM – неупорядоченное сравнение действительных чисел;
FUCOMP – неупорядоченное сравнение действительных чисел с выталкиванием;
FUCOMPP – неупорядоченное сравнение действительных чисел с двойным выталкиванием;
FICOM – сравнение с целым числом;
FICOMP – сравнение с целым числом с выталкиванием;
FCOMI – действительное сравнение с установкой EFLAGS;
FUCOMI – неупорядоченное сравнение действительных чисел с установкой EFLAGS;
FCOMIP – действительное сравнение с установкой EFLAGS и выталкиванием;
FUCOMIP – неупорядоченное сравнение действительных чисел с установкой EFLAGS и выталкиванием;
FTST – проверка вещественного числа (сравнение с 0);
FXAM – определить класс вещественного числа.

Команды трансцендентных операций
FSIN – синус;
FCOS – косинус;
FSINCOS – синус и косинус;

FPTAN – частичный тангенс;
FPATAN – частичный арктангенс;
F2xm1 – 2x–1;
Fyl2x – y*log2x;
Fyl2xp1 – y*log2(x+1).

Загрузка констант
Fld1 – загрузить число +1.0;
FLDZ – загрузить число +0.0;
FLDPI – загрузить число p;
Fldl2e – загрузить число log2e;
Fldln2 – загрузить число loge2;
Fldl2t – загрузить число $\log 2_{10}$;
Fldlg2 – загрузить число $\log 10_{2}$.

Команды управления сопроцессором
FINCSTP – увеличить указатель вершины стека сопроцессора;
FDECSTP – уменьшить указатель вершины стека сопроцессора;
FFREE – освободить действительный регистр;
FINIT – инициализировать сопроцессор после проверки на ошибку;
FNINIT – инициализировать сопроцессор без проверки на ошибку;
FCLEX – очистить флаги исключений сопроцессора после проверки на ошибку;
FNCLEX – очистить флаги исключений сопроцессора без проверки на ошибку;
FSTCW – сохранить регистр управления сопроцессора после проверки на ошибку;
FNTSCW – сохранить регистр управления сопроцессора без проверки на ошибку;
FLDCW – загрузить регистр управления сопроцессора;
FSTENV – сохранить окружение сопроцессора после проверки на ошибку;
FNSTENV – сохранить окружение сопроцессора без проверки на ошибку;
FLDENV – загрузить окружение сопроцессора;
FSAVE – сохранить состояние сопроцессора после проверки на ошибку;
FNSAVE – сохранить состояние сопроцессора без проверки на ошибку;
FRSTOR – возобновить состояние сопроцессора;
FSTSW – сохранить регистр состояния сопроцессора после проверки на ошибку;

FNSTSW – сохранить регистр состояния сопроцессора без проверки на ошибку;
WAIT/FWAIT – ожидать завершения работы сопроцессора;
FNOP – нет операции сопроцессора.

Команды управления состоянием сопроцессора и SIMD
FXSAVE – сохранить состояние сопроцессора и блока SIMD;
FXRSTOR – возобновить состояние сопроцессора и блока SIMD.

СПИСОК ЛИТЕРАТУРЫ

1. Кравець В. О. Системне програмування. Асемблер під Win32 API : навч. посіб. / В. О. Кравець, О. М. Рисований. – Харків : НТУ "ХПІ", 2008. – 512 с.

2. Крупник А. Асемблер. Самоучитель / А. Крупник. – СПб. : Питер, 2005. – 235 с.

3. Магда Ю. С. Ассемблер для процессоров Intel Pentium / Ю. С. Магда. – СПб. : Питер, 2006. – 410 с.

4. Методичні вказівки до виконання та оформлення курсового проекту з курсу «Системне програмування» для студентів спеціальностей: 7.05010201 «Комп'ютерні системи та мережі», 7.05010202 «Системне програмування», 7.05010203 «Спеціалізовані комп'ютерні системи» / уклад. О. М. Рисований – Харків : НТУ «ХПІ», 2013. – 184 с.

5. Методичні вказівки до виконання лабораторних робіт з курсу «Системне програмування» для студентів спеціальностей: 7.05010201 «Комп'ютерні системи та мережі», 7.05010202 «Системне програмування», 7.05010203 «Спеціалізовані комп'ютерні системи» / уклад. О. М. Рисований. – Харків : НТУ «ХПІ», 2013. – 216 с.

6. Рисований О. М. Системне програмування : підручник для студентів напряму "Комп'ютерна інженерія" вищих навчальних закладів / О.М. Рисований. – Харків : НТУ "ХПІ", 2010. – 912 с.

7. Рисований О.М. Системне програмування : навч. посіб. для студентів напрямку «Комп'ютерна інженерія» вищих навчальних закладів у 4 ч. Ч. І. Основи асемблера та його використання. / О. М. Рисований. – Харків : «Слово», 2015. – 336 с.

8. Рисований О.М. Системне програмування : навч. посіб. для студентів напрямку «Комп'ютерна інженерія» вищих навчальних закладів у 4 ч. Ч.2. Розширені можливості програмування на асемблері. / О. М. Рисований. – Харків : «Слово», 2015. – 224 с.

9. Рисований О.М. Системне програмування : навч. посіб. для студентів напрямку «Комп'ютерна інженерія» вищих навчальних закладів у 4 ч. Ч.3. Віконний інтерфейс на асемблері. / О. М. Рисований. – Харків : «Слово», 2015. – 272 с.

10. Рисований О.М. Системне програмування : навч. посіб. для студентів напрямку «Комп'ютерна інженерія» вищих навчальних закладів у 4 ч.. Ч.4. Програмування в Windows на асемблері. / О. М. Рисований – Харків : «Слово», 2015. – 224 с.

11. Рисований О. М. Цифрові пристрої і мікропроцесори. Архітектура і програмне забезпечення : навч. посіб. / О. М. Рисований, М. В. Грушенко. – Харків : ХУПС, 2005. – 384 с.

СОДЕРЖАНИЕ

Навчальне видання

РИСОВАНИЙ Олександр Миколайович

«СИСТЕМНЕ ПРОГРАМУВАННЯ»

Ч.1. « Програмування в середовищі masm64»

Навчально-методичний посібник

Російською мовою

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

В авторській редакції

http://blogs.kpi.kharkov.ua/v2/asm/knigi/

План 2016 р., поз. 93

Формат 60x84 1/16. Папір офісний.
Riso-друк. Гарнітура Times. Ум. друк. арк. 6,2. Наклад 50 прим.
Зам. № Ціна договірна.

Видавничий центр НТУ «ХПІ»
Свідоцтво про державну реєстрацію ДК № 3657 від 24.12.2009 р.
61002, Харків, вул. . Кирпичова, 2

Віддруковано в друкарні «Слово»
Адреса: 61002, Харків, вул. Лермонтовська, 27.