Лабораторная работа №3
Вывод строковых массивов на языке Ассемблера

Цель работы:
- знакомство с Макро Ассемблером MASM32;
- создание минимального Windows-приложения на ассемблере;
- изучение вызова функций API win32 из ассемблерных приложений.

Немного теории…

Далее будем полагать, что имеем дело с 32-битными процессорами, начиная с *Intel* 80386. Эти процессоры имеют 32-битную шину адреса, следовательно, могут адресовать 2^{32} = 4 294 967 296 = 4 Гб памяти.

Каждую Win32 программу Windows запускает в отдельном виртуальном пространстве. Это означает, что каждая Win32 программа будет иметь 4-Гб адресное пространство, но вовсе не означает, что каждая программа имеет 4 Гб физической памяти, а только то, что программа может обращаться по любому адресу в этих пределах.

Под Win32 есть только одна модель память: плоская, без 64-килобайтных сегментов. Память – это большое последовательное 4-гигабайтовое пространство. Можно не париться с сегментными регистрами, зато можно использовать любой сегментный регистр для адресации к любой точке памяти.

При программировании под Win32 необходимо помнить, что Windows использует регистры *esi*, *edi*, *ebp* и *ebx* для своих целей и не ожидает, что вы измените их значение. Если же вы используете какой-либо из этих четырёх регистров в вызываемой функции, то не забудьте восстановить их перед возвращением управления Windows.

ОС Windows предоставляет огромное количество ресурсов Windows-программам через Windows API (Application Programming Interface) - коллекцию полезных функций, располагающихся непосредственно в операционной системе и готовых для использования программами. Эти функции находятся в нескольких динамически подгружаемых библиотек (DLL), таких как *kernel32.dll*, *user32.dll* и *gdi32.dll*. *Kernel32.dll* содержит API функции, взаимодействующие с памятью и управляющие процессами, *user32.dll* контролирует пользовательский интерфейс, *gdi32.dll* отвечает за графические операции. Существуют также другие библиотеки. Код API-функций не включается в исполняемый файл.

Вызов системных функций API Win32 из программы на ассемблере подчиняется набору соглашений *stdcall*. Эти соглашения заключаются в следующем:

- регистр символов в имени функции не имеет значения. Например, функции с именами *ExitProcess* и *exitprocess* – это одна и та же функция;
- **аргументы передаются вызываемой функции через стек**. Если аргумент укладывается в 32-битное значение и не подлежит модификации вызываемой функцией, он обычно записывается в стек непосредственно с **по-**

мощью команды *push* (загрузка в стек). В остальных случаях программист должен разместить значение аргумента в памяти, а в стек записать 32-битный указатель на него. Таким образом, все передаваемые функции API параметры представляются 32-битными величинами, и количество байт, занимаемых в стеке для передачи аргументов, кратно четырем;

- вызывающая программа загружает аргументы в стек последовательно, начиная с последнего, указанного в описании функции, и заканчивая первым. После загрузки всех аргументов **программа вызывает функцию командой *call***;
- за возвращение стека в исходное состояние после возврата из функции API отвечает сама эта вызываемая функция. Программисту заботиться о восстановлении указателя стека *esp* нет необходимости, то есть выполнять команду *pop* (извлечение из стека) нет необходимости;
- вызываемая функция API гарантированно сохраняет регистры общего назначения *ebp*, *esi*, *edi*. Регистр *eax*, как правило, содержит возвращаемое значение. Состояние остальных регистров после возврата из функции API следует считать неопределенным. (Полный набор соглашений *stdcall* регламентирует также сохранение системных регистров *ds* и *ss*. Однако, для *flat*-модели памяти, используемой в Win32, эти регистры значения не имеют).

Итак, вот шаблон программы на ассемблере MASM32 для Win32

```
.386
.model flat, stdcall
option casemap: none
include C:\masm32\include\windows.inc
include C:\masm32\include\kernel32.inc
includelib C:\masm32\lib\kernel32.lib
.data
<инициализированные данные>
...
.data?
<неинициализированные данные>
...
.const
<константы>
...
.code
<метка>:
     <текст программы>
     ...
     push NULL
     call ExitProcess
end <метка>
```

.386 – это директива ассемблера, определяющая набор инструкций процессора, которые могут быть использованы в программе. По умолчанию транслятор полагает, что программа пишется для процессора i8086 и сопро-

цессора i8087. Для приложений win32 необходимо указывать либо .386, либо выше (.486, .586, .686) – в зависимости от того, собираетесь ли вы использовать возможности, предоставляемые процессорами/сопроцессорами последующих поколений;

.model – директива ассемблера, определяющая сегментную модель памяти приложения как плоскую (**flat**). Плоская модель памяти позволяет программе благодаря страничной адресации легко работать с 4 Гб виртуальной несегментированной памяти. Именно такая сегментная модель должна всегда использоваться при написании приложений для win32;

stdcall – говорит MASM32 о порядке передачи параметров. Согласно ему, данные передаются справа налево, но вызываемый ответственен за выравнивание стека. Платформа Win32 использует исключительно STDCALL;

option casemap: none – говорит MASM сделать метки чувствительными к регистрам, то есть ExitProcess и exitprocess - это различные имена;

include – директива ассемблера, добавляющая содержимое указанного после директивы файла в то место, где расположена директива. При использовании API-функций win32 в программе должны быть описаны их прототипы. Это можно сделать вручную или путём подключения соответствующего *include*-файла. Подключаемые файлы находятся в директории *c:\masm32\include*. Файлы подключения имеют расширение *.inc* и прототипы функций DLL находятся в *.inc* файле с таким же именем, как и у этой DLL. Например, *ExitProcess* экспортируется *kernel32.lib*, так что прототип *ExitProcess* находится в *kernel32.inc*;

includelib – директива, сообщающая ассемблеру, какие библиотеки использует программа.

Например, сервис функции *ExitProcess* предоставляется *dll*-библиотекой *kernel32.dll*, следовательно, при сборке приложения необходимо подключить библиотеку импорта *kernel32.lib*.

Также можно указать имена библиотек импорта не с помощью директивы *.includelib*, а в командной строке при запуске линкера;

.data – директива, определяющая секцию инициализированных данных программы;

.data? – директива, определяющая секцию неинициализированных данных программы. Если нужно предварительно выделить некоторое количество памяти, но не инициализировать ее, то данная секция предназначается для этого. Преимущество неинициализированных данных следующее: они не занимают места в исполняемом файле. Например, если вы выделите 10 кб в секции *.data?*, *exe*-файл не увеличиться на 10 кб. Вы всего лишь говорите компилятору, сколько места будет нужно, когда программа загрузится в память;

.const – директива, определяющая секцию констант. Константы не могут быть изменены программой.

Не обязательно задействовать все три секции. Объявляйте только те, которые хотите использовать;

.code – директива, определяющая секцию текста программы.

Все четыре директивы (секции) могут поделить адресное пространство на логические секции. Начало одной секции отмечает конец предыдущей. По факту есть две группы секций: данных и кода;

<метка>: – произвольная метка, устанавливающая границы кода. Обе метки должны быть идентичны. Весь текст программы должен располагаться между **<метка>:** и **end <метка>**;

push NULL – задание параметра функции ExitProcess через стек;

call ExitProcess – вызов API-функции завершения приложения. Здесь код выхода NULL, но он, может быть любым в пределах 32-разрядного целого числа. В полноценных приложениях нормой считается определение кода выхода в цикле обработки сообщений главного окна. Именно с помощью функции *ExitProcess* должно завершаться приложение win32, написанное на ассемблере.

Программирование процессоров x86 на языке Ассемблера для архитектуры Win32.

Программу на языке Ассемблера для Win32 можно писать как в специализированной среде разработки (SASM, RadASM и др.), так и в обычном блокноте.

Для программирования «в блокноте» скачайте и установите программу Notepad++ (https://notepad-plus-plus.org/downloads/).

Скачиваем и распаковываем архив masm32v11r.zip по ссылке http://www.masm32.com/download.htm. Запускаем файл установщика *install.exe*. Выбираем для установки диск *C*. После установки на диске *C* появится папка *masm32*.

Можно добавить пути в переменные среды Windows.

Для этого переходим: Этот компьютер -> Свойства -> Дополнительные параметры системы -> Переменные среды -> Системные переменные… переменная Path… Изменить…

В конец списка добавляем строки:

```
C:\masm32\bin
C:\masm32\help
C:\masm32\include
C:\masm32\lib
```

… сохраняем.

Создаем текстовый файл и переименовываем его в Test.asm.

Открываем файл в Notepad++ и пишем программу на языке Ассемблера, производящую вывод текстовой строки на экран консоли (не путайте консоль Windows и MS DOS).

```
;-----------------------------------------
; Программа на masm32 для консоли
; (вывод сообщения)
;-----------------------------------------

.386
.model flat, stdcall
option casemap:none

; Библиотеки и подключаемые файлы проекта
;-----------------------------------------
include C:\masm32\include\windows.inc
include C:\masm32\include\kernel32.inc
includelib C:\masm32\lib\kernel32.lib

; Сегмент данных
;-----------------------------------------
.data
string          db "Hello World!", 0Ah, 0h
sConsoleTitle   db "My first project", 0

.data?
stdout          dd ?
cWritten        dd ?

; Сегмент кода
;-----------------------------------------
.code
start:
; заголовок консоли (1)
      push offset sConsoleTitle
      call SetConsoleTitle

; получаем дескриптор (2)
      push STD_OUTPUT_HANDLE
      call GetStdHandle
      mov stdout, eax
      mov cWritten, ebx

; выводим в консоль строку (3)
      push NULL
      push offset cWritten
      push sizeof string
      push offset string
      push stdout
      call WriteConsole

; задержка (4)
      push INFINITE
      call Sleep
```

```
; завершаем процесс
      push NULL
      call ExitProcess
end start
```

Разбор программы.

(1) Функция *SetConsoleTitle* устанавливает строку для заголовка текущей консоли. Функция требует единственный параметр – указатель на строку символов, которую мы хотим вывести в заголовке окна. Строка должна заканчиваться нулем. Команда `push offset sConsoleTitle` помещает в стек (*push*) адрес (*offset*) строки символов (помеченной как *sConsoleTitle*). Ну а далее следует сам вызов (*call*) функции *SetConsoleTitle*.

Здесь для указания адреса используется префикс под названием *offset*. Это потому, что берется смещение (*offset*) относительно начала сегмента.

(2) Консоль можно использовать как устройство ввода (*input device*), устройство вывода (*output device*), устройство для отчета об ошибках (*error device*). Для того чтобы работать с этим устройством, мы должны получить его дескриптор при помощи API-функции *GetStdHandle*.

Единственный параметр, который от нас требует функция *GetStdHandle* – указание, на какое устройство мы желаем получить дескриптор:

– *STD_OUTPUT_HANDLE* – значение -10 – стандартное устройство ввода;

– *STD_OUTPUT_HANDLE* – значение -11 – стандартное выходное устройство;

– *STD_ERROR_HANDLE* – значение -12 – устройство стандартных ошибок.

После выполнения функции регистр *eax* содержит дескриптор стандартного вывода. Сохраняем этот дескриптор в переменную *stdout*.

(3) API-функция *WriteConsole* записывает символьную строку в экранный буфер консоли, начинающийся с текущей позиции курсора.

Параметры функции:

– *hConsoleOutput* – дескриптор экранного буфера;

– *lpBuffer* – указатель на массив символов (строку), которые будут записаны в экранный буфер консоли;

– *nNumberOfCharsToWrite* – число символов для записи (размер строки);

– *lpNumberOfCharsWritten* – указатель на переменную, которая принимает число фактических записей;

– *lpReserved* - зарезервировано. MSDN рекомендует просто передать NULL.

Последовательность ввода параметров функции обратная:

– командой `push NULL` заносим в стек значение *lpReserved;*

– командой `push offset cWritten` заносим в стек указатель на переменную, которая принимает число фактических записей;

– далее указываем число символов, которые мы хотим напечатать – `push sizeof string;`

– сама строка у нас определена в сегменте данных под именем `string`. Получить ее адрес мы можем при помощи `offset`. Укладываем все в одну строчку – `push offset string`;

– командой `push stdout` заносим дескриптор в стек.

(4) Чтобы консоль сразу же не закрылась, при помощи функции *Sleep* вызываем программную задержку. Функция *Sleep* приостанавливает работу по выполнению текущего потока на заданный промежуток времени. Функция требует один параметр *dwMilliseconds* – длительность задержки [мс]. Значение *INFINITE* вызывает бесконечную задержку.

```
push INFINITE
call Sleep
```

В папке с файлом *Test.asm* создаем текстовый файл *Run.bat* и добавляем в него в блокноте строку:

```
C:\masm32\bin\ml.exe /c /coff /Fl Test.asm
```

… сохраняем изменения.

Со значением ключей *ml.exe* можно ознакомиться здесь: https://wasm.in/blogs/kompiljacija-fajlov-asm-s-pomoschju-kompiljatora-ml-exe.74/.

После выполнения этой строки в папке появятся ещё два файла *Test.lst* и *Test.obj*.

Добавим в файл *Run.bat* ещё одну строку:

```
C:\masm32\bin\link.exe /SUBSYSTEM:CONSOLE /LIBPATH:C:\masm32\lib\
Test.obj
```

Подробную информацию о ключах link.exe можно посмотреть здесь http://bitfry.narod.ru/link.htm.

После выполнения этой строки появится файл *Test.exe*.

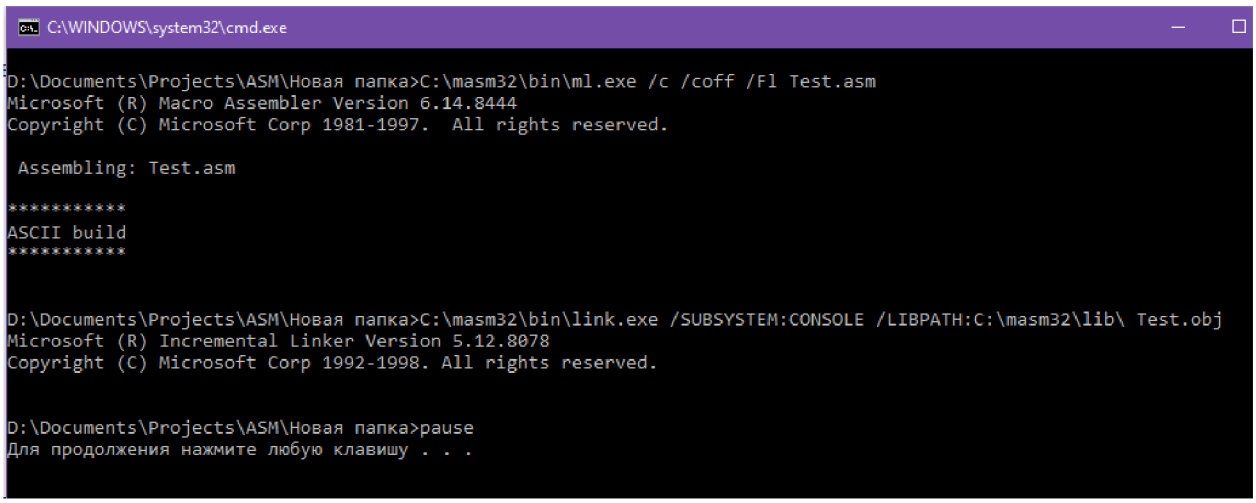
Также в файл *Run.bat* добавьте строки:

```
pause
del Test.obj
start Test.exe
```

```
1  C:\masm32\bin\ml.exe /c /coff /Fl Test.asm
2  C:\masm32\bin\link.exe /SUBSYSTEM:CONSOLE /LIBPATH:C:\masm32\lib\ Test.obj
3  pause
4  del Test.obj
5  start Test.exe
```

Сохраните изменения в файлах и запустите *Run.bat*.

В результате…



В результате выполнения файла Test.exe откроется консоль с заголовком «*My first project*» и содержанием «*Hello, World!*».

Задание 1.

Набрать текст приведеной выше программы, произвести ассемблирование и линковку, проверить работоспособность приложения.

Задание 2.

Переписать приведенный выше текст программы с использованием макрокоманды ***invoke***.

Макрокоманда *invoke*, позволяет записывать вызовы в более удобном виде. Например, следующий вызов:

```
push MB_OKCANCEL
push offset hello_title
push offset hello_mess
push 0
call MessageBox
```

с помощью макрокоманды *invoke* будет записан следующим образом:

```
invoke MessageBox, 0, addr hello_mess, addr hello_title, MB_OKCANCEL
```

Содержание отчёта.

1. Название работы.
2. Цель работы.
3. Постановка задачи.

4. Текст программ с комментариями.
5. Анализ результатов.
6. Выводы.