# ВВЕДЕНИЕ

К достоинствам языка ассемблера относятся: четкость, чистота кода, небольшой размер и высокая скорость выполнения скомпилированного кода, возможность непосредственного обращения к аппаратным средствам. Все, что только можно сделать на компьютере, можно реализовать на ассемблере. Он позволяет разбить необходимое действие на мельчайшие элементарные шаги. Программирование на ассемблере делает понимание принципов работы компьютера полным.

При написании программ на ассемблере от программиста требуется повышенное внимание к деталям. Языки высокого уровня дают возможность сосредоточиться на общей цели, не тратя время на мелочи. Но даже и в этом случае некоторые процедуры целесообразно писать на ассемблере, стыкуя их к основной программе, написанной на языке высокого уровня.

Язык ассемблера широко использовался программистами при написании программ под DOS. С появлением Windows возникла необходимость научиться программировать для новой системы.

Основные отличия программирования на ассемблере под Windows от программирования под DOS следующие:

– все приложения в Windows запускаются в 32-битном режиме с моделью памяти flat (данная модель памяти позволяет адресовать до 4Гб памяти непосредственно, забыв о 64кб лимите размера сегмента в DOS);

– другое принципиальное нововведение – отсутствие необходимости (и даже больше – самой возможности) программировать различные устройства на низком уровне. Операционная система, работающая в защищенном режиме, перехватывает непосредственные обращения к устройствам, а взамен предоставляет свыше двух тысяч системных вызовов.

В отличие от программирования под DOS, где программы, написанные на языках высокого уровня (ЯВУ) были мало похожи на свои аналоги, написанные на ассемблере, приложения под Win32 имеют гораздо больше общего. В первую очередь это связано с тем, что обращение к сервису операционной системы в Windows осуществляется посредством вызова функций, а не прерываний, что было характерно для DOS, а механизм вызова си-

стемных функций схож с механизмом вызова функций языками высокого уровня. Здесь нет передачи параметров в регистрах при обращении к сервисным функциям и, соответственно, нет и множества результирующих значений возвращаемых в регистрах общего назначения и регистре флагов. Следовательно, проще запомнить и использовать протоколы вызова функций системного сервиса.

Другими словами, программирование под Win32 ставит две задачи – оптимизированный вызов стандартных функций API, направленных на формирование интерфейса и работе с системными устройствами, и собственно реализацию нужного алгоритма.

Реализация алгоритма с использованием ассемблера, безусловно, является оптимальной – в этом случае «накладные расходы», связанные с использованием различных компиляторов, минимальны. Ассемблер в этом случае – предел, к которому стремится любой язык высокого уровня.

При реализации первой задачи неожиданно оказывается, что различия между ЯВУ и ассемблером крайне незначительны. Особенно хорошо это заметно при применении MS Visual C++, на котором оконное приложение создается с подробностью, сравнимой с ассемблерным кодом.

Современный ассемблер, к которому относится и TASM 5.0 фирмы Borland International Inc., целенаправленно развивал средства, которые ранее были характерны только для ЯВУ. К таким средствам можно отнести макроопределение вызова процедур, возможность введения шаблонов процедур (описание прототипов) и даже объектно-ориентированные расширения. Однако ассемблер сохранил такой инструмент, как макроопределения, вводимые пользователем, полноценного аналога которому нет ни в одном ЯВУ.

Все эти факторы позволяют рассматривать ассемблер как самостоятельный инструмент для написания приложений под платформы Win32.

Поэтому целью данного практикума является обучение разработке программ с использованием ассемблера. Для этого в практикум включены не только рекомендации по созданию программ для Windows на языке ассемблера, но и даны примеры про-

грамм с комментариями, а также задания для самостоятельного выполнения студентами.

Лабораторный практикум предполагает знание языков Assembler, C/C++, Pascal, а также наличие некоторых специфических знаний об общих принципах работы персонального компьютера.

К практикуму прилагается набор файлов, содержащих исходные и скомпилированные тексты примеров, минимальный набор утилит, необходимый для самостоятельного компилирования примеров, и некоторую справочную информацию.

При тестировании примеров использовался Borland Turbo Assembler and Tools 5.0, в том числе:

tasm.exe – Turbo Assembler Version 4.1,
tlink.exe – Turbo Link Version 7.1.30.1,
implib.exe – Borland Implib Version 2.0.112.1,
tlink32.exe – Turbo Link Version 1.6.71.0,
brcc32.exe – Borland Resource Compiler Version 5.00.

Рекомендуется использовать программы этих либо более новых версий. Совместимость с другими версиями не гарантирована.

## ОБЗОР ПРОГРАММИРОВАНИЯ В WIN32

### Принципы программирования

Разработка Win32-приложений сильно (качественно и количественно) отличается от аналогичной задачи под DOS. В основном это вызвано огромными различиями в работе самих операционных систем. Но программирование на ассемблере всегда отличалось от программирования на языках высокого уровня и требовало от программиста дополнительных знаний, дополнительной практики и большего профессионализма.

Поэтому, прежде чем начать создавать приложения для Win32, необходимо рассмотреть некоторые особенности системы Win32, которые были введены фирмой Microsoft.

1. Все Windows-приложения используют специальный формат исполняемых файлов – PE (Portable Executable, т.е. переносимый исполняемый формат). Данный формат имеют как исполня-

емые файлы, так и динамические библиотеки. Кроме того, есть еще формат *.VXD-драйверов – LE (Linear Executable, т.е. линейный исполняемый). В состав исполняемого PE-модуля входит и DOS-программа, так называемая «заглушка» (stub). Ее функция – вывести сообщение о невозможности выполнения при запуске программы под DOS. Фактически PE-заголовок просто следует после описания стандартного DOS-заголовка и программы-заглушки.

2. В системе Windows используются библиотеки системных функций API (Application Programming Interface), которые расположены в системных *.dll файлах (например, kernel32.dll, shell32.dll, user32.dll).

3. Чтобы активизировать системную функцию Windows, программа должна поместить в стек все параметры **от последнего к первому**, а затем передать управление дальней CALL. Данные функции сами освобождают стек и возвращают результат работы в регистре EAX. Такая договоренность о передаче параметров называется STDCALL. Такой (так называемый *«прямой»*) способ передачи параметров подходит только для функций с фиксированным числом параметров, но вызывающая сторона не должна заботиться об освобождении стека. Некоторые функции Windows API используют способ *«перевернутой»*, **от первого к последнему**, передачи параметров в стек. К таким функциям относят функции, которые могут получать переменное число параметров.

4. Имена всех системных функций Win32 модифицируются так, что перед именем функции ставится подчеркивание, а после знак @ и число байтов, которое занимают параметры, передаваемые ей в стеке. Так ExitProcess превращается в _ExitProcess@4. Реально это лишь небольшая процедура, которая просто передает управление на такую же метку, но с добавлением _ _imp_. То есть функция будет выглядеть как _ _imp_ _ ExitProcess@4.

Нужно отметить, что TLINK32 использует собственный способ вызова системных функций, который не позволяет обращаться непосредственно к _ _imp_-функциям. Это несколько увеличивает размер скомпилированной программы и может сказаться на скорости выполнения, зато мы получим возможность использовать упрощенный синтаксис вызова.

5. Кроме того, все функции API подразделяются на две группы: функции, которые поддерживают UNICODE и функции, которые поддерживают ANSI. Функции, поддерживающие UNICODE, имеют в конце названия букву W, а функции, поддерживающие ANSI – букву A.Например: GetProcAddressA – прототип функции GetProcAddress для ANSI; GetProcAddressW – прототип функции GetProcAddress для UNICODE. То есть программа, написанная для ANSI, может быть скомпилирована и для UNICODE, достаточно заменить A на W.

Различие UNICODE от ANSI заключается в том, что в ANSI для передачи одной буквы используется один байт, в зависимости от текущей кодовой страницы в различных кодировках, а в UNICODE используется два байта (так называемый WIDE-символ). Поэтому при использовании UNICODE все ссылки на строки включают в себя букву W. Так, например:

LPCTSTR (ANSI) = LPCWSTR (UNICODE)

LPTSTR (ANSI) = LPWSTR (UNICODE)

6. Все функции Windows API зависят от регистра букв (case sensitive).

7. Функции Windows сохраняют значение регистров EBP, ESI, EDI, EBX.

8. Строковые величины соответствуют стандарту языка C (null-terminated).

**Рекомендации**

1. В связи с тем, что система Windows ощутимо сложнее DOS, настоятельно рекомендуется программу создавать не «одним куском», а использовать несколько файлов, подключаемых директивой include. Сначала это может показаться бессмысленным, но создание сколько-нибудь объемного проекта за счет этого будет облегчено. Подобный подход пригодится и для языков высокого уровня. В предложенных примерах тоже используется разбивка на части и по возможности объясняется, для чего это необходимо.

2. В некоторых случаях для внешних процедур полезно бывает использовать псевдонимы при помощи equ, следующего вида:

```
DispatchMessage      equ   DispatchMessageA
```

Польза, кроме более читаемого кода, еще и в легкости перехода от ANSI к UNICODE. Достаточно будет заменить суффиксы на W, и сколько бы не было вызовов процедуры в программе, изменения будут действовать для каждого. Или в случае перехода от TASM к MASM можно достаточно быстро заменить упрощенные имена функций на полные, без утомительной правки объемного кода.

3. Поскольку в программировании используются вызовы системных функций Windows, встает логичный вопрос – где взять прототипы (описания) этих функций? Приводить полный перечень (их намного больше двух тысяч, и в разных версиях Windows они различны) не имеет смысла. Более целесообразно использование систем помощи языков высокого уровня (например, Delphi или Visual C), а еще лучше – справочника MSDN. Опыт работы с MSDN будет полезен тем, кто собирается серьезно разрабатывать программы для Windows вне зависимости от выбранного языка.

4. Для компиляции полезно написание пакетных командных файлов – это позволит отказаться от многократного набора одних и тех же команд.

## ЛАБОРАТОРНАЯ РАБОТА 1. ПРОСТЕЙШАЯ ПРОГРАММА

Традиционно первой программой на любом языке является вывод сообщения «Hello, World!». Пример программы на ассемблере для Win32:

hellowin.asm:

```
includelib   import32.lib ;Подключение библиотеки описния
                          ;системных функций
.386                      ;включение 32-битного режима
.model     flat           ;модель памяти FLAT
        extrn MessageBoxA:proc          ;Внешние процедуры
        extrn ExitProcess:proc
.data
mb_text      db 'Hello, World!',0       ;Текстовые константы
mb_title     db 'My first program',0    ;для выполнения
                                        ;программы
```

```
.code
start:    push 0                          ;Формирование
          push offset mb_title            ;параметров
          push offset mb_text             ;в стеке для;MessageBox -
          push 0                          ;прямая передача
          call MessageBoxA                ;Вызов MessageBox
          push 0                          ;параметр для ExitProcess
          call ExitProcess                ;завершение программы
          ends
          end    start
```

>tasm/ml hellowin.asm -> hellowin.obj (334b)
>tlink/Tpe/aa/c/x hellowin.obj-> hellowin.exe (4096 b)

Специфичные опции компиляции:

– опция **/ml** указывает, что в тексте программы во всех именах следует различать регистр букв, то есть MessageBoxA и MESSAGEBOXA неэквивалентны;

– опцией **/Tpe** дается указание сгенерировать EXE-файл. Если мы укажем опцию **/Tpd,** то на выходе компоновщика получим DLL-файл;

– опция **/aa** указывает компилятору, что нужно создать обычное приложение Windows, использующее GDI. Если указать **/ap**, то компоновщик сгенерирует консольное приложение Windows (примером консольного приложения может послужить FAR);

– опция **/c** аналогична опции **/ml**, она говорит компоновщику о том, что в файле должны различаться регистры букв у имен переменных и функций.

Результат:


Программа собственно состоит из двух функций: MessageBox и ExitProcess. Прототипы этих функций:

```
int MessageBox(
  HWND   hWnd,   //идентификатор окна-предка, из которого
вызывается MessageBox
```

```
  LPCSTR lpText, //указатель на null-terminated строку,
содержащую текст сообщения
  LPCSTR lpCaption,  //указатель на null-terminated строку,
содержащую заголовок сообщения
  UINT uType       //вид диалогового окна
         );
VOID ExitProcess(
  UINT uExitCode,  //код выхода
                    );
```

Если сравнить последовательность параметров в прототипе и последовательность помещения параметров в стек в программе, станет наглядно ясен принцип «прямой» передачи данных. Важный момент: после имени процедур отсутствует @<размер параметров>. **Это особенность TASM!** При использовании MASM будет необходимым указывать данный суффикс.

Сразу необходимо отметить следующее: множество различных уникальных типов параметров в прототипах функций не должно смущать программиста. Достаточно помнить, что с точки зрения ассемблера многие типы идентичны по занимаемому в памяти месту.

TASM позволяет упростить механизм вызова процедур. Если в директиве .model непосредственно указать STDCALL, то вызов процедуры станет еще больше напоминать язык высокого уровня – параметры можно перечислить через запятую после имени функции. Не следует забывать – **это макрос!** На самом деле происходит та же цепочка push. Это видно из листинга.

В дистрибутив TASM 5.0 также входит файл windows.inc, который содержит множество определений структур и констант. Можно использовать его непосредственно, а можно создать свой собственный include-файл, поместив в него необходимые в конкретном случае константы. Этот вариант может быть предпочтительней – можно легко найти и изучить необходимую структуру или константу.

В этот (или такой же) файл можно также вынести определения внешних процедур. В данной программе это не обязательно, но в более масштабных проектах множество определений будет визуально засорять программу, и разумней будет применить включаемые файлы.

Следующий пример – это тот же вывод «Hello, World!» с использованием STDCALL и msgbox.inc (файла с выдержками из windows.inc):

msgbox.inc:

```
MB_OK                = 0000H
MB_OKCANCEL          = 0001H
MB_ABORTRETRYIGNORE  = 0002H
MB_YESNOCANCEL       = 0003H
MB_YESNO             = 0004H
MB_RETRYCANCEL       = 0005H

MB_ICONHAND          = 0010H
MB_ICONQUESTION      = 0020H
MB_ICONEXCLAMATION   = 0030H
MB_ICONASTERISK      = 0040H

;Внешние процедуры
includelib   import32.lib
          extrn MessageBoxA:proc
          extrn ExitProcess:proc
```

hellown2.asm:

```
include   msgbox.inc
.386                              ;включение 32-битного режима
.model    FLAT,STDCALL            ;модель памяти FLAT,
                                  ;прямая передача параметров

.data
mb_text    db 'Hello, World!',0 ;Текстовые константы
mb_title   db 'Next program',0 ;для выполнения программы
.code
start:
        call MessageBoxA,0,offset mb_text, offset mb_title,
MB_ICONASTERISK+MB_OK
        call   ExitProcess,0        ;завершение программы
        ends
        end    start
```

>tasm/ml hellown2.asm -> hellown2.obj (354b)
>tlink/Tpe/aa/c/x hellown2.obj -> hellown2.exe (4096b)

Результат:


Как видно из примера, окно с предопределенным классом выводится достаточно просто, и оно схоже с языком высокого уровня.

## Порядок выполнения работы

1. Изучить основные сведения.
2. Выполнить ввод, трансляцию, построение кода программ и получить результаты ее работы.
3. Изучить содержимое exe-файла в отладчике td32.exe. Составить таблицу, в которой отразить название секций, их адрес в отладчике, описание содержимого.

## Содержание отчета о работе

1. Цель работы.
2. Текст программ.
3. Описание используемых API-функций.
4. Описание опций компиляции и построения исполняемого кода.
5. Таблица секций exe-файла.
5. Выводы по работе.

## СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ


1. Андреева А.А. и др. Программирование на языке ассемблера в операционной системе Windows: лабораторный практикум. Чебоксары, Чуваш. ун-т, 2006. 104 с.
2. Зубков С.В. Ассемблер для DOS, Windows и UNIX. М.: ДМК Пресс, 2015. 608 с.
3. Пирогов В.Ю. Ассемблер для Windows. СПб.: БХВ-Петербург, 2011. 864 с.
4. Аблязов Р. Программирование на ассемблере на платформе x86-64. М.: ДМК Пресс, 2016. 302 с.