https://nestor.minsk.by/kg/2008/17/kg81719.html

[https://nestor.minsk.by/kg/logo.jpg](https://nestor.minsk.by/kg/)

[](https://nestorclub.com/index.pl?act=NEWSSHOW&id=2020120301)

**Ассемблер под Windows для чайников**

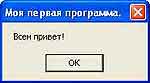
[главная](https://nestor.minsk.by/kg/) » [статьи](https://nestor.minsk.by/kg/arch.html) » [2008](https://nestor.minsk.by/kg/2008/) » [17](https://nestor.minsk.by/kg/2008/17/) » 01.05.2008‣программирование

**На сегодняшний день существует огромное количество языков программирования высокого уровня. На их фоне программирование на низкоуровневом языке — ассемблере — может на первый взгляд показаться чем-то устаревшим и нерациональным. Однако это только кажется. Следует признать, что ассемблер фактически является языком процессора, а значит, без него не обойтись, пока существуют процессоры. Основными достоинствами программирования на ассемблере являются максимальное быстродействие и минимальный размер получаемых программ.**

Недостатки зачастую обусловлены лишь склонностью современного рынка к предпочтению количества качеству. Современные компьютеры способны легко справиться с нагромождением команд высокоуровневых функций, а если нелегко — будьте добры обновите аппаратную часть вашей машины! Таков закон коммерческого программирования. Если же речь идет о программировании для души, то компактная и шустрая программа, написанная на ассемблере, оставит намного более приятное впечатление, нежели высокоуровневая громадина, обремененная кучей лишних операций. Бытует мнение, что программировать на ассемблере могут только избранные. Это неправда. Конечно, талантливых программистов-ассемблерщиков можно пересчитать по пальцам, но ведь так обстоит дело практически в любой сфере человеческой деятельности. Не так уж много найдется водителей-асов, но научиться управлять автомобилем сумеет каждый — было бы желание. Ознакомившись с данным циклом статей, вы не станете крутым хакером. Однако вы получите общие сведения и научитесь простым способам программирования на ассемблере для Windows, используя ее встроенные функции и макроинструкции компилятора. Естественно, для того, чтобы освоить программирование для Windows, вам необходимо иметь навыки и опыт работы в Windows. Сначала вам будет многое непонятно, но не расстраивайтесь из- за этого и читайте дальше: со временем все встанет на свои места.

Итак, для того, чтобы начать программировать, нам как минимум понадобится компилятор. Компилятор — это программа, которая переводит исходный текст, написанный программистом, в исполняемый процессором машинный код. Основная масса учебников по ассемблеру делает упор на использование пакета MASM32 (Microsoft Macro Assembler). Но я в виде разнообразия и по ряду других причин буду знакомить вас с молодым стремительно набирающим популярность компилятором FASM (Flat Assembler). Этот компилятор достаточно прост в установке и использовании, отличается компактностью и быстротой работы, имеет богатый и емкий макросинтаксис, позволяющий автоматизировать множество рутинных задач. Его последнюю версию вы можете скачать по адресу: [сайт](http://flatassembler.net/download.php) выбрав flat assembler for Windows. Чтобы установить FASM, создайте папку, например, "D:\FASM" и в нее распакуйте содержимое скачанного zip-архива. Запустите FASMW.EXE и закройте, ничего не изменяя. Кстати, если вы пользуетесь стандартным проводником, и у вас не отображается расширение файла (например, .EXE), рекомендую выполнить Сервис -> Свойства папки -> Вид и снять птичку с пункта Скрывать расширения для зарегистрированных типов файлов. После первого запуска компилятора в нашей папке должен появиться файл конфигурации — FASMW.INI. Откройте его при помощи стандартного блокнота и допишите в самом низу 3 строчки:  
[Environment]  
Fasminc=D:\FASM\INCLUDE  
Include=D:\FASM\INCLUDE

Если вы распаковали FASM в другое место — замените "D:\FASM\" на свой путь. Сохраните и закройте FASMW.INI. Забегая вперед, вкратце объясню, как мы будем пользоваться компилятором:  
1. Пишем текст программы, или открываем ранее написанный текст, сохраненный в файле .asm, или вставляем текст программы из буфера обмена комбинацией.  
2. Жмем F9, чтобы скомпилировать и запустить программу, или Ctrl+F9, чтобы только скомпилировать. Если текст программы еще не сохранен — компилятор попросит сохранить его перед компиляцией.  
3. Если программа запустилась, тестируем ее на правильность работы, если нет — ищем ошибки, на самые грубые из которых компилятор нам укажет или тонко намекнет.  
Ну, а теперь мы можем приступить к долгожданной практике. Запускаем наш FASMW.EXE и набираем в нем код нашей первой программы:  
  
include '%fasminc%/win32ax.inc'  
  
.data  
Caption db 'Моя первая программа.',0  
Text db 'Всем привет!',0  
  
.code  
start:  
invoke MessageBox,0,Text,Caption,MB\_OK  
invoke ExitProcess,0  
  
.end start



Жмем Run -> Run, или F9 на клавиатуре. В окне сохранения указываем имя файла и папку для сохранения. Желательно привыкнуть сохранять каждую программу в отдельную папку, чтобы не путаться в будущем, когда при каждой программе может оказаться куча файлов: картинки, иконки, музыка и прочее. Если компилятор выдал ошибку, внимательно перепроверьте указанную им строку — может, запятую пропустили или пробел. Также необходимо знать, что компилятор чувствителен к регистру, поэтому .data и .Data воспринимаются как две разные инструкции. Если же вы все правильно сделали, то результатом будет простейший MessageBox (рис. 1). Теперь давайте разбираться, что же мы написали в тексте программы. В первой строке директивой include мы включили в нашу программу большой текст из нескольких файлов. Помните, при установке мы прописывали в фасмовский ини-файл 3 строчки? Теперь %fasminc% в тексте программы означает D:\FASM\INCLUDE или тот путь, который указали вы. Директива include как бы вставляет в указанное место текст из другого файла. Откройте файл WIN32AX.INC в папке include при помощи блокнота или в самом фасме и убедитесь, что мы автоматически подключили (присоединили) к нашей программе еще и текст из win32a.inc, macro/if.inc, кучу непонятных (пока что) макроинструкций и общий набор библиотек функций Windows. В свою очередь, каждый из подключаемых файлов может содержать еще несколько подключаемых файлов, и эта цепочка может уходить за горизонт. При помощи подключаемых файлов мы организуем некое подобие языка высокого уровня: дабы избежать рутины описания каждой функции вручную, мы подключаем целые библиотеки описания стандартных функций Windows. Неужели все это необходимо такой маленькой программе? Нет, это — что-то вроде "джентльменского набора на все случаи жизни". Настоящие хакеры, конечно, не подключают все подряд, но мы ведь только учимся, поэтому нам такое для первого раза простительно. Далее у нас обозначена секция данных — .data. В этой секции мы объявляем две переменные — Caption и Text. Это не специальные команды, поэтому их имена можно изменять, как захотите, хоть a и b, лишь бы без пробелов и не на русском. Ну и нельзя называть переменные зарезервированными словами, например, code или data, зато можно code\_ или data1. Команда db означает "определить байт" (define byte). Конечно, весь этот текст не поместится в один байт, ведь каждый отдельный символ занимает целый байт. Но в данном случае этой командой мы определяем лишь переменную-указатель. Она будет содержать адрес, в котором хранится первый символ строки. В кавычках указывается текст строки, причем кавычки по желанию можно ставить и 'такие', и "такие" — лишь бы начальная кавычка была такая же, как и конечная. Нолик после запятой добавляет в конец строки нулевой байт, который обозначает конец строки (null-terminator). Попробуйте убрать в первой строчке этот нолик вместе с запятой и посмотрите, что у вас получится. Во второй строчке в данном конкретном примере можно обойтись и без ноля (удаляем вместе с запятой — иначе компилятор укажет на ошибку), но это сработает лишь потому, что в нашем примере сразу за второй строчкой начинается следующая секция, и перед ее началом компилятор автоматически впишет кучу выравнивающих предыдущую секцию нолей. В общих случаях ноли в конце текстовых строк обязательны! Следующая секция — секция исполняемого кода программы — .code. В начале секции стоит метка start:. Она означает, что именно с этого места начнет исполняться наша программа. Первая команда — это макроинструкция invoke. Она вызывает встроенную в Windows API-функцию MessageBox. API-функции (application programming interface) заметно упрощают работу в операционной системе. Мы как бы просим операционную систему выполнить какое-то стандартное действие, а она выполняет и по окончании возвращает нам результат проделанной работы. После имени функции через запятую следуют ее параметры. У функции MessageBox параметры такие:

1-й параметр должен содержать хэндл окна-владельца. Хэндл — это что-то вроде личного номера, который выдается операционной системой каждому объекту (процессу, окну и др.). 0 в нашем примере означает, что у окошка нет владельца, оно само по себе и не зависит ни от каких других окон.  
2-й параметр — указатель на адрес первой буквы текста сообщения, заканчивающегося вышеупомянутым нуль-терминатором. Чтобы наглядно понять, что это всего лишь адрес, сместим этот адрес на 2 байта прямо в вызове функции: invoke MessageBox,0,Text+2,Caption,MB\_OK и убедимся, что теперь текст будет выводиться без первых двух букв.  
3-й — указатель адреса первой буквы заголовка сообщения.  
4-й — стиль сообщения. Со списком этих стилей вы можете ознакомиться, например, в INCLUDE\EQUATES\ USER32.INC. Для этого вам лучше будет воспользоваться поиском в Блокноте, чтобы быстро найти MB\_OK и остальные. Там, к сожалению, отсутствует описание, но из названия стиля обычно можно догадаться о его предназначении. Кстати, все эти стили можно заменить числом, означающим тот, иной, стиль или их совокупность, например: MB\_OK + MB\_ICONEXCLAMATION. В USER32.INC указаны шестнадцатеричные значения. Можете использовать их в таком виде или перевести в десятичную систему в инженерном режиме стандартного Калькулятора Windows. Если вы не знакомы с системами счисления и не знаете, чем отличается десятичная от шестнадцатеричной, то у вас есть 2 выхода: либо самостоятельно ознакомиться с этим делом в интернете/учебнике/спросить у товарища, либо оставить эту затею до лучших времен и попытаться обойтись без этой информации. Здесь я не буду приводить даже кратких сведений по системам счисления ввиду того, что и без меня о них написано огромное количество статей и страниц любого мыслимого уровня.  
  
Вернемся к нашим баранам. Некоторые стили не могут использоваться одновременно — например, MB\_OKCANCEL и MB\_YESNO. Причина в том, что сумма их числовых значений (1+4=5) будет соответствовать значению другого стиля — MB\_RETRYCANCEL. Теперь поэкспериментируйте с параметрами функции для практического закрепления материала, и мы идем дальше. Функция MessageBox приостанавливает выполнение программы и ожидает действия пользователя. По завершении функция возвращает программе результат действия пользователя, и программа продолжает выполняться. Вызов функции ExitProcess завершает процесс нашей программы. Эта функция имеет лишь один параметр — код завершения. Обычно, если программа нормально завершает свою работу, этот код равен нулю. Чтобы лучше понять последнюю строку нашего кода — .end start, — внимательно изучите эквивалентный код:

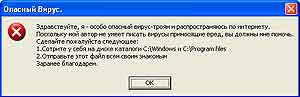
format PE GUI 4.0

**include '%fasminc%/win32a.inc'  
  
entry start  
  
section '.data' data readable writeable  
  
Caption db 'Наша первая программа.',0  
Text db 'Ассемблер на FASM — это просто!',0  
  
section '.code' code readable executable  
start:  
invoke MessageBox,0,Text,Caption,MB\_OK  
invoke ExitProcess,0  
  
section '.idata' import data readable writeable  
library KERNEL32, 'KERNEL32.DLL',\  
USER32, 'USER32.DLL'  
  
import KERNEL32,\  
ExitProcess, 'ExitProcess'  
  
import USER32,\  
MessageBox, 'MessageBoxA'**

Для компилятора он практически идентичен предыдущему примеру, но для нас этот текст выглядит уже другой программой. Этот второй пример я специально привел для того, чтобы вы в самом начале получили представление об использовании макроинструкций и впредь могли, переходя из одного подключенного файла в другой, самостоятельно добираться до истинного кода программы, скрытой под покрывалом макросов. Попробуем разобраться в отличиях. Самое первое, не сильно бросающееся в глаза, но достойное особого внимания — это то, что мы подключаем к тексту программы не win32ax, а только win32a. Мы отказались от большого набора и ограничиваемся малым. Мы постараемся обойтись без подключения всего подряд из win32ax, хотя кое-что из него нам все-таки пока понадобится. Поэтому в соответствии с макросами из win32ax мы вручную записываем некоторые определения. Например, макрос из файла win32ax:  
macro .data { section '.data' data readable writeable }

во время компиляции автоматически заменяет .data на section '.data' data readable writeable. Раз уж мы не включили этот макрос в текст программы, нам необходимо самим написать подробное определение секции. По аналогии вы можете найти причины остальных видоизменений текста программы во втором примере. Макросы помогают избежать рутины при написании больших программ. Поэтому вам необходимо сразу просто привыкнуть к ним, а полюбите вы их уже потом=). Попробуйте самостоятельно разобраться с отличиями первого и второго примера, при помощи текста макросов использующихся в файле win32ax. Скажу еще лишь, что в кавычках можно указать любое другое название секции данных или кода — например: section 'virus' code readable executable. Это просто название секции, и оно не является командой или оператором. Если вы все уяснили, то вы уже можете написать собственный вирус. Поверьте, это очень легко. Просто измените заголовок и текст сообщения:  
Caption db 'Опасный Вирус.',0  
  
Text db 'Здравствуйте, я — особо опасный вирус-троян и распространяюсь по интернету.',13,\  
'Поскольку мой автор не умеет писать вирусы, приносящие вред, вы должны мне помочь.',13,\  
'Сделайте, пожалуйста, следующее:',13,\  
'1.Сотрите у себя на диске каталоги C:\Windows и C:\Program files',13,\  
'2.Отправьте этот файл всем своим знакомым',13,\  
'Заранее благодарен.',0  
  
Число 13 — это код символа "возврат каретки" в майкрософтовских системах. Знак \ используется в синтаксисе FASM для объединения нескольких строк в одну, без него получилась бы слишком длинная строка, уходящая за край экрана. К примеру, мы можем написать start:, а можем — и st\  
ar\  
t:

t:



Компилятор не заметит разницы между первым и вторым вариантом.Ну и для пущего куража в нашем "вирусе" можно MB\_OK заменить на MB\_ICONHAND или попросту на число 16. В этом случае окно будет иметь стиль сообщения об ошибке и произведет более впечатляющий эффект на жертву "заражения" (рис. 2).  
Вот и все на сегодня. Желаю вам успехов и до новых встреч!  
Все приводимые примеры были протестированы на правильность работы под Windows XP и, скорее всего, будут работать под другими версиями Windows, однако я не даю никаких гарантий их правильной работы на вашем компьютере. Исходные тексты программ вы можете найти на форуме: [сайт](http://forum.sa-sec.org/index.php?showtopic=766)  
  
**BarMentaLisk, q@sa-sec.org SASecurity gr.**

*Компьютерная газета. Статья была опубликована в номере 17 за 2008 год в рубрике*[*программирование*](https://nestor.minsk.by/kg/abc/%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/)