**Win32ASM: Минимальное приложение — Архив WASM.RU**

Любое приложение обязано выполнить как минимум две не такие уж и простые задачи:  
  а) стартовать, получив при этом нормальный доступ к ресурсам рабочей среды,  
  и  
  б) завершиться, оставив эту рабочую среду работоспособной.  
     
 Открываем текстовый редактор блокнот и набираем там исходный текст минимального приложения на языке ассемблера.

Код (Text):

; Минимальное приложение на Assembler'e (minimal.asm)

 .386

 .model flat,stdcall

 includelib kernel32.lib

ExitProcess PROTO :DWORD

 .code

WinMain PROC

…

 push 0

 call ExitProcess

WinMain ENDP

end WinMain

 Код файла **minimal.bat**, набирается в редакторе блокнот.

**c:\masm32\bin\ml /c /coff minimal.asm**

  Ключ /c говорит о том, что мы хотим только оттранслировать исходный файл, но не компоновать.

  Ключ /coff означает, что мы хотим создать объектный файл в формате COFF (Common Object File Format), стандартном для "Окон" формате объектных файлов.

Дополнительные ключи:

/Fl - ключ для генерации файла-листинга компиляции, показывает строки с ошибками

/Sn – удаляет из листинга таблицу символов, что делает листинг короче и удобнее

/? – вывод на экран справки по ключам компилятора

Файл   minimal.batзапускается из «командной строки» меню «пуск/стандартные».

Теперь, если вы правильно набрали minimal.bat, в той же папке, что и minimal.asm, должен появиться объектный файл minimal.obj.  
  Далее приступим к созданию файла “exe”.

Код файла **minimal\_link.bat**

**c:\masm32\bin\link /SUBSYSTEM:WINDOWS /LIBPATH:c:\masm32\lib minimal.obj**

   Ключ /SUBSYSTEM с параметром WINDOWS говорит о том, что мы собираемся собрать загрузочный файл для подсистемы Windows. Другие возможные значения: CONSOLE - если мы собираемся делать программу с текстовым интерфейсом для MS DOS, NATIVE - если драйвер устройства, и POSIX - если мы собираемся писать программы, ориентированные на стандарты POSIX и более или менее переносимые под разные Юниксы (для которых эти стандарты и писались).  
  Ключ /LIBPATH указывает путь к библиотекам импорта, которые нам обязательно необходимо прилинковать к программе, если мы хотим использовать возможности, предоставляемые не только процессором, но и операционной системой.   
  Если вы все сделали правильно, то в нашей папке, наряду с исходным и объектным, должен появиться еще и третий файл - исполняемый.

  .386 - это директива ассемблера, определяющая набор инструкций процессора, которые могут быть использованы в программе. По умолчанию транслятор полагает, что программа пишется для процессора 8086 и сопроцессора 8087. 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 или нет.   
   .model flat, stdcall. Первый параметр - это модель памяти. Под Windows есть одна-единственная "правильная" модель памяти - flat, позволяющая нашей программе благодаря страничной адресации легко и просто работать с 4 Гб виртуальной несегментированной памяти.  Второй же параметр указывает на так называемое соглашение о вызове процедур. Каждый язык имеет свои "соглашения". Так, второй параметр может принимать значения: c, basic, fortran, pascal... При программировании под win32 на макроассемблере нужно использовать соглашение stdcall

 .includelib kernel32.lib - эта директива передается компоновщику и сообщает, что наша программа должна быть скомпонована с указанной библиотекой, в данном случае с kernel32.lib. Библиотека представляет собой некоторое количество готовых к употреблению процедур, оттранслированных в объектные файлы и собранных в файл под названием "библиотека".  
 Задействование ресурсов операционной системы производится при помощи так называемых функций Win API. А для того, чтобы получить доступ к этим ресурсам, вы должны обязательно cлинковать свою программу с соответствующей библиотекой импорта, в которой как раз и находятся подпрограммы, эти функции вызывающие.  
  Слово "слинковать" выделено не случайно. Это связано со следующей строчкой нашего исходника.  
  
  Посмотрите внимательно на тело WinMain. Там есть команда call, то есть вызов подпрограммы.

  Посмотрите на call ExitProcess - это вызов API'шной функции, завершающей работу нашей программы (аналог INT 20h).  
 На этапе транслирования ассемблер не имеет никакого понятия о том, что существует какой-то ExitProcess. Прототип - это своего рода уведомление транслятора о том, что "немного попозже я собираюсь обращаться к ExitProcess, и форма вызова для него такая-то". Далее следует сколько он принимает параметров, и какой они должны быть размерности.

  Как известно, каждая программа состоит из кода и данных. Раньше мы их называли сегментами. Теперь термин сегмент заменим на термин "секция", хотя, по большому счету, это одно и то же.  
  Определяются эти секции, следующими упрощенными директивами:  
  - .data - определяет секцию инициализированных данных;  
  - .data? - определяет секцию неинициализированных данных, для тех случаев, когда необходимо предварительно выделить определенное количество памяти, но инициализировать ее заранее нет необходимости. Сколько бы мы ни определяли неинициализированных элементов данных, размер файла программы на диске остается неизменным. Мы просто таким образом ставим систему в известность: "когда программа загрузится, я хочу иметь в своем распоряжении такой-то объем памяти";  
  - .const - определяет секцию констант, то есть элементов данных, которые наша программа не сможет (во всяком случае, не должна) изменять ни в коем разе;  
  - .code - код, то есть последовательность инструкций, которые должен выполнить компьютер.  
  Из всех вышеперечисленных секций в нашем исходнике есть только CODE. Это единственная секция, которая обязательно должна присутствовать в любой программе.

 В секции кода у нас есть одна-единственная процедура. О ее начале свидетельствует строчка WinMain PROC, где WinMain - это имя, которое вы можете заменить другим. А ее конец - это строчка WinMain ENDP.

  О конце модуля транслятору сообщает директива end, и в качестве параметра у нее метка, на которую будет передано управление при старте программы. В данном случае - это имя "главной", а в нашем случае еще и единственной, процедуры WinMain.  
  ; - этот комментарий..  
  
  Вызов функций API из программы на ассемблере подчиняется соглашению "stdcall". Это соглашение заключаются в следующем:  
  1. Регистр символов в имени функции не имеет значения. Например, функции с именами WindowsMustDie и windowsmustdie - это одна и та же функция. Здесь мы наблюдаем отличие от требования языков C и C++, в которых идентификаторы регистро-зависимы.   
  2. 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. В остальных случаях программист должен разместить значение аргумента в памяти, а в стек записать 32-битный указатель на него. Таким образом, все параметры, передаваемые функции API, представляются 32-битными величинами.  
  3. Вызывающая программа загружает аргументы в стек последовательно, начиная с последнего, указанного в описании функции, и кончая первым. После загрузки всех аргументов программа вызывает функцию командой call.  
  4. За возвращение стека в исходное состояние после возврата из функции API отвечает сама вызываемая функция. Программисту заботиться о восстановлении указателя стека esp нет необходимости.  
  5. 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.)  
  В применении к нашему примеру это означает следующее:  
  - аргументы мы должны передать через стек;  
  - сама фунция вызывается командой call;  
  - не следим за балансом push и pop при передаче параметров через стек, функция все сделает сама.