Другие темы раздела

FASM Уроки Iczelion'a на FASM https://www.cyberforum.ru/ fasm/ thread1240590.html
Уроки Iczelion'a на FASM Урок первый. MessageBox на FASM format PE GUI include 'win32ax.inc' ; import data in the same section invoke MessageBox,NULL,msgBoxText,msgBoxCaption,MB_OK ...

Побайтовый вывод файла FASM
Пытаюсь ввести в консоль файл в шестнадцатеричном виде, но происходит ошибка при выполнении. format PE console 4.0 include 'win32a.inc' xor ebx, ebx ; invoke CreateFile,\ ...

FASM Вывод адреса на консоль
Пытаюсь на консоль вывести адрес fin: invoke printf, не робит - как правильно надо? format PE console 4.0 entry start include 'win32a.inc' section '.data' data readable fin ...

FASM ГСЧ на макросах Всем привет. Понадобилось заюзать ГСЧ посредством макросов, чтобы каждый раз на стадии компиляции, использовалось уникальное значение. Учитывая семантику препроцессора (там чёрт ногу сломит),... https://www.cyberforum.ru/ fasm/ thread1213146.html

FASM Создание окна на fasm https://www.cyberforum.ru/ fasm/ thread1209394.html
Всем привет. Только что начал изучать ассемблер fasm. Возник первый вопрос: как создать окно? Прошу не просто дать мне код, а ещё объяснить что значит. Заранее благодарен

FASM Вызываем функции из clib (библиотека Си) в DOS
Вобщем, сбылась мечта идиота. Теперь, нежели писать свой ввод/вывод(особенно всегда напрягал ввод/вывод вещественных чисел на экран), можно воспользоваться стандартными ф-циями из библиотеки языка...

Организовать вычисления по формуле FASM
привет, всем активным участникам этого чудесного форума!!! помогите, пожалуйста, написать программу на Fasm Assembler. задание: Создать программу на языке Ассемблер, что позволяет организовать...

FASM Как сделать выход по ESC org 100h old dw 0 jmp start number dw 0 c dw 0 start: xor ax,ax mov es,ax cli https://www.cyberforum.ru/ fasm/ thread1161834.html

FASM Получение CLSID image/png https://www.cyberforum.ru/ fasm/ thread1160365.html
Всем ку! int GetEncoderClsid(const WCHAR* format, CLSID* pClsid) { UINT num = 0; // number of image encoders UINT size = 0; // size of the image encoder array in bytes...

FASM Вывод трех строк в один MessageBox Здравствуйте, помогите, пожалуйста, с такой проблемой: не могу вывести 3 строки (Год+Месяц+День) в один MessageBox Вот такой код: format PE GUI 4.0 entry start include 'win32ax.inc' include... https://www.cyberforum.ru/ fasm/ thread1142589.html

MikI
Ушел с форума
13987 / 7000 / **813**
Регистрация: 11.11.2010
Сообщений: 12,592

01.09.2014, 06:04 **[TC]** **0**

Мануал по flat assembler

01.09.2014, 06:04. **Просмотров** 99783. **Ответов** 50
Метки (Все метки)

Ответ

3.1.2 Импорт

Макросы импорта помогают формировать данные импорта для PE файла (обычно составляющие отдельную секцию). Есть два макроса для этой цели.

- Первый **library**, должен быть помещен непосредственно в начале объявления данных импорта, и он определяет, из каких библиотек будут импортированы функции. Он должен сопровождаться любым количеством пар параметров, каждая пара, является меткой для таблицы импорта из данной библиотеки, и символьной строки, определяющей название библиотеки. Например:

Assembler Выделить код

```
1  library kernel32,'KERNEL32.DLL',\
2      user32,'USER32.DLL'
```

  объявляет, импорт из этих двух библиотек.
- Для каждой из библиотек, тогда должна быть объявлена таблица импорта где-нибудь в данных импорта. Это делается макроинструкцией **import**, которая нуждается в первом параметре, чтобы определить метку для таблицы (такую же самую как объявлено ранее в макрокоманде **library**), и затем пары параметров которые содержат метку для импортированного указателя и символьной строки, определяющей название функции точно с таким же именем которое экспортируется библиотекой. Например, вышеупомянутое объявление библиотек может быть закончено со следующими объявлениями импорта:

Assembler Выделить код

```
1  import kernel32,\
2      ExitProcess,'ExitProcess'
3  import user32,\
4      MessageBeep,'MessageBeep',\
5      MessageBox,'MessageBoxA'
```

  Метки, определенные первыми параметрами в каждой паре переданные макрокоманде **import**, адресуют указатели на двойные слова, которые после загрузки PE заполняются адресами экспортируемых процедур.

Вместо символьной строки для названия процедуры, можно задать номер, чтобы определить импорт по ординалу, например:

Assembler Выделить код

Assembler Выделить код

```
1  import custom,\
2      ByName,'FunctionName',\
3      ByOrdinal,17
```

Макрос **import** оптимизирует данные импорта, поэтому только те функции, которые используются где-нибудь в программе, будут помещены в таблицы импорта, и если некоторая таблица импорта была бы пуста, то на нее, целая библиотека не будет ссылаются вообще. По этой причине удобно иметь законченную таблицу импорта для каждой библиотеки — пакет содержит такие таблицы для некоторых из стандартных библиотек, они сохранены в подкаталогах **APIA** И **APIW** и импортируют ASCII и Unicode варианты функций API. Каждый файл содержит одну таблицу импорта, с меткой в нижнем регистре с тем же самым именем как название файла. Так законченные таблицы для того, чтобы импортировать из библиотек KERNEL32.DLL и USER32.DLL могут быть определены, вот так (если ваша переменная среда **INCLUDE** указывает на каталог содержащий пакет стандартных файлов):

Assembler Выделить код

```
1  library kernel32,'KERNEL32.DLL',\
2      user32,'USER32.DLL'
3  include 'apia\kernel32.inc'
4  include 'apiw\user32.inc'
```

Добавлено через 41 секунду

3.1.3 Процедуры (32-разрядные)

Есть четыре макроинструкции для того, чтобы вызывать процедуры с параметрами, переданными в стек.

- stdcall вызывает непосредственно процедуру указанную первым параметром, используя STDCALL соглашение о вызовах. Остальная часть параметров переданных макроинструкции, определяет параметры для процедуры и сохраняется в стеке в обратном порядке.
- Макроинструкция invoke делает то же самое, однако процедура вызывается косвенно, через указатель, помеченный первым параметром. Таким образом, invoke может использоваться, чтобы вызвать процедуры с помощью указателей, определенных в таблицах импорта. Эта строка:

  Assembler Выделить код

  ```
  1  invoke MessageBox,0,szText,szCaption,MB_OK
  ```

  является эквивалентной этой:

  Assembler Выделить код

  ```
  1  stdcall [MessageBox],0,szText,szCaption,MB_OK
  ```

  и они обе преобразуются в этот код:

  Assembler Выделить код

  ```
  1      push MB_OK
  2      push szCaption
  3      push szText
  4      push 0
  5      call [MessageBox]
  ```

- ccall и cinvoke походят на stdcall и invoke, но они должны использоваться, чтобы вызвать процедуры, использующих C соглашение о вызовах, где стековый фрейм должен быть восстановлен вызывающей программой.

Чтобы определить процедуру, использующую стек для параметров и локальных переменных, Вы должны использовать макроинструкцию **proc**. В самой простой форме она должна сопровождаться названием для процедуры и затем именами для всех параметров которые требуется, например:

Assembler Выделить код

```
1  proc WindowProc,hwnd,wmsg,wparam,lparam
```

Запятая между названием процедуры и первым параметром является дополнительной. Код процедуры должен следовать в следующих строках, и заканчиваться макроинструкцией **endp**. Стековый фрейм устанавливается автоматически на входе в процедуру, регистр EBP используется как основание, при обращении к параметрам, так что Вы должны избегать использовать этот регистр для других целей. Названия, указанные для параметров используются, чтобы определить метки, базирующиеся на значении в регистре EBP, которые Вы можете использовать, чтобы обратиться к параметрам как к переменным. Например:

Assembler Выделить код

```
1  mov eax, [hwnd]
```

инструкции в процедуре, определенные как в вышеупомянутом примере, являются эквивалентом

Assembler Выделить код

```
1  mov eax, [ebp+8]
```

Область видимости этих меток ограничена процедурой, так что Вы можете использовать те же самые названия для других целей вне данной процедуры. Так как любые параметры помещаются в стек как двойные слова при вызове таких процедур, метки для параметров определяются как двойные данные слова по умолчанию, однако Вы можете определять размеры для

параметров с помощью двоеточия и оператора размера следующего за названием параметра . Предыдущий пример может быть перезаписан, который является снова эквивалентным:

Assembler — Выделить код

```
1  proc WindowProc,hwnd:DWORD,wmsg:DWORD,\
2       wparam:DWORD,lparam:DWORD
```

Если Вы определяете размер меньший чем двойное слово, данная метка обращается к младшей части целого двойного слова, сохраненного в стеке. Если Вы, определяете больший размер, подобно указателю из учетверенного слова, два параметра размером с двойное слово будут определены, чтобы содержать это значение, но помечены как одна переменная.

Название процедуры может также сопровождаться **stdcall** или ключевым словом, которое определит используемое соглашение о вызовах. Когда никакой такой тип не определен, значение по умолчанию используется, которое является эквивалентным **STDCALL**. Также ключевое слово **uses** может следовать, и после него, список регистров (разделенных пробелами), которые будут автоматически сохранены на входе в процедуру и восстановлены на выходе. В этом случае должна быть запятая между списком регистров и первым параметром. Так что полностью показанная инструкция процедуры могла бы выглядеть следующим образом:

Assembler — Выделить код

```
1  proc WindowProc stdcall uses ebx esi edi,\
2       hwnd:DWORD,wmsg:DWORD,wparam:DWORD,lparam:DWORD
```

Чтобы объявлять локальную переменную, Вы можете использовать макроинструкцию **local**, сопровождаемую одним или более объявлениями, разделенными запятыми, каждое, состоящее из названия для переменной, сопровождаемое двоеточием и типом переменной — любым из стандартных типов (должен быть в верхнем регистре) или названием структуры данных. Например:

Assembler — Выделить код

```
1  local hDC:DWORD,rc:RECT
```

Чтобы объявлять локальный массив, Вы должны за названием переменной указать размер массива, заключенный в квадратные скобки, например:

Assembler — Выделить код

```
1  local str[256]:BYTE
```

Другой способ определять локальные переменные состоит в том, чтобы объявить их в блоке, начатом с макроинструкции "**locals**" и законченным "**endl**", в этом случае, они могут быть определены точно так же как допустимые данные. Это объявление — эквивалент примера приведенного выше:

Assembler — Выделить код

```
1  locals
2       hDC dd ?
3       rc RECT
4  endl
```

Локальные переменные могут быть объявлены где-нибудь в процедуре, с единственным ограничением, что они должны быть объявлены прежде, чем они используются. Область видимости меток для переменных, определенных как локальные ограничена внутренней частью процедуры, Вы можете использовать те же самые имена для других целей вне процедуры. Если Вы даете немного инициализированных значений переменным, объявленным как локальные, макроинструкция генерирует команды, которые инициализируют эти переменные с заданными значениями и помещает их в ту же самую позицию в процедуре, где это объявление помещено.

ret помещенный где-нибудь в процедуре, генерирует заключительный код, необходимый, чтобы правильно выйти из процедуры, восстанавливая стековый фрейм и регистры, используемые процедурой. Если Вы должны произвести инструкцию возврата из подпрограммы, используйте мнемонику **retn** или **ret** с числовым операндом, что также заставляет это интерпретироваться как единственная инструкция.

Полное определение процедуры может выглядеть следующим образом:

Assembler — Выделить код

```
1  proc WindowProc uses ebx esi edi,hwnd,wmsg,wparam,lparam
2       local hDC:DWORD,rc:RECT
3       ; здесь ваш код
4       ret
5  endp
```

Вернуться к обсуждению:
Мануал по flat assembler

Следующий ответ

0

Programming
Эксперт
94731 / 64177 / **26122**
Регистрация: 12.04.2006

01.09.2014, 06:04

Готовые ответы и решения:

Сообщений: 116,782

Неофициальная разработка Flat assembler версии 2.0.0
Разработчик Flat assembler'a Tomasz Grysztar в одном из блогов сообщил о разработке новой...

Flat assembler ругается на PROC
Доброго времени суток. Есть программа, собственно вот что она делает: "На экране инициализировать...

Как подключить include к flat компилятору
Здравствуйте,как подключить include к flat компилятору?Требуется подключить include 'win32a.inc' к...

Flat Assembler
Со временем задачи стали нерешаемыми из-за ужасно медленной скорости. Уже давно хочу перейти на...

50