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

Мануал по flat assembler

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

Ответ

3.1.9 Текстовые кодировки

Макросы ресурсов используют **du** директиву, чтобы определить любые строки Unicode в ресурсах . Так как эта директива просто расширяет символы, обнуляя 16-разрядные значения, для строк содержащих некоторые символы не ASCII, **du** возможно, должна быть переопределена. Для части кодировок макросы, переопределяющие du, чтобы генерировать тексты должным образом в Unicode находятся в подкаталоге ENCODING. Например, если исходный текст закодирован с кодовой страницей Windows 1251 (русская кодовая страница в Windows), такая строка должна быть помещена где-нибудь в начале источника:

Assembler Выделить код

```
1  include 'encoding\win1251.inc'
```

3.2 Расширенные заголовочные файлы

Расширенные заголовочные файлы **win32ax.inc**, **win32wx.inc**, **win64ax.inc** и **win64wx.inc** обеспечивают все функциональные возможности основных и включают еще несколько особенностей, вовлекающих более сложные макрокоманды. Также, если формат PE не объявлен прежде, чем включены расширенные заголовочные файлы, то они объявляют это автоматически. **win32axp.inc**, **win32wxp.inc**, **win64axp.inc** и **win64wxp.inc** — варианты расширенных заголовочных файлов которые дополнительно выполняют проверку количества параметров при вызовах процедур.

3.2.1 Параметры процедуры

С расширенными заголовочными файлами макросы позволяют вызывать процедуры с большими типами параметров а не только с двойными словами. Прежде всего, когда символьная строка — передается как параметр для процедуры, это используется, чтобы определить строковые данные, размещенные среди кода, и передать процедуре указатель на эту строку. Это позволяет легко определять символьные строки которые не должны многократно использоваться только в строке вызывающей процедуру которая требует указателей на эти строки, например:

Assembler Выделить код

```
1  invoke MessageBox,HWND_DESKTOP,"Message","Caption",MB_OK
```

Если параметр — группа, содержащая некоторые значения, разделенные запятыми, это будет обработано тем же самым способом как простой символьный строковый параметр.

Если параметру предшествует слово addr это означает, что это значение — адрес двойного слова, и этот адрес нужно передать процедуре, даже если это не может быть сделано непосредственно — как в случае локальных переменных, которые имеют адреса относительно регистра EBP, в таком случае, регистр EDX используется временно, чтобы вычислить значение адреса и передать его процедуре. Например:

Assembler Выделить код

```
1      invoke RegisterClass,addr wc
```

в случае, если, когда *wc* — локальная переменная с адресом **EBP-100h**, это генерирует такую последовательность команд:

Assembler Выделить код

```
1      lea edx,[ebp-100h]
2      push edx
3      call [RegisterClass]
```

Однако, когда данный адрес — не соотносится ни с каким регистром, это будет сохранено непосредственно.

Если параметру предшествует слово double, это будет обработано как значение на 64 бита и передано процедуре как два 32-разрядных параметра. Например:

Assembler Выделить код

```
1  invoke glColor3d,double 1.0,double 0.1,double 0.1
```

передаст процедуре три параметра на 64 бита как шесть двойных слов. Если параметр после double — операнд в памяти, он не должен иметь оператора размера, double уже работает как переопределение размера.

Наконец, вызовы процедур могут быть вложены, когда одна процедура может использоваться как параметр для другой. В таком случае значение, возвращенное вложенной процедуре в EAX передают как параметр для процедуры, которая ее содержит. Пример такого вложения:

Assembler Выделить код

```
1  invoke MessageBox,<invoke GetTopWindow,[hwnd]>,\
2      "Message","Caption",MB_OK
```

Нет никаких пределов для глубины вложения вызовов процедур.

3.2.2 Структурирование исходников

Расширенные заголовочные файлы включают некоторые макроинструкции, которые помогают с простым структурированием в программе. **.data** и **.code** — только ярлыки для объявления разделов для данных и для кода. Макрос **.end** должен быть помещен в конце программы, с одним параметром, определяющим точку входа программы, и это также автоматически сгенерирует раздел импорта, используя все стандартные таблицы импорта.

Макрос **.if** генерирует часть кода, который проверяет некоторое простое условие во время выполнения, и в зависимости от результата продолжает выполнение следующего блока или пропускает его. Блок должен быть закончен .endif, но ранее также макрос .elseif может использоваться один или более раз и начинать код, который будет выполнен при некотором дополнительном условии, когда предыдущие условия не были выполнены, и **.else** как последний блок перед **.endif**, который будет выполнен, когда все условия были ложные.

Другой способ определить условие состоит в том, чтобы обеспечить два значения и оператор сравнения между ними — один из **=**,**<**,**>**, **<=**, **>=**, и **<>**. Значения сравниваются как беззнаковое. Также, если Вы запишете только единственное значение как отдельный параметр, это будет проверено, на равенство нулю, и условие будет истинно, только если это не ноль. For example:

Assembler Выделить код

```
1  .if eax
2      ret
3  .endif
```

generates the instructions, which skip over the ret when the EAX is zero. There are также some special symbols recognized as conditions: the **ZERO?** is true when the ZF flag is set, in the same way the **CARRY?**, **SIGN?**, **OVERFLOW?** and **PARITY?** correspond to the state of CF, SF, OF and PF flags.

The simple conditions like above can be composed into complex conditional expressions using the **&**, **|** operators for conjunction and alternative, the **~** operator for negation, and parenthesis. For example:

Assembler Выделить код

```
1  .if eax<=100 & ( ecx | edx )
2      inc ebx
3  .endif
```

will generate the compare and jump instructions that will cause the given block to get executed only when EAX is below or equal 100 and at the same time at least one of the ECX and EDX is not zero.

Макрос **.while** генерирует команды, которые повторят выполнение данного блока (заканчивающегося **.endw**), пока условие истинно. Условие должно следовать за **.while** и может быть определено тем же самым способом что и для **.if**.

Пара макроинструкций **.repeat** и **.until** определяет блок, который будет выполняться неоднократно, пока данное условие не будет выполнено — на сей раз условие должно следовать за **.until** макрокомандой, помещенной в конце блока, подобно:

Assembler Выделить код

```
1  .repeat
2      add ecx,2
3  .until ecx>100
```

Вернуться к обсуждению:
Мануал по flat assembler

0

Programming
Эксперт
94731 / 64177 / **26122**
Регистрация: 12.04.2006
Сообщений: 116,782

01.09.2014, 06:06

Готовые ответы и решения:

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