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

10.08.2014, 10:51 **[TC]** **0**

Мануал по flat assembler

10.08.2014, 10:51. **Просмотров** 99777. **Ответов** 50
Метки (Все метки)

Ответ

2.1.6 Инструкции передачи управления

"jmp" передает управление а заданное место. Адрес назначения может быть определен непосредственно в инструкции или косвенно через регистр или память, допустимый размер адреса зависит от того, какой переход, близкий или дальний, а также от того, какая инструкция, 16-битная или 32-битная. Операнд для близкого перехода должен быть размером "word" для 16-битной инструкции и размером "dword" для 32-битной инструкции. Операнд для дальнего перехода должен быть размером "dword" для 16-битной инструкции и размером "pword" для 32-битной инструкции. Прямая инструкция "jmp" содержит адрес назначения как часть инструкции, операндом, определяющим этот адрес, должно быть числовое выражение для близкого перехода и два числа, разделенные двоеточием, для дальнего перехода, первое определяет номер сегмента, второе - смещение внутри сегмента. Непрямая инструкция "jmp" получает адрес назначения через регистр или переменную-указатель, операндом должен быть регистр общего назначения или память. Для более подробной информации смотрите также 1.2.5.

Assembler Выделить код

```
1       jmp 100h          ; прямой близкий переход
2       jmp 0FFFFh:0      ; прямой дальний переход
3       jmp ax            ; непрямой близкий переход
4       jmp pword [ebx]   ; непрямой дальний переход
```

- "call" передает управление процедуре, сохраняя в стеке адрес инструкции, следующей за "call", для дальнейшего возвращения к ней инструкцией "ret". Правила для операндов такие же, что с инструкцией "jmp", но "call" не имеет короткого варианта в виде прямой инструкции, и поэтому не оптимизирована.
- "ret", "retn" и "retf" прекращают выполнение процедуры передают управление назад программе, которая изначально вызвала эту процедуру, используя адрес, который был сохранен в стеке инструкцией "call". "ret" это эквивалент "retn", которая возвращает из процедуры, которая была вызвана с использованием близкого перехода, тогда как "retf" возвращает из процедуры, которая была вызвана с использованием дальнего перехода. Эти инструкции подразумевают размер адреса, соответствующий текущей установке кода, но этот размер может быть изменен на 16-битный с помощью мнемоников "retw", "retnw" и "retfw" и на 32-битный с помощью "retd", "retnd" и "retfd". Все эти инструкции могут опционально предписывать непосредственный операнд, если добавить константу к указателю стека, они эффективно удаляют любые аргументы, которые вызывающая программа положила в стек перед выполнением инструкции "call".
- "iret" возвращает управление прерванной процедуре. Эта инструкция отличается от "ret" в том, что она также выводит из стека флаги в регистр флагов. Флаги сохраняются в стек механизмом прерывания. Инструкция подразумевает размер адреса, соответствующий текущей установке кода, но этот размер может быть изменен на 16-битный или на 32-битный с помощью мнемоников "iretw" или "iretd".

Условные инструкции перехода осуществляют или не осуществляют передачу управления в зависимости от состояния флагов CPU во время вызова этих инструкций. Мнемоники условных переходов могут быть получены добавлением условных мнемоников (смотри таблицу 3) к символу "j", например инструкция "jc" осуществляет передачу управления, если установлен флаг CF. Условные переходы могут быть только близкие и прямые и могут быть оптимизированы (смотри 1.2.5), операндом должно быть число, определяющее адрес назначения.

Таблица 3.1 Условия

Мнемоник	Тестируемое условие	Описание
o	OF = 1	переполнение
no	OF = 0	нет переполнения
c b nae	CF = 1	перенос меньше не больше и не равно
nc ae nb	CF = 0	нет переноса выше или равно не меньше
e z	ZF = 1	равно ноль
ne nz	ZF = 0	не равно не ноль
be na	CF or ZF = 1	ниже или равно не выше
a nbe	CF or ZF = 0	выше не ниже и не равно
s	SF = 1	отрицательное
ns	SF = 0	положительное
p pe	PF = 1	четное
np po	PF = 0	нечетное
l nge	SF xor OF = 1	меньше не больше и не равно
ge nl	SF xor OF = 0	больше или равно не меньше
le ng	(SF xor OF) or ZF = 1	меньше или равно не больше
g nle	(SF xor OF) or ZF = 0	больше не меньше и не равно

- "loop" - это условные переходы, которые используют значение из CX (или ECX) для определения количества повторений цикла. Все инструкции "loop" автоматически уменьшают на единицу CX (или ECX) и завершают цикл, когда CX (или ECX) равно нулю. CX или ECX используется в зависимости от от текущей установки разрядности кода - 16-ти или 32-битной, но вы можете принудительно использовать CX с помощью мнемоника "loopw", или ECX с помощью мнемоника "loopd".
- "loope" и "loopz" это синонимы этой инструкции, которые работают так же, как стандартный "loop", но еще завершают работу, если установлен ZF. "loopew" и "loopzw" заставляют использовать регистр CX, а "looped" и "loopzd" заставляют использовать регистр ECX.
- "loopne" и "loopnz" это тоже синонимы той же инструкции, которые работают так же, как стандартный "loop", но еще завершают работу, если ZF сброшен на ноль. "loopnew" и "loopnzw" заставляют использовать регистр CX, а "loopned" и "loopnzd" заставляют использовать регистр ECX.

Каждая инструкция "loop" требует операндом число, определяющее адрес назначения, причем это может быть только близкий переход (в пределах 128 байт назад и 127 байт вперед от адреса инструкции, следующей за "loop").

- "jcxz" переходит к метке, указанной в инструкции, если находит в CX ноль, "jecxz" делает то же, но проверяет регистр ECX. Правила для операндов такие же, как с инструкцией "loop".
- "int" активирует стандартный сервис прерывания, который соответствует числу, указанному как операнд в мнемонике. Это число должно находиться в пределах от 1 до 255. Стандартный сервис прерывания заканчивается мнемоникой "iret", которая возвращает управление инструкции, следующей за "int". "int3" кодирует короткое (в один байт) системное прерывание, которое вызывает прерывание 3. "into" вызывает прерывание 4, если установлен флаг OF.
- "bound" проверяет, находится ли знаковое число, содержащееся в указанном регистре в нужных пределах. Если число в регистре меньше нижней границы или больше верхней, вьзывается прерывание 5. Инструкция нуждается в двух операндах, первый - это тестируемый регистр, вторым должен быть адрес в памяти для двух знаковых чисел, указывающих границы. Операнды могут быть размером "word" или "dword".

Assembler Выделить код

Assembler Выделить код

```
1      bound ax,[bx]    ; тестирует слово на границы
2      bound eax,[esi]  ; тестирует двойное слово на границы
```

Вернуться к обсуждению:
Мануал по flat assembler

Следующий ответ

2

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

10.08.2014, 10:51

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