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, 06:00 **[TC]** **0**

Мануал по flat assembler

01.09.2014, 06:00. **Просмотров** 99777. **Ответов** 50
Метки (Все метки)

Ответ

2.3.6 Условный препроцессинг

При применении директивы "*match*" некоторый блок кода обрабатывается препроцессором и передается ассемблеру, только если заданная последовательность символов совпадает с образцом. Образец идет первым, заканчивается запятой, далее идут символы, которые должны подходить под образец, и далее блок кода, заключенный в фигурные скобки, как макроинструкция.
Есть несколько правил для построения выражения для сравнения, первое - это любые символьные знаки и строки в кавычках должны соответствовать абсолютно точно. В этом примере:

Assembler Выделить код

```
1      match +,+ { include 'first.inc' }
2      match +,- { include 'second.inc' }
```

Первый файл будет включен, так как "+" после запятой соответствует "+" в образце, а второй файл не будет включен, так как совпадения нет.
Чтобы соответствовать любому другому символу буквально, он должен предваряться знаком "=" в образце. Также чтобы привести в соответствие сам знак "=", или запятую должны использоваться конструкции "==" и "=,". Например, образец "=*a*==" будет соответствовать последовательности "*a*=".
Если в образце стоит некоторый символ имени, он соответствует любой последовательности, содержащей по крайней мере один символ и его имя заменяется на поставленную в соответствие последовательность везде в следующем блоке, аналогично параметрам в макроинструкции. На пример:

Assembler Выделить код

```
1      match a-b, 0-7
2       { dw a,b-a }
```

сгенерирует инструкцию "dw 0, 7-0". Каждое имя всегда ставится в соответствие как можно меньшему количеству символов, оставляя оставшиеся, то есть:

Assembler Выделить код

```
1      match a b, 1+2+3 { db a }
```

имя "a" будет соответствовать символу "1", оставляя последовательность "+2+3" в соответствие с "b". Но, таким образом:

Assembler Выделить код

```
1      match a b, 1 { db a }
```

для "b" ничего не остается, и блок вообще не будет обработан.
Блок кода, определенный директивой "*match*" обрабатывается так же, как любая макроинструкция, поэтому здесь могут использоваться любые операторы, специфичные для макроинструкций.
Что делает директиву "*match*" очень полезной, так это тот факт, что она заменяет символьные константы на их значения в поставленной в соответствие последовательности символов (то есть везде после запятой до начала блока кода) перед началом сопоставления. Благодаря этому директива может использоваться, например, для обработки некоторого блока кода в зависимости от выполнения условия, что данная символьная константа имеет нужное значение, например:

Assembler Выделить код

```
1      match =TRUE, DEBUG { include 'debug.inc' }
```

здесь файл будет включен, только если символьная константа "DEBUG" определена со значением "TRUE".

2.3.7 Порядок обработки

При сочетании разных свойств препроцессора важно знать порядок, в котором они обрабатываются. Как уже было отмечено, высший приоритет имеет директива "*fix*" и замены, ею определенные. Это полностью делается перед совершением любого другого препроцессинга, поэтому такой кусок кода:

Assembler Выделить код

```
1      V fix {
2        macro empty
3         V
4      V fix }
5         V
```

делает допустимое определение пустого макроса. Можно сказать, что директива "*fix*" и приоритетные константы обрабатываются на отдельной стадии, и весь остальной препроцессинг делается на результирующем коде.
Стандартный препроцессинг, который начинается после, на каждой строке начинается с распознавания первого символа. Сначала идет проверка на директивы препроцессора, и если ни одна из них не опознана, препроцессор проверяет, является ли первый символ макроинструкцией. Если макроинструкция не найдена, препроцессор переходит ко второму символу на строке, и снова начинает с проверки на директивы, список которых в этом случае ограничивается лишь "*equ*", так как только она может оказать вторым символом на строке. Если нет директивы, второй символ проверяется на структурную макроинструкцию, и если ни одна из этих проверок не дала положительного результата, символьные константы заменяются на их значения, и строка передается ассемблеру.
Продемонстрируем это на примере. Пусть "*foo*" - это макрос, а "*bar*" - это структура. Эти строки:

Assembler Выделить код

```
1      foo equ
2      foo bar
```

обе будут интерпретированы как вызовы макроса "*foo*", так как значение первого символа берет верх над значением второго.
Макроинструкции генерируют новые строки от их блоков определения, заменяя параметры на их значения и далее обрабатывая операторы "#" и "`". Оператор конверсии имеет высший приоритет, чем оператор сцепления.
После завершения этого, заново сгенерированная строка проходит через стандартный препроцессинг, как описано выше.
Хотя обычно символьные константы заменяются исключительно в строках, нет ни директив препроцессора, ни макроинструкций, встречается несколько особых ситуаций, где замены проводятся в частях строк, содержащих директивы. Первая - это определение символьной константы, где замены производятся везде после слова "equ" и результирующее значение присваивается новой константе (смотрите 2.3.2). Вторая такая ситуация - это директива "match", где замены производятся в символах, следующих за запятой перед сопоставлением их с образцом. Эти свойства могут использоваться, например, для сохранения списков, как,например, эта совокупность определений:

Assembler Выделить код

```
1      list equ
2
3      macro append item
4       {
5         match any, list \{ list equ list,item \}
6         match , list \{ list equ item \}
7       }
```

Здесь константа "*list*" инициализируется с пустым значением, и макрос "append" может использоваться для добавления новых пунктов к списку, разделяя их запятыми. Первое сопоставление в этом макросе происходит, только если значение списка не пусто (смотрите 2.3.6), таким образом новое его значение - это предыдущее с запятой и новым пунктом, добавленным в конец. Второе сопоставление происходит, только если список все еще пуст, и таким образом список определяется как содержащий только лишь новый пункт. Так, начиная с пустого списка, "append 1" определит "list equ 1", а "append 2", следующий за ним, определит "list equ 1,2". Может потребоваться использовать этот список как параметры к некоторому макросу. Но это нельзя сделать прямо - если "foo" это макрос, то в строке "foo list" символ "list" просто прошел бы как параметр к макросу, поскольку символьные константы на этой стадии ещё не развернуты. Для

этой цели снова оказывается удобна директива "match":

Assembler — Выделить код

```
1     match params, list { foo params }
```

Значение "*list*", если оно не пустое, соответствует ключевому слову "params", которое далее во время генерации строк, заключенных в фигурные скобки, заменяется на соответственное значение. Так, если "*list*" имеет значение "1,2", строка, указанная выше, сгенерирует строку, содержащую "foo 1,2", которая далее пройдет стандартный препроцессинг.
Есть ещё один особый случай - когда препроцессор собирается проверить второй символ и натыкается на двоеточие (что далее интерпретируется асемблером как определение метки), он останавливается в этом месте и заканчивает препроцессинг первого символа (то есть если это символьная константа, она развертывается) и если это все еще выглядит меткой, совершается стандартный препроцессинг, начиная с места после метки. Это позволяет поместить директивы препроцессора и макроинструкции после меток, аналогично инструкциям и директивам, обрабатываемым ассемблером, например:

Assembler — Выделить код

```
1     start: include 'start.inc'
```

Однако если метка во время препроцессинга разрушается (например, если у символьной константы пустое значение), происходит только замена символьных констант до конца строки.

Вернуться к обсуждению:
Мануал по flat assembler

Следующий ответ

0

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

01.09.2014, 06:00

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