Другие темы раздела

FASM Fasm dll https://www.cyberforum.ru/ fasm/ thread1252260.html
Как в fasm создать dll файл?

FASM Как использовать структуру sqlite3?
Хотелось бы прикрутить sqlite к своей проге с dll кой проблем нет. где взять структуры описанные в sqlite3.c

Делаем в IDE FASM'а кнопку Debug и дружим его с OllyDbg FASM
Делаем в IDE FASM'а кнопку Debug и дружим его с OllyDbg статья была взята здесь Статья посвящена всем любителям компилятора FASM и тем кто пишет код используя его IDE. Известно что любое...

FASM Как вывести время работы программы? Доброе время, суток! У меня такой вопрос, как вывести время работы программы? Скажем есть такая простенькая программа, которая считает от 1 миллиарда до 0, вот как сделать чтоб после того как она... https://www.cyberforum.ru/ fasm/ thread1248143.html

FASM Бегущая строка в текстмоде, нежно. Насилуем знакогенератор https://www.cyberforum.ru/ fasm/ thread1244262.html
Вотъ. Использованы куски из моей XVGA, писаные ещё в 1992, так что не обессудьте. В качестве мана пользовал Richard Wilton, "Programmer's Guide to PC and PS/2 Video Programming." ;FASM - сохранять...

Мануал по flat assembler FASM
flat assembler 1.71 Мануал программера перевод "flat assembler 1.71 Programmer's Manual" by Tomasz Grysztar перевод выполнили Paranoik и Mikl___ Содержание ...

FASM Уроки Iczelion'a на FASM
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 ...

FASM Побайтовый вывод файла 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,\ ... https://www.cyberforum.ru/ fasm/ thread1219549.html

FASM Вывод адреса на консоль https://www.cyberforum.ru/ fasm/ thread1219432.html
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

Mikl___
Ушел с форума
13980 / 6996 / **810**
Регистрация: 11.11.2010
Сообщений: 12,580

10.09.2014, 04:59 **[TC]** **0**

Руководство по препроцессору FASM

10.09.2014, 04:59. **Просмотров** 10863. **Ответов** 7
Метки (Все метки)

Ответ

9. Оператор *FIX* и макросы внутри макросов

В стародавние времена, в FASMe отсутствовала одна полезная возможность - создавать макросы внутри других макросов. Например, что бы при развёртывании макроса был бы определён новый макрос. Что-то вроде гипотетичного:

Assembler Выделить код

```
1  macro   declare_macro_AAA
2  {
3    macro AAA
4    {
5      db  'AAA',0
6    } ;завершаем определение AAA
7  }   ;завершаем определение declare_macro_AAA
```

Проблема в том, что когда макрос

Assembler Выделить код

```
1  declare_macro_AAA
```

обрабатывается препроцессором, первая найденная скобочка "**}**" считается завершением определения его, а не так как хотелось бы. Так же происходит и с другими символами и/или операторами (например, "**#**", "**`**", "**forward**", "**local**").

9.1. Explaination of fixes

Но со временем, была добавлена новая директива. Она работает подобно "**EQU**", но обрабатывается до любого другого препроцессинга. (За исключением предварительных операций, про которые говорится в разделе "Общие понятия" - они выполняются как бы до самого препроцессинга, но это уже внутренние детали, не слишком интересные). Директива эта называется **FIX**:

Синтаксис :

Assembler Выделить код

```
1  <name1> <fix name2>
```

Видно, что синтаксис такой же как у "**EQU**", но как я сказал, когда препроцессор обрабатывает часть кода, он смотрит, есть ли "**FIX**", а потом уже делает всё остальное. Например код:

Assembler Выделить код

```
1  a equ 1
2  b equ a
3  a b
```

Then preprocesisng happens like this:

Preprocessing line 1:
a - Preprocessor finds unknown word, skips it.
equ - "equ" is second word of line, so it remembers "a" equals rest of line ("b") and deletes line
Preprocessing line 2:
b - Preprocessor finds unknown word, skips it.
equ - "equ" is second word of line, so it remembers "b" equals rest of line ("a") and deletes line
Preprocessing line 3:
a - Preprocessor replaces "a" with "1"
b - Preprocessor replaces "b" with "a"

So it becomes:

Assembler Выделить код

```
1  1 a
```

But if we have

Assembler Выделить код

```
1  a fix 1
2  b fix a
3  a b
```

then it looks like:

Fixing line 1: No symbols to be fixed
Preprocessing line 1:
a - Preprocessor finds unknown word, skips it.
fix - "fix" is second word of line, so it remembers "a" is fixed to rest of line ("b") and deletes line
Fixing line 2: "a" is fixed to "1", so line becomes "b fix 1"
Preprocessing line 2:
b - Preprocessor finds unknown word, skips it.
fix - "fix" is second word of line, so it remembers "b" is fixed to rest of line ("1") and deletes line
Fixing line 3: "a" is fixed to "1", "b" is fixed to "1" so line becomes "1 1"
Preprocessing line 3:
1 - Preprocessor finds unknown word, skips it.
1 - Preprocessor finds unknown word, skips it.

This was only example to see how fixing works, usually it isn't used in this manner.

9.2. Using fixes for nested macro declaration

Now back to declaring macro inside macro - First, we need to know how are macros preprocessed. You can quite easily make it out yourself - on macro declaration macro body is saved, and when macro is being expanded preprocessor replaces line with macro usage by macro body and internally declares equates to handle arguments and continues with preprocessing of macro body. (of course it is more complicated but this is enough for understanding fixes).

So where was problem with declaring macro inside macro? First time compiler found "**}**" inside macro body it took it as end of macro body declaration, so there wasn't any way to include "**}**" in macro body. So we can easily fix this

Assembler Выделить код

```
1   macro a
2   {
3     macro b
4     %_
5        display 'Never fix before something really needs to be fixed'
6     _%
7   }
8   %_ fix {
9   _% fix }
10  a
11  b
```

Now preprocessing looks like (simplified)

1. Preprocessor loads declaration of macro "a"
2. Preprocessor loads declaration of fixes "%_" and "_%"
3. Preprocessor expands macro "a"
4. Preprocessor loads macro "b" declaration ("_%" and "%_" are fixed in each line before being handled by rest of preprocessor)
5. Preprocessor expands macro "b"

Here you see how important is placing of declaration of fixes, because macro body is fixed too before it's loaded by preprocessor. For example this won't work:

Assembler Выделить код

```
1  %_ fix {
2  _% fix }
3  macro a
4  {
5    macro b
6    %_
7       display 'Never fix before something really needs to be fixed, here you see it'
8    _%
9  }
10 a
11 b
```

Because "**%_**" and "**_%**" will be fixed before loading macro "**a**", so loading macro body will end at "**_%**" fixed to "**}**" and second "**}**" will remain there.

NOTE: Character "**%**" isn't special character for FASM's preprocessor, so you use it just like any normal character, like "**a**" or "**9**". It has special meaning AFTER preprocessing, and only when it is only char in whole word ("**%**" not "**anything%anything**").

We also need to fix other macro-releated operators:

Assembler Выделить код

```
1 %_ fix {
2 _% fix }
3 %local fix local
4 %forward fix forward
5 %reverse fix revese
6 %common fix common
7 %tostring fix `
```

Only **#** is special case, you can fix it, but there is a easier way. Every time preprocessor finds multiple **#**s, it removes one, so it is something like (this won't really work)

Assembler Выделить код

```
1 etc...
2 ###### fix #####
3 #####  fix ####
4 ####   fix ###
5 ###    fix ##
6 ##     fix #
```

So instead of using symbol fixed to "**#**" you can just use "**##**" etc.

9.3. Using fixes for moving parts of codes

You can also use fixes to move parts of code. In assembly programming is this useful especially when you break code into modules, but you want to have data and code grouped in separate segment/section, but defined in one file.

Right now this part of tutorial is **TODO**, I hope I will write it soon, for now you can look at JohnFound's Fresh's macro library, file

Assembler Выделить код

```
1 INCLUDE\MACRO\globals.inc
```

Я знаю, **FIX**ы могут смутить, и хорошо бы понимать внутренние детали работы препроцессора, но они предоставляют очень большие возможности. Создатель FASM'а сделал его настолько мощным, на сколько это возможно, даже за счёт некоторого ущерба удобочитаемости.

Assembler Выделить код

```
1 a   equ 10
2 b   fix 10
3 mov ax, a
4 mov bx, b
```

будет преобразован в:

Assembler Выделить код

```
1 mov ax, 10
2 mov bx, 10
```

Но при обработке такого кода:

Assembler Выделить код

Assembler Выделить код

```
1   equ fix =
2  a   equ 10
3  mov ax, a
```

в первой строк директива **FIX** скажет препроцессору поменять все **EQU** на **=**. Далее, перед обработкой следующей строки, препроцессор проверит, нет ли там пофиксеных идентификаторов. Так что в нашей второй строке **equ** будет заменено на **=**, и строка примет вид **a = 10**. Так что никакой другой обработки этой строки не будет выполнено. А значит, и третья строка не будет преобразовываться препроцессором, так как идентификатор a не будет определён директивой **EQU**. Результат всего этого будет такой:

Assembler Выделить код

```
1  a   = 10
2  mov ax, a
```

Директива **FIX** может быть использован и для определения макросов в макросах - того, что мы хотели сделать в нашем гипотетичном примере. Делается это подобным образом:

Assembler Выделить код

```
1   macro   declare_macro_AAA
2   {
3     macro AAA
4     %_
5       db  'aaa',0
6     _%
7   }
8
9   %_ fix {
10  _% fix }
11
12  declare_macro_AAA
```

Здесь, препроцессор найдёт объявление макроса **declare_macro_AAA** и определит его, далее будет два **FIX**, и потом использование макроса **declare_macro_AAA**. Так что он преобразует это в:

Assembler Выделить код

```
1   macro   declare_macro_AAA
2   {
3     macro AAA
4     %_
5       db  'aaa',0
6     _%
7   }
8
9   %_ fix {
10  _% fix }
11
12  macro   AAA
13  %_
14      db  'aaa',0
15  _%
```

и теперь уже содержимое нового макроса будет обработано препроцессором. Далее будут заменены аргументы FIXов, и получится:

Assembler Выделить код

```
1   macro   declare_macro_AAA
2   {
3     macro AAA
4     %_
5       db  'aaa',0
6     _%
7   }
8
9   macro   AAA
10  {
11      db  'aaa',0
12  }
```

как мы и хотели.
Подобным образом можно пофиксить все остальные проблематичные вещи:

Assembler Выделить код

Assembler Выделить код

```
1  macro   declare_macro_TEXT
2  {
3    macro TEXT [arg]
4    %_
5      %forward
6      db  %x arg
7    _%
8  }
9
10 %_ fix {
11 _% fix }
12 %forward fix forward
13
14 declare_macro_TEXT
15
16 %x fix `
17
18 TEXT abc,def
```

В этом примере нужно обратить внимание на один момент: строка **%x fix `** должна находиться после **declare_macro_TEXT**. Если б она находилась до, то **%x** было бы пофиксено во время развёртывания макроса, и тогда **`arg** приняло бы вид **'arg'**, следовательно макрос **TEXT** был бы объявлен так:

Assembler Выделить код

```
1 macro   TEXT [arg]
2 {
3   forward
4     db 'arg'    ;строка не зависит от аргументов
5 }
```

Но, в нашем случае он будет:

Assembler Выделить код

```
1 macro   TEXT [arg]
2 {
3   forward
4     db `arg     ;имена аргументов превращаются в строки
5 }
```

Этот пример показывает, как важно местонахождение **FIX**.
Иногда необходимо фиксить идентификаторы дважды:

Assembler Выделить код

```
1  macro m1
2  {
3    macro m2
4    %_
5      macro m3 [arg]
6      %%_
7        db arg
8      _%%
9    _%
10 }
11
12 %%_ fix %_
13 _%% fix _%
14 %_ fix {
15 %_ fix }
16
17 m1
18 m2
19 m3
```

Символы фиксятся даже во время препроцессинга других **FIX**, так что код выше не будет работать, если порядок будет такой:

Assembler Выделить код

```
1 %_ fix {
2 %_ fix }
3 %%_ fix %_
4 _%% fix _%
```

В этом случае строка

Assembler Выделить код

```
1 %%_ fix %_
```

была бы пофиксена сразу же после

Assembler — Выделить код

```
1  %_ fix {
```

, так что все последующие **%%_** сразу же преобразовались бы в **}**. То же самое и для

Assembler — Выделить код

```
1  _%% fix _%
```

.

Заключение

Не забывайте читать документацию FASM. Практически всё, что есть в туториале, можно найти там. Может быть написано и немного сложнее для изучения, но лучше подойдёт в качестве справочной информации. Не так сложно запомнить - 99% пользователей FASM научились его использовать по этой документации и при помощи форума.

Вернуться к обсуждению:
Руководство по препроцессору FASM

Следующий ответ

3

IT_Exp
Эксперт
87844 / 49110 / **22898**
Регистрация: 17.06.2006
Сообщений: 92,604

10.09.2014, 04:59

Требуется директива препроцессору
Создаю проект "Консольное приложение" на Visual C#. Код : #include <stdio.h> int main(void) {...

Видимость переменных и директивы препроцессору, не видит поле
Есть поле public float zoomSpeed = 0; Есть метод, в нем строки для разных платформ. void...

При создании файла заголовка в Code::Blocks вставляются какие-то команды препроцессору.
Вот что появляется при создании файла rectangle.hpp: #ifndef RECTANGLE_HPP_INCLUDED #define...

Руководство
Как вообще по spring 4 его руководство читать, может кто-нибудь переведет или че путное есть а не...

0