



FASM

[Форум программистов и сисадминов Киберфорум](#) > [Форум программистов](#) > [Низкоуровневое программирование](#) > [Assembler, MASM, TASM](#) > [FASM](#)

[Войти](#)[Регистрация](#)[Восстановить пароль](#)[Правила](#)[Карта](#)[Блоги](#)[Сообщество](#)[Поиск](#)

Другие темы раздела

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

Делаем в IDE FASM'a кнопку Debug и дружим его с OllyDbg FASM
Делаем в IDE FASM'a 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 Бегущая строка в текстмде, нежно. Насилуем знакогенератор <https://www.cyberforum.ru/fasm/thread1244262.html>

Вот. Используются куски из моей XVGA, писанные 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...

FASM Уроки Iczelion'a на FASM

Уроки Iczelion'a на FASM Урок первый. MessageBox in FASM format PE GUI
include 'win32ax.inc' ; import data in the same section invoke
MessageBox,NULL,msgBoxText,msgBoxCaption,MB_OK ...

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 Как использовать структуру sqlite3?

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

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>

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 Miki____
Содержание ...

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 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>

Miki

Ушел с форума



13958 / 6977 / 806

Регистрация:

11.11.2010

Сообщений:

12,550

09.09.2014, 12:56 [ТС]

0

Руководство по препроцессору FASM

09.09.2014, 12:56. Просмотров 10410. Ответов 2

Метки ([Все метки](#))**Ответ****4. Простые макросы без аргументов****4.1. Определение простых макросов**

Используя **EQU** можно делать наиболее простые замены в исходном тексте при обработке препроцессором. Большими возможностями обладают макросы. Командой **MACRO** можно создавать собственные инструкции.

Синтаксис:

Assembler

[Выделить код](#)

```
1 macro name
2 {
3 ; тело макроса
4 }
```

Когда препроцессор находит директиву **macro**, он определяет макрос с именем **name**. Далее, встретив в исходном тексте строку, начинающуюся с **name**, препроцессор заменит **name** на тело макроса - то, что указано в определении между скобками { и }. Имя макроса может быть любым допустимым идентификатором, а тело макроса - всё, что угодно, за исключением символа }, который означает завершение тела макроса.

Например:

Assembler

[Выделить код](#)

```
1 macro a
2 {
3 push eax
4 }
5 xor eax, eax
6 a
```

будет заменено на:

Assembler

[Выделить код](#)

```
1 xor eax, eax
2 push eax
```

Или:

Assembler[Выделить код](#)

```

1 macro a
2 {
3 push    eax
4 }
5 macro b
6 {
7 push    ebx
8 }
9 b
10 a

```

получим:

Assembler[Выделить код](#)

```

1 push    ebx
2 push    eax

```

Разумеется, макросы не обязательно оформлять так, как выше, можно делать и так:

Assembler[Выделить код](#)

```

1 macro push5 {push dword 5}
2 push5

```

получим:

Assembler[Выделить код](#)

```

1 push    dword 5

```

Или:

Assembler[Выделить код](#)

```

1 macro push5 {push dword 5}
2 }

```

с тем же самым результатом. Скобочки можете размещать как хотите.

4.2. Вложенные макросы

Макросы могут быть вложенными один в другой. То есть, если мы переопределим макрос, будет использовано последнее определение. Но если в теле нового определения содержится тот же макрос, то будет использовано предыдущее определение. Посмотрите пример:

Assembler[Выделить код](#)

```

1 macro a { mov ax, 5}
2
3 macro a
4 {
5     a
6     mov bx, 5
7 }
8
9 macro a
10 {
11     a
12     mov cx, 5
13 }
14     a

```

в результате получим:

Assembler[Выделить код](#)

```

1     mov ax, 5
2     mov bx, 5
3     mov cx, 5

```

Или такой пример:

Assembler[Выделить код](#)

Assembler

[Выделить код](#)

```

1  macro  a {1}
2  a
3
4  macro  a {
5      a
6      2 }
7  a
8
9  macro  a {
10     a
11     3 }
12
13 a

```

получим:

Assembler

[Выделить код](#)

```

1      1
2
3      1
4      2
5
6      1
7      2
8      3

```

4.3. Директива PURGE. Отмена определения макроса

Как и в случае с директивой **EQU**, можно отменить определение макроса. Для этого используется директива **PURGE** с указанием имени макроса.

Синтаксис:

Assembler

[Выделить код](#)

```
1 purge name
```

Пример:

Assembler

[Выделить код](#)

```

1 a
2 macro a {1}
3 a
4 macro a {2}
5 a
6 purge a
7 a
8 purge a
9 a

```

получим:

Assembler

[Выделить код](#)

```

1 a
2 1
3 2
4 1
5 a

```

Если применить PURGE к несуществующему макросу, ничего не произойдёт.

4.4. Поведение макросов

Имя макроса будет заменено его телом не только в том случае, если оно расположено в начале строки. Макрос может находиться в любом месте исходного текста, где допустима мнемоника инструкции (например, **add** или **mov**). Всё потому, что основное предназначение макросов - имитировать инструкции. Единственное исключение из этого правила - макросы недопустимы после префиксов инструкций (**rep**).

Пример:

Assembler

[Выделить код](#)

Assembler

[Выделить код](#)

```

1 macro   CheckErr
2 {
3     cmp eax, -1
4     jz   error
5 }
6
7     call    Something
8 a: CheckErr    ; здесь макросу предшествует метка, всё Ок.
```

получим:

Assembler

[Выделить код](#)

```

1     call    Something
2 a: cmp eax,-1
3     jz   error
```

Пример #2:

Assembler

[Выделить код](#)

```

1 macro   stos0
2 {
3     mov al, 0
4     stosb
5 }
6     stos0    ;это место инструкции, будет замена.
7 here: stos0    ;это тоже место инструкции.
8     db  stos0 ;здесь инструкции не место, замены не будет.
```

получим:

Assembler

[Выделить код](#)

```

1     mov al, 0
2     stosb
3 here: mov al, 0
4     stosb
5     db  stos0
```

Возможно переопределять (**overload**) инструкции посредством макросов. Так как препроцессор ничего об инструкциях не знает, он позволяет использовать мнемонику инструкции в качестве имени макроса:

Assembler

[Выделить код](#)

```

1 macro   pusha
2 {
3     push eax ebx ecx edx ebp esi edi
4 }
5 macro   popa
6 {
7     pop edi esi ebp edx ecx ebx eax
8 }
```

эти две новые инструкции будут экономить по четыре байта в стеке, так как не сохраняют ESP (правда, занимают побольше места, чем реальные инструкции 😊). Всё же, переопределение инструкций не всегда хорошая идея - кто-нибудь читая Ваш код может быть введен в заблуждение, если он не знает, что инструкция переопределена. Также, возможно переопределять директивы ассемблера:

Assembler

[Выделить код](#)

```

1 macro   use32
2 {
3     align 4
4     use32
5 }
6
7 macro   use16
8 {
9     align 2
10    use16
11 }
```

5. Макросы с фиксированным количеством аргументов

5.1. Макросы с одним аргументом

Макросы могут иметь аргумент. Аргумент представляет собой какой-либо идентификатор, который будет повсюду заменён в теле макроса тем, что будет указано при использовании.

Синтаксис:

Assembler

[Выделить код](#)

```
1 macro <name> <argument> { <тело макроса> }
```

Например:

Assembler

[Выделить код](#)

```
1 macro add5 where
2 {
3     add where, 5
4 }
5
6     add5    ax
7     add5    [variable]
8     add5    ds
9     add5    ds+2
```

получим:

Assembler

[Выделить код](#)

```
1     add ax, 5
2     add [variable], 5
3     add ds, 5 ;такой инструкции не существует
4               ;но препроцессор это не волнует.
5               ;ошибка появится на стадии ассемблирования.
6     add ds+2,5 ;ошибка синтаксиса, как и ранее
7               ;определится при анализе синтаксиса (parsing).
```

(разумеется, комментарии в результате работы препроцессора не появятся 🤖)

5.2. Макросы с несколькими аргументами

У макросов может быть несколько аргументов, разделённых запятыми ",":

Assembler

[Выделить код](#)

```
1 macro movv where, what
2 {
3     push    what
4     pop     where
5 }
6
7 movv    ax, bx
8 movv    ds, es
9 movv    [var1], [var2]
```

преобразуется в:

Assembler

[Выделить код](#)

```
1     push    bx
2     pop     ax
3
4     push    es
5     pop     ds
6
7     push    [var2]
8     pop     [var1]
```

Если несколько аргументов имеют одно и тоже имя, то будет использован первый из них 🤖.

Если при использовании макроса указать меньше аргументов, чем при определении, то значения неуказанных будет пустым:

Assembler

[Выделить код](#)

```
1 macro    pupush a1, a2, a3, a4
2 {
3     push    a1 a2 a3 a4
4     pop     a4 a3 a2 a1
5 }
6
7     pupush  eax, dword [3]
```

получим:

Assembler

[Выделить код](#)

```
1     push    eax dword [3]
2     pop     dword [3] eax
```

Если в аргументе макроса необходимо указать запятую как символ (","), тогда необходимо аргумент заключить в скобочки из символов < и >.

Assembler

[Выделить код](#)

```
1 macro   safe_declare name, what
2 {
3     if used name
4         name    what
5     end if}
6
7 safe_declare    var1, db 5
8 safe_declare    array5, <dd 1,2,3,4,5>
9 safe_declare    string, <db "привет, я просто строка",0>
```

получим:

Assembler

[Выделить код](#)

```
1  if used var1
2      var1    db 5
3  end if
4
5  if used array5
6      array5  dd 1,2,3,4,5
7  end if
8
9  if used string
10     string  db "привет, я просто строка",0
11 end if
```

Конечно же, можно использовать символы < и > и внутри тела макроса:

Assembler

[Выделить код](#)

```
1 macro   a arg {db arg}
2 macro   b arg1,arg2 {a <arg1,arg2,3>}
3 b    <1,1>,2
4 получим:
5 db 1,1,2,3
```

5.3. Директива "LOCAL"

Возможно, появится необходимость объявить метку внутри тела макроса:

Assembler

[Выделить код](#)

```
1 macro   pushstr string
2 {
3     call    behind ;помещаем в стек адрес string и переходим к behind
4     db      string, 0
5 behind:
6 }
```

но если использовать такой макрос 2 раза, то и метка **behind** будет объявлена дважды, что приведёт к ошибке. Эта проблема решается объявлением локальной метки **behind**. Это и делает директива **LOCAL**. Синтаксис:

Assembler

[Выделить код](#)

```
1 local    label_name
```

Директива должна применяться внутри тела макроса. Все метки **label_name** внутри макроса становятся локальными. Так что, если макрос используется дважды никаких проблем не появляется:

Assembler

[Выделить код](#)

```
1 macro   pushstr string
2 {
3     local behind
4     call    behind
5     db      string,0
6 behind:
7 }
8
9 pushstr 'aaaaa'
10 pushstr 'bbbbbbbb'
11 call    something
```

На самом деле, **behind** заменяется на **behind?XXXXXXXX**, где **XXXXXXXX** - какой-то шестнадцатеричный номер генерируемый препроцессором. Последний пример может быть преобразован к чему-то вроде:

Assembler

[Выделить код](#)

Assembler

[Выделить код](#)

```

1      call    behind?00000001
2      db      'aaaaa', 0
3  behind?00000001:
4      call    behind?00000002
5      db      'bbbbbbbb', 0
6  behind?00000002:
7      call    something

```

Заметьте, Вы не сможете напрямую обратиться к метке содержащей **?**, так как это специальный символ в FASM, поэтому он и используется в локальных метках. К примеру, **aa?bb** рассматривается как идентификатор **aa**, специальный символ **?** и идентификатор **bb**.

Если Вам нужно несколько локальных меток - не проблема, их можно указать в одной директиве LOCAL, разделив запятыми:

Assembler

[Выделить код](#)

```

1  macro  pushstr string ;делает то же, что и предыдущий макрос
2  {
3      local addr, behind
4      push  addr
5      jmp  behind
6  addr  db  string,0
7  behind:
8  }

```

Всегда хорошо бы начинать все локальные метки макросов с двух точек **..** - это значит, что они не будут менять текущую глобальную метку. К примеру:

Assembler

[Выделить код](#)

```

1  macro  pushstr string
2  {
3      local behind
4      call  behind
5      db  string, 0
6  behind:
7  }
8
9  MyProc:
10     pushstr 'aaaa'
11     .a:

```

будет преобразовано в:

Assembler

[Выделить код](#)

```

1  MyProc:
2      call    behind?00000001
3      db      'aaaa', 0
4  behind?00000001:
5  .a:

```

в результате получим метку **behind?00000001.a** вместо **MyProc.a**. Но если в примере выше **behind** заменить на **..behind**, текущая глобальная метка не изменится и будет определена метка **MyProc.a**:

Assembler

[Выделить код](#)

```

1  macro  pushstr string
2  {
3      local ..behind
4      call  ..behind
5      db  string,0
6  ..behind:
7  }
8
9  MyProc:
10     pushstr 'aaaa'
11     .a:

```

Вернуться к обсуждению:

[Руководство по препроцессору FASM](#)

[Следующий ответ](#)

2



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

Заказываю контрольные, курсовые, дипломные и любые другие студенческие работы [здесь](#).

[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