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
13958 / 6977 / **806**
Регистрация: 11.11.2010
Сообщений: 12,550

09.09.2014, 12:57 **[ТС]** **0**

Руководство по препроцессору FASM

09.09.2014, 12:57. **Просмотров** 10410. **Ответов** 7
Метки (Все метки)

Ответ

5.4. Оператор объединения

У макроязыка FASMa есть ещё одна возможность - манипуляции с идентификаторами. Делается это оператором **#**, который объединяет два идентификатора в один. К примеру, **a#b** становится **ab**, а **aaa bbb#ccc ddd** -- **aaa bbbccc ddd**.

Оператор **#** может быть использован только внутри тел макросов, а объединение символов происходит после замены аргументов макроса параметрами. Так что его можно использовать для создания новых идентификаторов из переданных в макрос параметров:

Assembler Выделить код

```
1   macro   string name, data
2   {
3     local ..start
4   ..start:
5   name db data,0
6   sizeof.#name =  $ - ..start
7   }
8
9   string  s1,'нудные макросы'
10  string  s2,<'а вот и я',13,10,'заставлю тебя их видеть во сне'>
```

получим:

Assembler Выделить код

```
1  ..start?00000001:
2  s1  db 'нудные макросы',0
3  sizeof.s1 = $ - ..start?00000001
4
5  ..start?00000002:
6  s2  db 'а вот и я',13,10,'заставлю тебя их видеть во сне',0
7  sizeof.s2 = $ - ..start?00000002
```

так что для всех строк, создаваемых этим макросом будет определён идентификатор **sizeof.имя строки**, равный количеству байт строки.
Оператор **#** способен так же объединять символьные строки:

Assembler Выделить код

Assembler Выделить код

```
1  macro   debug name
2  {
3      db 'name: '#b,0
4  }
5  debug   '1'
6  debug   'foobar'
```

будет:

Assembler Выделить код

```
1  db  'name: 1',0
2  db  'name: foobar',0
```

Это полезно при передаче аргументов из макроса в макрос:

Assembler Выделить код

```
1   macro   pushstring string
2   {
3     local ..behind
4       call    ..behind
5       db  string,0
6   ..behind:
7   }
8   macro   debug string
9   {
10      push    MB_OK
11      push 0 ;empty caption
12      pushstring  'debug: '#string    ;принимает один аргумент
13      push    0               ;нет окна-предка
14      call    [MessageBox]
15  }
```

Обратите внимание, нельзя использовать **#** совместно с идентификаторами, определёнными **local**, так как **local** обрабатывается препроцессором раньше, чем **#**. Из-за этого подобный код работать не будет:

Assembler Выделить код

```
1  macro   a arg
2  {
3    local name_#arg
4  }
5  a   foo
```

5.5. Оператор "`"

Существует оператор, преобразующий идентификатор в символьную строку. Он так же может быть использован только внутри макросов:

Assembler Выделить код

```
1  macro   proc name
2  {
3     name:
4     log  `name   ;log - макрос, принимающий параметр-строку
5  }
6  proc    DummyProc
```

получим:

Assembler Выделить код

```
1  DummyProc:
2      log 'DummyProc'
```

Пример посложнее, с использованием "**#**"

Assembler Выделить код

```
1  macro   proc name
2  {
3       name:
4       log 'начинается подпрограмма: '#`name
5  }
6  proc    DummyProc
7  retn
8  proc    Proc2
9  retn
```

будет:

Assembler Выделить код

Assembler Выделить код

```
1  DummyProc:
2  log 'начинается подпрограмма: DummyProc'
3  retn
4  Proc2:
5  log 'начинается подпрограмма: Proc2'
6  retn
```

6. Макросы с групповыми аргументами

6.1. Определение макросов с групповым аргументом

У макросов могут быть так называемые групповые аргументы. Это позволяет использовать переменное количество аргументов. При определении макроса, групповой аргумент заключается в квадратные скобочки "**[**" и "**]**":

Синтаксис:

Assembler Выделить код

```
1  macro   name arg1, arg2, [grouparg]
2  {
3   <тело макроса>
4  }
```

Среди аргументов в определении макроса, групповой аргумент должен быть последним. Групповой аргумент может содержать несколько значений:

Assembler Выделить код

```
1  macro   name arg1,arg2,[grouparg] {}
2  name    1,2,3,4,5,6
```

В этом примере значение **arg1** равно 1, **arg2** равно 2, а **grouparg** равно 3,4,5 и 6

6.2. Директива "*COMMON*"

Для работы с групповыми аргументами применяются специальные директивы препроцессора. Они могут быть использованы только внутри тела макроса имеющего групповой аргумент. Первая такая директива - это "**COMMON**". Она означает, что после нее имя группового аргумента будет замещаться всеми аргументами сразу:

Assembler Выделить код

```
1  macro   string [grp]
2  {
3    common
4    db    grp,0
5  }
6
7  string  'aaaaaa'
8  string  'line1',13,10,'line2'
9  string  1,2,3,4,5
```

получим:

Assembler Выделить код

```
1  db  'aaaaaa',0
2  db  'line1',13,10,'line2',0
3  db  1,2,3,4,5,0
```

6.3. Директива "*FORWARD*"

Аргументы можно обрабатывать и по-отдельности. Для этого служит директива "**FORWARD**". Часть тела макроса после этой директивы обрабатывается препроцессором для каждого аргумента из группы:

Assembler Выделить код

```
1  macro   a arg1,[grparg]
2  {
3  forward
4      db  arg1
5      db  grparg
6  }
7
8  a   1,'a','b','c'
9  a   -1, 10, 20
```

будет:

Assembler Выделить код

Assembler Выделить код

```
1   db  1
2   db  'a'
3   db  1
4   db  'b'
5   db  1
6   db  'c'
7
8   db  -1
9   db  10
10  db  -1
11  db  20
```

Директива "**FORWARD**" работает по умолчанию для макросов с групповыми аргументами, так что предыдущий пример можно сделать так:

Assembler Выделить код

```
1  macro   a arg1,[grparg]
2  {
3      db  arg1
4      db  grparg
5  }
```

6.4. Директива "*REVERSE*"

"**REVERSE**" - это аналог "**FORWARD**", но обрабатывает группу аргументов в обратном порядке - от последнего к первому:

Assembler Выделить код

```
1  macro   a arg1,[grparg]
2  {
3  reverse
4      db  arg1
5      db  grparg
6  }
7
8  a   1,'a','b','c'
```

получим:

Assembler Выделить код

```
1  db  1
2  db  'c'
3  db  1
4  db  'b'
5  db  1
6  db  'a'
```

6.5. Комбинирование директив управления группами

Три вышеупомянутые директивы могут разделять тело макроса на блоки. Каждый блок обработается препроцессором после предыдущего. Например:

Assembler Выделить код

```
1   macro   a [grparg]
2   {
3     forward
4       f_#grparg:  ;оператор объединения
5     common
6       db  grparg
7     reverse
8       r_#grparg:
9   }
10
11  a   1,2,3,4
```

будет:

Assembler Выделить код

```
1  f_1:
2  f_2:
3  f_3:
4  f_4:
5  db  1,2,3,4
6  r_4:
7  r_3:
8  r_2:
9  r_1:
```

6.6. Директива *LOCAL* в макросах с групповыми аргументами

У локальных меток в макросах есть ещё одно полезное свойство. Если директива "**LOCAL**" находится внутри блока "**FORWARD**" или "**REVERSE**", то уникальное имя метки сгенерируется для каждого аргумента из группы, и в последующих блоках "**FORWARD**" и/или "**REVERSE**" для каждого аргумента будет использована соответствующая ему метка:

Assembler Выделить код

```
macro   string_table [string]
{
  forward       ;таблица указателей на строки
    local   addr    ;локальная метка для строки
    dd  addr        ;указатель на строку
  forward       ;строки
    addr    db string,0 ;создаём и завершаем нулём
}
string_table    'aaaaa','bbbbbb','5'
```

получим:

Assembler Выделить код

```
dd  addr?00000001
dd  addr?00000002
dd  addr?00000003
addr?00000001   db 'aaaaa',0
addr?00000002   db 'bbbbbb',0
addr?00000003   db '5',0
```

Другой пример с блоком "**REVERSE**":

Assembler Выделить код

```
macro   a [x]
{
forward
  local here
  here  db x
reverse
  dd    here
}
a   1,2,3
```

будет:

Assembler Выделить код

```
here?00000001   db 1
here?00000002   db 2
here?00000003   db 3
dd  here?00000003
dd  here?00000002
dd  here?00000001
```

Как видно, метки используется с соответствующими аргументами и в "**FORWARD**"- и в "**REVERSE**"-блоках.

6.7. Макросы с несколькими групповыми аргументами

Возможно использовать и несколько групповых аргументов. В этом случае определение макроса не будет выглядеть как:

Assembler Выделить код

```
macro   a [grp1],[grp2]
```

так как тут не ясно какой аргумент какой группе принадлежит. Исходя из этого делают так:

Assembler Выделить код

```
macro   a [grp1,grp2]
```

В этом случае каждый нечётный аргумент относится к группе **grp1**, а каждый чётный - к **grp2**:

Assembler Выделить код

```
macro   a [grp1,grp2]
{
  forward
    l_#grp1:
  forward
    l_#grp2:
}

a   1,2,3,4,5,6
```

будет:

Assembler Выделить код

```
1  l_1:
2  l_3:
3  l_5:
4  l_2:
5  l_4:
6  l_6:
```

Или ещё:

Assembler Выделить код

```
1  macro   ErrorList [name,value]
2  {
3    forward
4      ERROR_#name = value
5  }
6  ErrorList \
7      NONE,0,\
8      OUTOFMEMORY,10,\
9      INTERNAL,20
```

получим:

Assembler Выделить код

```
1  ERROR_NONE = 0
2  ERROR_OUTOFMEMORY = 10
3  ERROR_INTERNAL = 20
```

Конечно же, может быть больше двух групп аргументов:

Assembler Выделить код

```
1  macro   a [g1,g2,g3]
2  {
3    common
4      db  g1
5      db  g2
6      db  g3
7  }
8  a   1,2,3,4,5,6,7,8,9,10,11
```

будет:

Assembler Выделить код

```
1  db  1,4,7,10
2  db  2,5,8,11
3  db  3,6,9
```

7. Условный препроцессинг

В действительности, FASM не имеет директив для условного препроцессинга. Но директива ассемблера "**if**" может быть использована совместно с возможностями препроцессора для получения тех же результатов, что и при условном препроцессинге. (Но в этом случае увеличивается расход памяти и времени).
Как известно, оператор "**if**" обрабатывается во время ассемблирования. Это значит, что условие в этом операторе проверяется после обработки исходного текста препроцессором. Именно это обеспечивает работу некоторых логических операций.

Я не буду рассказывать о деталях времени ассемблирования (логических операциях вроде "**&**", "**|**" и тому подобном) - читайте об этом в документации FASM. Я лишь расскажу об операторах проверки условия используемых препроцессором.

Вернуться к обсуждению:
Руководство по препроцессору FASM

Следующий ответ

2

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

09.09.2014, 12:57

Готовые ответы и решения:

Вызываю dll (написанную на vc++2008) из Fasm. Через 40 секунд вылет из программы.Без вызова dll из Fasm программа не вылетает.
Программа на vc++2008: #include "MathFuncsDll.h" #include <stdexcept> using namespace std; ...

Вопрос по препроцессору C
Хотел спросить по поводу вычислений на этапе компиляции. Допустим, есть вот такой код #defyme...

Директива препроцессору #pragma
Что это за директива такая? Для чего предназначена? Если не затруднит, можно привести примеры?

Требуется директива препроцессору
у меня проблема такого плана (опишу все действия сначала, т.к. не уверен в их правильности):...

7