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

7.1. Оператор "*EQ*"

Простейший логический оператор - это "**EQ**". Он всего лишь сравнивает два идентификатора - одинаковы ли их значение. Значение **abcd eq abcd** - *истина*, а **abcd eq 1** - *ложь* и так далее... Это полезно для сравнения символов, которые будут обработаны препроцессором:

Assembler Выделить код

```
1  STRINGS equ ASCII
2  if STRINGS eq ASCII
3       db  'Oh yeah',0
4  else if STRINGS eq UNICODE
5       du  'Oh yeah',0
6  else
7       display 'unknown string type'
8  end if
```

после обработки препроцессором, это примет вид:

Assembler Выделить код

```
1  if ASCII eq ASCII
2       db  'Oh yeah',0
3  else if ASCII eq UNICODE
4       du  'Oh yeah',0
5  else
6       display 'unknown string type'
7  end if
```

Здесь только первое условие (**ASCII eq ASCII**) выполняется, так что будет ассемблировано только

Assembler Выделить код

```
1  db 'Oh yeah',0
```

Другой вариант:

Assembler Выделить код

Assembler Выделить код

```
1  STRINGS equ UNICODE   ;разница здесь, UNICODE вместо ASCII
2  if STRINGS eq ASCII
3      db  'Oh yeah',0
4  else if STRINGS eq UNICODE
5      du  'Oh yeah',0
6  else
7      display 'unknown string type'
8  end if
```

получим:

Assembler Выделить код

```
1  if UNICODE eq ASCII
2      db  'Oh yeah',0
3  else if UNICODE eq UNICODE
4      du  'Oh yeah',0
5  else
6      display 'unknown string type'
7  end if
```

Тут уже первое условие (**UNICODE eq ASCII**) будет ложно, второе (**UNICODE eq UNICODE**) - верно, будет ассемблироваться

Assembler Выделить код

```
1  du 'Oh yeah',0
```

Несколько лучшее применение этого - проверка аргументов макросов, вроде:

Assembler Выделить код

```
1   macro   item type,value
2   {
3      if type eq BYTE
4       db  value
5      else if type eq WORD
6       dw  value
7      else if type eq DWORD
8       dd  value
9      else if type eq STRING
10      db  value,0
11     end if
12  }
13  item    BYTE,1
14  item    STRING,'aaaaaa'
```

будет:

Assembler Выделить код

```
1   if BYTE eq BYTE
2       db  1
3   else if BYTE eq WORD
4       dw  1
5   else if BYTE eq DWORD
6       dd  1
7   else if BYTE eq STRING
8       db  1,0
9   end if
10  if STRING eq BYTE
11      db  'aaaaaa'
12  else if STRING eq WORD
13      dw  'aaaaaa'
14  else if STRING eq DWORD
15      dd  'aaaaaa'
16  else if STRING eq STRING
17      db  'aaaaaa',0
18  end if
```

ассемблироваться будут только две команды:

Assembler Выделить код

```
1  db  1
2  db  'aaaaaa',0
```

Подобно всем другим операторам препроцессора, "**EQ**" может работать с пустыми аргументами. Это значит, что, например, "**if eq**" верно, а **if 5 eq** - *ложно* и т.п.

Пример макроса:

Assembler Выделить код

```
1  macro   mov dest,src,src2
2  {
3    if src2 eq
4      mov dest, src
5    else
6      mov dest, src
7      mov src, src2
8    end if
9  }
```

здесь, если есть третий аргумент, то будут ассемблироваться две последних команды, если нет - то только первая.

7.2. Оператор "*EQTYPE*"

Ещё один оператор - "**EQTYPE**". Он определяет, одинаков ли тип идентификаторов.
Существующие типы:

- отдельные строки символов, заключённые в кавычки (те, которые не являются частью численных выражений)
- вещественные числа
- любые численные выражения, например, **2+2** (любой неизвестный символ будет рассматриваться как метка, так что он будет считаться подобным выражением)
- адреса - численные выражения в квадратных скобках (учитывая оператор размерности и префикс сегмента)
- мнемоники инструкций
- регистры
- операторы размерности
- операторы **NEAR** и **FAR**
- операторы **USE16** и **USE32**
- пустые аргументы (пробелы, символы табуляции)

Пример макроса, который позволяет использовать переменную в памяти в качестве счётчика в инструкции **SHL** (например **shl ax, [myvar]**):

Assembler Выделить код

```
1   macro   shl dest, count
2   {
3     if count eqtype [0]       ;если count - ячейка памяти
4       push    cx
5       mov cl, count
6       shl dest, cl
7       pop cx
8     else              ;если count другого типа
9       shl dest, count ;просто используем обычную shl
10    end if
11  }
12  shl ax, 5
13  byte_variable   db 5
14  shl ax, [byte_variable]
```

получится:

Assembler Выделить код

```
1   if 5 eqtype [0]
2       push    cx
3       mov cl, 5
4       shl ax, cl
5       pop cx
6   else
7       shl ax, 5
8   end if
9   byte_variable   db 5
10
11  if [byte_variable] eqtype [0]
12      push    cx
13      mov cl, [byte_variable]
14      shl ax, cl
15      pop cx
16  else
17      shl ax, [byte_variable]
18  end if
```

в результате обработки условий конечный результат будет:

Assembler Выделить код

```
1      shl ax, 5
2  byte_variable   db 5
3      push    cx
4      mov cl, [byte variable]
5      shl ax, cl
6      pop cx
```

Заметьте, что **shl ax, byte [myvar]** не будет работать с этим макросом, так как условие **byte [variable] eqtype [0]** не выполняется. Читаем дальше.
Когда мы сравниваем что-то посредством "**EQTYPE**", то это что-то может быть не только единичным идентификатором, но и их комбинацией. В таком случае, результат **eqtype** *истина*, если не только типы, но и порядок идентификаторов совпадают. К примеру, **if eax 4 eqtype ebx name** - верно, так как **name** - это метка, и её тип - численное выражение.

Пример расширенной инструкции **mov**, которая позволяет перемещать данные между двумя ячейками памяти:

Assembler Выделить код

```
1   macro   mov dest,src
2   {
3     if dest src eqtype [0] [0]
4       push    src
5       pop dest
6     else
7       mov dest,src
8     end if
9   }
10
11  mov [var1], 5
12  mov [var1], [var2]
```

преобразуется препроцессором в:

Assembler Выделить код

```
1   if [var1] 5 eqtype [0] [0]  ;не верно
2       push    5
3       pop [var1]
4   else
5       mov [var1],5
6   end if
7
8   if [var1] [var2] eqtype [0] [0]  ;верно
9       push    [var2]
10      pop [var1]
11  else
12      mov [var1], [var2]
13  end if
```

и будет ассемблировано в:

Assembler Выделить код

```
1       mov [var1], 5
2       push    [var2]
3       pop [var1]
```

Хотя более удобно для восприятия реализовать макрос используя **логический оператор *И*** - **&**:

Assembler Выделить код

```
1  macro   mov dest,src
2  {
3    if (dest eqtype [0]) & (src eqtype [0])
4      push    src
5      pop dest
6    else
7      mov dest, src
8    end if
9  }
```

Пример с использованием "**EQTYPE**" с четырьмя аргументами приведён для демонстрации возможностей, обычно проще использовать в таких случаях "**&**". Кстати, в качестве аргументов, возможно использовать некорректные выражения - достаточно, чтобы лексический анализатор распознал их тип. Но это не является документированным, так что не будем этот обсуждать.

7.3. Оператор "*IN*"

Бывают случаи, когда в условии присутствует слишком много "**EQ**":

Assembler Выделить код

```
1   macro   mov a,b
2   {
3     if (a eq cs) | (a eq ds) | (a eq es) | (a eq fs) | \
4        (a eq gs) | (a eq ss)
5       push    b
6       pop a
7     else
8       mov a, b
9     end if
10  }
```

Вместо применения множества **логических операторов *ИЛИ*** - **|**, можно использовать специальный оператор "**IN**". Он проверяет, присутствует ли идентификатор слева, в списке идентификаторов справа. Список должен быть заключён в скобочки "**<**" и "**>**", а идентификаторы в нём разделяются запятыми:

Assembler Выделить код

```
1  macro   mov a,b
2  {
3    if a in <cs,ds,es,fs,gs,ss>
4      push    b
5      pop a
6    else
7      mov a, b
8    end if
9  }
```

Это так же работает для нескольких идентификаторов (как и "**EQ**"):

Assembler Выделить код

```
1  if dword [eax] in <[eax], dword [eax], ptr eax, dword ptr eax>
```

8. Структуры

В FASM, структуры практически тоже самое, что и макросы. Определяются они посредством директивы **STRUC**: Синтаксис:

Assembler Выделить код

```
1  struc   <name> <arguments>  { <тело структуры> }
```

Отличие от макросов заключается в том, что в исходном тексте перед структурой должна находиться некое "**имя**" - *имя объекта-структуры*. Например:

Assembler Выделить код

```
1  struc   a {db 5}
2  a
```

это не будет работать. Структуры распознаются только после имен, как здесь:

Assembler Выделить код

```
1  struc   a {db 5}
2  name    a
```

подобно макросу, это преобразуется препроцессором в:

Assembler Выделить код

```
1  db  5
```

Смысл имени в следующем - оно будет добавлена ко всем идентификаторам из тела структуры, которые начинаются с точки. Например:

Assembler Выделить код

```
1  struc   a {.local:}
2  name1   a
3  name2   a
```

будет:

Assembler Выделить код

```
1  name1.local:
2  name2.local:
```

Таким образом можно создавать структуры вроде тех, что есть в языках высокого уровня абстракции:

Assembler Выделить код

```
1  struc   rect left,right,top,bottom  ;аргументы как у макроса
2  {
3      .left   dd left
4      .right  dd right
5      .top    dd top
6      .bottom dd bottom
7  }
8  r1 rect 0,20,10,30
9  r2 rect ?,?,?,?
```

получим:

Assembler Выделить код

Assembler Выделить код

```
1  r1.left     dd 0
2  r1.right    dd 20
3  r1.top      dd 10
4  r1.bottom   dd 30
5  r2.left     dd ?
6  r2.right    dd ?
7  r2.top      dd ?
8  r2.bottom   dd ?
```

Поскольку, используемой структуре всегда должно предшествовать имя, препроцессор однозначно отличает их от макросов. Поэтому имя структуры может совпадать с именем макроса - в каждом случае будет выполняться нужная обработка.

Существуют хитрый приём, позволяющий не указывать аргументы, если они равны 0:

Assembler Выделить код

```
1  struc   ymmv arg
2  {
3      .member dd arg+0
4  }
5  y1  ymmv 0xACDC
6  y2  ymmv
```

будет:

Assembler Выделить код

```
1  y1.member   dd 0xACDC+0
2  y2.member   dd +0
```

Как говорилось ранее, если значение аргумента не указанно, то в теле макроса или структуры вместо него ничего не подставляется. В этом примере "плюс" ("**+**") используется или как бинарный оператор (то есть с двумя операндами), или как унарный (с одним операндом) оператор.
ПРИМЕЧАНИЕ: часто используется так же макрос или структура **struct**, которая определяется для расширения возможностей при определении структур. Не путайте **struct** и **struc**.

Вернуться к обсуждению:
Руководство по препроцессору FASM

Следующий ответ

3

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

10.09.2014, 04:59

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

Powered by vBulletin® Version 3.8.9
Copyright ©2000 - 2020, vBulletin Solutions, Inc.