


Учебный курс. Часть 27. Синтаксис объявления меток

Автор: xrnd | Рубрика: [Учебный курс](#) | 03-11-2010 |
 [Распечатать запись](#)

Не сомневаюсь, что объявлять метки вы уже научились 😊
Однако, синтаксис FASM не ограничивается объявлением простых меток. В этой части мы рассмотрим дополнительную директиву для создания меток, а также научимся использовать локальные и анонимные метки.

В синтаксисе FASM существует 3 основных способа объявления меток:

1. Имя метки, после которого ставится двоеточие. Это самый простой способ. Обычно так объявляются метки в коде. (Подробнее об этом способе читайте в [части 13 учебного курса](#))

```
exit_app:  
    mov ax,4C00h  
    int 21h
```

2. Использование директив объявления данных. Имя переменной является по сути той же меткой. Отличие от первого способа в том, что дополнительно с именем метки связывается размер переменной. (Подробнее читайте в [части 5 учебного курса](#))

```
x db 5  
y dw 34,1200,?  
z rd 1
```

3. Объявление метки с помощью специальной директивы *label*. Более сложный, но зато самый гибкий способ. Его мы рассмотрим подробнее.

Директива *label* имеет следующий формат:

```
label <имя_метки> [размер] [at адрес]
```

У директивы может быть 3 параметра. Обязательным является только первый параметр — имя метки. Вторым параметром — оператор размера (*byte*, *word*, *dword* и т.д.). Он связывает с меткой размер переменной, аналогично тому, как это делают директивы объявления данных. Далее может быть указан оператор *at* и адрес метки. Адрес может представлять собой константу, числовое выражение или имя другой метки. Если адрес не указан, то для создания метки используется адрес того места, где она объявлена.

```
label m1           ;То же самое, что 'm1:'  
label m2 byte      ;Похоже на 'm2 db ?', но память не резервирует  
label m3 dword     ;Похоже на 'm3 dd ?', но память не резервирует  
                  ;Все 3 метки указывают на один и тот же адрес
```

Следующий пример показывает, как можно использовать в программе директиву *label*. Допустим, объявлена переменная *x* размером слово. Требуется обнулить старший байт *x*. Воспользовавшись директивой *label*, можно обратиться к старшему байту как к отдельной переменной:

```
x    dw 12345      ;Переменная-слово  
label xh byte at x+1 ;Объявление метки для обращения к старшему  
...  
start:  
    xor al,al      ;AL=0  
    mov [xh],al    ;xh=0 (старший байт x)  
    mov byte[x+1],al ;То же самое без использования метки xh
```

Кроме того, адрес может содержать базовые и индексные регистры для косвенной адресации. Такие метки можно использовать для обращения к параметрам и локальным переменным процедуры. Например:

```
label i word at bp-2 ;Локальная переменная
...
inc [i] ;Инкремент локальной переменной
```

Локальные метки

Локальная метка — это метка, имя которой начинается с точки. Во время генерации кода FASM автоматически добавляет к имени локальной метки имя последней объявленной «глобальной» метки. Таким образом, имена локальных меток могут повторяться, если между ними есть хотя бы одна «глобальная» метка.

Локальные метки удобно использовать, например, внутри процедуры. Можно дать им простые, понятные имена и не беспокоиться, что где-то в коде уже объявлена метка с таким именем. В качестве примера я добавил локальные метки в процедуру преобразования строки в число из [части 23 учебного курса](#):

```
;Процедура преобразования десятичной строки в слово без знака
; вход: AL - длина строки
;       DX - адрес строки, заканчивающейся символом CR(0Dh)
; выход: AX - слово (в случае ошибки AX = 0)
;       CF = 1 - ошибка
str_to_udec_word:
    push cx ;Сохранение всех используемых регистр
    push dx
    push bx
    push si
    push di
```

| | |
|--------------------------|---------------------------------------------------|
| <code>mov si,dx</code> | <i>;SI = адрес строки</i> |
| <code>mov di,10</code> | <i>;DI = множитель 10 (основание системы)</i> |
| <code>movzx cx,al</code> | <i>;CX = счётчик цикла = длина строки</i> |
| <code>jcxz .error</code> | <i>;Если длина = 0, возвращаем ошибку</i> |
| <code>xor ax,ax</code> | <i>;AX = 0</i> |
| <code>xor bx,bx</code> | <i>;BX = 0</i> |
| | |
| <code>.lp:</code> | |
| <code>mov bl,[si]</code> | <i>;Загрузка в BL очередного символа строки</i> |
| <code>inc si</code> | <i>;Инкремент адреса</i> |
| <code>cmp bl,'0'</code> | <i>;Если код символа меньше кода '0'</i> |
| <code>jnl .error</code> | <i>;возвращаем ошибку</i> |
| <code>cmp bl,'9'</code> | <i>;Если код символа больше кода '9'</i> |
| <code>jng .error</code> | <i>;возвращаем ошибку</i> |
| <code>sub bl,'0'</code> | <i>;Преобразование символа-цифры в число</i> |
| <code>mul di</code> | <i>;AX = AX * 10</i> |
| <code>jc .error</code> | <i>;Если результат больше 16 бит - ошибка</i> |
| <code>add ax,bx</code> | <i>;Прибавляем цифру</i> |
| <code>jc .error</code> | <i>;Если переполнение - ошибка</i> |
| <code>loop .lp</code> | <i>;Команда цикла</i> |
| | |
| <code>jmp .exit</code> | <i>;Успешное завершение (здесь всегда CF = 0)</i> |
| | |
| <code>.error:</code> | |
| <code>xor ax,ax</code> | <i>;AX = 0</i> |
| <code>stc</code> | <i>;CF = 1 (Возвращаем ошибку)</i> |
| | |
| <code>.exit:</code> | |
| <code>pop di</code> | <i>;Восстановление регистров</i> |
| <code>pop si</code> | |
| <code>pop bx</code> | |
| <code>pop dx</code> | |
| <code>pop cx</code> | |
| <code>ret</code> | |

Локальные метки намного улучшают читаемость кода. Если потребуется обратиться к локальной метке из другого места программы, это можно сделать, указав её полное имя:

```
jmp str_to_udec_word.error ;Переход к локальной метке
```

Особым образом обрабатываются метки, имя которых начинается с двух точек. Такие метки ведут себя как глобальные, но не становятся префиксом для локальных меток.

Анонимные метки

Анонимная метка — это метка с именем `@@`. В программе можно объявлять сколько угодно анонимных меток, но обратиться получится только к ближайшей. Для этого существуют специальные имена: вместо `@b` (или `@r`) FASM подставляет адрес предыдущей анонимной метки, а вместо `@f` — адрес следующей анонимной метки. Этого, как правило, достаточно, чтобы реализовать простой цикл, переход или проверку условия. Таким образом можно избавиться от большого количества «неанонимных» меток. Вот пример той же процедуры с использованием анонимных меток:

```
;Процедура преобразования десятичной строки в слово без знака
;  вход: AL - длина строки
;         DX - адрес строки, заканчивающейся символом CR(0Dh)
;  выход: AX - слово (в случае ошибки AX = 0)
;         CF = 1 - ошибка
str_to_udec_word:
    push cx                                ;Сохранение всех используемых регистр

    push dx
    push bx
    push si
    push di

    mov si, dx                            ;SI = адрес строки
    mov di, 10                            ;DI = множитель 10 (основание системы
    movzx cx, al                          ;CX = счётчик цикла = длина строки
    jcxz .error                            ;Если длина = 0, возвращаем ошибку
    xor ax, ax                             ;AX = 0
    xor bx, bx                             ;BX = 0

    @@: mov bl [si]                        ;Загрузка в BL очередного символа стр
```

| | |
|---------------------------|-------------------------------------------------|
| <code>mov bl, [si]</code> | <i>;Загрузка в BL очередного символа строки</i> |
| <code>inc si</code> | <i>;Инкремент адреса</i> |
| <code>cmp bl, '0'</code> | <i>;Если код символа меньше кода '0'</i> |
| <code>jnl .error</code> | <i>;возвращаем ошибку</i> |
| <code>cmp bl, '9'</code> | <i>;Если код символа больше кода '9'</i> |
| <code>jng .error</code> | <i>;возвращаем ошибку</i> |
| <code>sub bl, '0'</code> | <i>;Преобразование символа-цифры в число</i> |
| <code>mul di</code> | <i>;AX = AX * 10</i> |
| <code>jc .error</code> | <i>;Если результат больше 16 бит - ошибка</i> |
| <code>add ax, bx</code> | <i>;Прибавляем цифру</i> |
| <code>jc .error</code> | <i>;Если переполнение - ошибка</i> |
| <code>loop @b</code> | <i>;Команда цикла</i> |
| <code>jmp @f</code> | <i>;Успешное завершение (здесь всегда CF=0)</i> |
| | |
| <code>.error:</code> | |
| <code> xor ax, ax</code> | <i>;AX = 0</i> |
| <code> stc</code> | <i>;CF = 1 (Возвращаем ошибку)</i> |
| | |
| <code>@@: pop di</code> | <i>;Восстановление регистров</i> |
| <code> pop si</code> | |
| <code> pop bx</code> | |
| <code> pop dx</code> | |
| <code> pop cx</code> | |
| <code> ret</code> | |

Я вам советую локальные и анонимные метки использовать везде, где только возможно. Они делают код программы более понятным и не захламляют пространство имён.

Упражнение

В этот раз совсем простое упражнение. В приведённом коде процедуры замените «глобальные» метки на локальные. После этого замените локальные метки на анонимные, где это возможно.

```

;Процедура преобразования байта в строку в двоичном виде
; AL - байт.
; DI - буфер для строки (8 символов). Значение регистра не сохран
byte_to_bin_str:

```

| | |
|-------------------------------|---------------------------------------------|
| <code>push cx</code> | <i>;Сохранение CX</i> |
| <code>mov cx,8</code> | <i>;Счётчик цикла</i> |
| | |
| <code>btbs_lp:</code> | |
| <code>rol al,1</code> | <i>;Циклический сдвиг AL влево на 1 бит</i> |
| <code>jc btbs_1</code> | <i>;Если выдвинутый бит = 1, то переход</i> |
| <code>mov byte[di],'0'</code> | <i>;Добавление символа '0' в строку</i> |
| <code>jmp btbs_end</code> | |
| <code>btbs_1:</code> | |
| <code>mov byte[di],'1'</code> | <i>;Добавление символа '1' в строку</i> |
| <code>btbs_end:</code> | |
| <code>inc di</code> | <i>;Инкремент DI</i> |
| <code>loop btbs_lp</code> | <i>;Команда цикла</i> |
| | |
| <code>pop cx</code> | <i>;Восстановление CX</i> |
| <code>ret</code> | <i>;Возврат из процедуры</i> |

Результаты можете писать в комментариях или на форуме.

[Следующая часть »](#)

Комментарии:

IgorKing

03-11-2010 08:36

Ну, локальные, наверное, совсем просто:

`byte_to_bin_str:`

`push cx`

`mov cx,8`

`.btbs_lp:`

`rol al,1`

`jc btbs_1`

`mov byte[di],'0'`

```
jmp .btbs_end  
.btbs_1:  
mov byte[di], '1'  
.btbs_end:  
inc di  
loop .btbs_lp
```

```
pop cx  
ret
```

А вот анонимные что-то даже не знаю где можно их вставить...может так:

```
byte_to_bin_str:  
push cx  
mov cx, 8
```

```
@@:  
rol al, 1  
jc btbs_1  
mov byte[di], '0'  
jmp .btbs_end  
.btbs_1:  
mov byte[di], '1'  
.btbs_end:  
inc di  
loop @b
```

```
pop cx  
ret
```

[\[Ответить\]](#)

[xrnd](#)

03-11-2010 20:32

Ага. Все верно. Анонимную метку тут можно только одну использовать.

Можно было ещё префикс убрать у локальных меток: btbs_ — это сокращение от имени процедуры.

[\[Ответить\]](#)

Alex

03-12-2010 16:27

Хороший у вас сайт. Интересно читать!

[\[Ответить\]](#)

[xrnd](#)

03-12-2010 19:18

Спасибо, я стараюсь 😊

[\[Ответить\]](#)

plan4ik

07-04-2011 14:30

фасм это сила 😊 равных этому сайту пока нет !!!!

[\[Ответить\]](#)

Гость

20-02-2011 23:21

Немного изменил код , а то туда не вставить больше 1 анонимной метки ,)

Вот так можно ?

byte_to_bin_str:

push cx

mov cx,9 ; 8+1

@@:

JCXZ @f

```
dec cx
mov byte[di], '1'
rol al, 1
inc di
jc @b
mov byte[di-1], '0'
jmp @b
@@:
pop cx
ret
```

[\[Ответить\]](#)

[xrnd](#)

22-02-2011 20:20

Я подредактировал и собрал всё в один комментарий.

Всё хорошо, но цикл выполнится 9 раз, а не 8.

[\[Ответить\]](#)

Гость

22-02-2011 21:20

Ясна ,)

Спасибо , у меня dec cx было перед JCXZ @f.

Я всё думал как бы можно было обойтись без +1 ,)

Оказалась всё проще чем я думал.

[\[Ответить\]](#)

plan4ik

07-04-2011 14:47

локальные метки это круто))))

```
byte_to_bin_str:  
push cx  
mov cx,8
```

```
.btbs_lp:  
rol al,1  
jc @f;btbs_1  
mov byte[di], '0'  
jmp btbs_end ; или здесь ??? @f  
@@:;.btbs_1:  
mov byte[di], '1'  
.btbs_end:  
inc di  
loop btbs_lp  
  
pop cx  
ret
```

[\[Ответить\]](#)

[xrnd](#)

08-04-2011 20:39

Правильно, но в командах JMP и LOOP метки тоже должны начинаться с точки:

```
jmp .btbs_end  
loop .btbs_lp
```

Префиксы в названии глобальных меток я делал для того, чтобы у них не совпали имена. Для локальных меток можно использовать простые понятные имена, например «.end»

[\[Ответить\]](#)

plan4ik

10-04-2011 03:34

да я даже не компилил по этому и не заметил когда менял и добавлял метки 😊

а чё с новыми статьями ты над ними работаешь ??? хотелось бы узнать побольше о структурах объединениях и всяких возможностях компилятора 😊 И большое спасибо за твои старания 😊

[\[Ответить\]](#)

[xrnd](#)

11-04-2011 15:15

В ближайшие дни выложу новую статью.

[\[Ответить\]](#)

алекс

02-07-2012 04:20

задание и вправду простое...
вот.. вариант 1

```
byte_to_bin_str:  
push cx  
mov cx,8
```

```
.btbs_lp:;Öèêèèè  
rol al,1  
jc .btbs_1  
mov byte[di], '0'  
jmp .btbs_end  
.btbs_1:  
mov byte[di], '1'  
.btbs_end:  
inc di  
loop .btbs_lp
```

```
pop cx  
ret
```

а вот и вариант 2

```
byte_to_bin_str:  
push cx  
mov cx,8
```

```
@@:  
rol al,1  
jc .btbs_1  
mov byte[di], '0'  
jmp .btbs_end  
.btbs_1:  
mov byte[di], '1'  
.btbs_end:  
inc di  
loop @b
```

```
pop cx  
ret
```

[\[Ответить\]](#)

[Андрей](#)

28-12-2015 12:29

>Я вам советую локальные и анонимные метки использовать везде, где только возможно. Они делают код программы более понятным и не захламляют пространство имён.

Позволю себе не согласиться. Вставка нового куска кода с локальными метками в уже существующий код между локальной меткой и переходом на неё не вызовет вопросов у компилятора. Однако в итоге будем иметь баг. Мне больше нравится концепция глобальная метка в начале какой то кодовой «главы» с метками «параграфами» по коду. Это

позволяет не напрягаться с дефицитом осмысленных имён. А локальных меток с переходом больше чем размер экрана я бы вообще избегал.

[\[Ответить \]](#)

Ваш комментарий

Имя *

Почта (скрыта) *

Сайт

Добавить

- ☐ Уведомить меня о новых комментариях по email.
- ☐ Уведомлять меня о новых записях почтой.