Учебный курс. Часть 28. Основы создания макросов

Автор: xrnd | Рубрика: Учебный курс | 16-11-2010 |
Распечатать запись

Отличительной особенность FASM является очень гибкая и мощная поддержка макросов. В этой статье мы рассмотрим лишь основы создания макросов, так как эта тема довольно обширна и рассказать всё в одной статье не получится.

Что же такое макросы? Макросы — это шаблоны для генерации кода. Один раз создав макрос, мы можем использовать его во многих местах в коде программы. Макросы делают процесс программирования на ассемблере более приятным и простым, а код программы получается понятнее. Макросы позволяют расширять синтаксис ассемблера и даже добавлять собственные «команды», которых нет в процессоре.

Обработкой макросов занимается *препроцессор* FASM. Преобразование исходного кода в исполняемый код FASM выполняет в два этапа. Первый этап — *препроцессирование*, а второй — собственно *ассемблирование* или *компиляция*. На первом этапе происходит вычисление всех числовых выражений, вместо констант и названий меток подставляются их фактические значения, вместо макросов подставляется сгенерированный код. На втором этапе все данные и машинные команды преобразуются в соответствующие байты, и в результате получается исполняемый файл требуемого формата.

Синтаксис создания макроса

Для создания макроса используется директива *macro*. Эта директива имеет следующий синтаксис:

```
macro < название_макроса> [< список_параметров>]
{
    < тело_макроса>
}
```

После директивы пишется название макроса, а также может быть указан список параметров (операндов). Параметры в списке перечисляются через запятую. Внутри фигурных скобок записывается тело макроса. Кстати, для коротких макросов можно писать всё это в одной строке:

```
macro <название_макроса> [<список_параметров>] { <тело_макроса> }
```

Тело макроса — это код, который подставляется в то место, где макрос будет вызван. Создание макроса является по сути лишь его объявлением, в этом месте программы никакого кода сгенерировано не будет! Поэтому объявления макросов обычно размещают в самом начале программы или в отдельном файле.

Примеры макросов

В качестве примера рассмотрим простой макрос без параметров, который предназначен для завершения программы:

```
macro exit_app
{
    mov ax,4C00h
    int 21h
}
```

После того, как макрос объявлен, в нужном месте программы достаточно написать exit_app. Туда препроцессор FASM автоматически подставит 2 команды, записанные в теле макроса. Создадим ещё один полезный макрос, предназначенный для такой часто используемой операции, как вывод строки:

```
macro print_str str
{
    mov ah,9
    mov dx,str
    int 21h
}
```

У этого макроса есть один параметр — адрес строки. При генерации кода вместо str будет подставлен тот параметр, который указан при вызове макроса. Обратите внимание, что код будет генерироваться в месте каждого вызова макроса! В этом главное отличие макроса от процедуры. Код процедуры содержится в программе только в одном экземпляре, а вызывается она с помощью передачи управления командой CALL.

Теперь добавим эти макросы в программу «hello, world!» из части 6 учебного курса:

```
 1 ; Макрос выхода из программы
 2 macro exit_app
 3 {
 4     mov ax,4C00h    ;Здесь только объявление макроса, код не г
 5     int 21h
 6 }
 7
 8 ; Макрос вывода строки
 9 macro print_str str
10 {
11     mov ah,9        ;Здесь тоже код не генерируется
12     mov dx,str
13     int 21h
```

```
14 }
15
16 ;-------------------------------------------------------------
17 use16                ;Генерировать 16-битный код
18 org 100h             ;Программа начинается с адреса 100h
19
20     print_str hello ;Вывод строки
21     exit_app        ;Выход из программы
22
23 ;-------------------------------------------------------------
24 hello db 'Hello, macro world!$'
```

В результате основная программа состоит всего из двух строчек, и это уже не похоже на ассемблер, это — макроассемблер FASM 🙂

Расширение системы команд

Макросы можно использовать для расширения системы команд. Например, часто в программе приходится обнулять регистры. Создадим специальный макрос для этой цели:

```
; Макрос - команда обнуления регистра
macro clr reg { xor reg,reg }
```

Теперь обнулять регистры в программе можно так:

```
    clr ax          ;AX=0
    clr si          ;SI=0
    clr bl          ;BL=0
```

Макросы с переменным количеством параметров

Возможно также создавать макросы с переменным количеством параметров. В этом случае имя параметра записывается в квадратных скобках. Для генерации кода

макрос вызывается столько раз, сколько параметров ему было передано. Например, можно написать специальные макросы для улучшения команд PUSH и POP:

```
; Макрос - улучшенная команда push
macro push [arg] { push arg }
; Макрос - улучшенная команда pop
macro pop [arg] { pop arg }
```

Несмотря на то, что название макроса совпадает с именем команды, в теле макроса это имя считается именем команды (иначе получилась бы бесконечная рекурсия 🙂). Данные макросы позволяют помещать в стек и извлекать из стека сразу несколько операндов, что упрощает код программы. Пример использования:

```
    push ax,word[si],5
    pop  dx,cx,ax
```

В результате препроцессором будет сгенерирован следующий код:

```
    push ax
    push word[si]
    push 5
    pop dx
    pop cx
    pop ax
```

Директива include

Возможно, вам захочется написать собственный набор макросов и использовать их в своих программах. В этом случае удобно поместить макросы в отдельный файл и воспользоваться директивой включения файла *include*. Синтаксис директивы *include* очень прост:

```
include 'путь/к/файлу'
```

Путь к файлу указывается в одинарных кавычках и может быть относительным (по отношению к компилируемому файлу) или полным (начиная от буквы диска или корневого каталога системы). Если включаемый файл находится в той же папке, то достаточно указать только имя файла. Расширение файла может быть любым, но обычно используют «inc» или «asm».

Препроцессор FASM читает указанный файл и подставляет код из него вместо директивы *include*. В отдельный файл можно также вынести часто используемые процедуры, отдельные функциональные блоки программы или даже объявления данных.

Если записать макросы в отдельный файл ‘mymacro.inc’, то программа «hello, world!» станет ещё короче:

```
 1 include 'mymacro.inc'
 2
 3 use16                 ;Генерировать 16-битный код
 4 org 100h              ;Программа начинается с адреса 100h
 5
 6     print_str hello ;Вывод строки
 7     exit_app        ;Выход из программы
 8
 9 ;-----------------------------------------------------------
10 hello db 'Hello, macro world!$'
```

Упражнение

Напишите макрос для определения максимального значения. У макроса должно быть 3 операнда: второй и третий сравниваются между собой, больший из них помещается на

место первого. Результаты можете писать в комментариях или на форуме.

Следующая часть »

Комментарии:

Alex
03-12-2010 14:42

```
macro max a,b,c{
cmp a,b
cmovb a,b

cmp a,c
cmovb a,c
}

mov ax,’2′
mov bx,’4′
mov cx,’3′

max ax,bx,cx
```

Вот пример для нахождения максимума из трех значений. Аргументами могут быть только регисты

[Ответить]

xrnd
03-12-2010 14:51

Хороший пример. Но он не совсем соответствует упражнению.

[Ответить]

fufel
12-12-2010 17:20

Здравствуте! Не совсем понял задание, поэтому сделал как понял.

```
include ‘mymacro.inc’
use16
org 100h
macro max a,b,c
{
cmp b,c
jg obm
xchg a,c
jmp @f
obm:
xchg a,b
@@:
}

mov al,2
mov bl,4
mov cl,3
max al,bl,cl
exit_d
```

[Ответить]

xrnd
12-12-2010 21:59

Хороший макрос. Если вместо XCHG поставить MOV — получится как раз по заданию 😉

У этого макроса только один недостаток — его нельзя использовать в коде несколько раз — FASM ругнётся, что метка obm уже объявлена.

[Ответить]

argir
05-01-2011 22:58

```
use16
org 100h
jmp start
macro max op1,op2,op3;op2,3 могут быть регистрами или
цифрами,op1 регистр
{
mov ax,op2
mov bx,op3
cmp ax,bx
ja @f
xchg ax,bx
@@: mov op1,ax
}
start:
max cx,3,5
mov ax,4C00h
int 21h
```

[Ответить]

xrnd
11-01-2011 00:18

В принципе, всё правильно.
Но тут могут возникнуть проблемы, если каким-то из операндов будут AX или BX.
Например:
max cx,bx,ax

Более сложный вариант макроса требует проверки операндов.
Но в статье об этом не рассказывается, поэтому и так неплохо.

[Ответить]

Гость
20-02-2011 23:02

macro xmax x1, x2,x3 ; переменные 16 битные проверки на равенства x2 и x3 нет
{
mov dx, x3
sbb dx ,x2; тут нет особой разницы cmp или просто вычитание так как переменные
jc f@; если c =1 , когда x2 больше x3 перейти по метки
push x3; это выполнится если c=0 push вместо mov
pop x1
@@:
jnc f@; если c =0 , то x1 найден выход
push x2 ; это выполняется если изначально c =1
pop x1
@@:
}

[Ответить]

Гость
20-02-2011 23:25

А можно использовать не посредственно значения ?
xmax 1,3,5
Или только регистры и переменные ?

[Ответить]

Гость
20-02-2011 23:48

macro push [arg] { push arg }
Ну с этем способом передачи параметров в макрос понятно ,)

Можно как не буть так
macro arg1, arg2, arg3
{
mov ax , arg3
}
В принципе и через стёк можно но придётся создавать тогда переменные arg1, arg2, arg3 для такова макроса заранее.

Можно ли в макросе объявлять переменные в памяте ?
Передовая размер и имя в качестве параметров.
ну или как не буть так
macro x
{
rw 1
}
P.S Понятно , что ассемблер на такое неспособен , но компилятор может и не такое потянуть.

[Ответить]

xrnd
22-02-2011 20:39

Макрос на то и макрос 🙂 Операнды не передаются в него через стек или регистры, а подставляются. Причём во время компиляции кода.

Объявлять переменные в памяти можно.
Например, если требуется какие-то сложные структуры в памяти объявить, то можно придумать для этого специальные макросы.

[Ответить]

xrnd
22-02-2011 20:34

Зависит от того, как написать макрос.

Так как в первый операнд записывается результат, то он не должен быть непосредственным значением. Второй и третий тоже одновременно не могут быть непосредственными значениями — какой смысл генерировать код для сравнения констант, если можно сразу подставить результат.

[Ответить]

xrnd
22-02-2011 20:32

Лучше всё-таки использовать CMP.
Команда SBB тут не подходит, она учитывает значение флага CF.
К тому же, макрос нельзя будет использовать, если что-то хранится в регистре DX.

push/pop можно в большинстве случаев заменить одной командой MOV.

[Ответить]

diger
21-02-2011 20:06

помогите . нужно сделать так , чтобы первый аргумент макроса подставлялся до пробела…

короче надо перевести на FASM код:

```
%macro ISR_NOERRCODE 1
global isr%1
isr%1:
cli
push byte 0
```

```
push byte %1
jmp isr_common_stub
%endmacro

%macro ISR_ERRCODE 1
global isr%1
isr%1:
cli
push byte %1
jmp isr_common_stub
%endmacro
```

[Ответить]

Гость
21-02-2011 22:12

А это как
синтаксис macro []
Там пробелы вроде не как не фигурируют.

Вот например

```
use16
org 100h
macro x reg, arg2, arg3 ; 1 регистр , 2 числовые константы
которые создаёт ;компилятор , в конечном коде их небудит
{
if reg in ; это условие если регистр один из 4 выполняется
add reg, arg2 ; к указному регистру + arg2
add reg, arg3
end if ; конец цыкла
}
x cx,2,5
mov ax,4C00h
int 21h
```

«eq» проверяет такие значения на тождественность.(на равенство)
«in» проверяет, принадлежит ли данное значение к списку (1 из варианта)
Попробуй с помощью условия сделать , что тебе надо ,)
Или напиши , что имена тебе надо , я какрас сейчас макросы пытаюсь изучать ,)

[Ответить]

xrnd
22-02-2011 20:51

Здесь нужно использовать специальный оператор ‘#’, чтобы добавить аргумент к названию метки. Вроде так:

```
macro ISR_NOERRCODE arg
{
isr#arg:
    cli
    push 0
    push arg
    jmp isr_common_stub
}

macro ISR_ERRCODE arg
{
isr#arg:
    cli
    push arg
    jmp isr_common_stub
}
```

[Ответить]

Гость
21-02-2011 22:16

```
use16
org 100h
macro x reg, arg2, arg3
{
if reg in <ax,bx,dx,cx>
add reg, arg2
add reg, arg3
end if
}
x cx,2,5
mov ax,4C00h
int 21h
```

[Ответить]

xrnd
22-02-2011 20:54

И что делает этот макрос? 🙂

[Ответить]

Гость
22-02-2011 22:02

Ну складывает ,)
reg =arg2+reg3

reg может быть ax,bx,dx,cx , я хотел показать пример с условиями.
А Упражнение так решить можно ,)

```
use16
org 100h
macro x reg, arg2, arg3
{
if arg2 — arg3 ; Если arg2 меньше arg3
mov reg ,arg2 ; то
```

else if arg2 — arg3 ; Если arg2 больше arg3
mov reg ,arg3 ;
else ; Если не больше и не меньше значит равны
mov reg ,arg3 ;
end if ;
}
x cx,2,5
mov ax,4C00h
int 21h
Результат mov cx 5
Правда , так нельзя ,значения должны быть фиксированными.
А эти конструкция применяются для , создания кода в зависимости от условия.
Условия проверяет компилятор и создаёт конечный код ассемблера.

[Ответить]

Гость
22-02-2011 22:12

Условия проверяется 1 раз при компиляции ,)

[Ответить]

xrnd
23-02-2011 00:14

С фиксированными значениями не интересно 🙂 2 и 5 можно сравнить без макроса.
Ты попробуй определить, параметр регистр или переменная в памяти и сгенерировать правильный код.
Например, если 2 и 3 операнды в памяти, то их нельзя сравнить командой CMP, нужно хотя бы один поместить в регистр.

[Ответить]

Гость
23-02-2011 13:12

Я не нашёл как проверить , является аргумент , простым значением или переменой.
2,3 аргумент роли не играют , и могут быть регистром , переменой в памяти или не посредственым значением.

```
use16
org 100h
macro x arg1, arg2, arg3
{
if arg1 in ; если 1 параметр 16битный регистр
mov arg1, arg2 ; вместо arg1 будит подставлено имя регистра
cmp arg1, arg3
jc @f
push arg1
@@:
jnc @f;
push arg3
@@:
pop arg1
; else if ; тут должна быть проверка , переменная это или непосредственное знач.
else ; по задумки выполняется когда 2 преведущех условия лож
push dx
mov dx,arg2
cmp dx, arg3
jc @f
mov [arg1],arg2
@@:
jnc @f;
mov [arg1],arg3
@@:
pop dx
end if
}
```

```
x cx,2,5
x x1,2,5
mov ax,4C00h
int 21h
x1 dw 1
```

[Ответить]

Гость
23-02-2011 13:13

if arg1 in — ax,bx,dx,cx,si,di — если 1 параметр 16битный регистр

[Ответить]

Гость
23-02-2011 13:34

Хотя нет с значениями из памяти подставляется память (
Но как проверять , переменная это или , не посредственное значение ?

[Ответить]

xrnd
25-02-2011 22:30

Скорее всего, никак.
В случае ошибки код просто не скомпилируется.
Кстати, есть ещё 8-битные регистры 🙂

[Ответить]

Mihahail
08-03-2011 16:30

Не понял, если есть макросы, то зачем теперь выносить код в процедуры.. или я что-то путаю?

[Ответить]

xrnd
09-03-2011 23:00

Есть большая разница между макросами и процедурами. И они используются для разных целей.

Код процедуры содержится в программе только в одном экземпляре.
А макрос генерирует код в месте каждого своего вызова.
Поэтому, например, длинный повторяющийся кусок кода лучше сделать процедурой, чем макросом.
Макросы дают большую гибкость, так как получают параметры на этапе компиляции программы. Процедуры работают и получают параметры только на этапе выполнения.

Так что хорошо знать и то, и другое 🙂

[Ответить]

Mihahail
11-03-2011 23:38

Тогда почему не использовать процедуры?
точнее не так
наверно лучше спросить, чем получение параметров на этапе компиляции выгоднее получения параметров при выполнеии.
Думаю просто мы скидываем нагрузку с проца.
Так? =)
Хотя я думаю, что тут или перативка тратится на длинную программу с макросами, или процессор больше обрабатывает переходы в процедуры, и приходится выбирать между

потреблением оперативки и процессорных ресурсов.
Поправьте, если ошибаюсь.

[Ответить]

xrnd
12-03-2011 23:20

Выгоднее тем, что получив параметры на этапе компиляции, можно сгенерировать лучший код, изменить команды. С помощью процедур такое не сделать.

Например, макросы для создания процедур proc и endp генерируют разные команды в зависимости от параметров.

[Ответить]

Mihahail
13-03-2011 11:31

>>Например, макросы для создания процедур proc и endp генерируют разные
>>команды в зависимости от параметров.

Вот оно что! спасибо вам.
Кстати, будут новые статьи?

[Ответить]

xrnd
20-03-2011 22:26

Обязательно будут!

[Ответить]

plan4ik
15-04-2011 00:07

большая просьба сделай дополнительную статью по макросам (расширеную) так сказать для уже знакомых с базовыми знаниями макро-программирования … так как макросы это основное орудие фасма… и потом твои статьи наберут еще больше внимания и роспостранения)))
может бить даже книгу напишешь по фасму и по ней будут учится буржуи за бугром 😉 …

[Ответить]

xrnd
15-04-2011 00:33

Я хочу написать даже несколько статей про макросы, так как эта тема очень обширная и практически нигде толком не описана. На васме есть статья, но на полное описание макросов она не претендует.

Насчет буржуев не знаю, для них переводить придётся 🙂

[Ответить]

plan4ik
15-04-2011 09:47

ради такой инфы начнут по русски читать со словарем на коленях))

[Ответить]

andrew.NET
10-06-2012 15:43

```
macro max res, a, b
{
cmp a, b
jg .more
jle .less
.more
mov [res], [a]
jmp .fin
.less
mov [res], [b]
jmp .fin
.fin
}
```

[Ответить]

алекс
04-07-2012 19:25

```
macro min_max a,b,c
{
push ax
mov ax,b
cmp ax,c
jg @f
mov ax,c
@@
mov a,ax
pop ax
}
```

[Ответить]

Дима
17-11-2013 11:45

Дано: последовательность n байт (символов) (n <100) Вывести отдельно "большие" буквы отдельно "маленькие" буквы.
Какой Макрос под ету задачку нада ато не пойму!

[Ответить]

Ваш комментарий

Имя *

Почта (скрыта) *

Сайт

Добавить

Уведомить меня о новых комментариях по email.

Уведомлять меня о новых записях почтой.