Лабораторная работа 5. 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 расширить 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.

Директива include

Синтаксис:

```
include    'file name'
```

Эта директива вставляет содержимое файла `file name` в исходный текст. Вставленный текст, естественно, тоже будет обработан препроцессором. Имя файла (и путь к нему, если он есть) должны быть заключены в кавычки " или апострофы '.

В частности, можно описать некоторые макросы в отдельном файле, а затем, с помощью данной директивы включить их в свою программу – получится своего рода библиотека макросов.

Директива EQU

Синтаксис:

```
name1 equ name2
```

Это команда говорит препроцессору, что необходимо заменить все последующие `name1` на `name2`.

Например:

```
count equ 10 ; это команда препроцессора  
mov    ecx, count
```

преобразуется в:

```
mov    ecx, 10
```

Ещё пример:

```
mov    eax, count  
count equ 10  
mov    ecx, count
```

преобразуется в:

```
mov    eax, count  
mov    ecx, 10
```

потому что препроцессор заменит `count` только после директивы `equ`.

Даже это работает:

```
10    equ 11
mov   ecx, 10
```

после обработки препроцессором, получим:

```
mov   ecx, 11
```

Директива **RESTORE**

Можно заставить препроцессор прекратить заменять идентификаторы, определённые директивой EQU. Это делает директива RESTORE

Синтаксис:

```
restore    name1
```

name1 - это идентификатор определённый ранее в директиве EQU. После этой команды name больше не будет заменяться на name2.

Например:

```
mov   eax, count
count equ 10
mov   eax, count
restore    count
mov   eax, count
```

получим:

```
mov   eax, count
mov   eax, 10
mov   eax, count
```

Обратите внимание, что для определений сделанных директивой EQU работает принцип стека. То есть, если мы два раза определим один и тот же идентификатор используя EQU, то после однократного использования RESTORE значение идентификатора будет соответствовать определённому первой директивой EQU.

Например:

```
mov   eax, count
count equ 1
mov   eax, count
count equ 2
mov   eax, count
count equ 3
mov   eax, count
restore    count
mov   eax, count
restore    count
mov   eax, count
restore    count
mov   eax, count
```

получим:

```
mov   eax, count
```

```
mov    eax, 1
mov    eax, 2
mov    eax, 3
mov    eax, 2
mov    eax, 1
mov    eax, count
```

Если попытаться выполнить RESTORE большее количество раз, чем было сделано EQU, никаких предупреждений выдано не будет. Значение идентификатора будет неопределенно.

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.

Пример:

```
use16
org 100h

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

macro endl
{
    mov ah, 2
    mov dl, 10
    int 21h
    mov dl, 13
    int 21h
}

print_str s
endl
mov ah,8
int 21h
```

```
int 20h
```

```
s db "hello world!$"
```

Директива LOCAL

Возможно, появится необходимость объявить метку внутри тела макроса:

```
macro ToBeOrNotToBe choice
{
    cmp choice, '1'
    jnz .next
    print_str ToBe
    jmp .next2
.next:
    print_str NotToBe
.next2:
}
```

но если использовать такой макрос 2 раза, то и метки `.next` и `.next2` будут объявлены дважды, что приведёт к ошибке. Эта проблема решается объявлением локальных меток с помощью директивы `LOCAL`.

Синтаксис:

```
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.

```
macro ToBeOrNotToBe choice
{
    local .next, .next2
    cmp choice, '1'
    jnz .next
    print_str ToBe
    jmp .next2
.next:
    print_str NotToBe
.next2:
}
```

На самом деле, `.next` заменяется на `.next?XXXXXXXX`, где `XXXXXXXX` - какой-то шестнадцатеричный номер генерируемый препроцессором.

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

```
macro pushstr string
{
    local    behind
    call behind
    db    string, 0
behind:
}
```

```
MyProc:
  pushstr  'aaaa'
.a:
```

будет преобразовано в:

```
MyProc:
  call behind?00000001
  db  'aaaa', 0
behind?00000001:
.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`:

```
macro pushstr string
{
  local  ..behind
  call ..behind
  db  string,0
..behind:
}
```

```
MyProc:
  pushstr  'aaaa'
.a:
```

Пример. Ввести строку с клавиатуры, вывести её на экран.

```
use16
org 100h
```

```
macro print_str str
{
  pusha
  mov ah, 9
  mov dx, str
  mov bx, dx      ;1й байт хранит макс. число символов в строке
  inc bx          ;а 2й - фактическое
  mov al, [bx]   ;фактическое число символов в регистр AL
  add bl, al     ;прибавляем это число к bl, чтобы найти конец
строки
  inc bl         ;и переходим к следующему символу
  mov byte [bx], '$' ;ставим там конец строки
  add dx, 2      ;пропускаем первые 2 байта
  int 21h       ;и выводим строку
  popa
}
```

```
macro endl
{
  pusha
  mov ah, 2
  mov dl, 10
  int 21h
}
```

```

    mov dl, 13
    int 21h
    popa
}

mov dx, s1 ;адрес строки
mov [s1], 51 ;1-й байт - макс. количество символов в строке
mov ah, 0ah ;функция DOS для ввода строки
int 21h
endl ;вызываем макрос перехода на новую строку

print_str s1 ;макрос вывода введенной строки
endl ;вызываем макрос перехода на новую строку

mov ah, 8
int 21h

int 20h
;-----
s1 db 53 dup(?)

```

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Выбрать вариант в соответствии со списком. Данные вводить с клавиатуры. Результат выводить на экран. При решении использовать макросы.

ВАРИАНТЫ

1. Сколько раз в введенной строке встречается число 10?
2. Повторить все символы строки заданное количество раз.
3. Подсчитать в строке число букв А и В, если букв А больше, чем В, то удалить в строке все символы В.
4. Подсчитать количество слов в данном предложении.
5. Определить, сколько раз в строке встречается данный символ.
6. Дана символьная строка, заканчивается точкой. Найти длину самого длинного слова.
7. Имеется строка символов, содержащая буквы латинского алфавита и цифры. Определить количество цифр в строке.
8. Преобразовать строку, заменив все «:» запятыми, встречающиеся среди первых $[n/2]$ символов и заменив точками все «!», встречающиеся среди символов, стоящих после $[n/2]$ символов.
9. В предложении найти слова, начинающиеся и заканчивающиеся на одну и ту же букву.
10. Заменить в заданной строке знак "!" на сочетание "???".
11. Определить сколько раз в тексте встречаются гласные буквы.
12. В данной строке подсчитать сумму цифр, содержащихся в ней.
13. В заданной строке удвоить каждый символ.
14. Вводится строка. Поставить запятую после каждого пробела данной строки.
15. Вводится строка. Удалить из строки лишние пробелы.
16. Вводится строка. Удалить из строки пробелы, стоящие перед запятыми.
17. Дана строка и число n. Верно ли, что в ней есть по крайней мере n подряд идущих букв а?
18. Дана строка. Удалить из нее последовательности символов, расположенные между скобками. Скобки также удалить. После первой открывающей скобки другие открывающие скобки игнорируются. Если скобка не закрывается до конца строки, то удалять все до конца строки. Закрывающая скобка без парной открывающей игнорируется.
19. Заменить в исходной строке все 0 на заданный символ.
20. В исходной строке вместо заданного символа вставьте 0.
21. Удалить из строки все одинаковые символы, идущие подряд.

22. Добавить в строку недостающие пробелы (после знаков препинания обязательно должен быть пробел).
23. Подсчитать в строке число букв А и В. Найти разницу N между этими числами и вывести N раз букву, которая встречается чаще.
24. Определить является ли заданная строка палиндромом без учета регистра.
25. Написать программу, разбивающую текст на строки длиной не N символов.
26. Поменять местами в предложении все 0 и 1.