

**Весна 2021**

# **Системное программное обеспечение**

Онлайн-лекции

## **Лекция №8: Макросы**

Доцент, к.т.н. ГОЛЬЦОВ Александр Геннадьевич

# Гибкость проекта

- Использование макросов, определяемых символов, директив условной компиляции позволяет создавать гибкие проекты: один и тот же исходный текст в зависимости от настроек порождает исполнимые файлы с весьма разнообразными возможностями и функциями.
- Справедливо в равной мере для ЯВУ и ассемблерных языков.

# Макрос

- Макрос - это описание последовательности команд.
- Макрос имеет имя и набор параметров.
- Как и процедура, макрос описывает некоторое действие, которое многократно нужно выполнить в разных местах программы.
- Содержимое макроса целиком копируется в код программы в том месте, где прописано обращение к макросу. В программе получается множество копий макроса.
- В отличие от процедуры, макрос не вызывается и из него не производится возврат.
- Для действий в 1-3 команды макросы обычно эффективнее процедур.

# Описание макроса

Имя MACRO параметры

..... набор команд .....

ENDM ; перед ENDM не дублируется имя

- Макрос - абстрактная сущность, сам по себе (пока не вызван) он не существует, **можно описывать вне сегмента кода** (и любого другого).
- MACRO занимает место под код там, где "вызывается", а не при объявлении (аналогично EQU).
- Параметры (формальные параметры макроса) - это просто идентификаторы (без типов), перечисленные через запятую.
- Параметры могут упоминаться внутри макроса по тем же правилам, что и любые символы, определенные через EQU. Чему именно будут соответствовать эти символы - определяется при "вызове" макроса (обращении к нему).

# Обращение к макросу

## Имя аргументы

- В качестве аргументов перечисляются метки, символы, имена регистров и вообще любые последовательности символов.
- При вызове макроса задается соответствие каждого из параметров какой-то метке, символу, регистру. Как будто бы дополнительно для конкретного вызова макроса устанавливается связь

параметр equ аргумент

- Везде, где внутри макроса упомянут параметр, в данном конкретном случае подставляется указанный аргумент.
- При подстановке аргумента макрос должен оставаться синтаксически корректным.
- Аргументы могут не передаваться при обращении, хотя объявлены в макросе; проверить наличие переданного параметра можно внутри макроса (см. ниже).

# Примеры макросов

; Сдвиг DX:AX влево

ShlDxAx MACRO

shl ax,1

rcl dx,1

ENDM

.....  
ShlDxAx ; без параметров

shl ax,1  
rcl dx,1



# Примеры макросов

```
Swap MACRO Src, Dst
    push Dst
    mov Dst, Src
    pop Src
ENDM
```

.....  
Swap AX, BX

push ax  
mov ax,bx  
pop bx

.....  
Swap AX, N ; переменная

push N  
mov N,ax  
pop ax

.....  
Swap M, N ; нельзя  
Swap ds,es ; нельзя  
Swap 1,2 ; нельзя

А так лучше:

```
push dst
push src
pop dst
pop src
```

# Метки внутри макросов

- Просто метку упомянуть нельзя - она будет подставляться в код каждый раз при упоминании макроса, т.е. дублироваться.
- Внутри макроса в директиве LOCAL через запятую перечисляются локальные символы:

```
Delay MACRO n  
    LOCAL m1  
    mov    cx,n  
m1: loop  m1  
ENDM
```



# Макрос с данными

```
.model small
.stack 100h
Message MACRO s
    LOCAL str
    .data
    str db s,13,10,'$'
    .code
    mov ah,9
    mov dx,offset str
    int 21h
ENDM
```

```
.code
    mov     ax,@data
    mov     ds,ax

    Message 'Error!'
    Message 'Execution terminated'

    mov     ax,4C00h
    int     21h
end
```

```
cs:0000 B8354F      mov     ax,4F35
cs:0003 8ED8        mov     ds,ax
cs:0005 B409        mov     ah,09
cs:0007 BA0000      mov     dx,0000
cs:000A CD21        int     21
cs:000C B409        mov     ah,09
cs:000E BA0900      mov     dx,0009
cs:0011 CD21        int     21
cs:0013 B8004C      mov     ax,4C00
cs:0016 CD21        int     21
cs:0018 0000        add     [bx+si],al
cs:001A 0000        add     [bx+si],al
cs:001C 0000        add     [bx+si],al
```

```
ds:0000 45 72 72 6F 72 21 0D 0A Error!J
ds:0008 24 45 78 65 63 75 74 69 $Executi
ds:0010 6F 6E 20 74 65 72 6D 69 on termi
ds:0018 6E 61 74 65 64 0D 0A 21 natedJ$
```

# Оператор &

- & - катенация (слияние строк) - позволяет "дописать" содержимое параметра после или перед знаком:

```
mData MACRO name,sze,n,str
    name&Count dw sze n
    name&Msg db '-- &str& --'
ENDM
```

```
.data
mData Apples,w,25,Яблоки
.code
```

```
ApplesCount dw 25
ApplesMsg db '-- Яблоки --'
```

```
.....
mov ax,ApplesCount
mov di, offset ApplesMsg
```

# Оператор < >

- То, что заключено в угловые скобки, считается одним единым параметром вызываемого макроса

SomeMacros 2, <; text>

- первый параметр - 2
- второй параметр - точка с запятой и слово text
- слово text при вызове макроса не является комментарием, но может им стать после подстановки в макрос

# Оператор %

- Делает параметром вычисленное значение выражения, а не само выражение

org 500

SomeLabel label byte

.....

SomeMacros %SomeLabel+5, 12

- параметрами являются 505 и 12, SomeLabel не "вписывается" в макрос, на вход подается сразу результат, число 505

# Условная компиляция

- Директивы условной компиляции применимы везде, в том числе - внутри макросов.
- Простейшая форма:  
IF expr  
    команды  
ENDIF
- Выражение - см. лекцию про определение данных, любое выражение-константа, вычисляемое на этапе компиляции и не содержащее ссылок на регистры или значение ячеек памяти.
- В выражении допустимы имена меток и символы, определяемые через EQU или =

# Условная компиляция

- Полная форма:

начальное\_условие

команды

ELSEIF выражение

команды

ELSEIF выражение

команды

.....

ELSE

команды

ENDIF

# Формулировка начального условия

- IF/IFE *expr* - если *expr* = 1/0
- IFDEF/IFNDEF *sym* - если определен или не определен символ (метка, процедура, EQU и т.п.)
- IFB/IFNB *arg* - если аргумент пустой/не пустой
- IFIDN *arg1*, *arg2* - если аргументы 1 и 2 идентичны
- IFDIF *arg1*, *arg2* - если аргументы 1 и 2 разные
- IFINDI/IFDIFI *arg1*, *arg2* - то же при условии неразличимости больших и малых букв

# Пример условной конструкции

```
InitialValue equ 1000
```

```
.....
```

```
InitAX MACRO
```

```
    IFDEF InitialValue
```

```
        mov ax, InitialValue
```

```
    ELSE
```

```
        mov ax,0
```

```
        ; или сгенерировать случайным образом
```

```
    ENDIF
```

```
ENDM
```



# Директива EXITM

- Используется совместно с директивами условной компиляции
- Позволяет как бы "выйти" из середины макроса, т.е. не копировать код макроса, расположенный после директивы, в точку обращения к макросу.

# Спецмакрос REPT

- Не имеет имени, интерпретируется сразу там, где встречается.
- Подобно директиве DUP, размножающей элементы данных, позволяет размножить свое содержимое.

- Формат:

REPT количество

..... команды .....

ENDM

- Пример:

REPT 3

shl ax,1

ENDM



shl ax,1

shl ax,1

shl ax,1

# Спецмакрос IRP

- Не имеет имени, интерпретируется сразу там, где встречается.

- Формат:

IRP параметр, <список значений>

... упоминается параметр .....

ENDM

- Дублирует тело столько раз, сколько элементов в списке значений, для каждого

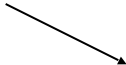
- Пример:

IRP s, <'One', 'Two', 'Three'>

db s

ENDM

db 'One'  
db 'Two'  
db 'Three'



# Спецмакрос IRPC

- Не имеет имени, интерпретируется сразу там, где встречается.

- **Формат:**

## IRPC параметр, строка ; не в кавычках

... упоминается параметр .....

# ENDM

- Дублирует тело столько раз, сколько символов в списке значений, **для каждого символа**

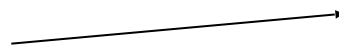
- Пример:

IRPC num, 1234

push num

ENDM

```
push 1
push 2
push 3
push 4
```



; умножает AX на константу, рез. в DX:AX

```
SmartMul MACRO factor
    IF factor eq 1
        cwd
        EXITM
    ENDIF
    fast=0
    count=1
    bit=2
    REPT 16
        IF factor eq bit
            fast=1
            EXITM
        ENDIF
        count=count+1
        bit=bit shl 1
    ENDM
    IF fast
        cwd
        REPT count
            shl ax,1
            rcl dx,1
        ENDM
    ELSE
        mov dx,factor
        imul dx
    ENDIF
ENDM
```

## имер сложного макроса

В этом макросе описан нетривиальный алгоритм, при интерпретации которого генерируется код в том месте, где прописано каждое обращение к этому макросу ("вызов").

# Продолжение примера

mov ax,15  
SmartMul 1  
SmartMul 16  
SmartMul 7

mov	ax,000F
cbw	
xor	dx,dx
shl	ax,1
rcl	dx,1
shl	ax,1
rcl	dx,1
shl	ax,1
rcl	dx,1
shl	ax,1
rcl	dx,1
mov	dx,0007
mul	dx

# Макросы и процедуры

- Для вызова процедур часто вместе с процедурой удобно разработать макрос.
- Перед вызовом процедуры часто параметры нуждаются в некоторой дополнительной обработке.
- Макрос помещает нужным образом в нужные места параметры и вызывает процедуру.

# Пример макроса для процедуры

```
_PrnStr proc
    arg StrAddr:dword
    push bp
    mov  bp,sp

    lds  dx, StrAddr


    mov  ah,9
    int  21h
    pop  bp
    ret
_PrnStr endp
```

```
PrnStr Macro s
    push ds
    push seg s
    push offset s
    call _PrnStr
    add  sp,4
    pop  ds
ENDM
```

```
.data
Hello db 'Hello, world$'
```

.....

PrnStr Hello



```
push    ds
push    4F36
push    0000
call    0000
add     sp,0004
pop     ds
```



# Универсальный макрос вызова процедуры

```
invoke macro subr,par0,par1,par2,par3,par4,par5,par6,par7
  irp par,par7,par6,par5,par4,par3,par2,par1,par0
    ifnb <par>
      push par
    endif
  endm
  call subr
endm
```

---

```
invoke _PrnStr, offset(Hello), seg(Hello)
add    sp,4
```

# Директива INCLUDE

## INCLUDE имя\_файла\_ASM

- Позволяет включить в состав исходного текста (файла .ASM) содержимое другого файла, упомянутого в директиве INCLUDE, в том месте, где упомянута директива
- Для малых проектов удобно иметь библиотеки макросов или процедур в отдельном файле (например, [mymacros.asm](#)) и включать их в тот или иной проект, упоминая как `INCLUDE mymacros.asm`
- Дополнительно активно применяются директивы условной компиляции.

Спасибо за внимание.