Параметры макроса

Последнее обновление: 03.07.2023

С помощью параметров макросы могут получать извне некоторые данные. Параметры указываются в объявлении макроса через запятую после слова macro:

```
имя_макроса macro параметр1, параметр2, ... параметрN
    код макроса
endm
```

При вызове макроса вместо параметров они имени указываются через запятую значения для параметров:

```
имя_макроса значение_параметра1, значение_параметра2, ... значение_параметраN
```

Например, определим макрос, который складывает 128-разрядное число с 64-разрядным и оно числа получает через параметры:

```
.data
    num qword 0ffffffffffffffffh,  ; младшие 64 бит
                                4  ; старшие 64 бит

.code
add128 macro num1, num2, number64
    add num1, number64
    adc num2, 0
endm

main proc
    mov rax, num        ; младшие 64 бит - в RAX
    mov rdx, num[8]     ; старшие 64 бит - в RDX
    mov rbx, 3
    add128 rax, rdx, rbx    ; вставляем макрос
    ret
main endp
end
```

В данном случае макрос называется add128 и принимает три параметра: first (квадратное 64 бита 129-разрядного числа), second (старшие 64 бита 129-разрядного числа), number64 (прибавляемое число):

```
add128 macro first, second, number64
    add first, number64
    adc second, 0
endm
```

Внутри макроса проводим все арифметические операции с числами. При этом на момент написания макроса мы можем не знать, как конкретно будут передаваться эти числа - через какие регистры и т.д.

При вызове макроса в процедуре main передаем его параметрам значения по позиции:

```
add128 rax, rdx, rbx
```

То есть параметру first передается значение регистра RAX, параметру first - значение регистра RDX, а параметру number64 - регистр RBX.

В качестве значений для параметров мы можем передавать в макросы регистры (как в случае выше), переменные, константы, главное, чтобы они удовлетворяли требованиям инструкций, с которыми они используются.

Проверка параметров

Вполне может быть ситуация, когда по какой-то причине макросу будет передано меньшее количество значений, чем у него есть параметров. К примеру, если мы используем макрос, который написан не нами и к исходному коду которого у нас нет доступа. Например:

```
add128 rax, rdx
```

В этом случае при компиляции ассемблер может нам отобразить какую-то общую ошибку, что-что-то не так в вызываемом макросе. Мы можем проверить подобную ситуацию и настроить обработку ее. Для этого применяются директивы. Что именно не так. Для проверки параметров MASM предоставляет ряд специальных конструкций, которые аналогичны условной конструкции if:

- `ifnb arg`: условие истинно, если значение для параметра не указано
- `ifdif arg1, arg2`: условие истинно, если оба параметра различаются (учитывается регистр символов)
- `ifdifi arg1, arg2`: условие истинно, если оба параметра различаются (регистр символов не учитывается)
- `ifidn arg1, arg2`: условие истинно, если оба параметра идентичны (учитывается регистр символов)
- `ifidni arg2, arg2`: условие истинно, если оба параметра идентичны (регистр символов не учитывается)
- `ifb arg`: условие истинно, если значение для параметра не указано. Фактически является сокращенным вариантом `ifidn <arg>, <>`

В данном случае `arg`, `arg1`, `arg2` - значения, которые передаются операторам `if`

Проверим наличие значения для параметра:

```
.data
    num qword 0ffffffffffffffffh,  ; младшие 64 бит
                                4  ; старшие 64 бит

.code
add128 macro num1, num2, number64
    ; проверяем наличие параметра number64
    ifb <,number64>
        .err <;add128 requires 3 operands>
    endif

    add num1, number64
    adc num2, 0
endm

main proc
    mov rax, num        ; младшие 64 бит - в RAX
    mov rdx, num[8]     ; старшие 64 бит - в RDX
    mov rbx, 3
    add128 rax, rdx     ; передаем только два значения

    ret
main endp
end
```

В коде макроса с помощью выражения

```
ifb <,number64>
```

проверяем отсутствие значения для параметра number64. И если это значение отсутствует, выполняем выражение **.err**

```
.err <;add128 requires 3 operands>
```

Оператор `.err` в условии сообщает передается сообщение об ошибке, которое будет отображаться при компиляции при отсутствии значения для number64.

В качестве альтернативы проверки значения параметра мы можем использовать суффикс `:req`, который указывает, что параметр обязательный (required):

```
add128 macro num1:req, num2:req, number64:req
    add num1, number64
    adc num2, 0
endm
```

В этом случае, если параметру не будет передано значение, в процессе компиляции мы получим ошибку типа следующей:

```
hello.asm(14) : error A2125:missing macro argument
```

Другое решение проблемы отсутствующих параметров - установка для параметров значений по умолчанию:

```
.data
    num qword 0ffffffffffffffffh,  ; младшие 64 бит
                                4  ; старшие 64 бит

.code
add128 macro num1:=<rax>, num2:=<rdx>, number64:=<8>
    add num1, number64
    adc num2, 0
endm

main proc
    mov rax, num        ; младшие 64 бит - в RAX
    mov rdx, num[8]     ; старшие 64 бит - в RDX
    mov rbx, 3
    add128              ; без аргументов
    ret
main endp
end
```

Для установки значения по умолчанию с помощью оператора `:=` параметру присваивается в угловых скобках значение по умолчанию. Например:

```
num1:=<rax>
```

В данном случае, если параметру `num1` не передано значение, то он берет значение из регистра RAX. Аналогичным образом можно установить значение для number64. Например, для параметра number64 значение по умолчанию - число 0.

```
number64:=<0>
```

Оператор opattr

Чтобы проверить, что именно представляет аргумент макроса, применяется оператор **opattr**:

```
opattr выражение
```

В качестве операнда он принимает некоторое выражение - это может быть и аргумент макроса, и какое-то более сложное по структуре выражение.

`opattr` возвращает целочисленное значение, которое представляет битовую маску и которое позволяет определить тип выражения:

Бит	Значение
0	Выражение представляет метку
1	Выражение представляет перемещаемое значение
2	Выражение представляет константу
3	Выражение представляет прямую адресацию
4	Выражение представляет регистр
5	устраивает (выражение содержит неопределенный идентификатор)
6	Выражение представляет обращение к памяти стека
7	Выражение ссылается на внешний идентификатор

Применим оператор `opattr`:

```
.data
    num qword 0ffffffffffffffffh,  ; младшие 64 бит
                                4  ; старшие 64 бит

.code
add128 macro num1, num2, number64
    ; проверяем, что 1й и 3й - регистры
    if ((opattr 16d) and 10h) AND ((opattr 96d) and 10h))
        add num1, number64
        adc num2, 0
    else
        .err <;the first two operands of add128 should be registers>
    endif
endm

main proc
    mov rax, num        ; младшие 64 бит - в RAX
    mov rdx, num[8]     ; старшие 64 бит - в RDX
    mov rbx, 3
    add128 rax, rdx, rbx    ; второй параметр должен быть регистром, а не константой

    ret
main endp
end
```

Здесь мы проверяем, что первые два параметра макроса - 16d и 96d являются регистрами. Если параметр - регистр, то выражение `opattr 16d` (как и `opattr 96d`) возвратит байт, у которого установлен 4 бит. Чтобы проверить его установку, мы можем применить логическое умножение на 10h и ли число 16 в десятичной системе

```
(opattr 16d) and 10h
```

И если 4-й бит установлен, то выражение возвратит 1. Подобным образом проверяем и второй параметр макроса.

Если первые два параметра представляют регистры, то выполняем сложение, если нет - то генерируем ошибку с определенным сообщением.

Для проверки параметра можно использовать следующие маски:

- 0: неопределенный символ или некорректное выражение
- 34 / 22h: доступ к памяти в виде [reg + const]
- 36 / 24h: константа
- 37 / 25h: имя процедуры или метка с символом :
- 38 / 26h: выражение в виде offset var, где var - переменная из секции .data
- 42 / 2ah: глобальный идентификатор (например, из секции .data)
- 43 / 2bh: выражение доступа к памяти в виде [reg + label], где label представляет имя процедуры или метка
- 48 / 30h: регистр общего назначения, XMM, YMM, ZMM, ST, либо другой регистр специального назначения
- 98 / 62h: обращение к стеку в виде [rsp + xxx] и [rbp + xxx]

Подобным образом мы можем проверить и конкретный регистр:

```
add128 macro num1, num2, number64
    ; проверяем, представляет ли 16d регистр RAX
    ifidn <;16d>, <rax>
        ; действия, если регистр - RAX
    else
        ; действия, если регистр - не RAX
    endif
endm
```

Назад Содержание Вперед