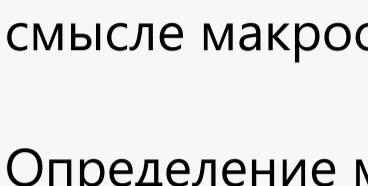


Макросы

Определение макросов

Последнее обновление: 03.07.2023



В ассемблере макрос представляет идентификатор, который ассемблер преобразует в дополнительный код. Макросы позволяют заменить длинные повторяющиеся последовательности код гораздо более короткими последовательностями. В каком-то смысле макросы MASM можно назвать процедурами/функциями времени компиляции.

Определение макроса в общем случае имеет следующую форму:

```
1 имя_макроса macro аргументы_макроса
2     код макроса
3 endm
```

Сначала идет объявление макроса - имя макроса, затем оператор `macro` и после указываются аргументы макроса. Аргументы необязательны, макрос может вообще не иметь аргументов. Завершается макрос оператором `endm`. Между объявлением макроса и `endm` располагается собственно код макроса.

Рассмотрим пример макроса, который складывает 128-разрядное число с 64-разрядным:

```
1 .data
2     num qword 0xffffffffffffffffh,      ; младшие 64 бит
3                                     4      ; старшие 64 бит
4 .code
5 add128 macro
6     add rax, rbx
7     adc rdx, 0
8 endm
9
10 main proc
11     mov rax, num          ; младшие 64 бит - в RAX
12     mov rdx, num[8]        ; старшие 64 бит - в RDX
13     mov rbx, 3
14     add128                ; вставляем макрос
15     ret
16 main endp
17 end
```

В данном случае макрос называется `add128`:

```
1 add128 macro
2     add rax, rbx
3     adc rdx, 0
4 endm
```

Здесь подразумеваем, что 128-разрядное число располагается в регистрах RDX:RAX: младшие 64 битов в RAX, а старшие 64 битов - в RDX. Прибавляемое 64-битное число передается через регистр RBX. Работа макроса проста - складываем 64-битное число RBX с младшими 64 битами RAX и, если при этом устанавливается флаг переноса, прибавляем бит к старшим 64-битам в RDX.

Здесь 128-битное число представляет массив `num` из двух подчисел `qword`, которое загружается в RAX и RDX. Для вставки макроса в код применяется имя макроса. То есть макрос не вызывается в процессе выполнения, его код вставляется в место вызова макроса на этапе компиляции. То есть фактически мы получим программу:

```
1 .data
2     num qword 0xffffffffffffffffh,      ; младшие 64 бит
3                                     4      ; старшие 64 бит
4 .code
5 main proc
6     mov rax, num          ; младшие 64 бит - в RAX
7     mov rdx, num[8]        ; старшие 64 бит - в RDX
8     mov rbx, 3
9     add rax, rbx
10    adc rdx, 0
11    ret
12 main endp
13 end
```

Макросы или функции?

В приведенном выше примере мы могли бы определить макрос сложения чисел в виде отдельной процедуры. И тут может возникнуть вопрос: что выбрать - макросы или функции? Минусом макросов является то, что их код вставляется в каждое место, где они используются. Что ведет к общему увеличению объема программы. С другой стороны, макросы позволяют избежать переходов и сохранений/восстановлений адреса следующей инструкции, что положительно влияет на производительность. Макросы немного быстрее, чем вызовы процедур, потому что вы не выполняете вызов и соответствующие инструкции `ret`. Кроме того, с макросами сам код становится чуть более читабельным. В общем случае рекомендуется применять макросы для коротких, критичных по времени частей программы. Тогда как процедуры применяются для более длинных блоков кода и когда время выполнения не так критично.

[Назад](#) [Содержание](#) [Вперед](#)



Помощь сайту

[Помощь сайту](#)

Юмани:

410011174743222

Номер карты:

4048415020898850

[Телеграмм](#)

[Вконтакте](#) | [Телеграм](#) | [Донаты/Помощь сайту](#)

Contacts: metanit22@mail.ru

Copyright © Евгений Попов, metanit.com, 2026. Все права защищены.