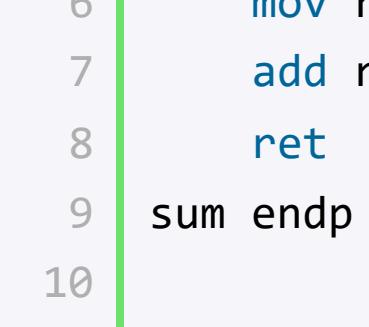


Параметры

Последнее обновление: 02.07.2023



Через параметры процедуры могут принимать из вне некоторые значения. Обычно для передачи параметров применяются регистры. Но поскольку количество регистров ограничено, и они могут использоваться для других целей, также можно передавать значения параметров через стек или через глобальные переменные. Можно комбинировать различные подходы.

Если параметров немного, распространенным способом является передача значений в процедуру через регистры. Например, определим простейшую процедуру, которая получает извне два числа и складывает их:

```
1 .code
2 ; Процедура sum принимает два параметра
3 ; RCX - первое число
4 ; RDX - второе число
5 sum proc
6     mov rax, rcx ; в RAX копируем число из RCX
7     add rax, rdx ; складываем с числом из RDX
8     ret
9 sum endp
10
11 main proc
12     mov rcx, 3 ; первый параметр для процедуры sum
13     mov rdx, 4 ; второй параметр для процедуры sum
14     call sum
15     ret
16 main endp
17 end
```

Здесь мы предполагаем, что два числа в процедуру sum будут передаваться через регистры RCX и RDX. В процедуре sum помещаем в RAX первое число из RCX и складываем его с числом из RDX. В итоге после выполнения этой программы в регистре RAX будет число 7.

Если мы сами пишем все процедуры программы на ассемблере, то мы можем установить себе свои собственные правила, каким образом, через какие именно регистры будут передаваться параметры в процедуру. Однако если мы действуем какие-то внешний функционал, например, взаимодействуем с функциями C/C++, то нам придется также применять те условности, которые накладывают эти функции и конкретные ОС. В частности, в при вызове функций C/C++ на Windows надо следовать следующим соглашениям:

- Вызывающий код передает первые четыре параметра в регистры, а в стеке надо резервировать память для этих параметров.
- Для передачи в функцию первых четырех параметров (целочисленных) используются регистры RCX, RDX, R8 и R9 соответственно. Если параметры представляют числа с плавающей точкой, то они передаются через регистры XMM0, XMM1, XMM2 и XMM3.
- Параметры всегда представляют собой 8-байтовые значения.
- Вызывающий код должен зарезервировать для параметров в стеке как минимум 32 байта, даже если параметров меньше пяти (плюс 8 байтов для каждого дополнительного параметра, если параметров пять или более).

Передача параметров через стек

Еще один способ передачи параметров представляет передача через стек:

```
1 .code
2 sum proc
3     mov rax, [rsp+24] ; RAX = 1
4     add rax, [rsp+16] ; RAX = RAX + 2 = 3
5     add rax, [rsp+8] ; RAX = RAX + 3 = 6
6     ret
7 sum endp
8
9 main proc
10    push 1
11    push 2
12    push 3
13    call sum
14    add rsp, 24
15    ret
16 main endp
17 end
```

В функции main процедуре sum через стек передаются три параметра с помощью инструкции push

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

Поскольку инструкция push помещает в стек 8-байтные значения, то в данном случае три добавленных числа займут в стеке пространство в 24 байта.

После вызова процедуры sum это пространство очищается

```
1 add rsp, 24
```

В процедуре sum для получения параметров используем косвенную адресацию и смещение относительно указателя RSP. При вызове процедуры sum стек условно будет выглядеть следующим образом:

Регистр RSP будет указывать на ячейку с адресом возврата. Соответственно, чтобы получить тот или иной параметр, нам надо использовать смещение 8, 16 или 24:

```
1 mov rax, [rsp+24] ; RAX = 1
```

В итоге после выполнения программы в регистре RAX будет число 6.

Однако добавление в стек с помощью инструкции push может быть не оптимальным способом. В частности, выделенная память для стека может быть избыточна. В нашем случае для наших трех чисел необязательно выделять 24 байта, можно обойтись гораздо меньшим объемом. В этом случае вызывающий код может вручную помещать значения в нужные области стека, используя регистр RSP и смещения:

```
1 .code
2 sum proc
3     xor rax, rax ; обнуляем регистр
4     mov ax, [rsp+14] ; AX = 1
5     add ax, [rsp+12] ; AX = AX + 2 = 3
6     add ax, [rsp+10] ; AX = AX + 3 = 12
7     ret
8 sum endp
9
10 main proc
11    sub rsp, 8 ; резервируем для параметров 8 байт
12    mov ax, 2
13    mov [rsp+6], ax
14    mov ax, 4
15    mov [rsp+4], ax
16    mov ax, 6
17    mov [rsp+2], ax
18    call sum
19    add rsp, 8 ; восстанавливаем указатель стека
20    ret
21 main endp
22 end
```

Также в стек помещаются три числа, но теперь они представляют 2-байтные числа. Соответственно нам нужно всего лишь 6 байт. Однако поскольку стек должен быть выровнен по 8 байтам, и соответственно выделенное пространство должно быть не меньше общего размера помещаемых элементов и также должно быть кратным 8, поэтому выделяем 8 байт, где 2 байта будут незаполненными. Далее помещаем данные через регистр AX в соответствующую область в стеке:

```
1 mov ax, 1
2 mov [rsp+6], ax
```

При получении данных в процедуре sum при установке смещения по прежнему учитываем 8 байтов адреса возврата:

```
1 mov ax, [rsp+14] ; RAX = 2
```

Стоит отметить, что при взаимодействии с функциями со сторонними API, например, с функциями C/C++ и API операционных систем есть свои ограничения на работу со стеком. Например, на Windows при работе с C/C++ требуется, чтобы каждый элемент в стеке все равно было равен 8 байтам, даже несмотря на то, что в реальности ему достаточно места. Например:

```
1 .code
2 sum proc
3     xor rax, rax ; обнуляем регистр
4     mov ax, [rsp+24] ; AX = 1
5     add ax, [rsp+16] ; AX = AX + 2 = 3
6     add ax, [rsp+8] ; AX = AX + 3 = 6
7     ret
8 sum endp
9
10 main proc
11    sub rsp, 24 ; резервируем для параметров 24 байта
12    mov ax, 1
13    mov [rsp+16], ax
14    mov ax, 2
15    mov [rsp+8], ax
16    call sum
17    add rsp, 24
18    ret
19 main endp
20 end
```

Здесь также по сути помещаем в стек 2-байтные числа. Соответственно нам нужно всего лишь 6 байт. Однако поскольку стек должен быть выровнен по 8 байтам, и соответственно выделенное пространство должно быть не меньше общего размера помещаемых элементов и также должно быть кратным 8, поэтому выделяем 24 байта, где 6 байт будут незаполненными. Далее помещаем данные через регистр AX в соответствующую область в стеке:

```
1 mov ax, 1
2 mov [rsp+16], ax
```

В итоге получится, что из 8 байт в стеке только младшие 2 байта будут содержать значимую информацию - число 1. Остальные 6 байт по сути будут содержать случайные значения, условно говоря, "мусор". Но поскольку при получении соответствующего значения из стека мы его получаем также в 2-байтный регистр

```
1 mov ax, [rsp+24] ; AX = 1
```

то проблем с интерпретацией значения или получением мусорных значений у нас не будет - мы опять же берем только 2 младших байта в регистр.

Определение параметров с директивой proc

В MASM директива proc позволяет сразу при определении процедуры определить и ее параметры в виде:

```
1 имя_процедуры proc параметр1:тип, параметр2:тип, ... параметрN:тип
```

После слова proc перечисляются через запятую параметры, где в каждом определении параметра указывается имя параметра и через двоеточие его тип. Параметры могут представлять различных стандартных типа - byte, word, dword и т.д. (массивы не допускаются), однако в стеке для каждого параметра в любом случае необходимо резервировать 8 байт:

```
1 .code
2 sum proc
3     xor rax, rax ; обнуляем регистр
4     mov ax, [rsp+24] ; AX = 1
5     add ax, [rsp+16] ; AX = AX + 2 = 3
6     add ax, [rsp+8] ; AX = AX + 3 = 6
7     ret
8 sum endp
9
10 main proc
11    sub rsp, 24 ; резервируем для параметров 24 байта
12    mov ax, 1
13    mov [rsp+16], ax
14    mov ax, 2
15    mov [rsp+8], ax
16    call sum
17    add rsp, 24
18    ret
19 main endp
20 end
```

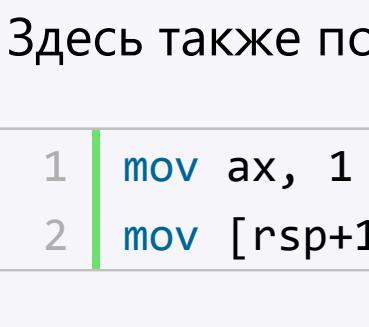
Процедура sum принимает три параметра - a, b и c, которые представляют типы byte, word и dword. При этом MASM автоматически связывает определенные значения из стека с этими параметрами. Это довольно удобно, поскольку нам можно не использовать регистр RSP и смещения для получения значений параметров.

Для передачи значений через стек нам надо учитывать, что для каждого параметра в стеке должно выделяться 8 байт, и что в стеке они должны располагаться в порядке, обратном определению. Так, последний параметр - c, который имеет тип dword (массивы не допускаются), однажды в стек помещается значение для этого параметра

```
1 mov eax, 8
2 mov [rsp+16], eax ; параметр c
```

Далее помещаем в стек значение для параметра b, и в конце - значение параметра a.

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



Помощь сайту

[Помощь сайту](#)

Юмани:

410011174743222

Номер карты:

4048415020898850

Телеграмм

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

Contacts: metanit22@mail.ru

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