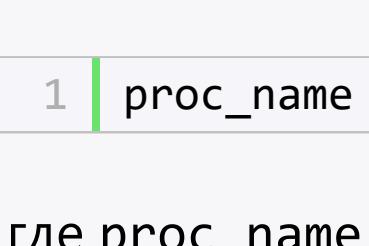




Процедуры

Определение и вызов процедур

Последнее обновление: 02.07.2023



Процедуры или функции в ассемблере позволяют разбить программу на подпрограммы, где каждая подпрограмма выполняет какой-то определенный набор действий.

Для определения процедуры применяется выражение

```
1 proc_name proc
```

где proc_name - имя процедуры.

Завершается определение процедуры выражением

```
1 proc_name endp
```

где proc_name - также имя процедуры.

Между proc_name proc и proc_name endp идет произвольный набор инструкций. Перед завершением процедуры помещается инструкция **ret**, которая передает управление в вызывающий код. Например:

```
1 .code
2 setReg proc      ; начало процедуры setReg
3     mov rax, 10
4     ret
5 setReg endp      ; конец процедуры setReg
6
7 main proc
8     ret
9 main endp
10 end
```

Здесь определены две процедуры - главная функция main и процедура setReg. В процедуре setReg устанавливается значение регистра rax. При компиляции подобной программы (допустим, она находится в файле hello.asm) с помощью команды

```
m164 hello.asm /link /entry:main
```

благодаря флагу /entry:main будет создавать файл, при запуске которого выполняется процедура с именем main. Но в нашем случае эта процедура пока ничего не делает, а процедура setReg автоматически не выполняется. Чтобы выполнить одну процедуру/функцию в другой, необходимо использовать инструкцию **call**, после которой указывается имя выполняемой процедуры:

```
1 call proc_name
```

Инструкция **call** помещает в стек 64-битный адрес инструкции, которая идет сразу после вызова. Значение, которое вызов помещает в стек, называется адресом возврата. Когда процедура завершает выполнение, для возвращения к вызывающему коду она выполняет инструкцию **ret**. Команда ret извлекает 64-битный адрес возврата из стека и косвенно передает управление на этот адрес.

Например, выполним процедуру setReg в функции main:

```
1 .code
2 setReg proc      ; начало процедуры setReg
3     mov rax, 10
4     ret
5 setReg endp      ; конец процедуры setReg
6
7 main proc
8     call setReg ; вызов процедуры setReg
9     ret
10 main endp
11 end
```

Вызываемые процедуры могут, в свою очередь, вызывать другие процедуры. Например:

```
1 .code
2 inner proc
3     add rax, 1
4     ret
5 inner endp
6
7 outer proc
8     call inner
9     add rax, 1
10    ret
11 outer endp
12
13 main proc
14     mov rax, 1
15     call outer
16     ret
17 main endp
18 end
```

Здесь функция main вызывает процедуру outer, а та вызывает процедуру inner. В итоге к завершению программы в регистре RAX будет число 3.

Стек и процедуры

При работе со стеком в процедурах следует учитывать, что вызов процедуры с помощью инструкции **call** помещает в стек адрес возврата. При завершении процедуры инструкция **ret** извлечет этот адрес возврата из стека и перейдет по этому адресу. Таким образом, выполнение вернется в код, где была вызвана процедура. Поэтому при вызове инструкции **ret** (при завершении процедуры) адрес возврата должен быть в верхушке стека.

Но при невнимательности это требование может быть нарушено. Например:

```
1 .code
2 sum proc
3     push rbx          ; сохраняем регистр RBX в стек
4     add rax, rbx
5     ret                ; регистр RBX НЕ восстанавливаем
6 sum endp
7
8 main proc
9     mov rax, 1
10    mov rbx, 2
11    call sum
12    ret
13 main endp
14 end
```

Здесь вызывается процедура sum, в которой в стек сохраняется регистр RBX. Однако в конце процедуры регистр RBX не восстанавливается. Поэтому в качестве адреса возврата будет рассматриваться значение регистра RBX, которое при вызове инструкции **ret** будет находиться в верхушке стека. В итоге поведение программы неопределено, и скорее всего она завершится ошибкой.

Другой пример - извлечение адреса возврата до завершения процедуры:

```
1 .code
2 sum proc
3     pop rbx           ; извлекаем данные из стека в регистр RBX
4     add rax, rbx
5     ret                ; адрес возврата неопределен
6 sum endp
7
8 main proc
9     mov rax, 1
10    mov rbx, 2
11    call sum
12    ret
13 main endp
14 end
```

Здесь в регистр RBX извлекаются данные из стека - по сути в него извлекается адрес возврата. В результате опять же поведение программы неопределено.

Поэтому процедура должна извлекать из стека все ранее сохраненные в ней данные и извлекать ровно столько, сколько было сохранено, чтобы адрес возврата сохранялся в стеке и к концу программы оказался в верхушке стека.

[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.