

Сохранение регистров и переменных при вызове процедур

Последнее обновление: 02.07.2023



Процедура может активно задействовать регистры для различных задач. Например:

```

1 .code
2 sum proc
3     mov rbx, 5
4     mov rax, 10
5     add rax, rbx
6     ret
7 sum endp
8
9 main proc
10    mov rbx, 125
11    call sum
12    add rax, rbx ; RAX = 20
13    ret
14 main endp
15
16 end

```

Для примера здесь определена процедура sum, которая устанавливает регистры RBX и RAX и складывает их значения, помещая результат в регистр RAX.

В функции main помещаем в регистр RBX число 125, затем вызываем процедуру sum. Далее результат процедуры sum - содержимое регистра RAX складываем с числом из регистра RAX. Однако, поскольку вызов процедуры sum изменил значение регистра RBX, то в после ее вызова в этом регистре будет не число 125, а число 5. Соответственно итоговый результат может отличаться от ожидаемого. В данном случае в регистре RAX будет число 20, а не 140.

Конечно, мы могли бы выбрать разные регистры, но данный пример - упрощение, в реальной процедуре одновременно может быть задействовано для сложных вычислений множество регистров. Кроме того, разрабатываемые процедуры могут использоваться во внешней программе, и при разработке процедуры мы можем не знать, какие именно регистры и как именно будет использовать внешняя программа. Поэтому хорошей, а иногда и необходимой практикой при вызове процедуры является сохранение значений используемых регистров в стек, а при завершении процедуры - их восстановление. Например:

```

1 .code
2 sum proc
3     push rbx ; сохраняем в стек регистр rbx
4     mov rbx, 5
5     mov rax, 10
6     add rax, rbx
7     pop rbx ; восстанавливаем из стека регистр rbx
8     ret
9 sum endp
10
11 main proc
12    mov rbx, 125
13    call sum
14    add rax, rbx ; RAX = 140
15    ret
16 main endp
17
18 end

```

В данном случае в начале процедуры sum сохраняем значение rbx, а при завершении восстанавливаем его. Подобным образом стоит сохранять и восстанавливать все используемые регистры в стек. Регистр RAX в данном случае не сохраняется, так как мы ожидаем, что через него функция main получить результат процедуры sum.

При этом в принципе необязательно сохранять регистры в вызываемой процедуре, мы могли бы это сделать и в вызывающей процедуре:

```

1 .code
2 sum proc
3     mov rbx, 5
4     mov rax, 10
5     add rax, rbx
6     ret
7 sum endp
8
9 main proc
10    mov rbx, 125
11    push rbx ; сохраняем в стек регистр rbx
12    call sum
13    pop rbx ; восстанавливаем из стека регистр rbx
14    add rax, rbx
15    ret
16 main endp
17
18 end

```

Однако сохранение регистров в вызываемой процедуре может быть более предпочтительным, так как функция main может вызывать кучу других процедур и постоянно сохранять/восстанавливать регистры при каждом вызове значительно утяжелит код функции main.

Стоит отметить, что при взаимодействии с внешними API может потребоваться в обязательном порядке сохранять данные некоторых регистров. Так, если мы взаимодействуем с Windows API, то есть общие соглашения, что при вызове функций WinAPI необходимо хранить в стеке, а после вызова восстанавливать регистры RBX, RBP, RDI, RSI, RSP, R12, R13, R14 и R15, а также XMM6 - XMM15.

Сохранение и восстановление переменных

Все, что касается регистров, также можно отнести и к глобальным переменным, которые используются в процедурах:

```

1 .data
2     num word 45
3 .code
4 sum proc
5     mov num, 15
6     mov rax, 10
7     add ax, num ; AX = AX + num = 10 + 15 = 25
8     ret
9 sum endp
10
11 main proc
12     call sum
13     add ax, num ; AX = AX + num = 25 + 15 = 40
14     ret
15 main endp
16
17 end

```

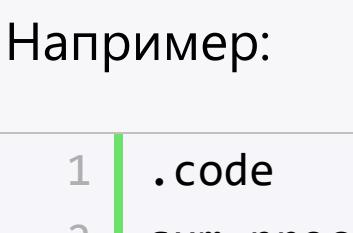
В данном случае в функции main мы ожидаем, что переменная num будет равна 45, но вызов процедуры sum меняет ее значение на 45. И подобным образом в вызываемой процедуре переменную можно сохранить в стек:

```

1 .data
2     num word 45
3 .code
4 sum proc
5     push num ; сохраняем значение num в стек
6     mov num, 15
7     mov rax, 10
8     add ax, num ; AX = AX + num = 10 + 15 = 25
9     pop num ; восстанавливаем num из стека
10    ret
11 sum endp
12
13 main proc
14     call sum
15     add ax, num ; AX = AX + num = 25 + 45 = 70
16     ret
17 main endp
18
19 end

```

[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.