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 переменные. Можно комбинировать различные подходы.

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

```
.code
; Процедура sum принимает два параметра
; RCX - первое число
; RDX - второе число
sum proc
    mov rax, rcx    ; в RAX копируем число из RCX
    add rax, rdx    ; складываем с числом из RDX
    ret
sum endp

main proc
    mov rcx, 3      ; первый параметр для процедуры sum
    mov rdx, 4      ; второй параметр для процедуры sum
    call sum
    ret
main endp
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, какие именно регистры будут передавать свои параметры в процедуру. Однако если мы задействуем какие-то внешние функционал, например, взаимодействуем с функциями C/C++, то нам придется также применять те условности, которые накладывают эти функции и конкретные ОС. В частности, в вызов 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:

```
.code
sum proc
    mov rax, [rsp+24]   ; RAX = 1
    add rax, [rsp+16]   ; RAX = RAX + 2 = 3
    add rax, [rsp+8]    ; RAX = RAX + 3 = 6
    ret
sum endp

main proc
    push 1
    push 2
    push 3
    call sum
    add rsp, 24
    ret
main endp
end
```

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

```
push 1
push 2
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

```
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:

```
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 избыточно. 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:

```
.code
sum proc
    xor rax, rax        ; обнуляем регистр
    mov ax, [rsp+14]    ; RAX = 2
    add ax, [rsp+12]    ; RAX = RAX + 4 = 6
    add ax, [rsp+10]    ; RAX = RAX + 6 = 12
    ret
sum endp

main proc
    sub rsp, 8  ; резервируем для параметров 8 байт
    mov ax, 2
    mov [rsp+6], ax
    mov ax, 4
    mov [rsp+4], ax
    mov ax, 6
    mov [rsp+2], ax
    call sum
    add rsp, 8  ; восстанавливаем указатель стека
    ret
main endp
end
```

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

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

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

```
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 момент работы с C/C++ с требуется, чтобы каждый элемент в стеке все также было равен 8 байтам, даже несмотря на то, что в реальности ему достаточно места. Например:

```
.code
sum proc
    xor rax, rax        ; обнуляем регистр
    mov ax, [rsp+24]    ; AX = 1
    add ax, [rsp+16]    ; AX = AX + 2 = 3
    add ax, [rsp+8]     ; AX = AX + 3 = 6
    ret
sum endp

main proc
    sub rsp, 24  ; резервируем для параметров 24 байта
    mov ax, 1
    mov [rsp+16], ax
    mov ax, 2
    mov [rsp+8], ax
    mov ax, 3
    mov [rsp], ax
    call sum
    add rsp, 24
    ret
main endp
end
```

Здесь также по сути помещаем в стек 2-байтные числа:

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

В итоге получаются, 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

```
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:

```
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 различные стандартные типа - byte, word, dword и т.д, массивы не допускаются), однако в стеке для каждого параметра в любом случае необходимо резервировать 8 байт:

```
.code
sum proc a:byte, b:word, c:dword
    mov eax, c          ; EAX = 8
    add eax, dword ptr b    ; EAX = EAX + 4 + 12
    add eax, dword ptr a    ; EAX = EAX + 2 + 14
    ret
sum endp

main proc
    sub rsp, 24  ; резервируем для параметров 24 байта
    mov eax, 8
    mov [rsp+16], eax   ; параметр c
    mov ax, 4
    mov [rsp+8], ax     ; параметр b
    mov al, 2
    mov [rsp], al       ; параметр a
    call sum
    add rsp, 24
    ret
main endp
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. Поэтому вначале в стек помещаем значение для этого параметра

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

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

Назад Содержание Вперед