Практика 07: процедуры.

Ранее был рассмотрен один из способов группировки команд – макросы. Недостаток макросов в том, что вызов макроса означает команду для компилятора подставить вместо имени макроса его содержимое. Таким образом происходит многократное копирование одинакового кода по ходу приложения, что приводит к разрастанию программы. Чтобы избежать этого необходимо не копировать код, а обращаться к нему из разных мест. Для этого существует механизм процедуры.

Процедура представляет собой код, который может выполняться многократно и к которому можно обращаться из разных частей программы. Обычно процедуры предназначены для выполнения каких-то отдельных, законченных действий программы и поэтому их иногда называют подпрограммами. В других языках программирования процедуры могут называться функциями или методами, но по сути это всё одно и то же. Для работы с процедурами предназначены две команды

- CALL
- RET

С помощью команды **CALL** выполняется **вызов процедуры**. Эта команда работает почти также, как команда безусловного перехода (JMP), но с одним отличием — одновременно в стек сохраняется текущее значение регистра IP. Это позволяет потом вернуться к тому месту в коде, откуда была вызвана процедура. В качестве операнда указывается адрес перехода, который может быть

непосредственным значением (меткой), 32-ух или 64-разрядным регистром (кроме сегментных) в зависимости от используемой архитектуры или ячейкой памяти, содержащей адрес.
Возврат из процедуры выполняется командой **RET**. Эта команда восстанавливает значение из вершины стека в регистр IP. Таким образом, выполнение программы продолжается с команды, следующей сразу после команды CALL. Обычно код процедуры заканчивается этой командой. Команды CALL и RET не изменяют значения флагов (кроме некоторых особых случаев в защищенном режиме).
Процедуры размещаются в любом сегменте кода приложения. Размещение должно быть таким, чтобы не произошло произвольного перехода в процедуру. Например, можно разместить процедуру до точки старта кода, или после системного вызова ExitProcess. Или обойти код процедуры безусловным переходом.
Существует 2 типа вызовов процедур:

- **ближним** называется **вызов процедуры**, которая находится в текущем сегменте кода
- **дальний вызов** — это вызов процедуры в другом сегменте.

Соответственно существуют 2 вида команды RET — для ближнего и дальнего возврата. Компилятор FASM автоматически определяет нужный тип машинной команды, поэтому в большинстве случаев не нужно об этом беспокоиться.
Приведем пример процедуры, которая ничего не делает (опустим не относящиеся к примеру части программы).

<..>
Call myproc
<..>
Push 0
Call [ExitProcess]
Myproc:
Nop
Ret
<..>

Очень часто возникает необходимость передать процедуре какие-либо параметры. Самый простой способ передать параметры — это поместить их в регистры перед вызовом процедуры.
Кроме передачи параметров часто нужно получить какое-то значение из процедуры. Существуют разные способы возврата значения из процедуры, но самый часто используемый — это поместить значение в один из регистров. Обычно для этой цели используют регистр EAX, хотя можно использовать любой.

Задание :

Сделать процедуру смены вывода цвета текста. Для передачи цвета использовать регистр EDX.

Решение:
<..>
mov edx,0x0006
Call setcolor
<..>

```
setcolor:
    push STD_OUTPUT_HANDLE
    call [GetStdHandle]
    push edx
    push eax
    call [SetConsoleTextAttribute]
    ret
```

Хорошим тоном считается сохранение регистров, которые процедура изменяет в ходе своего выполнения. Это позволяет вызывать процедуру из любой части кода и не беспокоиться, что значения в регистрах будут испорчены. Обычно регистры сохраняются в стеке с помощью команды PUSH, а перед возвратом из процедуры восстанавливаются командой POP. Естественно, восстанавливать их надо в обратном порядке. Примерно вот так:

```
setcolor:
    push eax
    push STD_OUTPUT_HANDLE
    call [GetStdHandle]
    push edx
    push eax
    call [SetConsoleTextAttribute]
    pop eax
Ret
```

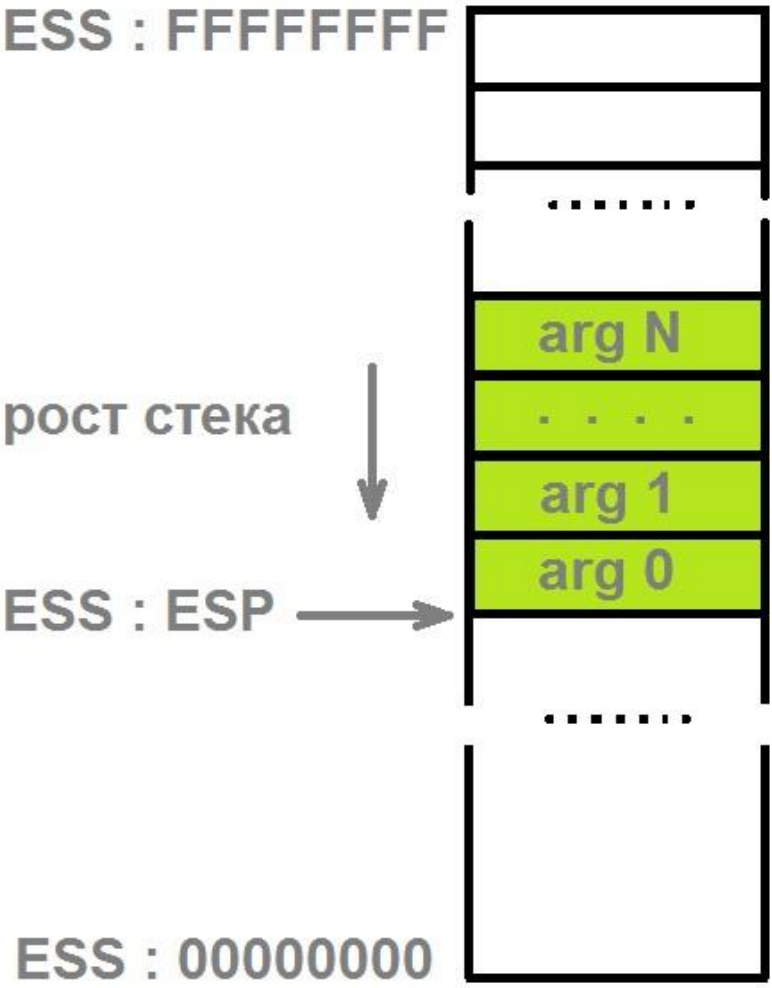
Параметры процедуры через стэк.

Передать большое число параметров, или информации в процедуру через регистры нельзя. Зато через стек передать можно практически сколько угодно. Однако, обращение к параметрам в стеке происходит медленнее. Если вы оптимизируете программу по скорости выполнения, то имеет смысл передавать параметры через регистры.

Перед вызовом процедуры параметры необходимо поместить в стек с помощью команды PUSH. Здесь существует два варианта: параметры могут помещаться в стек в прямом или в обратном порядке. Обычно используется обратный порядок: параметры помещаются в стек, начиная с последнего, так что перед вызовом процедуры на вершине стека оказывается первый параметр. Например:

```
; Данные
arg0    dw 0
arg1    dw 12
...
argN    dw 345
; Код
push [argN]
push ...
push [arg1]
push [arg0]
call myproc
```

Перед выполнением команды CALL стек будет иметь следующий вид:

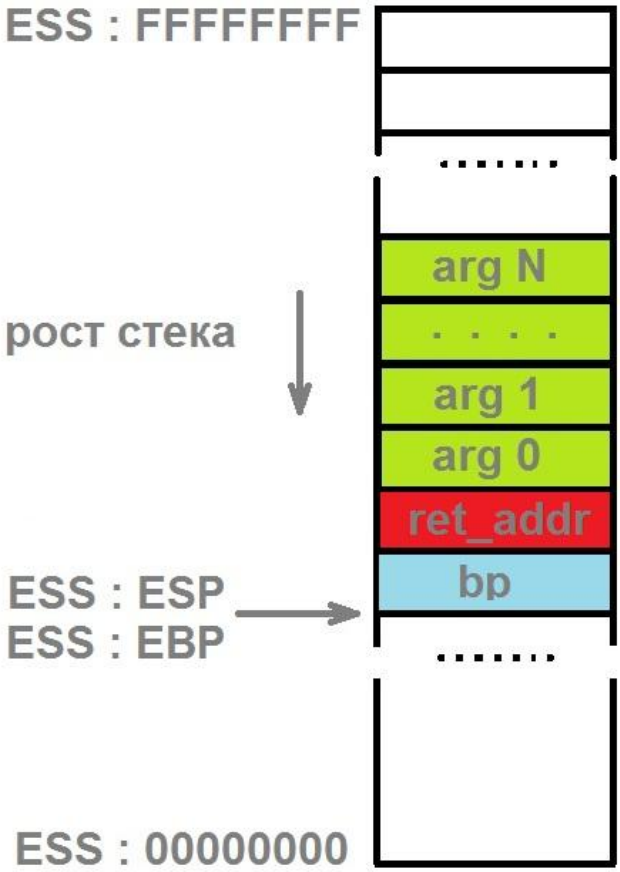


Для обращения к параметрам внутри процедуры обычно используют регистр EBP. В самом начале процедуры содержимое регистра EBP сохраняется в стеке и в него копируется значение

регистра ESP. Это позволяет «запомнить» положение вершины стека и адресовать параметры относительно регистра EBP. Кстати, подобное соглашение об обращении к параметрам процедуры является еще одним поводом не рекомендовать использовать стек для хранения переменных программы.

```
;Процедура
myproc:
   push ebp
   mov ebp,esp
   …
```

При выполнении кода процедуры стек будет иметь следующую структуру:



Здесь ret_addr обозначает адрес возврата, помещаемый в стек командой вызова процедуры, а ebp — сохранённое значение регистра EBP. В нашем случае стек имеет ширину 32 бит, поэтому первый параметр будет доступен как word[ebp+8], второй как word[ebp+12] и так далее.

Mov eax,[ebp+8]
Mov ebx,[epb+12]
...

После того, как процедура выполнилась, необходимо очистить стек, вытолкнув из него параметры. Тут тоже существует 2 способа:

- стек может быть очищен самой процедурой
- стек может быть очищен кодом, который эту процедуру вызывал

Для первого способа используется команда RET с одним операндом, который должен быть равен количеству байтов, выталкиваемых из стека. В нашем случае он должен быть равен количеству параметров, умноженному на 4.

Ret 8; для процедуры с 2 параметрами

Для второго способа нужно использовать команду RET без операндов. Стек восстанавливается после выполнения процедуры путём прибавления значения к ESP. С помощью такого способа программируются процедуры с переменным количеством параметров. Процедура не знает, сколько ей будет передано параметров, поэтому очистка стека должна выполняться вызывающим кодом.

call myproc
add esp,8 ; для процедуры с 2 параметрами в 32bit

Совокупность таких особенностей, как способ и порядок передачи параметров, механизм очистки стека, сохранение определённых

регистров в процедуре и некоторых других называется соглашениями вызова. Соблюдение этих соглашений является важным при вызове из программы компонентов, написанных на других языках программирования, или функций ОС, или при предусматривании вызова процедур в других программах. В остальных случаях можно не соблюдать соглашения.

Задание:

Переписать процедуру установки цвета. Параметр передавать через стек.

Решение:

setcolor:

```
push ebp
mov ebp,esp
push eax
push ecx
push STD_OUTPUT_HANDLE
call [GetStdHandle]
mov ecx,[ebp+8]
push ecx
push eax
call [SetConsoleTextAttribute]
pop ecx
pop eax
pop ebp
ret 4
```

Задание:

Написать процедуру вывода текста. Строку передавать через стек, длину высчитывать.

Способы вызова процедуры.

Команда call по сути своей аналогична jmp, то есть осуществляет переход по указанному адресу памяти. Существует несколько способов указания адреса вызываемой процедуры:

- по метке процедуры
- по адресу в регистре
- по адресу в переменной
- прочее

Вызов по метке мы и использовали

call myproc

Вызов по адресу в регистре

mov eax,myproc
call eax

Вызов по адресу в переменной

.data
myprocaddr DD myproc
.text
call [myprocaddr]

Эту конструкцию мы используем при вызове системных вызовов. Объясняется это просто — внутри включаемых файлов (мы подключаем win32ax.inc, остальные модули подключаются уже внутри данного) есть переменные, которые хранят адреса процедур системных вызовов ядра.
Прочие варианты вызова — например, задание адреса через регистр и смещение - **EAX+ESI**) используются редко.

Задание :

*написать рекурсивную процедуру, выводящую на экран «ёлочку» из символов *. Цветовой атрибут строки равен количеству символов в строке+1. Например для основания «ёлочки» равного 5 вывод будет выглядеть так:*

*****
****
***
**
*

Домашнее задание:

написать процедуры вывода строки на экран и ввода строки и числа с клавиатуры.