Практика 08: процедуры, часть 2. Директива Include.

Обращение к параметрам, указывая смещения относительно регистра BP, хоть и естественный для программиста на ASM способ, но не совсем удобный. Поэтому в таких ассемблерах, как TASM и MASM, существуют специальные директивы, позволяющие создавать процедуры быстро и удобно. В FASM таких директив нет! Но они и не нужны — то же самое можно сделать с помощью макросов.
Макросы находятся в файле proc32.inc, который уже подключен к приложению (он подключается в win32a.inc, который в свою очередь подключается в win32ax.inc, который мы подключили в явном виде).
Для создания процедуры используется следующий синтаксис:

```
proc < имя_процедуры>[,][< список_параметров>]
   ...
   ret
endp
```

После proc указывается имя процедуры. Далее через запятую список параметров. Между именем процедуры и списком параметров запятую ставить не обязательно (можно просто поставить пробел).
Для возврата из процедуры следует использовать команду RET без операндов. Завершается процедура макросом endp.
Например, объявим процедуру с тремя параметрами:

```
proc myproc a,b,c
   mov ax,[b]
   ...
   ret
endp
```

Внутри процедуры обращаться к параметрам можно как к простым переменным — с помощью квадратных скобок.

По умолчанию размер параметров считается равным ширине стека, то есть в нашем случае 32-бит. Если требуется передавать процедуре байт или двойное слово, то нужно дополнительно указать размер. Это можно сделать, поставив двоеточие после имени параметра.
Варианты возможны следующие:

- BYTE – 1 байт
- WORD – 2 байт
- DWORD – 4 байт
- PWORD – 6 байт
- QWORD – 8 байт
- TBYTE – 10 байт
- DQWORD – 16 байт

Приведенный выше пример может трансформироваться в следующее:

```
proc myproc a:BYTE,b:DWORD,c:DWORD
   mov ax,[b]
   ...
   ret
endp
```

Дополнительно после имени процедуры можно указать тип соглашения вызова — кто отвечает за очистку стека при вызове процедуры

- stdcall – очищает процедура
- c – очищает вызывающий код

указывается редко, обычно оставляют значением по умолчанию stdcall.
Макросы PROC и ENDP позволяют также организовать сохранение и восстановление регистров, используемых кодом процедуры. Для этого после имени процедуры нужно указать ключевое слово uses и список регистров через пробел. Регистры будут помещены в стек при входе в процедуру (в порядке их записи) и восстановлены перед возвратом.
Например, добавим сохранение регистров к нашей процедуре:

```
proc myproc2 c uses ax bx cx dx,a:BYTE,b:WORD,c:DWORD
   mov cl,[a]
   mov bx,[b]
   mov ax,word[c]
   mov dx,word[c+2]
   ...
   ret
endp
```

Если объявление процедуры получается слишком длинным, можно продолжить его на следующей строке, добавив символ \ в конец первой строки (это работает и с любыми другими макросами):

```
proc myproc4 c uses ax bx cx dx,\
        a:BYTE,b:WORD,c:DWORD
```

Вызов процедуры осуществляется один из следующих макросов

- stdcall
- invoke
- ccall
- cinvoke

Первые две аналогичны и рассчитаны на формат вызова stdcall, а вторые две на формат c. Таким образом, наша процедура должна вызываться следующим образом

```
stdcall myproc,10,20,30
```

напомним, что с аналогичным макросом invoke мы сталкивались на первой практике. Использовать же будем stdcall.

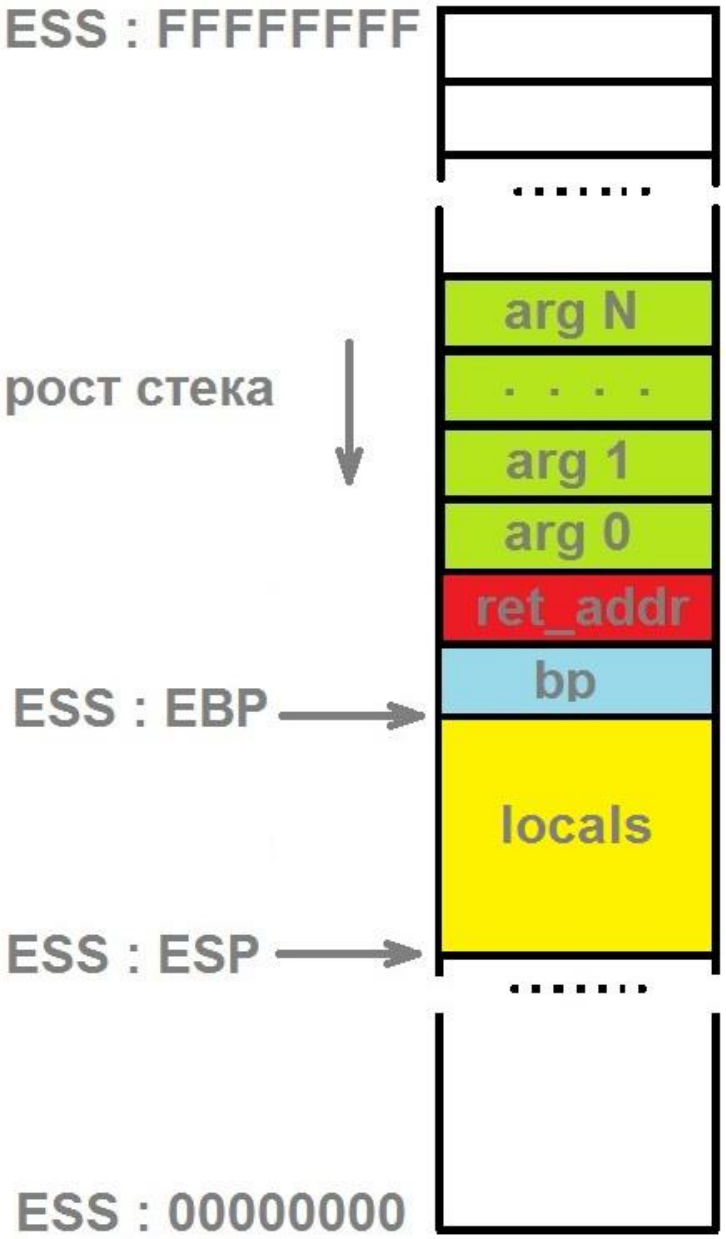
Задание 1:
переписать процедуру установки цвета, используя макросы PROC и ENDP

Локальные переменные.

Чтобы создать локальные переменные в процедуре, необходимо выделить для них память. Эта память выделяется в стеке. Для этого достаточно вычесть из регистра ESP значение, равное суммарному размеру всех локальных переменных в процедуре. Так как ширина стека равна 32 бит, то это значение должно быть кратно 4 байтам. При выходе из процедуры нужно восстановить указатель стека. Обычно это выполняется командой mov esp,ebp (В ebp сохраняется значение sp при входе в процедуру, как в случае с параметрами, передаваемыми через стек). Код процедуры с локальными переменными будет выглядеть следующим образом:

```
proc myproc
    push bp
    mov bp,sp
    sub sp,locals_size
    ...
    mov sp,bp
    pop bp
    ret
endp
```

После выполнения начальных команд процедуры — называемых прологом — стек выглядит следующим образом :



Locals это область под локальные переменные. В следующем примере выделяется две локальные переменные по 4 байта :

```
proc myproc
    push ebp
    mov ebp,esp
    sub esp,8
    ...
    mov dword[ebp-8],1234
    mov dword[ebp-4],10
    ...
    mov esp,ebp
    pop ebp
    ret
endp
```

После выполнения кода пролога первая локальная переменная будет находиться по адресу ebp-8, а вторая по адресу ebp-4. Обратите внимание, что стековые переменные должны быть явно инициализированы. Так как память выделяется в стеке, то изначально в них будет всякий мусор (а вовсе не нули).
Для локальных переменных существует понятие области видимости — область программы, в которой доступна переменная. Обычно в ассемблере область видимости ограничена процедурой, создавшей локальную переменную. Рассмотренный способ размещения локальных переменных тоже не очень удобен и нагляден. В файле proc32.inc предусмотрено несколько макросов для объявления локальных переменных. С их помощью есть три варианта объявления локальных переменных

1. local + директивы объявления данных
2. local + операторы размера
3. locals + endl

В первом случае объявление локальных переменных выглядит так

```
local x dw 10
local y dw ?, z dd 128
```

Во втором случае так

local x:WORD
local y WORD, z DWORD

в этом случае не предусмотрено объявление значений, но зато легко создаются массивы

local buffer[256]:BYTE

что означает объявление переменной buffer с размером 256 байт
В третьем случае так

locals
x DW 10
y DW ?
Z DD 128
endl

используются только стандартные директивы объявления переменных.
Технически переменные всё так же хранятся в стеке и получают память кратно 8 байт.
Обращение к переменным внутри процедуры происходит также, как к параметрам и глобальным переменным. Например:

mov ax,[x]

Директива Include.

Директива include позволяет включать на этапе компиляции содержимое другого файла в программу. Вызвать директиву можно в любом месте программы, например:
в файл write.asm запишем код вывода строки

push STD_OUTPUT_HANDLE
call [GetStdHandle]

push 0
push outputnumber
push 255
push outputstr
push eax
call [WriteConsole]

а в самой программе вместо кода вывода строки включаем файл

include write.asm

Но обычно в включаемых файлах записывают макросы или процедуры.

Задание 3:
создать свой файл endprogram.inc, содержащий процедуру endprogram, Процедура выводит надпись Press any key… , ждёт нажатия 1 клавиши и завершает работу приложения.

Домашнее задание:
создать свой файл myprocs.inc, содержащий написанные ранее процедуры вывода строки на экран, ввода строки и числа с клавиатуры.