https://clck.ru/9Khfe

Пятиминутка

1. Изначально ZF, CF и OF имеют значение 0.

   Какие значения будут у флагов ZF, CF, OF после исполнения этого кода?

```
mov eax, 167
mov ecx, 24
cmp eax, ecx
shl eax, cl
```

2. Изначально eax имеет значение 0.

Выберите фрагменты кода, после выполнения которых в eax окажется 32

2.1.
```
inc eax
shl eax, 5
```

2.2.
```
inc eax
inc eax
mov ebx, 5
mul ebx
```

2.3.
```
mov eax, 0x20
```

2.4.
```
lea eax, [eax+32]
```

2.5.
```
dec eax
xor eax, 0x20
not eax
```

2.6.
```
inc eax
inc eax
lea eax, [eax+eax*8-2]
```

2.7.
```
inc eax
test eax, eax
jnz continue
shl eax, 5
continue:
```

3. Опишите, что делает код:

```
; В esi находится указатель на строку в конце которой находится
нулевой байт
cld ; Устанавливает флаг DF в 0
loop:
  lodsb
  test al, al
  jnz loop
std ; Устанавливает флаг DF в 1
dec esi
dec esi
loop2:
  lodsb
  cmp al, ' '
  je loop2
inc esi
inc esi
mov byte ptr [esi], 0
```

Стек и вызов функций

Мельник Богдан

Стек

Стек (англ. *stack* — стопка; читается *стэк*) — структура данных, представляющая собой список элементов, организованных по принципу *LIFO* (англ. *last in — first out*, «последним пришёл — первым вышел»).

Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно снять верхнюю.

Аппаратная поддержка стека

Регистры

SS (Stack Segment) — сегментный регистр, в зависимости от режима адресации в нём могут находиться разные значения либо адрес, либо дескриптор сегмента памяти.
Обычно равен DS и ES.

В паре с регистром ESP (Stack Pointer) однозначно устанавливают местоположение в памяти верхушки стека.

Регистры

ESP 0007FFC4

0007FFBC	00000009	
0007FFC0	0007FFF8	
0007FFC4	7C81776F	RETURN to kernel32.7C81776F
0007FFC8	7C910228	ntdll.7C910228
0007FFCC	FFFFFFFF	
0007FFD0	7FFD5000	
0007FFD4	80544DFD	
0007FFD8	0007FFC8	
0007FFDC	85F182A8	
0007FFE0	FFFFFFFF	End of SEH chain
0007FFE4	7C839AB0	SE handler
0007FFE8	7C817778	kernel32.7C817778
0007FFEC	00000000	
0007FFF0	00000000	
0007FFF4	00000000	
0007FFF8	0100739D	NOTEPAD.<ModuleEntryPoint>

PUSH

PUSH — **СНАЧАЛА** вычитает ESP на размер двойного слова и **ЗАТЕМ** устанавливает в нужное значение SS:[ESP].

PUSH

0007FFBC	00000009
0007FFC0	0007FFF8
0007FFC4	7C81776F
0007FFC8	7C910228
0007FFCC	FFFFFFFF
0007FFD0	7FFD5000
0007FFD4	80544DFD
0007FFD8	0007FFC8
0007FFDC	85F182A8
0007FFE0	FFFFFFFF
0007FFE4	7C839AB0
0007FFE8	7C817778
0007FFEC	00000000
0007FFF0	00000000
0007FFF4	00000000
0007FFF8	0100739D

push 0x12345678

0007FFBC	00000009
0007FFC0	12345678
0007FFC4	7C81776F
0007FFC8	7C910228
0007FFCC	FFFFFFFF
0007FFD0	7FFD5000
0007FFD4	80544DFD
0007FFD8	0007FFC8
0007FFDC	85F182A8
0007FFE0	FFFFFFFF
0007FFE4	7C839AB0
0007FFE8	7C817778
0007FFEC	00000000
0007FFF0	00000000
0007FFF4	00000000
0007FFF8	0100739D

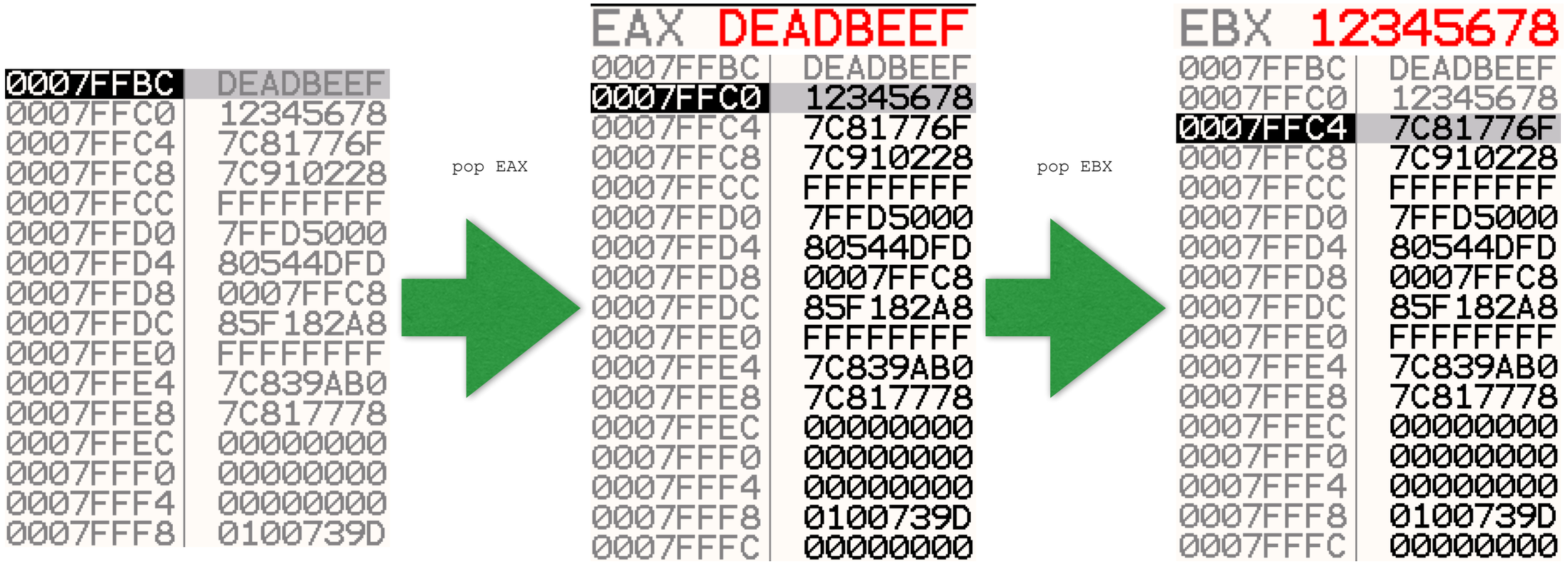
push 0xDEADBEEF

0007FFBC	DEADBEEF
0007FFC0	12345678
0007FFC4	7C81776F
0007FFC8	7C910228
0007FFCC	FFFFFFFF
0007FFD0	7FFD5000
0007FFD4	80544DFD
0007FFD8	0007FFC8
0007FFDC	85F182A8
0007FFE0	FFFFFFFF
0007FFE4	7C839AB0
0007FFE8	7C817778
0007FFEC	00000000
0007FFF0	00000000
0007FFF4	00000000
0007FFF8	0100739D

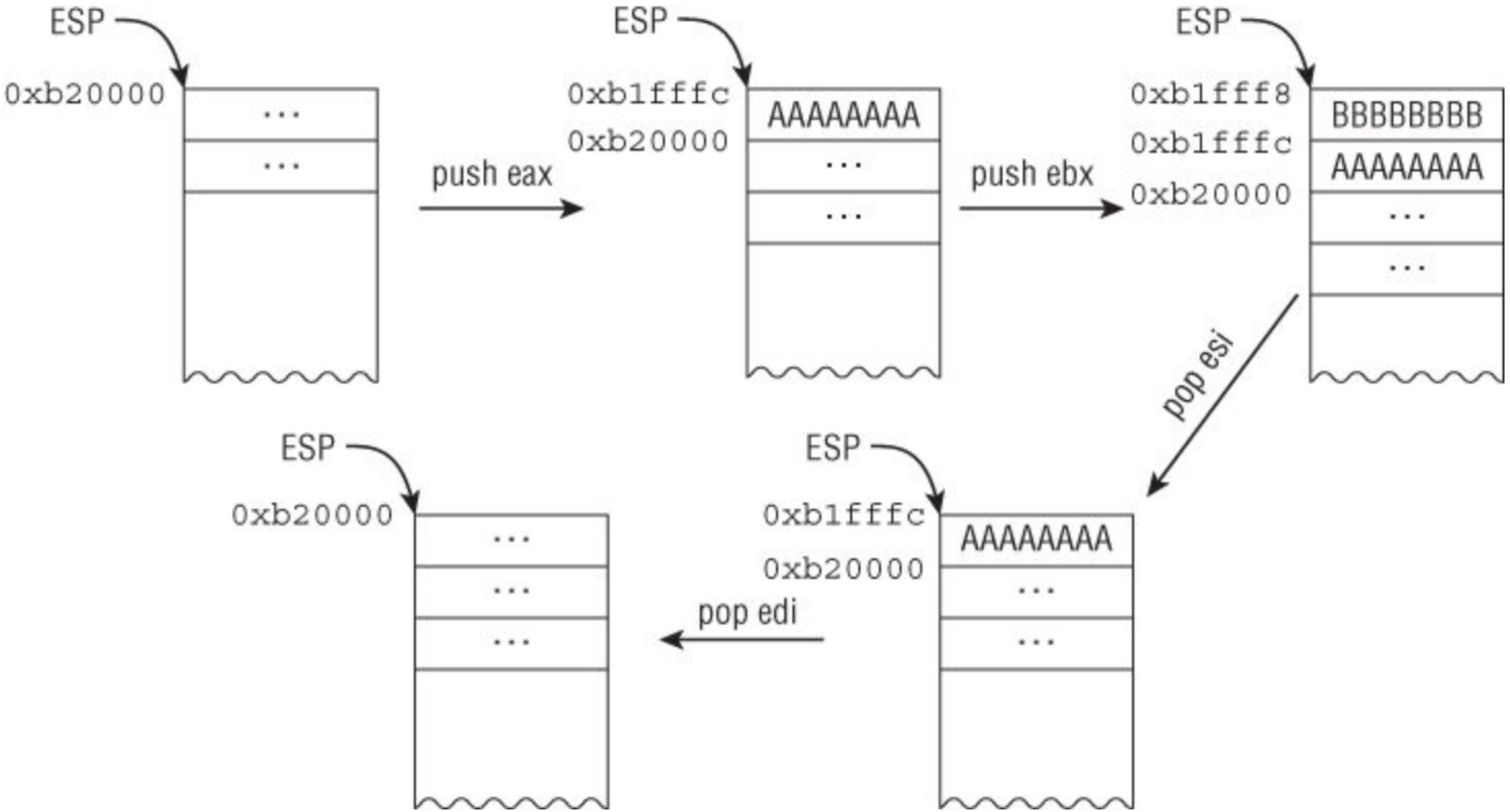
POP

POP — **СНАЧАЛА** возвращает значение SS:[ESP] и **ЗАТЕМ** увеличивает ESP на размер двойного слова.

POP


0007FFBC DEADBEEF
0007FFC0 12345678
0007FFC4 7C81776F
0007FFC8 7C910228
0007FFCC FFFFFFFF
0007FFD0 7FFD5000
0007FFD4 80544DFD
0007FFD8 0007FFC8
0007FFDC 85F182A8
0007FFE0 FFFFFFFF
0007FFE4 7C839AB0
0007FFE8 7C817778
0007FFEC 00000000
0007FFF0 00000000
0007FFF4 00000000
0007FFF8 0100739D
pop EAX
EAX DEADBEEF
0007FFBC DEADBEEF
0007FFC0 12345678
0007FFC4 7C81776F
0007FFC8 7C910228
0007FFCC FFFFFFFF
0007FFD0 7FFD5000
0007FFD4 80544DFD
0007FFD8 0007FFC8
0007FFDC 85F182A8
0007FFE0 FFFFFFFF
0007FFE4 7C839AB0
0007FFE8 7C817778
0007FFEC 00000000
0007FFF0 00000000
0007FFF4 00000000
0007FFF8 0100739D
0007FFFC 00000000
pop EBX
EBX 12345678
0007FFBC DEADBEEF
0007FFC0 12345678
0007FFC4 7C81776F
0007FFC8 7C910228
0007FFCC FFFFFFFF
0007FFD0 7FFD5000
0007FFD4 80544DFD
0007FFD8 0007FFC8
0007FFDC 85F182A8
0007FFE0 FFFFFFFF
0007FFE4 7C839AB0
0007FFE8 7C817778
0007FFEC 00000000
0007FFF0 00000000
0007FFF4 00000000
0007FFF8 0100739D
0007FFFC 00000000

Ещё пример



Картинка взята из Practical Reverse Engineering

Вызов функций

Одного JMP/JCC для вызова функций нам недостаточно.

Почему?

Вызов функций

Одного JMP/JCC для вызова функций нам недостаточно.

Чтобы вернуться из функции нам нужен второй JMP :)

(На самом деле нет)

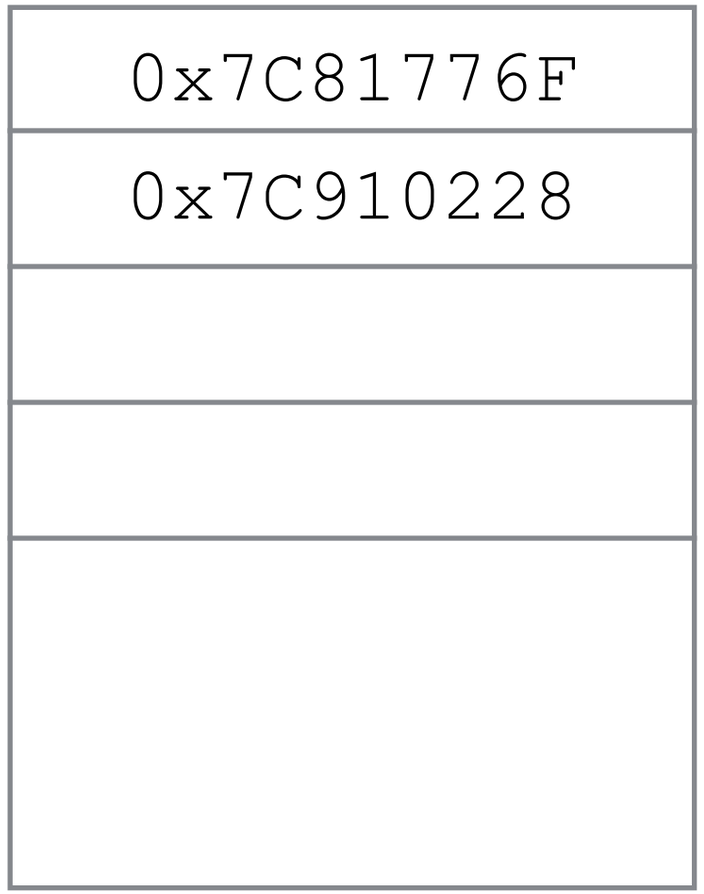
Вызов функций

Для вызова функций было придумано несколько инструкций.

Самые важные это **CALL** и **RET**

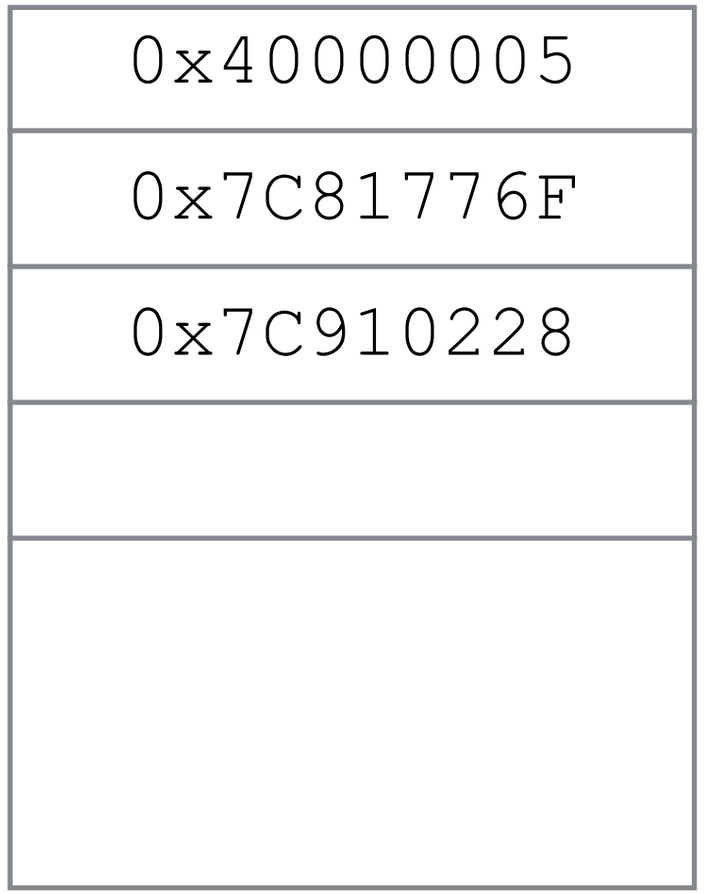
CALL и RET

```
0x40000000 call foo
0x40000005 mov ebx, 1

0x40001000 foo:
0x40001000 mov eax, 1
0x40001005 ret
```



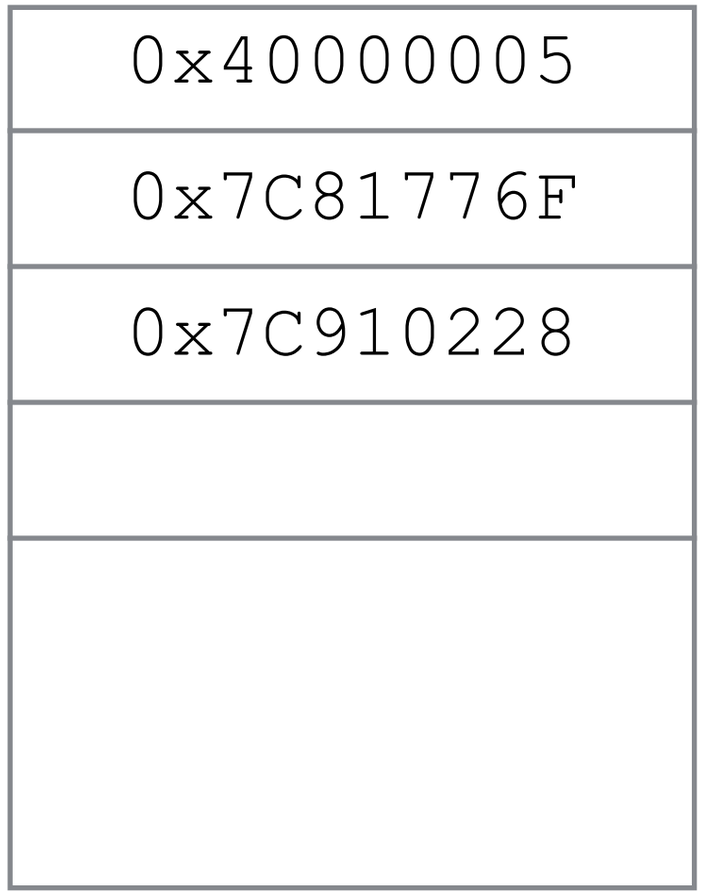
CALL и RET

```
0x40000000 call foo
0x40000005 mov ebx, 1

0x40001000 foo:
0x40001000 mov eax, 1
0x40001005 ret
```



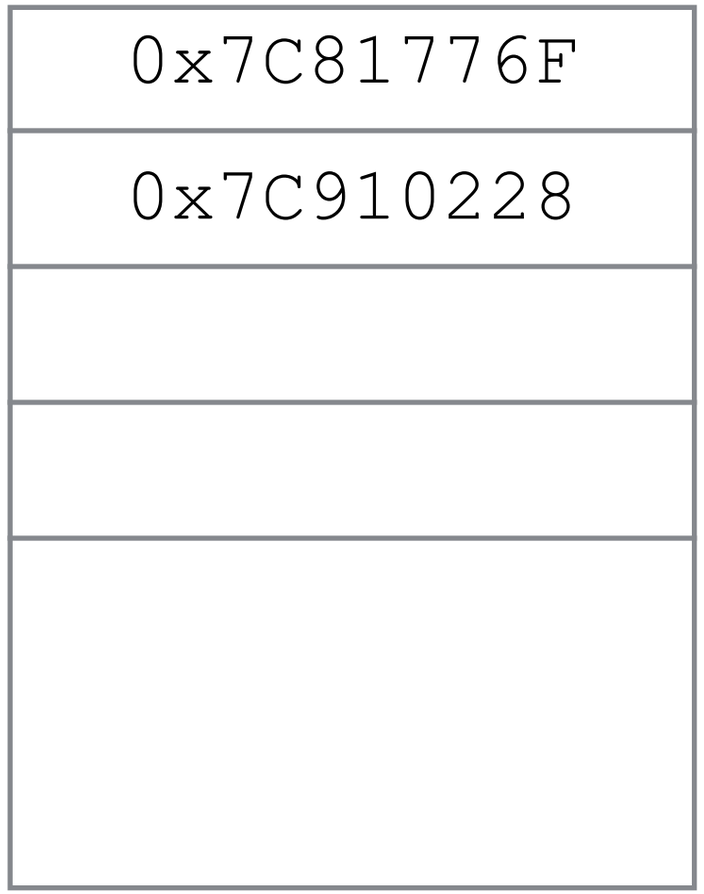
CALL и RET

```
0x40000000 call foo
0x40000005 mov ebx, 1

0x40001000 foo:
0x40001000 mov eax, 1
0x40001005 ret
```



CALL и RET

```
0x40000000 call foo
0x40000005 mov ebx, 1

0x40001000 foo:
0x40001000 mov eax, 1
0x40001005 ret
```



Передача аргументов

Самый простой и логичный способ передавать аргументы через регистры, но это не всегда возможно.

Передача аргументов

	CDECL	STDCALL	FASTCALL
Parameters	Pushed on the stack from right-to-left. Caller must clean up the stack after the call.	Same as `CDECL` except that the callee must clean the stack.	First two parameters are passed in `ECX` and `EDX`. The rest are on the stack.
Return value	Stored in `EAX`.	Stored in `EAX`.	Stored in `EAX`.
Non-volatile registers	`EBP`, `ESP`, `EBX`, `ESI`, `EDI`.	`EBP`, `ESP`, `EBX`, `ESI`, `EDI`.	`EBP`, `ESP`, `EBX`, `ESI`, `EDI`.

Картинка взята из Practical Reverse Engineering

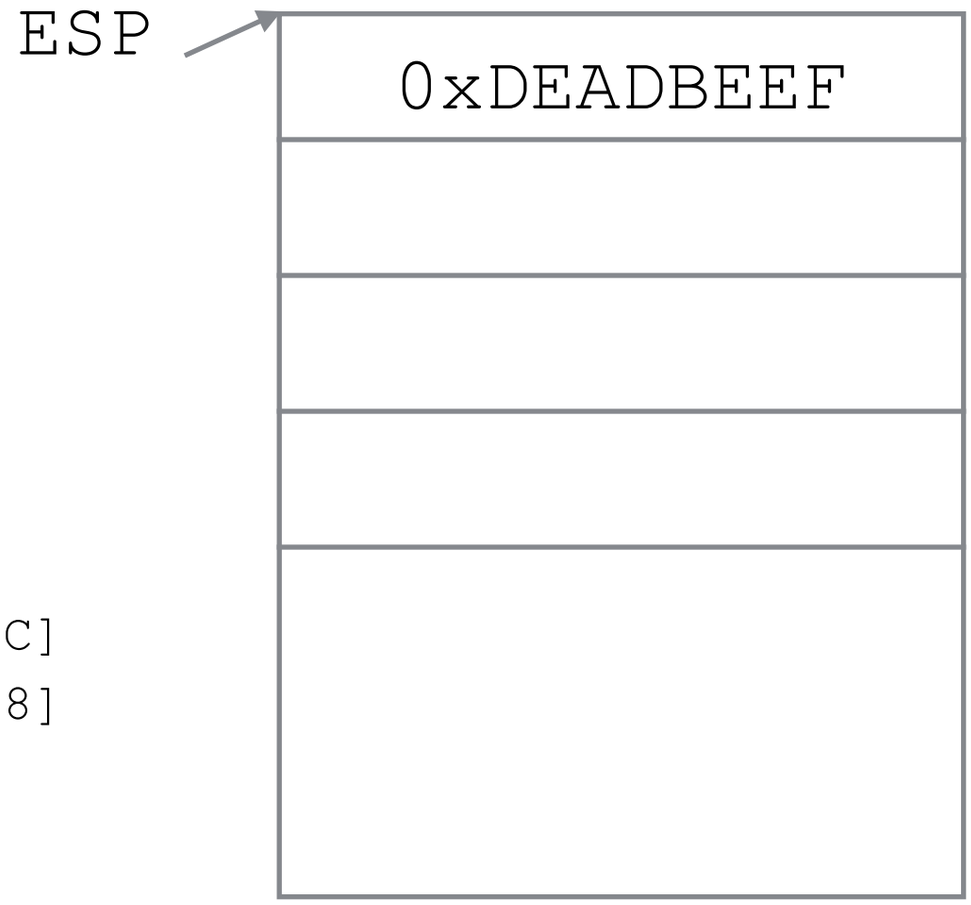
Аргументы и локальные переменные

Вся работа с локальными переменными и аргументами функции происходит через регистр EBP.

Аргументы и локальные переменные

```
0x40000000 push 1
0x40000002 push 2
0x40000004 call foo
0x40000009 add esp, 8
0x4000000C test eax, eax
```

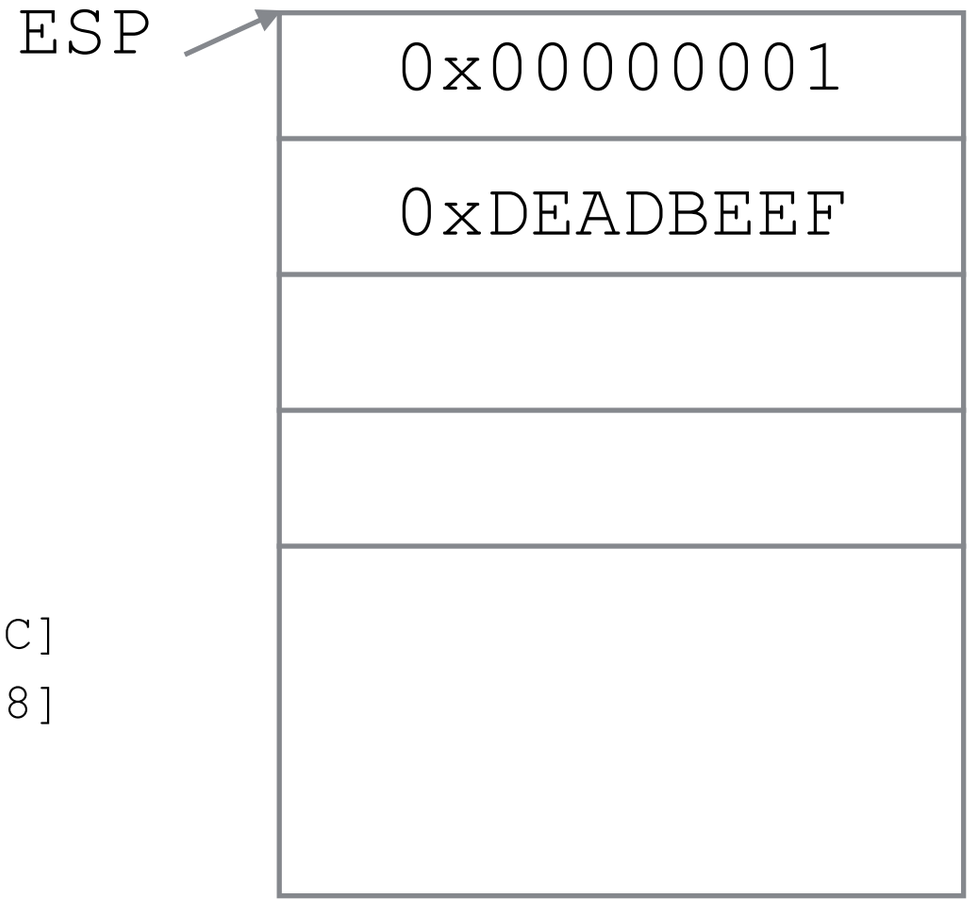
```
0x40001000 foo:
0x40001000 push ebp
0x40001001 mov ebp, esp
0x40001003 sub esp, 4
0x40001006 mov eax, dword ptr [ebp+C]
0x40001009 add eax, dword ptr [ebp+8]
0x4000100C mov dword ptr [ebp-4], 1
0x40001013 add esp, 4
0x40001016 pop ebp
0x40001017 ret
```



Аргументы и локальные переменные

```
0x40000000 push 1
0x40000002 push 2
0x40000004 call foo
0x40000009 add esp, 8
0x4000000C test eax, eax
```

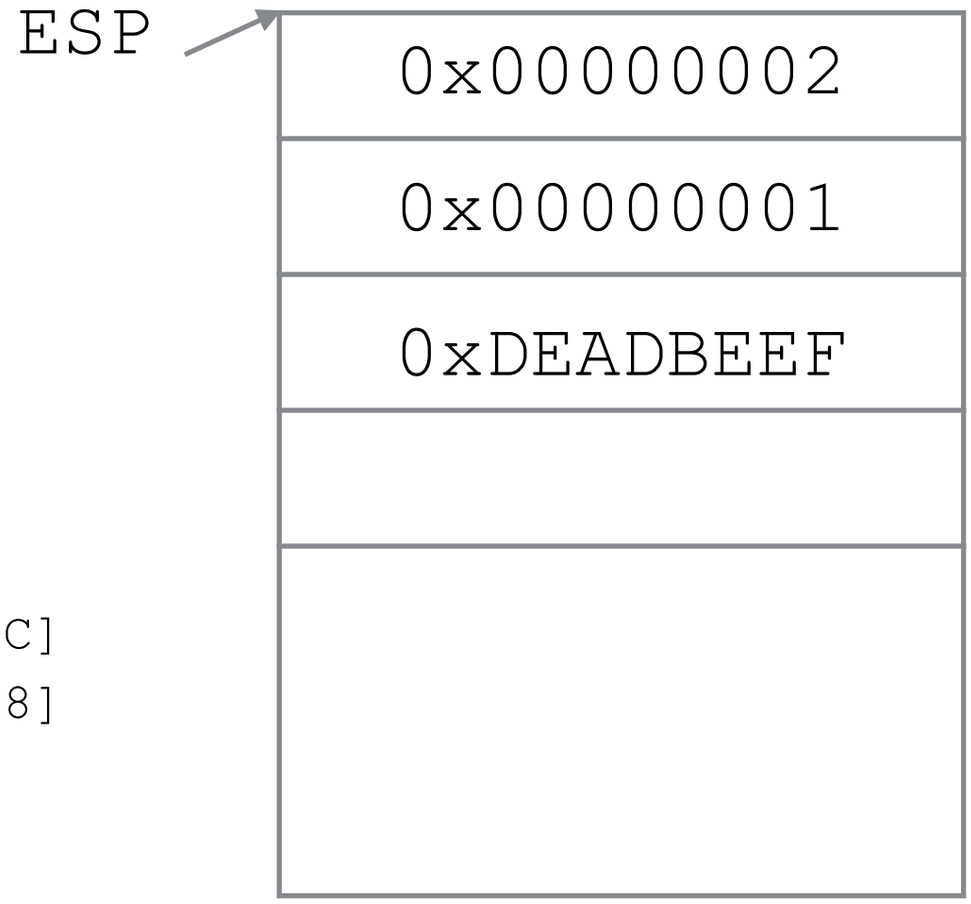
```
0x40001000 foo:
0x40001000 push ebp
0x40001001 mov ebp, esp
0x40001003 sub esp, 4
0x40001006 mov eax, dword ptr [ebp+C]
0x40001009 add eax, dword ptr [ebp+8]
0x4000100C mov dword ptr [ebp-4], 1
0x40001013 add esp, 4
0x40001016 pop ebp
0x40001017 ret
```



Аргументы и локальные переменные

```
0x40000000 push 1
0x40000002 push 2
0x40000004 call foo
0x40000009 add esp, 8
0x4000000C test eax, eax
```

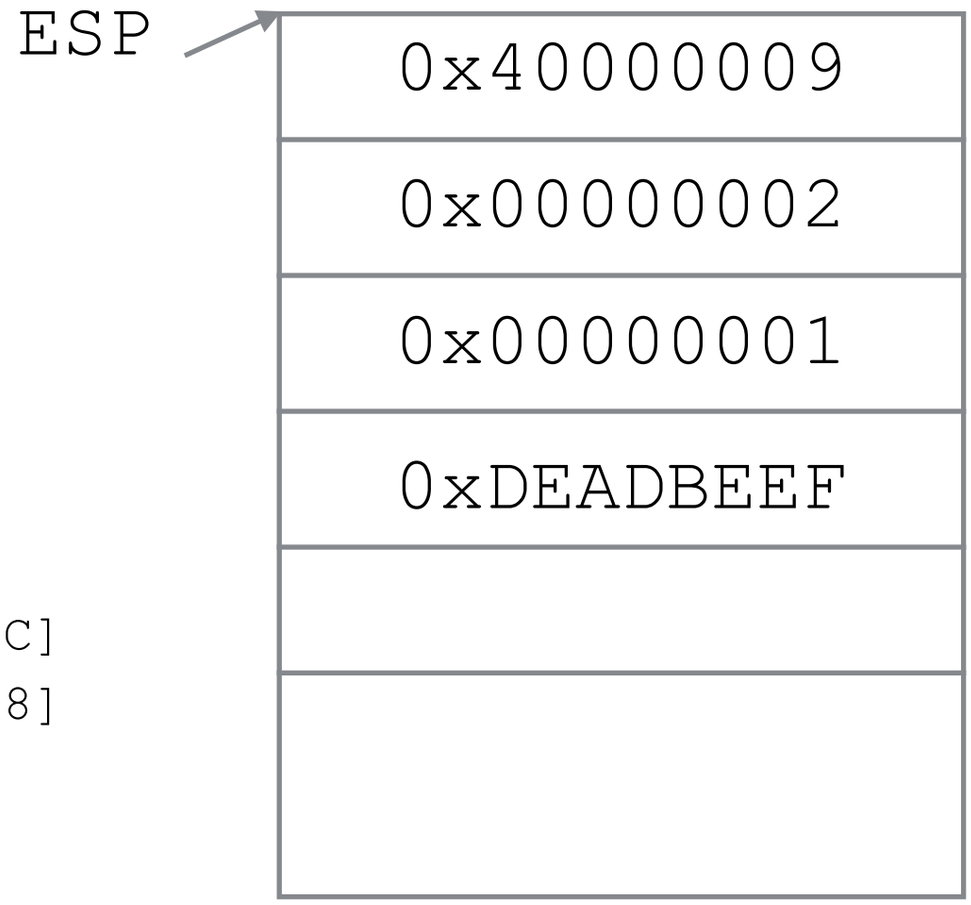
```
0x40001000 foo:
0x40001000 push ebp
0x40001001 mov ebp, esp
0x40001003 sub esp, 4
0x40001006 mov eax, dword ptr [ebp+C]
0x40001009 add eax, dword ptr [ebp+8]
0x4000100C mov dword ptr [ebp-4], 1
0x40001013 add esp, 4
0x40001016 pop ebp
0x40001017 ret
```



Аргументы и локальные переменные

```
0x40000000 push 1
0x40000002 push 2
0x40000004 call foo
0x40000009 add esp, 8
0x4000000C test eax, eax
```

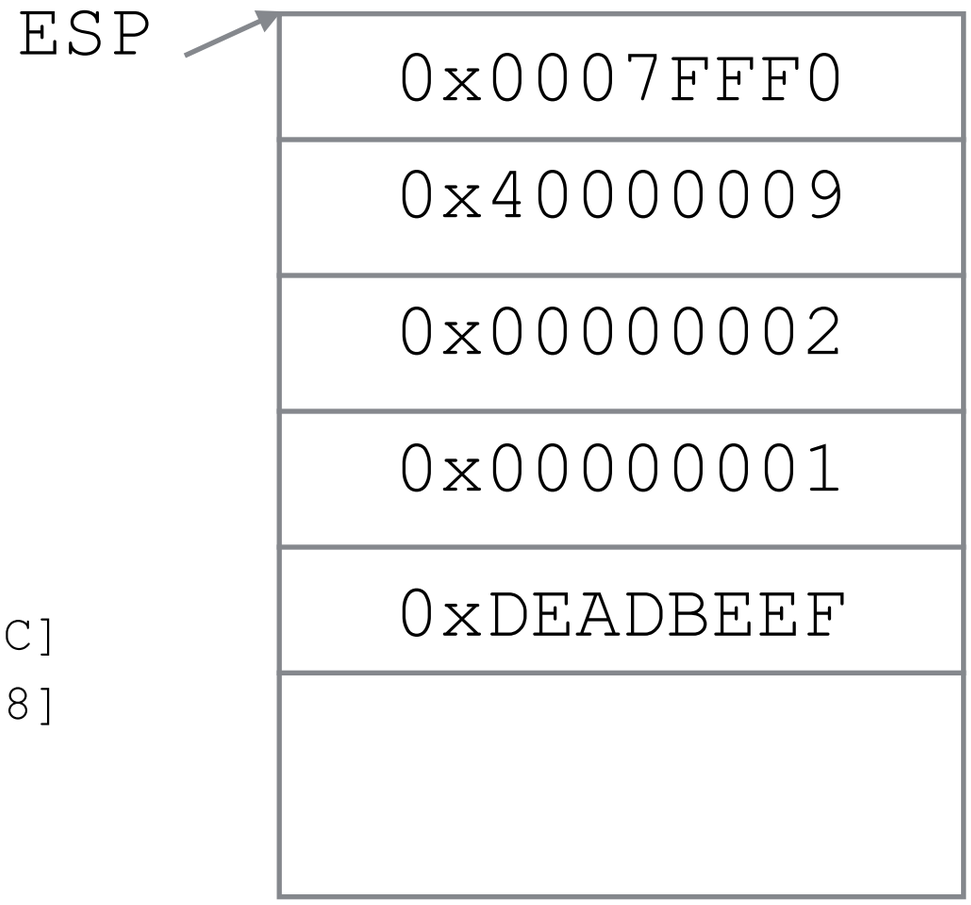
```
0x40001000 foo:
0x40001000 push ebp
0x40001001 mov ebp, esp
0x40001003 sub esp, 4
0x40001006 mov eax, dword ptr [ebp+C]
0x40001009 add eax, dword ptr [ebp+8]
0x4000100C mov dword ptr [ebp-4], 1
0x40001013 add esp, 4
0x40001016 pop ebp
0x40001017 ret
```



Аргументы и локальные переменные

```
0x40000000 push 1
0x40000002 push 2
0x40000004 call foo
0x40000009 add esp, 8
0x4000000C test eax, eax
```

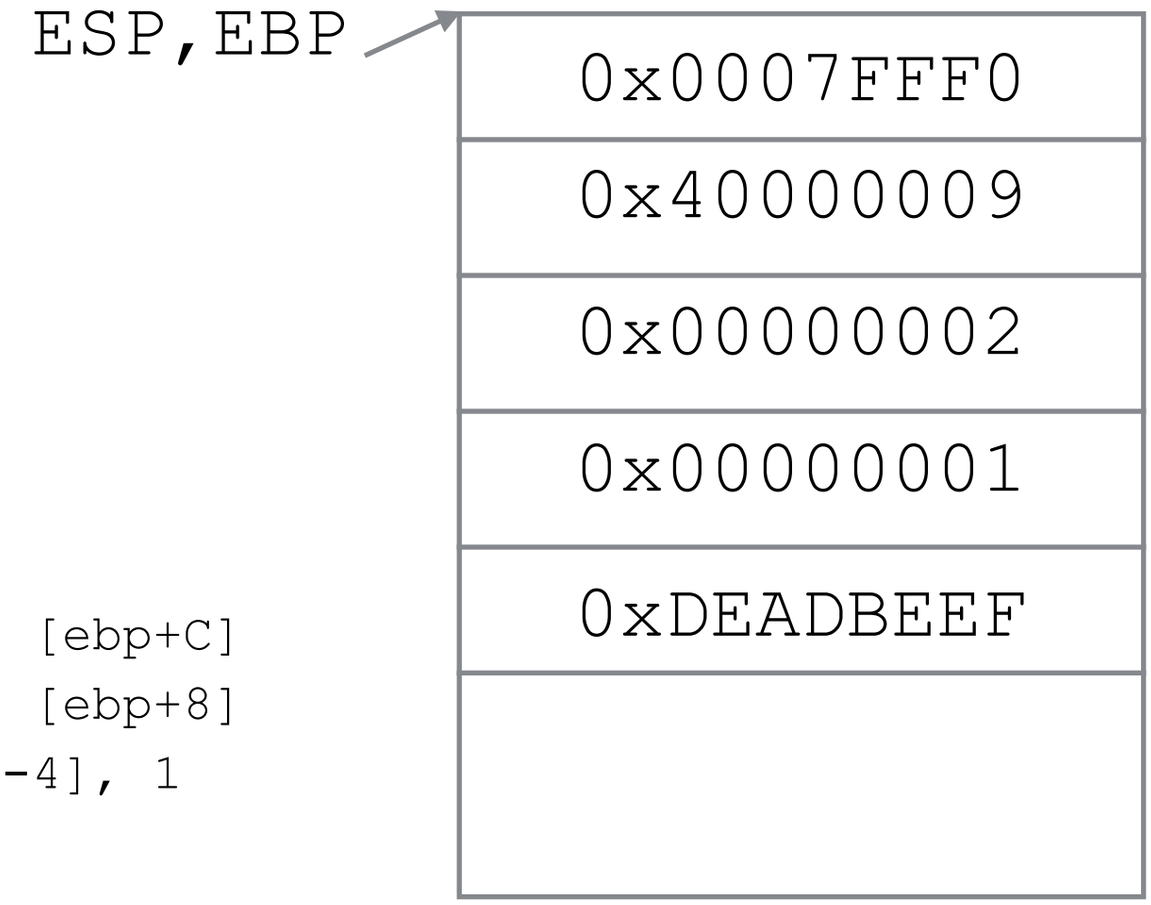
```
0x40001000 foo:
0x40001000 push ebp
0x40001001 mov ebp, esp
0x40001003 sub esp, 4
0x40001006 mov eax, dword ptr [ebp+C]
0x40001009 add eax, dword ptr [ebp+8]
0x4000100C mov dword ptr [ebp-4], 1
0x40001013 add esp, 4
0x40001016 pop ebp
0x40001017 ret
```



Аргументы и локальные переменные

```
0x40000000 push 1
0x40000002 push 2
0x40000004 call foo
0x40000009 add esp, 8
0x4000000C test eax, eax
```

```
0x40001000 foo:
0x40001000 push ebp
0x40001001 mov ebp, esp
0x40001003 sub esp, 4
0x40001006 mov eax, dword ptr [ebp+C]
0x40001009 add eax, dword ptr [ebp+8]
0x4000100C mov dword ptr [ebp-4], 1
0x40001013 add esp, 4
0x40001016 pop ebp
0x40001017 ret
```



Аргументы и локальные переменные

```
0x40000000 push 1
0x40000002 push 2
0x40000004 call foo
0x40000009 add esp, 8
0x4000000C test eax, eax
```

```
0x40001000 foo:
0x40001000 push ebp
0x40001001 mov ebp, esp
0x40001003 sub esp, 4
0x40001006 mov eax, dword ptr [ebp+C]
0x40001009 add eax, dword ptr [ebp+8]
0x4000100C mov dword ptr [ebp-4], 1
0x40001013 add esp, 4
0x40001016 pop ebp
0x40001017 ret
```

Аргументы и локальные переменные

```
0x40000000 push 1
0x40000002 push 2
0x40000004 call foo
0x40000009 add esp, 8
0x4000000C test eax, eax
```

```
0x40001000 foo:
0x40001000 push ebp
0x40001001 mov ebp, esp
0x40001003 sub esp, 4
0x40001006 mov eax, dword ptr [ebp+C]
0x40001009 add eax, dword ptr [ebp+8]
0x4000100C mov dword ptr [ebp-4], 1
0x40001013 add esp, 4
0x40001016 pop ebp
0x40001017 ret
```

Аргументы и локальные переменные

```
0x40000000 push 1
0x40000002 push 2
0x40000004 call foo
0x40000009 add esp, 8
0x4000000C test eax, eax
```

```
0x40001000 foo:
0x40001000 push ebp
0x40001001 mov ebp, esp
0x40001003 sub esp, 4
0x40001006 mov eax, dword ptr [ebp+C]
0x40001009 add eax, dword ptr [ebp+8]
0x4000100C mov dword ptr [ebp-4], 1
0x40001013 add esp, 4
0x40001016 pop ebp
0x40001017 ret
```

Аргументы и локальные переменные

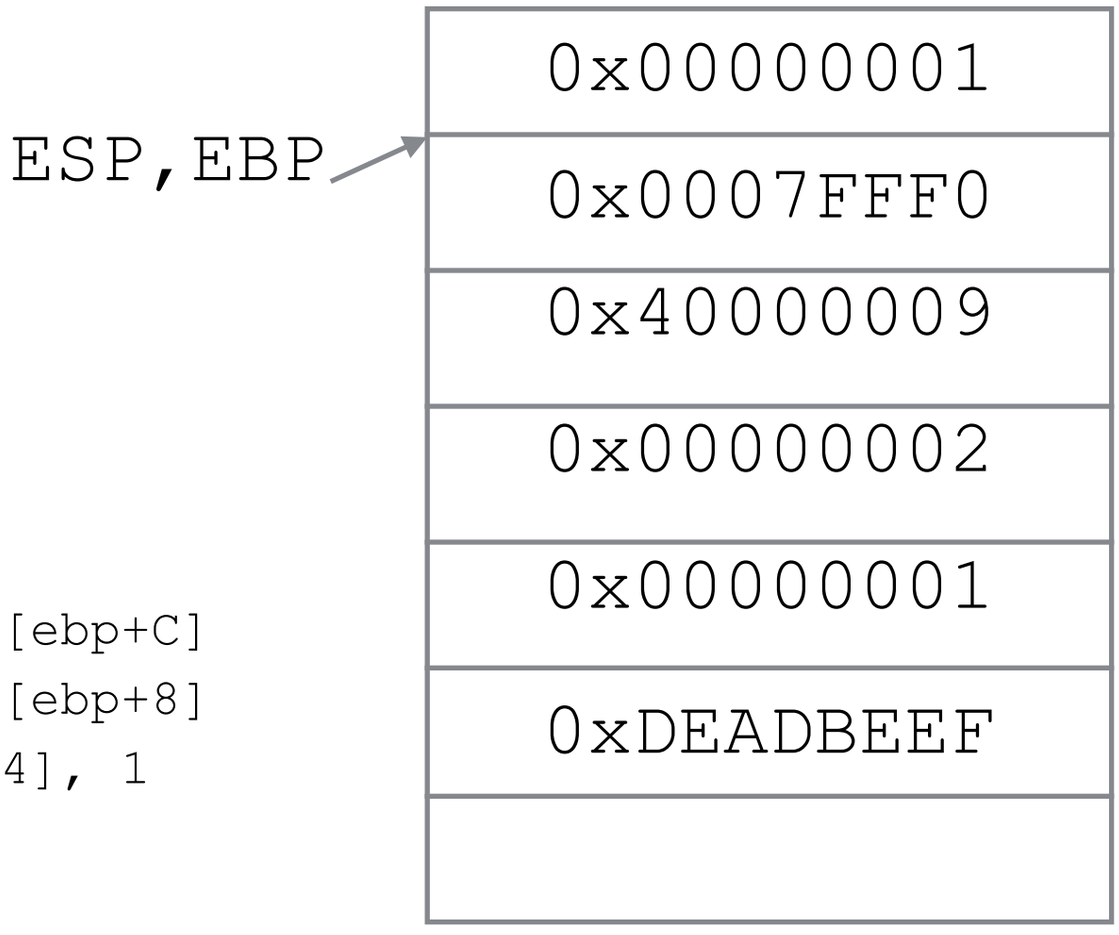
```
0x40000000 push 1
0x40000002 push 2
0x40000004 call foo
0x40000009 add esp, 8
0x4000000C test eax, eax
```

```
0x40001000 foo:
0x40001000 push ebp
0x40001001 mov ebp, esp
0x40001003 sub esp, 4
0x40001006 mov eax, dword ptr [ebp+C]
0x40001009 add eax, dword ptr [ebp+8]
0x4000100C mov dword ptr [ebp-4], 1
0x40001013 add esp, 4
0x40001016 pop ebp
0x40001017 ret
```

Аргументы и локальные переменные

```
0x40000000 push 1
0x40000002 push 2
0x40000004 call foo
0x40000009 add esp, 8
0x4000000C test eax, eax
```

```
0x40001000 foo:
0x40001000 push ebp
0x40001001 mov ebp, esp
0x40001003 sub esp, 4
0x40001006 mov eax, dword ptr [ebp+C]
0x40001009 add eax, dword ptr [ebp+8]
0x4000100C mov dword ptr [ebp-4], 1
0x40001013 add esp, 4
0x40001016 pop ebp
0x40001017 ret
```



Аргументы и локальные переменные

```
0x40000000 push 1
0x40000002 push 2
0x40000004 call foo
0x40000009 add esp, 8
0x4000000C test eax, eax
```

```
0x40001000 foo:
0x40001000 push ebp
0x40001001 mov ebp, esp
0x40001003 sub esp, 4
0x40001006 mov eax, dword ptr [ebp+C]
0x40001009 add eax, dword ptr [ebp+8]
0x4000100C mov dword ptr [ebp-4], 1
0x40001013 add esp, 4
0x40001016 pop ebp
0x40001017 ret
```

Аргументы и локальные переменные

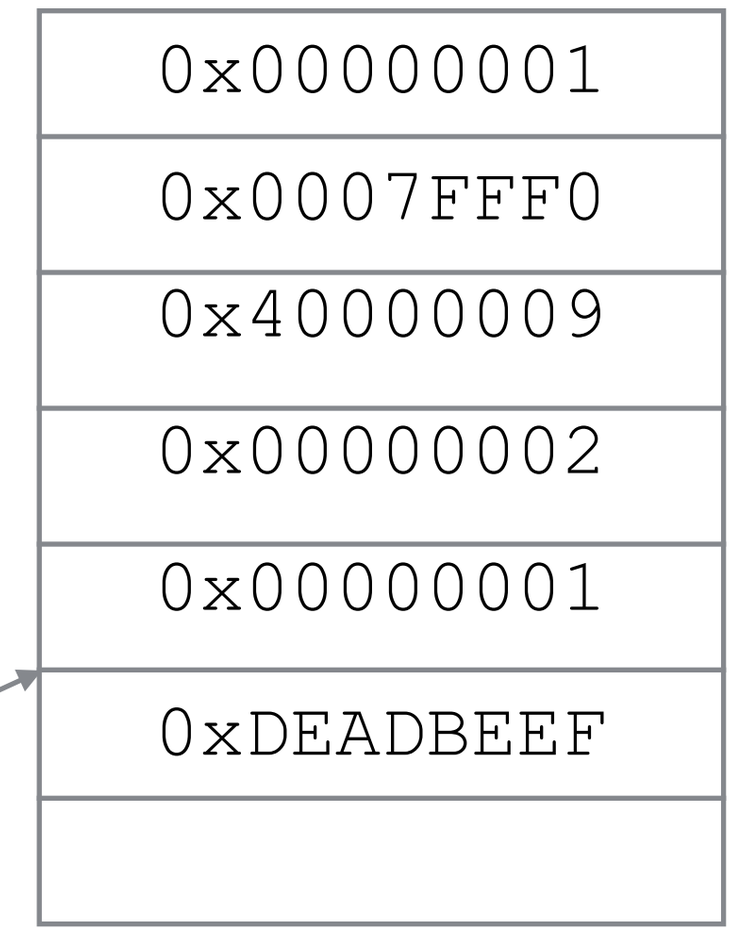
```
0x40000000 push 1
0x40000002 push 2
0x40000004 call foo
0x40000009 add esp, 8
0x4000000C test eax, eax
```

```
0x40001000 foo:
0x40001000 push ebp
0x40001001 mov ebp, esp
0x40001003 sub esp, 4
0x40001006 mov eax, dword ptr [ebp+C]
0x40001009 add eax, dword ptr [ebp+8]
0x4000100C mov dword ptr [ebp-4], 1
0x40001013 add esp, 4
0x40001016 pop ebp
0x40001017 ret
```

Аргументы и локальные переменные

```
0x40000000 push 1
0x40000002 push 2
0x40000004 call foo
0x40000009 add esp, 8
0x4000000C test eax, eax
```

```
0x40001000 foo:
0x40001000 push ebp
0x40001001 mov ebp, esp
0x40001003 sub esp, 4
0x40001006 mov eax, dword ptr [ebp+C]
0x40001009 add eax, dword ptr [ebp+8]
0x4000100C mov dword ptr [ebp-4], 1
0x40001013 add esp, 4
0x40001016 pop ebp
0x40001017 ret
```



Аргументы и локальные переменные

Не cdecl, stdcall, fastcall едины.
Бывают другие варианты передачи аргументов и адресации локальных переменных.

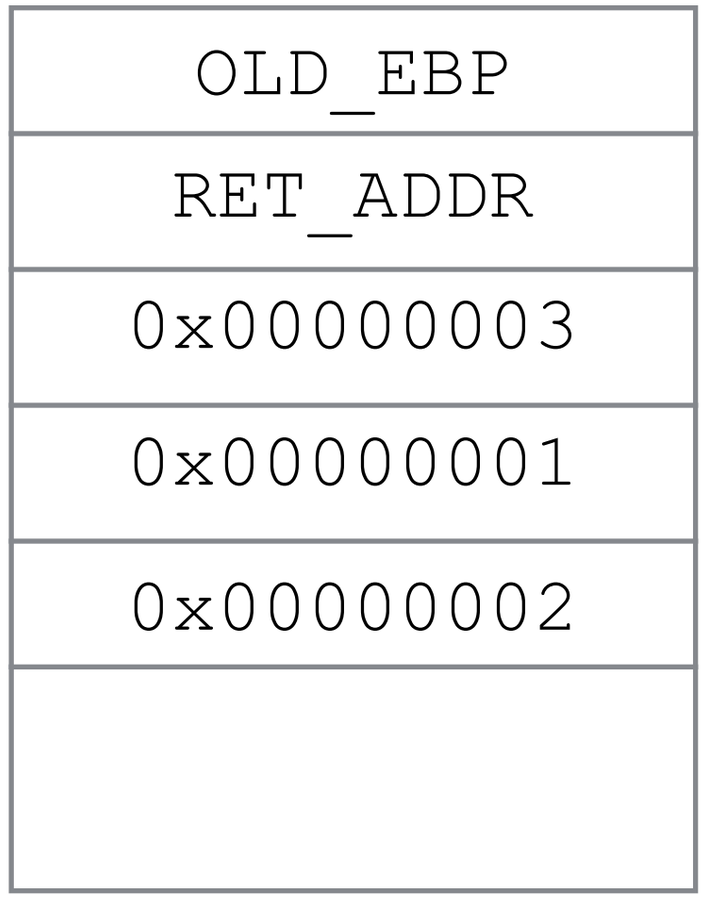
Использование адресации через ESP вместо EBP освобождает ещё один регистр для свободного использования.

int __cdecl foo(3, 1, 2) == ?

```
foo:
 push ebp
 mov ebp, esp
 mov eax, dword ptr [ebp+10]
 pop ebp
 ret
```

int __cdecl foo(3, 1, 2) == ?

```
foo:
 push ebp
 mov ebp, esp
 mov eax, dword ptr [ebp+10]
 pop ebp
 ret
```



```
foo:
 push ebp
 mov ebp, esp
 sub esp, 8
 mov ecx, dword ptr [ebp+8]
 mov dword ptr [ebp-4], 0
 mov dword ptr [ebp-8], 1
 test ecx, ecx
 je foo_exit
foo_loop:
 mov eax, dword ptr [ebp-4]
 mov ebx, dword ptr [ebp-8]
 mov dword ptr [ebp-4], ebx
 add eax, ebx
 mov dword ptr [ebp-8], eax
 dec ecx
 jne foo_loop
foo_exit:
 mov eax, dword ptr [ebp-4]
 add esp, 8
 pop ebp
 ret
```