Архитектура ЭВМ

Лекция 5. Стек и подпрограммы

к.ф.-м.н. Филонов Павел Владимирович
filonovpv@gmail.com

Московский Государственный Технический Университет
Гражданской Авиации

22 сентября 2013 г.

Сегодня поговорим о ...

Сегодня поговорим о ...

- Что такое стек?

Сегодня поговорим о ...

- Что такое стек?
- Процессор имеет им пользоваться

Сегодня поговорим о ...

- Что такое стек?
- Процессор имеет им пользоваться
- Зачем он нужен?

Сегодня поговорим о ...

- Что такое стек?
- Процессор имеет им пользоваться
- Зачем он нужен?
- Чем помогут подпрограммы?

Сегодня поговорим о ...

- Что такое стек?
- Процессор имеет им пользоваться
- Зачем он нужен?
- Чем помогут подпрограммы?
- А как они вообще работают?

Сегодня поговорим о ...

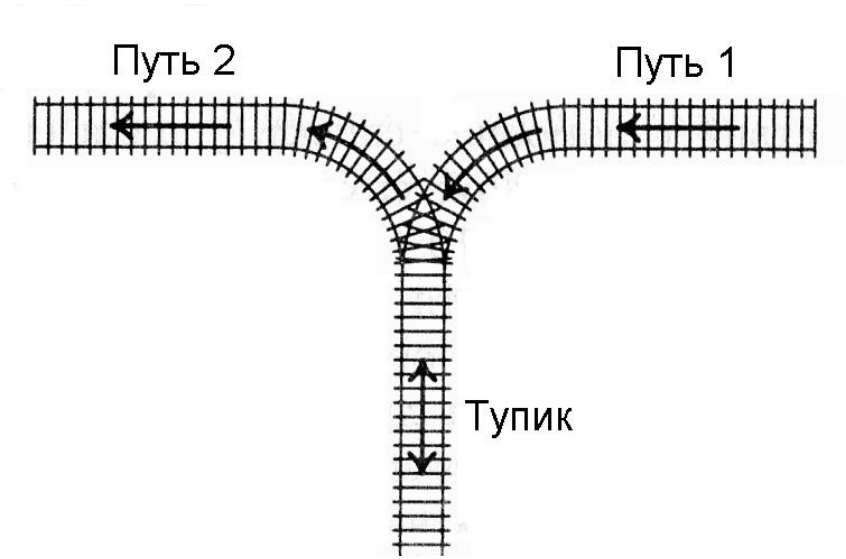
- Что такое стек?
- Процессор имеет им пользоваться
- Зачем он нужен?
- Чем помогут подпрограммы?
- А как они вообще работают?
- Стековый фрейм — взрыв мозга или неизбежное зло?

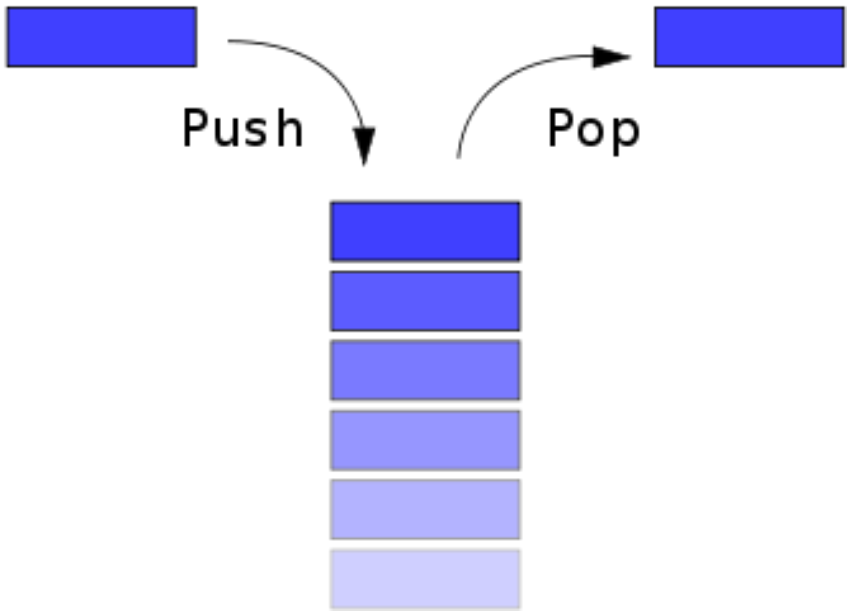
Сегодня поговорим о ...

- Что такое стек?
- Процессор имеет им пользоваться
- Зачем он нужен?
- Чем помогут подпрограммы?
- А как они вообще работают?
- Стековый фрейм — взрыв мозга или неизбежное зло?
- Разные соглашения о вызове подпрограмм

Стек

Стек (*англ.* Stack) — структура данных, организованная по принципу LIFO (Last In — First Out, последним пришёл — первым вышел)



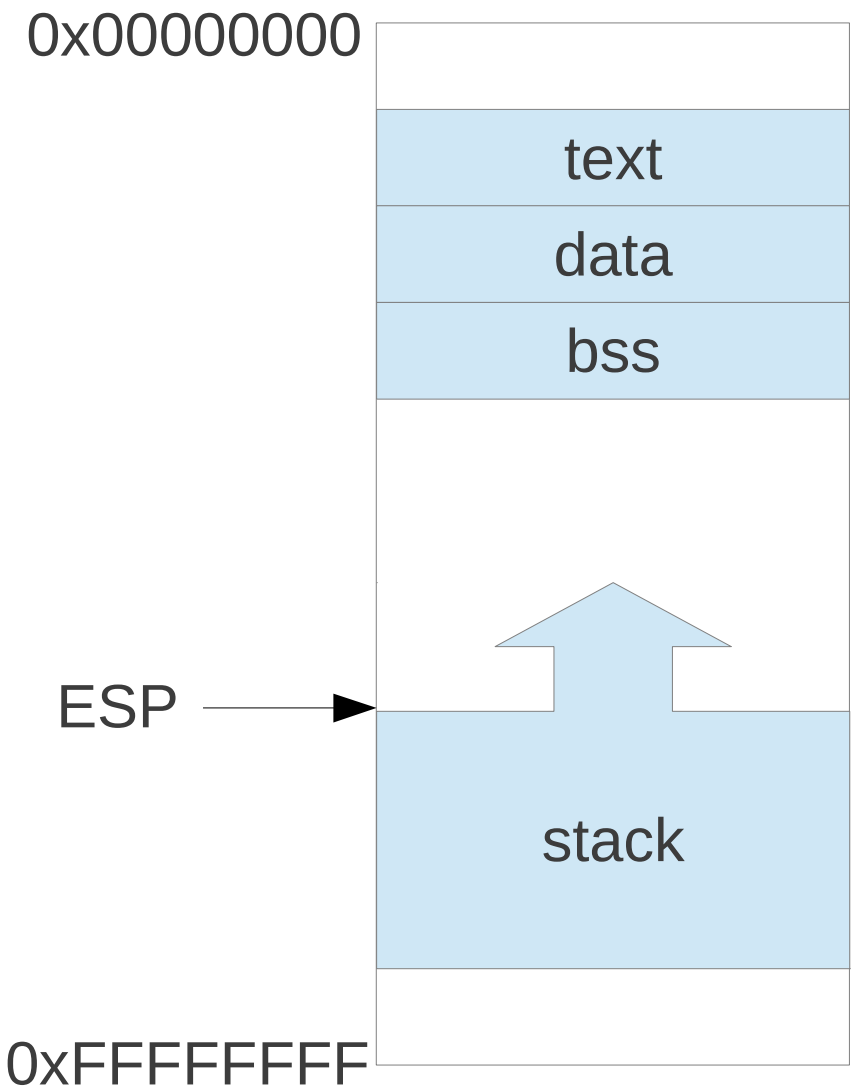


Push — положить элемент на вершину стека

Pop — снять элемент с вершины стека

Организация стека в архитектуре x86

- Секция стека располагается в старших адресах памяти.
- Адрес вершины стека хранится в регистре ESP.
- Вершина стека растёт в сторону уменьшения адресов.



Команды работы со стеком

Положить в стек

Команда **push** помещает операнд на вершину стека и уменьшает ESP на размер операнда.

- push — положить слово (2 байта) или двойное слово (4 байта)
- pushf (pushfd) — положить в стека EFALGS (4 байта)
- pusha (pushad) — положить в стек все 4-байтовые регистры (EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI)

Взять со стека

Команда **pop** извлекает с вершины стека данные, записывает их в операнд и увеличивает ESP на размер операнда

- pop — извлечь слово (2 байта) или двойное слово (4 байта)
- popf (popfd) — извлечь из стека EFALGS (4 байта)
- popa (popad) — извлечь из стека все 4-байтовые регистры (EDI, ESI, EBP, EBX, EDX, ECX, EAX). Значение ESP пропускается

Пример (правильное скобочное выражение)

Правильное скобочное выражение

```
 1  ▷ возвращает True, либо False
 2  c ← следующий символ
 3  while не конец файла
 4        do if c = '('
 5              then push c
 6              else  if c = ')'
 7                       then pop t
 8                            if t ≠ c
 9                               then return False
10           c ← следующий символ
11  if стек не пуст
12     then return False
13     else  return True
```

Подпрограммы

Подпрограммой называется некоторая обособленная часть программного кода, которая может быть **вызвана** из главной программы (или из другой подпрограммы.)

Под **вызовом** понимается временная передача управления подпрограмме с тем, чтобы, когда подпрограмма сделает свою работу, она вернула управление в точку, откуда её вызвали.

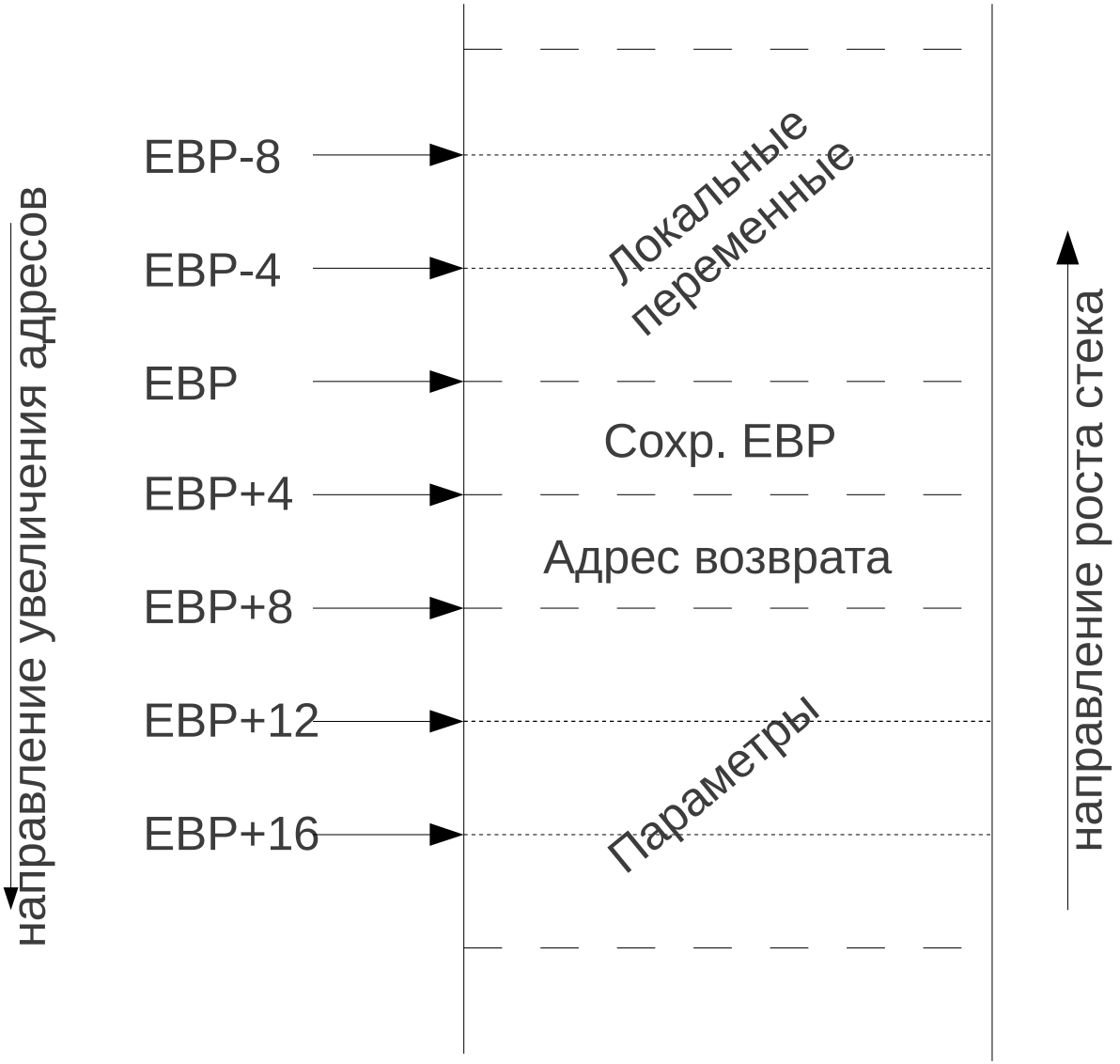
При вызове подпрограммы необходимо запомнить **адрес возврата**, т.е. адрес машинной команды, следующей за командой вызова подпрограммы.

При **вызове** подпрограмме могут передаваться параметры, влияющие на её работу.

Подпрограммы используют в ходе своей работы **локальные переменные**.

Стековый фрейм

В современных вычислительных системах **параметры**, **адрес возврата** и **локальные переменные** подпрограмм хранятся в области стековой памяти, называемой **стековым фреймом**.



Вызов и возврат из подпрограммы

Инструкция `call <метка>` сохраняет в стеке адрес возврата и передаёт управление на указанную метку.
Инструкция `ret` выталкивает из стека адрес возврата и передаёт управление на него.

Действия при входе в подпрограмму

1. сохранить в стеке значение EBP
2. записать в EBP текущий адрес вершины стека (ESP)
3. вычесть из ESP число байт под локальные переменные
4. сохранить в стеке все регистры, которые будут изменены в подпрограмме (включая EFLAGS)

Действия при выходе из подпрограммы

1. восстановить значение всех регистров (включая EFLAGS)
2. записать в ESP значение EBP
3. восстановить старое значение EBP из стека

Пример (Алгоритм Евклида)

Главная программа

```
%include "stud_io.inc"

global _start

section .text
_start:
    push 169     ; b := 169
    push 39      ; a := 39
    call gcd     ; eax := gcd(a,b)
    add esp, 8   ;
    push eax
    call printint
    add esp, 4

    FINISH
```

Подпрограмма gcd

```
gcd:    push ebp
        mov ebp, esp
        push ebx
        push edx
        pushf
while:  cmp [ebp+12], dword 0 ; b <> 0
        je wend
        mov eax, [ebp+8]  ; eax := a
        mov ebx, [ebp+12] ; ebx := b
        cdq               ; eax -> [edx:eax]
        idiv ebx          ; edx := a mod b
        mov [ebp+8],  ebx ; a := b
        mov [ebp+12], edx ; b := edx
        jmp while
wend:   mov eax, [ebp + 8] ; return a
        popf
        pop edx
        pop ebx
        mov esp, ebp
        pop ebp
        ret
```

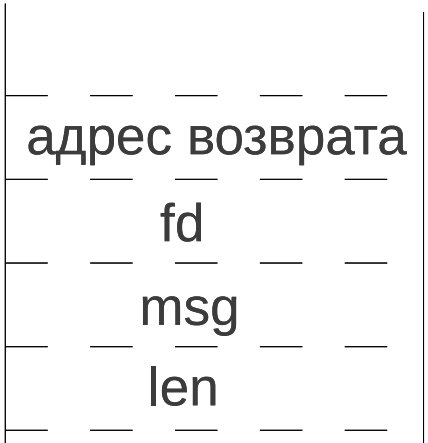
Соглашения о вызове подпрограмм

Что регламентируется:

- как передаются параметры (через регистры или через стек)
- в каком порядке
- кто восстанавливает стек (вызываемая или вызывающая подпрограмма)
- как возвращается результат

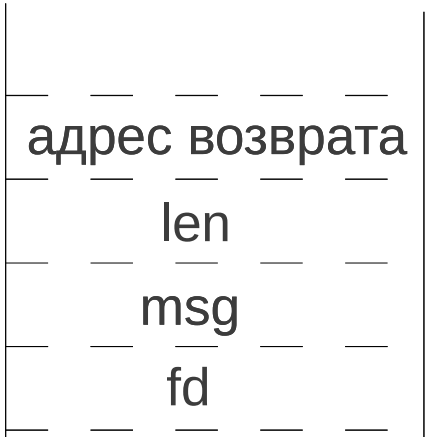
func(fd, msg, len)

cdelc:




```
push len
push msg
push fd
call func
add esp, 12
```

pascal:




```
push fd
push msg
push len
call func
```