© 2013 МГУ/ВМиК/СП

Лекция 14

27 марта

© 2013 МГУ/ВМиК/СП

Рекурсивная функция

```
/* Рекурсивный popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

```
pcount_r:
    push   ebp
    mov    ebp, esp
    push   ebx
    sub    esp, 4
    mov    ebx, dword [ebp + 8]
    mov    eax, 0
    test   ebx, ebx
    je   .L3
    mov    eax, ebx
    shr    eax, 1
    mov    dword [esp], eax
    call   pcount_r
    mov    edx, ebx
    and    edx, 1
    lea    eax, [edx + eax]
.L3:
    add    esp, 4
    pop    ebx
    pop    ebp
    ret
```

- Регистры
  - eax, edx используются без предварительного сохранения
  - ebx используется, сохраняется в начале и восстанавливается в конце

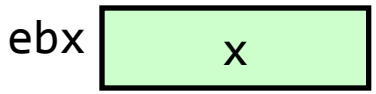
© 2013 МГУ/ВМиК/СП

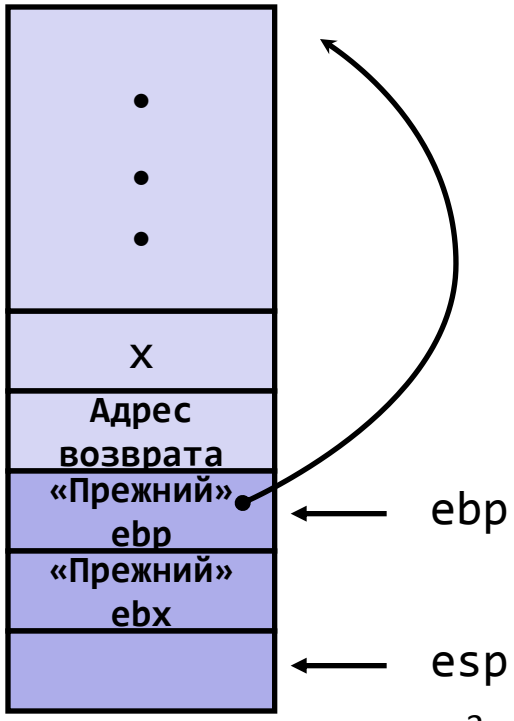
Рекурсивный вызов (1/5)

```
/* Рекурсивный popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

```
pcount_r:
    push   ebp
    mov    ebp, esp
    push   ebx
    sub    esp, 4
    mov    ebx, dword [ebp + 8]
     • • •
```

- Действия
  - Сохраняем значение ebx на стеке
  - Выделяем место для размещения аргумента вызова (рекурсивного)
  - Размещаем значение x в ebx





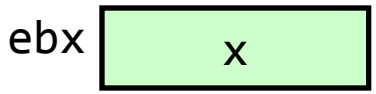
© 2013 МГУ/ВМиК/СП

Рекурсивный вызов (2/5)

```
/* Рекурсивный popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

```
     • • •
    mov   eax, 0
    test  ebx, ebx
    je   .L3
     • • •
.L3:
     • • •
    ret
```

- Действия
  - Если x == 0, выходим из функции
    - Регистр eax содержит 0

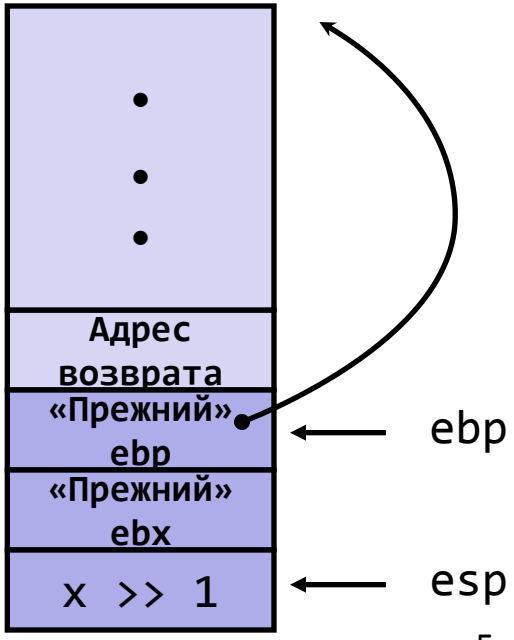


© 2013 МГУ/ВМиК/СП

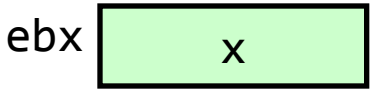
Рекурсивный вызов (3/5)

```
/* Рекурсивный popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

```
 • • •
mov    eax, ebx
shr    eax, 1
mov    dword [esp], eax
call   pcount_r
 • • •
```



- Действия
  - Сохраняем x >> 1 на стеке
  - Выполняем рекурсивный вызов
- Результат
  - eax содержит возвращенное значение
  - ebx содержит неизменное значение x



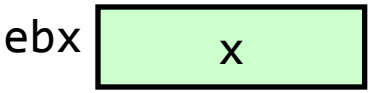
© 2013 МГУ/ВМиК/СП

Рекурсивный вызов (4/5)

```
/* Рекурсивный popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

```
 • • •
mov    edx, ebx
and    edx, 1
lea    eax, [edx + eax]
 • • •
```

- Состояние регистров
  - eax содержит значение полученное от рекурсивного вызова
  - ebx содержит x
- Действия
  - Вычисляем (x & 1) + возвращенное значение
- Результат
  - Регистр eax получает результат работы функции



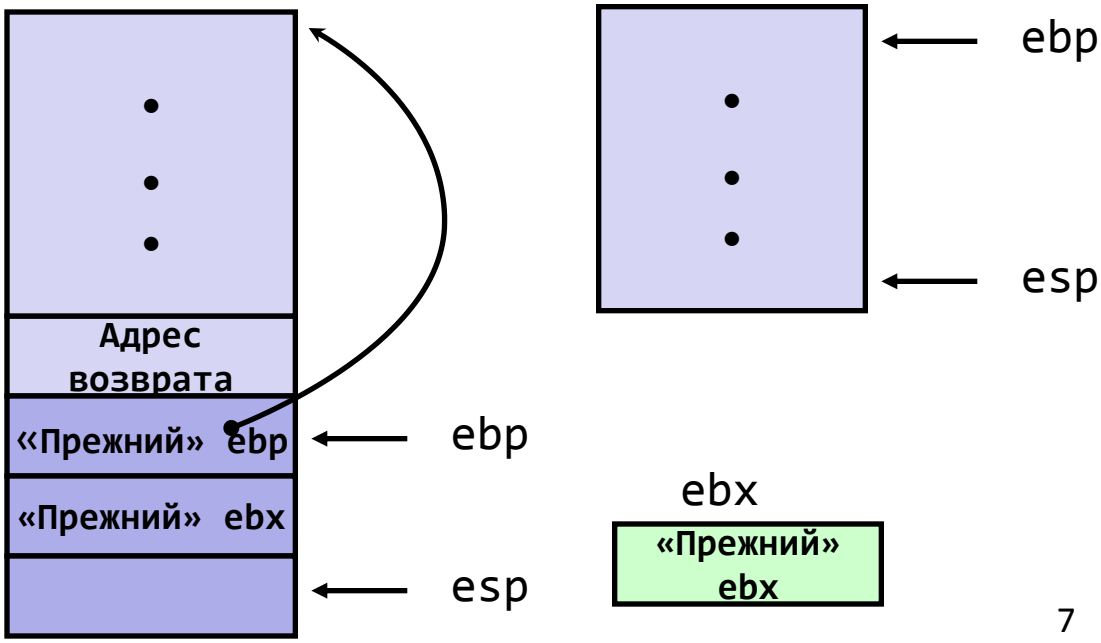
© 2013 МГУ/ВМиК/СП

Рекурсивный вызов (5/5)

```
/* Рекурсивный popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

```
    • • •
L3:
    add esp,  4
    pop  ebx
    pop  ebp
    ret
```

- Действия
  - Восстанавливаем значения ebx и ebp
  - Восстанавливаем esp



© 2013 МГУ/ВМиК/СП

Рекурсия. Заключение.

- Не используются дополнительные приемы
  - Создание фреймов гарантирует, что каждый вызов располагает персональным блоком памяти
    - Хранятся регистры и локальные переменные
    - Хранится адрес возврата
  - Общее соглашение о сохранении регистров препятствует порче регистров различными вызовами
  - Стековая организация поддерживается порядком вызовов и возвратов из функций
    - Если P вызывает Q, тогда Q завершается до того, как завершится P
    - Последним пришел, первым ушел
- Аналогично при неявной рекурсии
  - P вызывает Q; Q вызывает P

© 2013 МГУ/ВМиК/СП

Передача указателей на локальные переменные в качестве параметров

Создаем указатель

```
/* Вычисляем x + 3 */
int add3(int x) {
  int localx = x;
  incrk(&localx, 3);
  return localx;
}
```

Разыменовываем указатель

```
/* увеличиваем значение на k */
void incrk(int *ip, int k) {
  *ip += k;
}
```

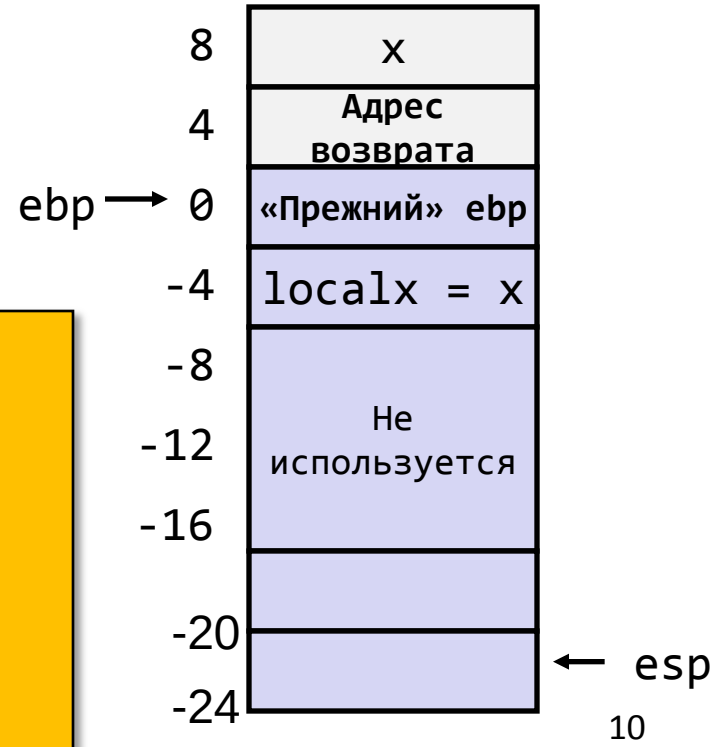
- add3 создает указатель и передает его в incrk

© 2013 МГУ/ВМиК/СП

Создаем и инициализируем локальные переменные

```
int add3(int x) {
  int localx = x;
  incrk(&localx, 3);
  return localx;
}
```

- Переменную localx необходимо размещать на стеке
  - Причина: создается указатель на нее
- Вычисляем указатель: ebp - 4



Начало функции add3

```
add3:
   push  ebp
   mov   ebp, esp
   sub   esp, 24  ; выделяем 24 байта
   mov   eax, dword [ebp + 8]
   mov   dword [ebp - 4], eax
        ; Присваиваем localx значение x
```

© 2013 МГУ/ВМиК/СП

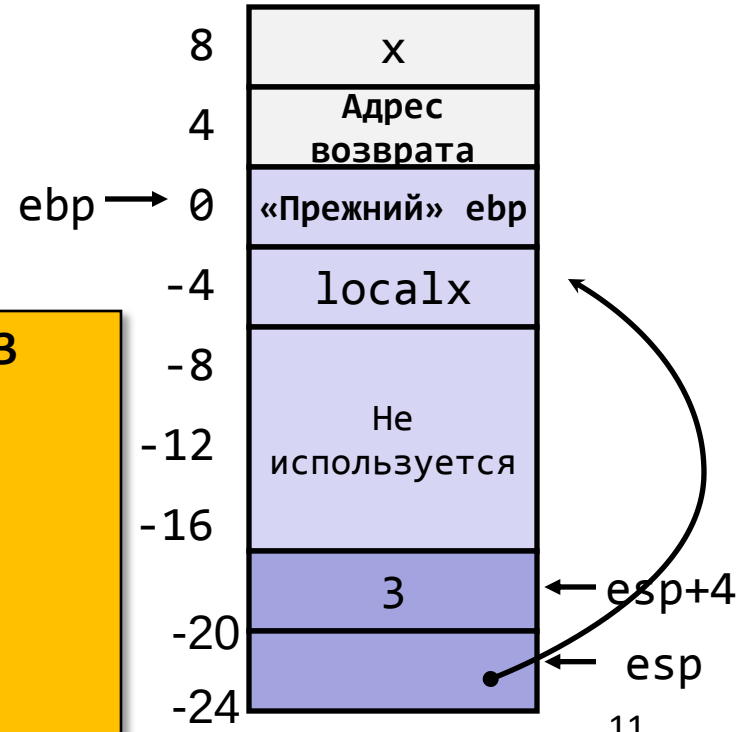
Создаем фактический параметр - указатель

```
int add3(int x) {
  int localx = x;
  incrk(&localx, 3);
  return localx;
}
```

- Инструкция lea используется для вычисления адреса localx

Продолжение функции add3

```
mov dword [esp + 4], 3 ; 2-ой параметр = 3
lea eax, [ebp - 4]     ; &localx
mov dword [esp], eax   ; 1-ый параметр =
                       ;          &localx
call incrk
```



© 2013 МГУ/ВМиК/СП

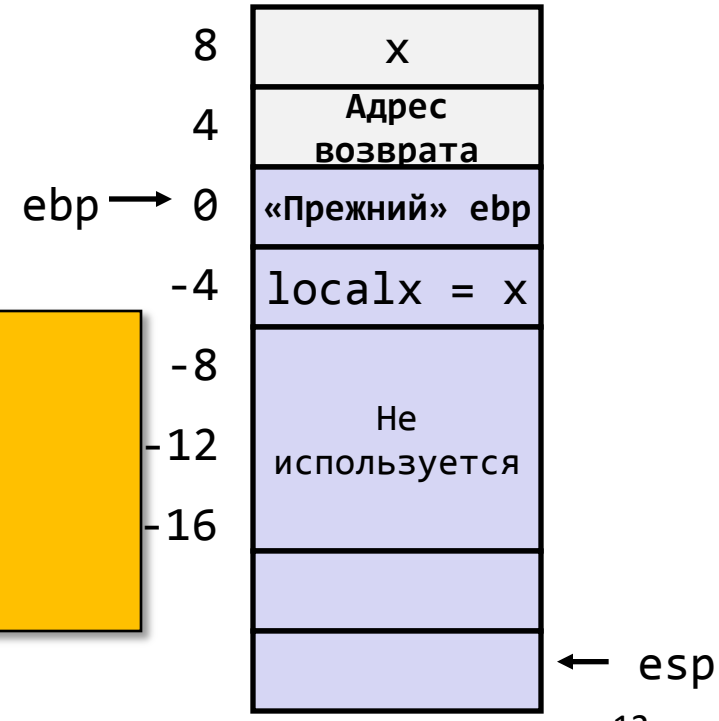
Возвращаемое значение – локальная переменная

```
int add3(int x) {
  int localx = x;
  incrk(&localx, 3);
  return localx;
}
```

- Извлекаем из стека localx – возвращаемое значение функции

Конец функции add3

```
 mov eax, dword [ebp - 4]
  ; возвращаемое значение = localx
 leave
 ret
```



© 2013 МГУ/ВМиК/СП

Соглашение CDECL

- Размещение параметров
  - Стек
- Порядок параметров
  - «обратный»: от «верхушки» стека ко «дну»
- Какие регистры могут быть использованы вызванной функцией
  - EAX, EDX, ECX
- Очистка стека от аргументов вызванной функции
  - Очищает вызвавшая функция
- Возвращаемое функцией значение
  - EAX
  - EAX:EDX
  - Через память

© 2013 МГУ/ВМиК/СП

Размещение параметров и возвращаемого значения

- sizeof == 4
  - Целиком занимает машинное слово, помещаемое на стек
  - EAX
- sizeof < 4
  - Занимает младшие байты слова на стеке
  - AX, AL
- sizeof == 8 (long long)
  - Два машинных слова в естественном порядке
  - EAX:EDX
- Массивы ≡ указатели
- Структуры
  - ???

© 2013 МГУ/ВМиК/СП

Какие есть компиляторы и соглашения о вызове функций?

- Microsoft
- Borland
- Watcom
- Gnu
- Intel

- Digital Mars
- Codeplay

- Модели памяти
  - 16, 32, 64 разряда

- Операционные системы
  - Linux (FreeBSD, …)
  - Windows
  - Mac OS

Agner Fog http://www.agner.org/optimize/

© 2013 МГУ/ВМиК/СП

Параметр – структура

```
typedef struct {
   short a;
   short b;
   short c;
} t_short;

short g(t_short x, t_short y) {
   return x.a * x.b - x.c + y.a;
}
```

```
g:
   push    ebp
   mov     ebp, esp
   movzx   eax, word [ebp+10]
   imul    ax,  word [ebp+8]
   sub     ax,  word [ebp+12]
   add     ax,  word [ebp+16]
   pop     ebp
   ret
```

© 2013 МГУ/ВМиК/СП

Возвращаемое значение - структура

```
typedef struct {
  int x;
  int y;
  int z;
} triple;

triple f2(int a, int b, int c) {
   triple v = {a, b, c};
   return v;
}

void f(triple *p) {
   *p = f2(1, 2, 3);
}
```

```
f:
  push    ebp                    ; (1)
  mov     ebp, esp               ; (2)
  sub     esp, 24                ; (3)
  mov     eax, 1                 ; (4)
  mov     ecx, 3                 ; (5)
  mov     edx, 2                 ; (6)
  mov     dword [esp+4], eax     ; (7)
  mov     dword [esp+12], ecx    ; (8)
  mov     dword [esp+8], edx     ; (9)
  mov     eax, dword [ebp+8]     ; (10)
  mov     dword [esp], eax       ; (11)
  call    f2                     ; (12)
  sub     esp, 4                 ; (13)
  leave                          ; (14)
  ret                            ; (15)
```

© 2013 МГУ/ВМиК/СП

Возвращаемое значение - структура

```
typedef struct {
  int x;
  int y;
  int z;
} triple;

triple f2(int a, int b, int c) {
   triple v = {a, b, c};
   return v;
}

void f(triple *p) {
   *p = f2(1, 2, 3);
}
```

```
f2:
   push    ebp                    ; (1)
   mov     ebp, esp               ; (2)
   mov     eax, dword [ebp+8]     ; (3)
   mov     edx, dword [ebp+20]    ; (4)
   mov     dword [eax+8], edx     ; (5)
   mov     edx, dword [ebp+16]    ; (6)
   mov     dword [eax+4], edx     ; (7)
   mov     edx, dword [ebp+12]    ; (8)
   mov     dword [eax], edx       ; (9)
   pop     ebp                   ; (10)
   ret     4                     ; (11)
```

© 2013 МГУ/ВМиК/СП

Функция main

```
#include <stdio.h>

void nullify(int argc, char* argv[]) {
}

int main(int argc, char* argv[]) {
   nullify(argc, argv);
   return 0;
}
```

```
CMAIN:
   lea     ecx, [esp+4]   ; (1)
   and     esp, -16       ; (2)
   push    dword [ecx-4]  ; (3)
   push    ebp            ; (4)
   mov     ebp, esp       ; (5)
   push    ecx            ; (6)
   sub     esp, 20        ; (7)
   ; ...

nullify:
        ret
```

© 2013 МГУ/ВМиК/СП

Функция main

```
#include <stdio.h>

void nullify(int argc, char* argv[]) {
}

int main(int argc, char* argv[]) {
   nullify(argc, argv);
   return 0;
}
```

```
CMAIN:
   ; …
   mov     eax, dword [ecx+4] ; (8)
   mov     dword [esp+4], eax ; (9)
   mov     eax, dword [ecx]   ; (10)
   mov     dword [esp], eax   ; (11)
   call    nullify            ; (12)
   ; ...

nullify:
        ret
```

© 2013 МГУ/ВМиК/СП

Функция main

```
#include <stdio.h>

void nullify(int argc, char* argv[]) {
}

int main(int argc, char* argv[]) {
   nullify(argc, argv);
   return 0;
}
```

```
CMAIN:
   ; …
   mov     eax, 0        ; (13)
   add     esp, 20       ; (14)
   pop     ecx           ; (15)
   pop     ebp           ; (16)
   lea     esp, [ecx-4]  ; (17)
   ret

nullify:
        ret
```

© 2013 МГУ/ВМиК/СП

Пример вызова malloc

```
#include <stdlib.h>

struct chain;

typedef struct chain {
   int val;
   struct chain *next;
} t_chain, *p_chain;
```

```
p_chain insert(p_chain p, int val) {
   if ((0 == p) || (p->val > val)) {
      p_chain np =
         (p_chain)malloc(sizeof(t_chain));
      np->val = val;
      np->next = p;
      return np;
   } else {
      p->next = insert(p, val);
      return p;
   }
}
```

© 2013 МГУ/ВМиК/СП

Пример вызова malloc

```
p_chain insert(p_chain p, int val) {
   if ((0 == p) || (p->val > val)) {
      p_chain np =
         (p_chain)malloc(sizeof(t_chain));
      np->val = val;
      np->next = p;
      return np;
   } else {
      p->next = insert(p, val);
      return p;
   }
}
```

```
%include 'io.inc'

section .text

CEXTERN malloc

insert:
   push    ebp
   mov     ebp, esp
   sub     esp, 24
   ; ...
```

© 2013 МГУ/ВМиК/СП

Пример вызова malloc

```
p_chain insert(p_chain p, int val) {
   if ((0 == p) || (p->val > val)) {
      p_chain np =
         (p_chain)malloc(sizeof(t_chain));
      np->val = val;
      np->next = p;
      return np;
   } else {
      p->next = insert(p, val);
      return p;
   }
}
```

```
insert:
   ; ...
   mov      dword [ebp-4], esi
   mov      esi, dword [ebp+8]
   mov      dword [ebp-8], ebx
   mov      ebx, dword [ebp+12]
   ; ...
```

© 2013 МГУ/ВМиК/СП

Пример вызова malloc

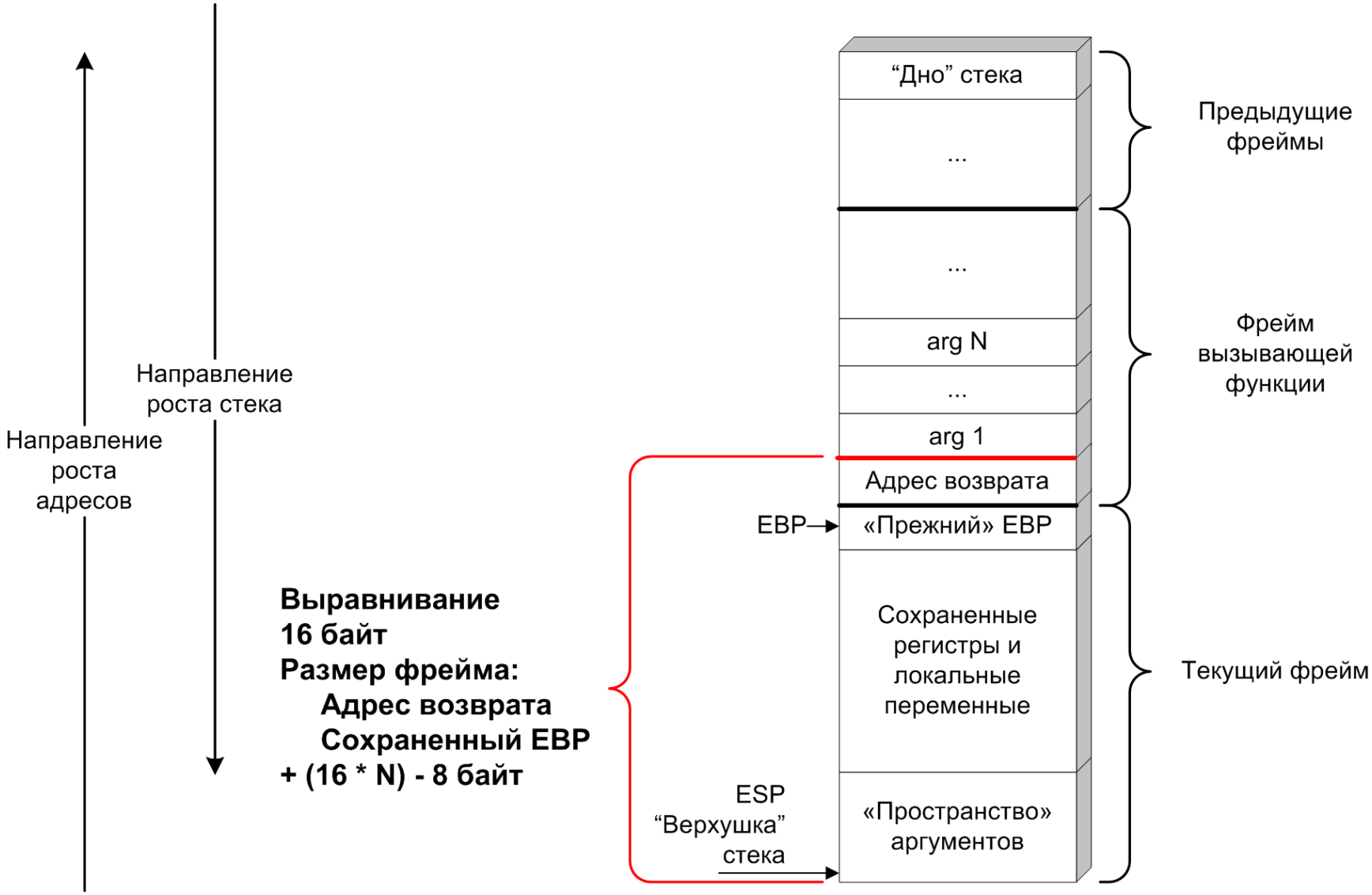
```
p_chain insert(p_chain p, int val) {
   if ((0 == p) || (p->val > val)) {
      p_chain np =
         (p_chain)malloc(sizeof(t_chain));
      np->val = val;
      np->next = p;
      return np;
   } else {
      p->next = insert(p, val);
      return p;
   }
}
```

```
   test    esi, esi
   je      .L2
   cmp     dword [esi], ebx
   jle     .L3
.L2:
   mov     dword [esp], 8
   call    malloc
   mov     dword [eax], ebx
   mov     dword [eax+4], esi
   mov     ebx, dword [ebp-8]
   mov     esi, dword [ebp-4]
   mov     esp, ebp
   pop     ebp
   ret
.L3:
```

© 2013 МГУ/ВМиК/СП

Пример вызова malloc

```
p_chain insert(p_chain p, int val) {
   if ((0 == p) || (p->val > val)) {
      p_chain np =
         (p_chain)malloc(sizeof(t_chain));
      np->val = val;
      np->next = p;
      return np;
   } else {
      p->next = insert(p, val);
      return p;
   }
}
```

```
   ; …
.L3:
   mov     dword [esp+4], ebx
   mov     dword [esp], esi
   call    insert
   mov     dword [esi+4], eax
   mov     eax, esi
   mov     ebx, dword [ebp-8]
   mov     esi, dword [ebp-4]
   mov     esp, ebp
   pop     ebp
   ret
```

© 2013 МГУ/ВМиК/СП