© 2012 МГУ/ВМиК/СП

Лекция 12

21 марта

© 2012 МГУ/ВМиК/СП

Промежуточные итоги

- Типы данных языка Си: скалярные типы, составные типы
- Машинные типы данных
- Вычисление выражений
- Арифметические, логические, битовые команды
- Управляющие операторы
- Условная и безусловная передача управления

© 2012 МГУ/ВМиК/СП

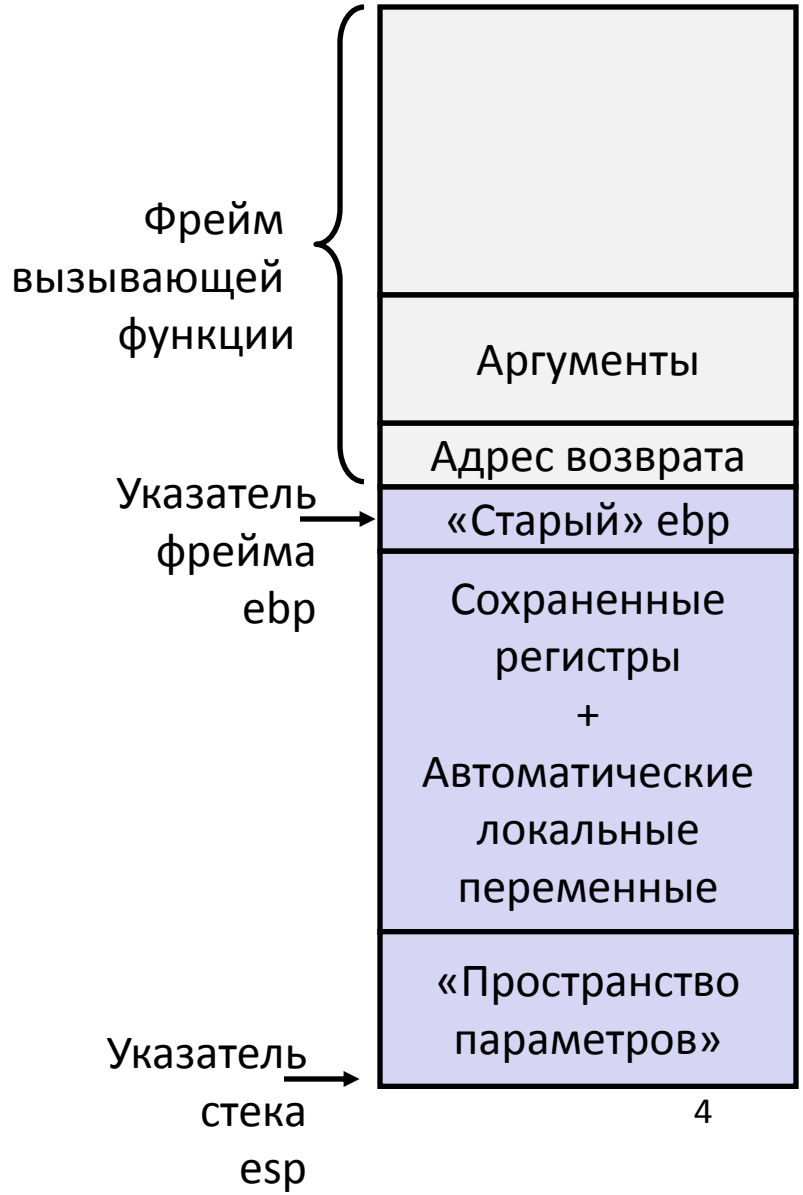
Далее

- Функции
  - Примеры …
  - Рекурсия
  - Выравнивание стека
  - Различные соглашения о вызове функций
  - Отказ от указателя фрейма
  - Переменное число параметров
  - Вызов по указателю
- Операции над строками
- Числа с плавающей точкой

© 2012 МГУ/ВМиК/СП

Поддержка функций на уровне аппаратуры

- Аппаратный стек
  - Команды PUSH и POP
  - Регистр ESP
  - Стек растет вниз
- Вызов/возврат
  - Команды CALL и RET
- Состояние текущего вызова
  - Фрейм – блок памяти на стеке
  - Регистр EBP
  - Содержимое
    - Фактические аргументы/формальные параметры
    - Адрес возврата
    - Автоматические локальные переменные
    - Вспомогательные переменные

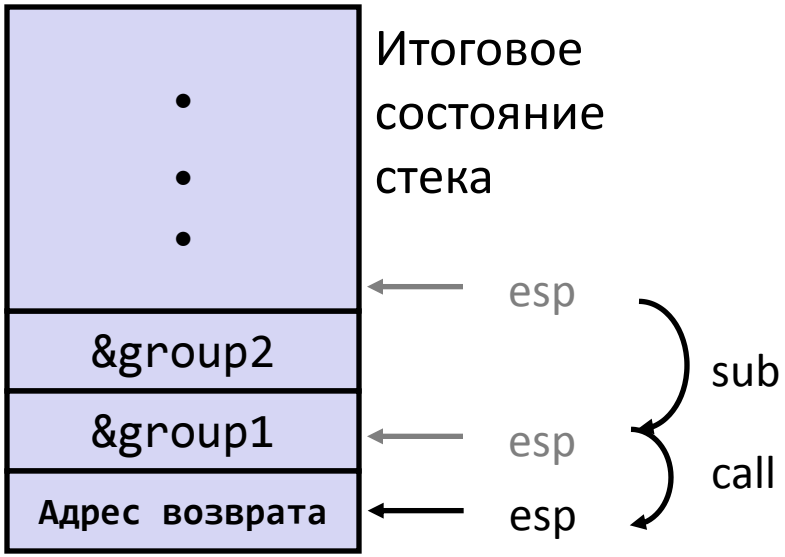


© 2012 МГУ/ВМиК/СП

```
int group1 = 101;
int group2 = 106;

void call_swap() {
  swap(&group1, &group2);
}
```

```
void swap(int *xp, int *yp) {
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```



Вызываем swap из call_swap

```
call_swap:
    • • •
    sub     esp, 8
    mov     dword [esp + 4], group2
    mov     dword [esp], group1
    call    swap
    • • •
```

© 2012 МГУ/ВМиК/СП

```c
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

```
swap:
    push    ebp
    mov     ebp, esp
    push    ebx

    mov     edx, dword [ebp + 8]
    mov     ecx, dword [ebp + 12]
    mov     ebx, dword [edx]
    mov     eax, dword [ecx]
    mov     dword [edx], eax
    mov     dword [ecx], ebx

    pop     ebx
    pop     ebp
    ret
```

Пролог

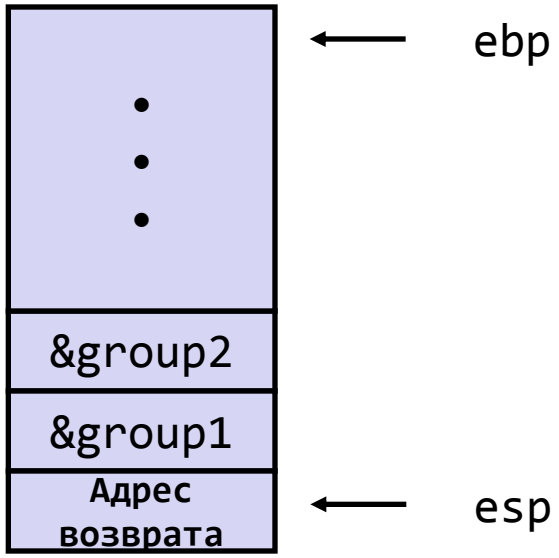
Тело функции

Эпилог

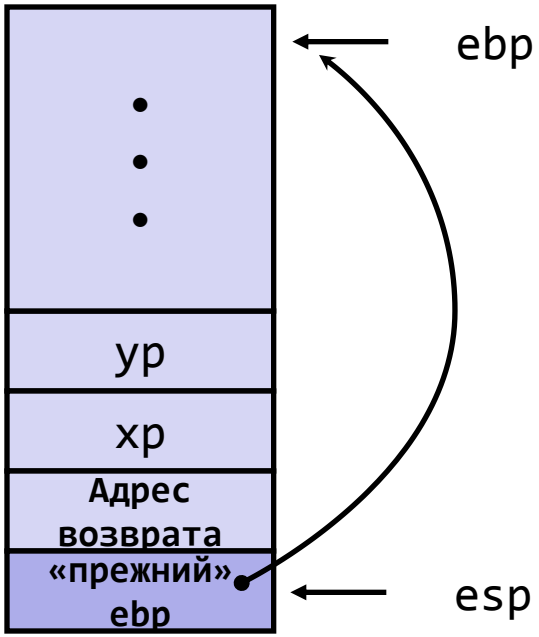
© 2012 МГУ/ВМиК/СП

Swap: как работает пролог функции. №1

Начальное состояние стека



Текущее состояние стека

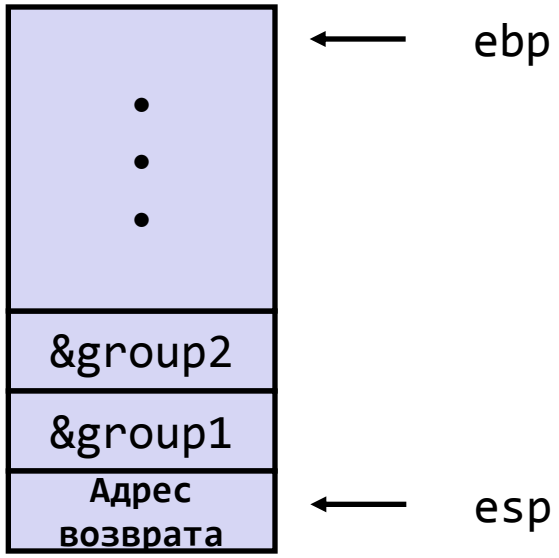



```
swap:
    push ebp        ; (1)
    mov  ebp, esp   ; (2)
    push ebx        ; (3)
```

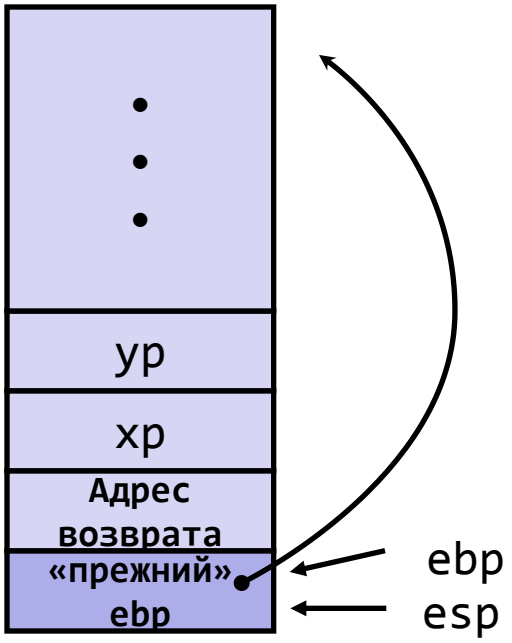
© 2012 МГУ/ВМиК/СП

Swap: как работает пролог функции. №2

Начальное состояние стека



Текущее состояние стека

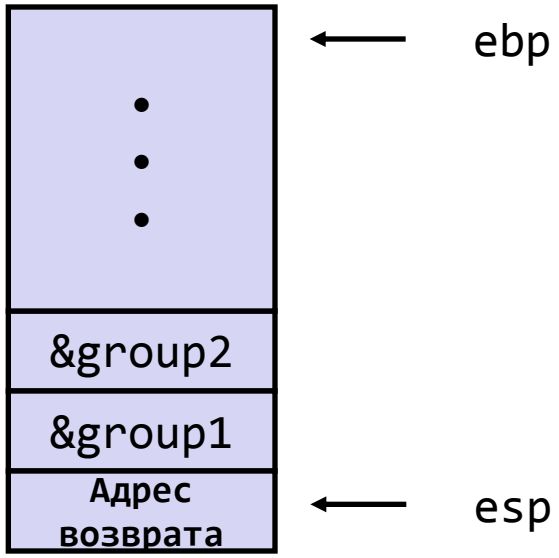



```
swap:
    push ebp        ; (1)
    mov  ebp, esp   ; (2)
    push ebx        ; (3)
```

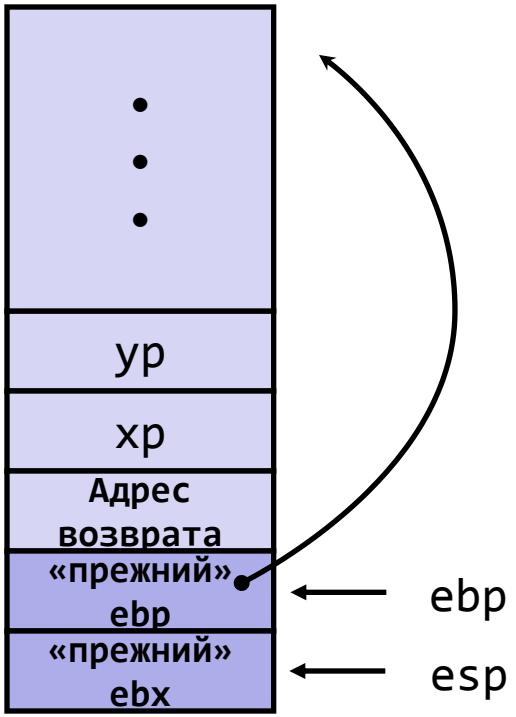
© 2012 МГУ/ВМиК/СП

Swap: как работает пролог функции. №3

Начальное состояние стека



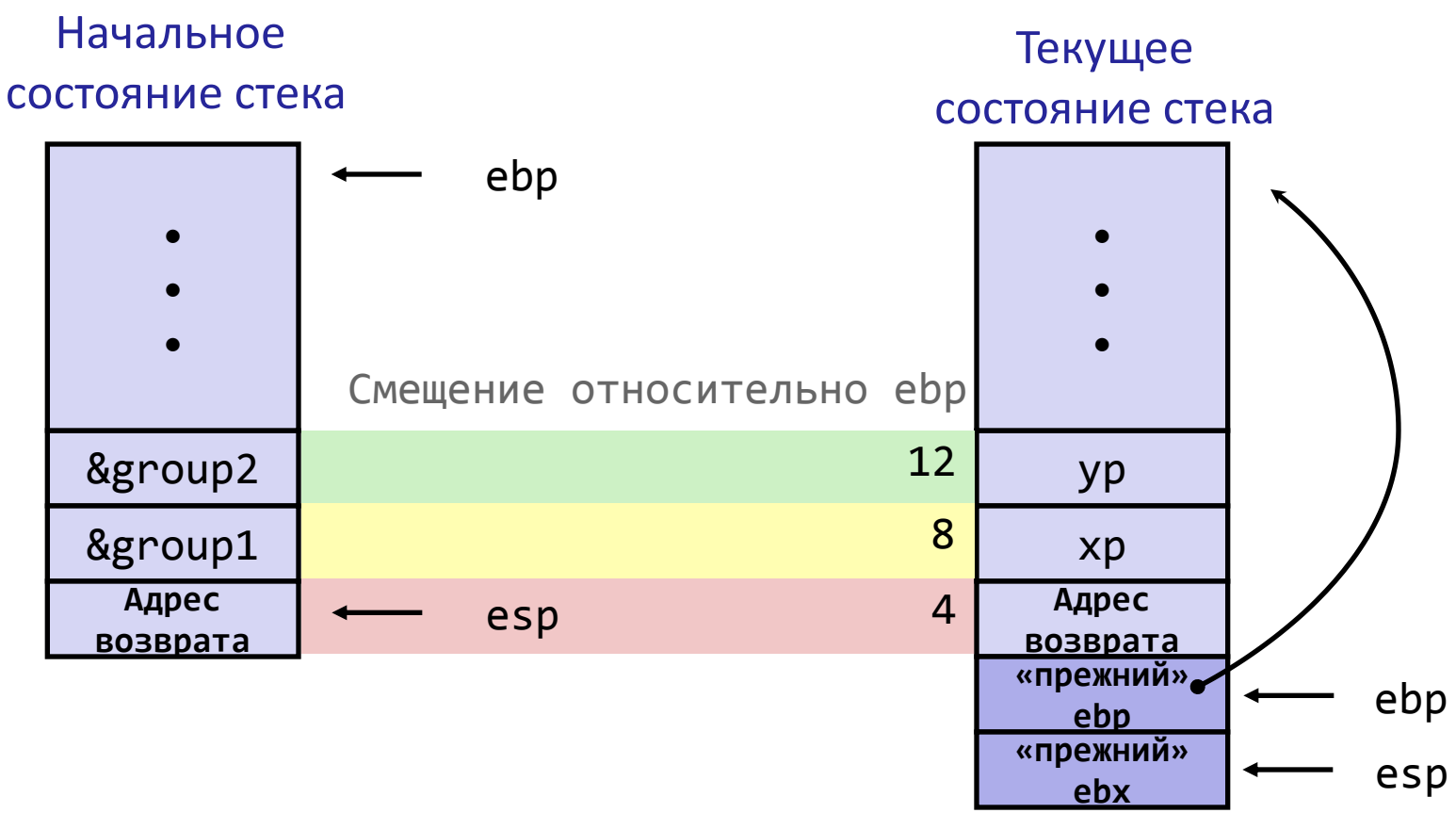
Текущее состояние стека




```
swap:
    push ebp        ; (1)
    mov  ebp, esp   ; (2)
    push ebx        ; (3)
```

© 2012 МГУ/ВМиК/СП

Тело функции swap

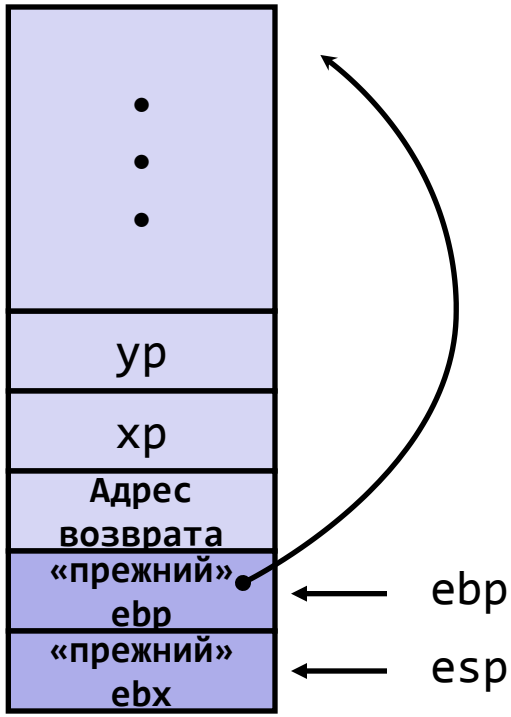



```
mov edx, dword [ebp + 8]  ; извлекаем параметр xp
mov ecx, dword [ebp + 12] ; извлекаем параметр yp
. . .
```

© 2012 МГУ/ВМиК/СП

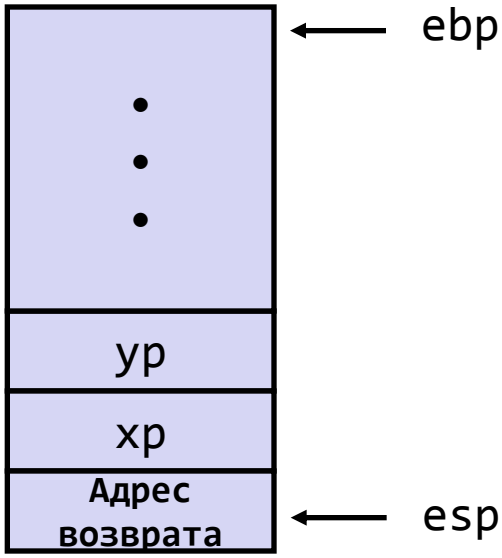
Swap: как работает эпилог функции

Стек до эпилога




```
pop    ebx
pop    ebp
```

Итоговое состояние стека

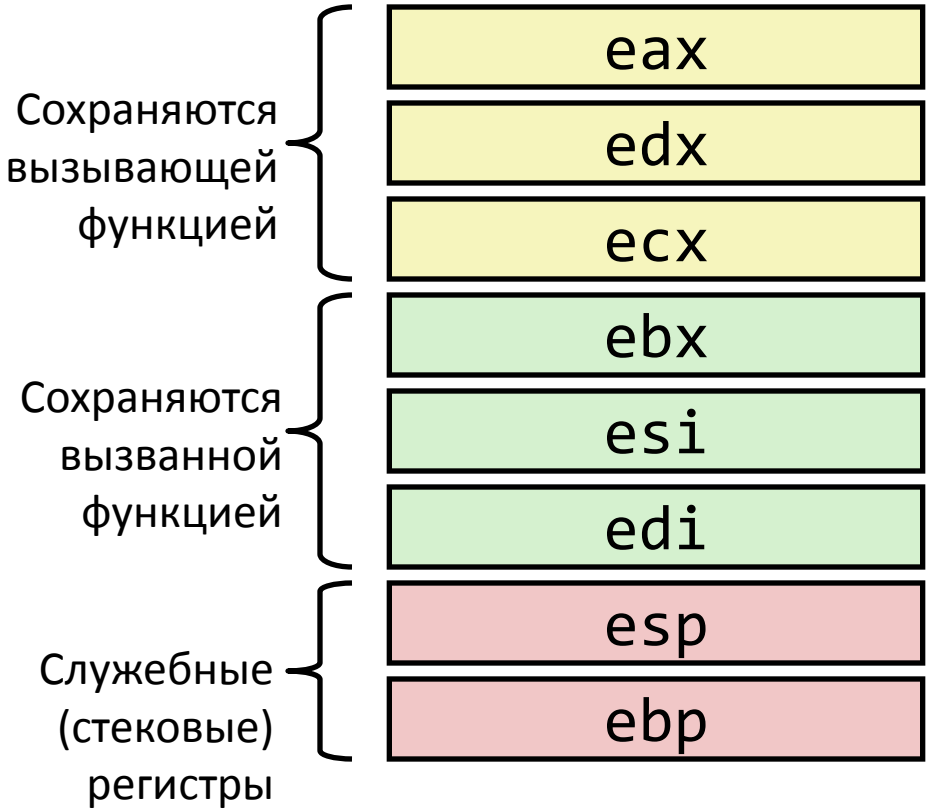


- Что происходит
  - Сохраняется и восстанавливается значение регистра ebx, …
  - … но не eax, ecx, edx

© 2012 МГУ/ВМиК/СП

Сохранение регистров в IA32/Linux+Windows

- eax, edx, ecx
  - Вызывающая функция сохраняет эти регистры перед call, если планирует использовать позже
- eax
  - Используется для возврата значения, если возвращается целый тип
- ebx, esi, edi
  - Вызванная функция сохраняет значения этих регистров, если планирует ими воспользоваться
- esp, ebp
  - Сохраняются вызванной функцией
  - Восстанавливаются перед выходом из функции



© 2012 МГУ/ВМиК/СП

Задача

тело Си-функции

```
{
    *p = d;
    return x-c;
}
```

p, d, x, c
формальные
параметры
функции

Соответствующий ассемблерный код

```
;   типовой пролог

    movsx edx, byte [ebp+12]
    mov   eax, [ebp+16]
    mov [eax], edx
    mov   eax, word [ebp+8]
    mov   edx, [ebp+20]
    sub   edx, eax
    mov   eax, edx

;   типовой эпилог
```

Требуется восстановить объявление функции: порядок параметров, их типы, тип возвращаемого значения

© 2012 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
```

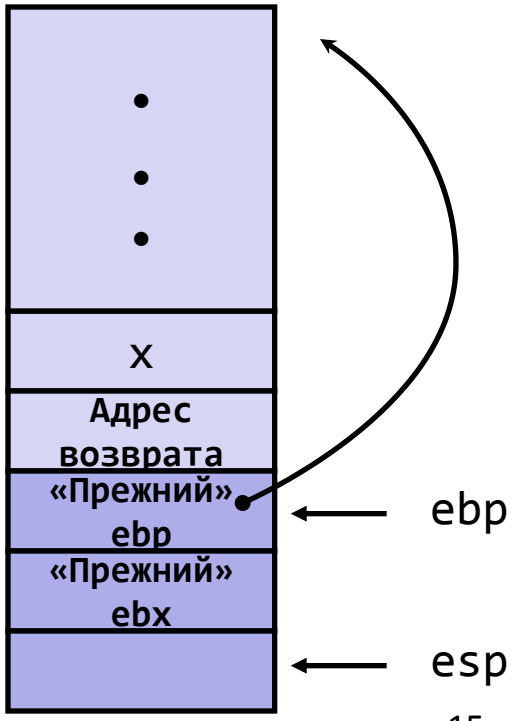
- Регистры
  - eax, edx используются без предварительного сохранения
  - ebx используется, сохраняется в начале и восстанавливается в конце

© 2012 МГУ/ВМиК/СП

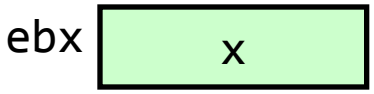
Рекурсивный вызов (1/5)

```
/* 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



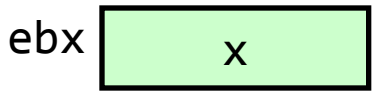
© 2012 МГУ/ВМиК/СП

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

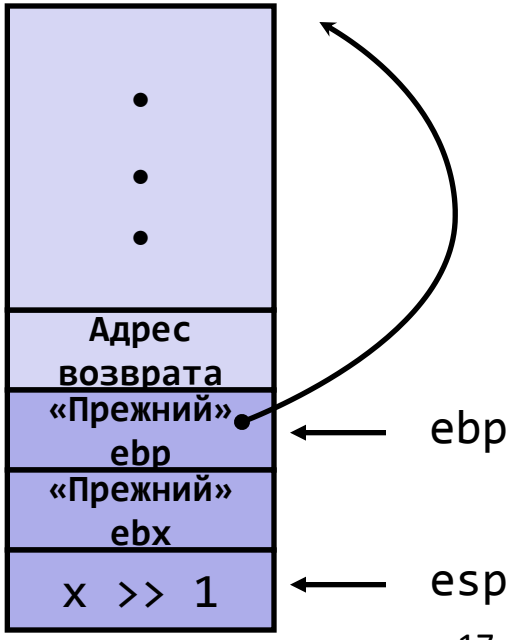


© 2012 МГУ/ВМиК/СП

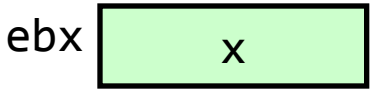
Рекурсивный вызов (3/5)

```
/* 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



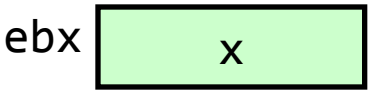
© 2012 МГУ/ВМиК/СП

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



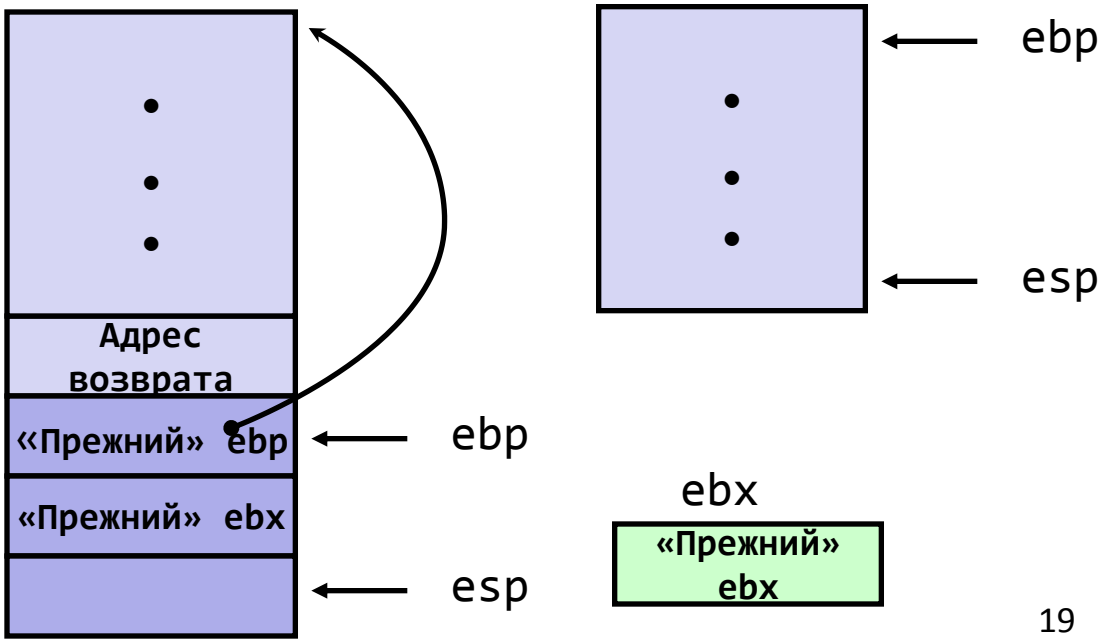
© 2012 МГУ/ВМиК/СП

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



© 2012 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

© 2012 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;
}
```

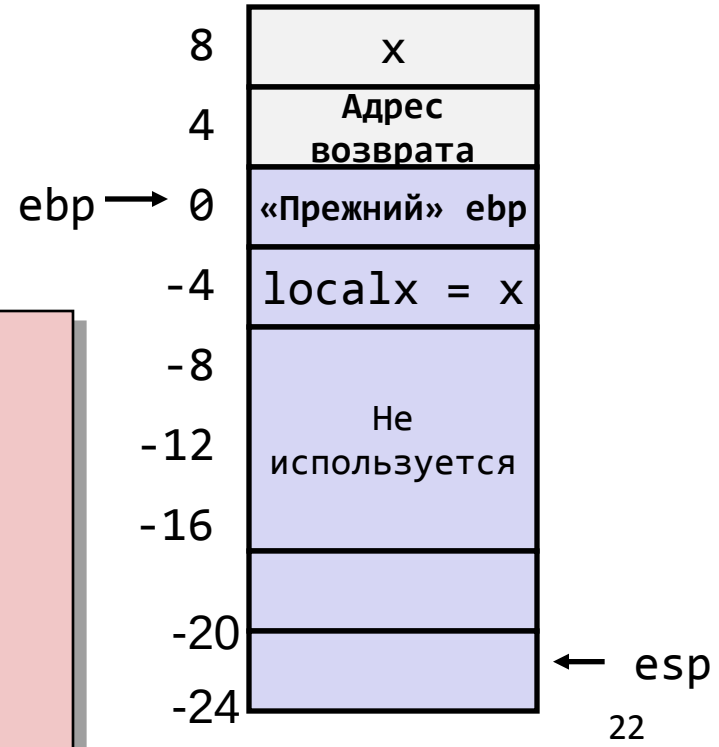
- add3 создает указатель и передает его в incrk

© 2012 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
```

© 2012 МГУ/ВМиК/СП

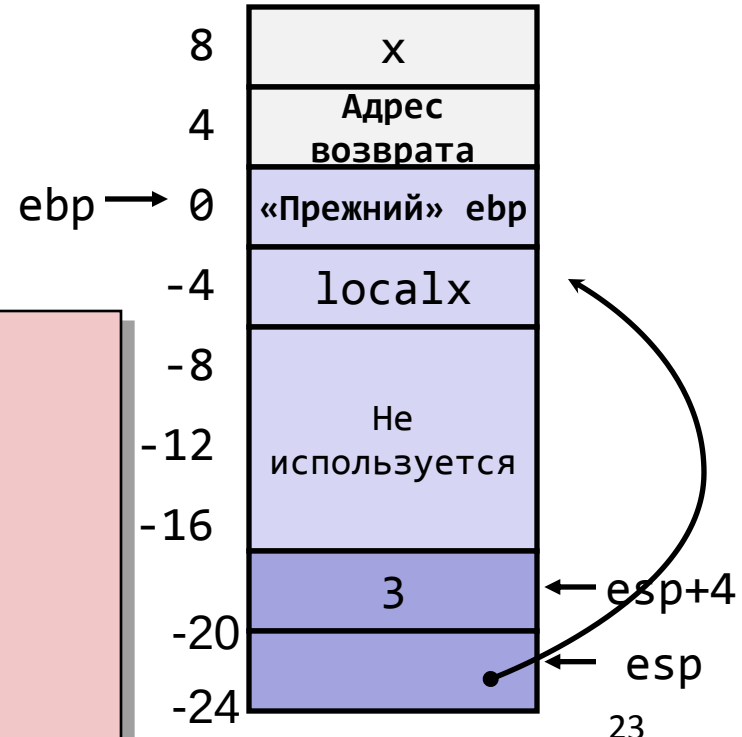
Создаем фактический параметр - указатель

```
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
```



© 2012 МГУ/ВМиК/СП

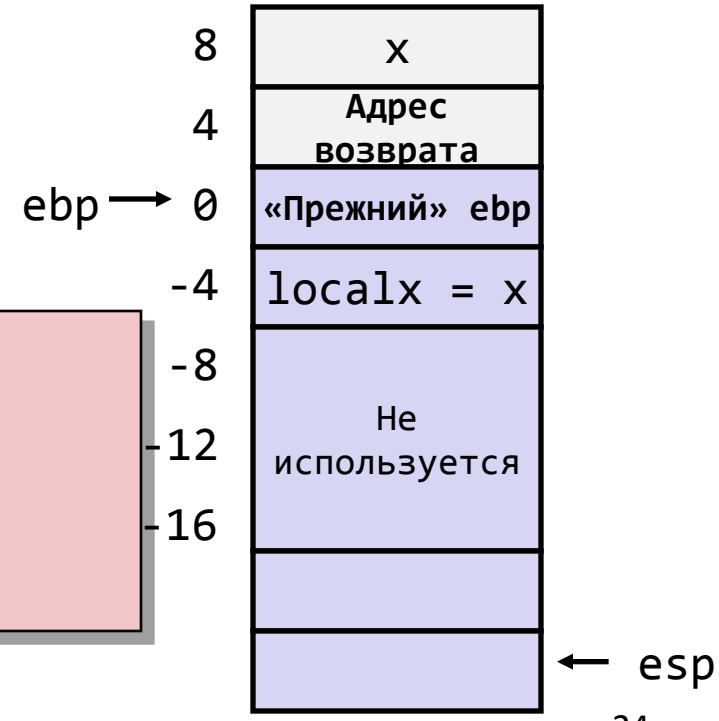
Возвращаемое значение – локальная переменная

```
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
```



© 2012 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