© 2014 МГУ/ВМК/СП

Лекция 12

22 марта

© 2014 МГУ/ВМК/СП

Типы данных языка Си
Итоги

- Размещение переменных
  - Классы памяти: автоматическая, статическая, динамическая
  - Регистр вместо памяти (при определенных условиях)
- Массивы в языке Си
  - Непрерывная последовательность байт в памяти
  - Выравнивание всего массива удовлетворяет требования к выравниванию для каждого его элемента
  - Имя массива – указатель на его первый элемент
  - Нет никаких проверок выхода за границы
- Структуры
  - Память под поля выделяется в порядке объявления этих полей
  - Помещаются пропуски между полями и в конце всей структуры для с целью выравнивания
- Объединения
  - Объявленные поля перекрываются в памяти
  - Способ жестокого обмана системы типов языка Си

© 2014 МГУ/ВМК/СП

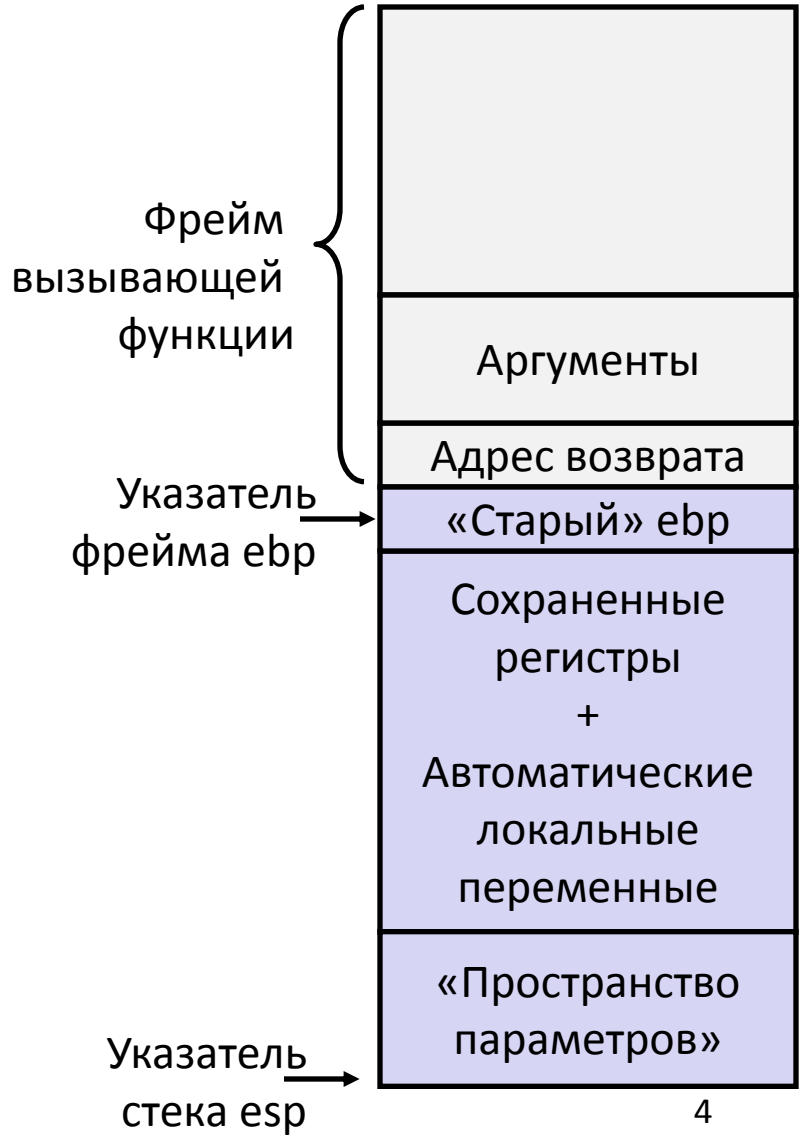
Далее…

- ***Функции***
  - ***Рекурсия***
  - ***Выравнивание стека***
  - ***Различные соглашения о вызове функций***
    - ***cdecl/stdcall/fastcall, отказ от указателя фрейма***
    - ***Соглашение вызова для x86-64***
  - ***Переменное число параметров***
  - ***Вызов по указателю***
  - ***Ассемблерные вставки***
  - ***Переполнение буфера, эксплуатация ошибок, механизмы защиты***
- Операции над строками
- Организация динамической памяти
- Числа с плавающей точкой

© 2014 МГУ/ВМК/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

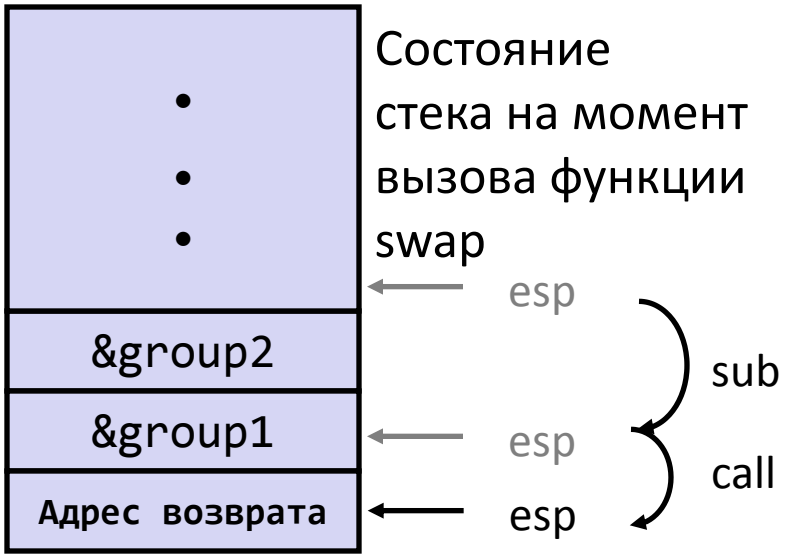


© 2014 МГУ/ВМК/СП

```
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
    • • •
```

© 2014 МГУ/ВМК/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

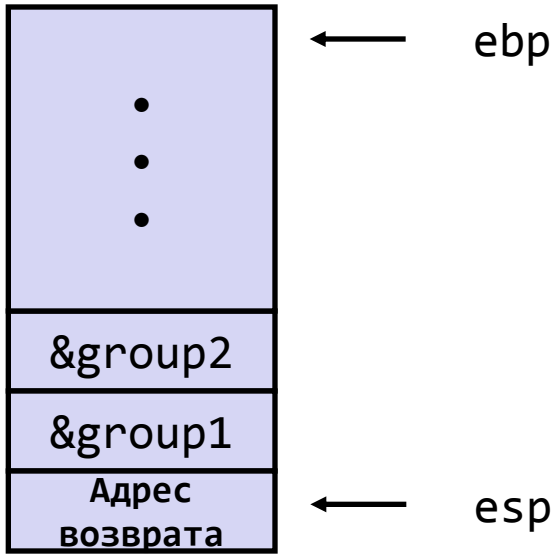
Тело функции

Эпилог

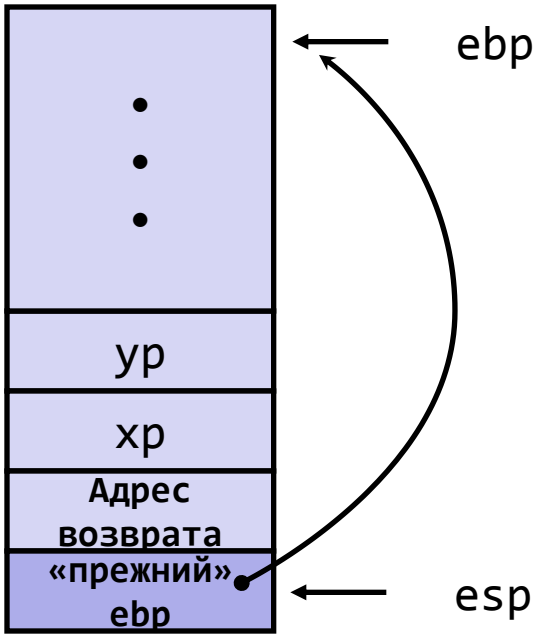
© 2014 МГУ/ВМК/СП

Swap: как работает пролог функции. №1

Начальное состояние стека



Текущее состояние стека

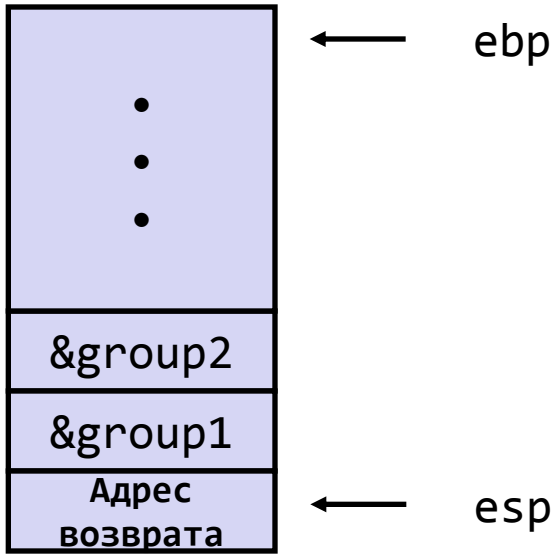



```
swap:
    push ebp        ; (1)
    mov  ebp, esp   ; (2)
    push ebx        ; (3)
```

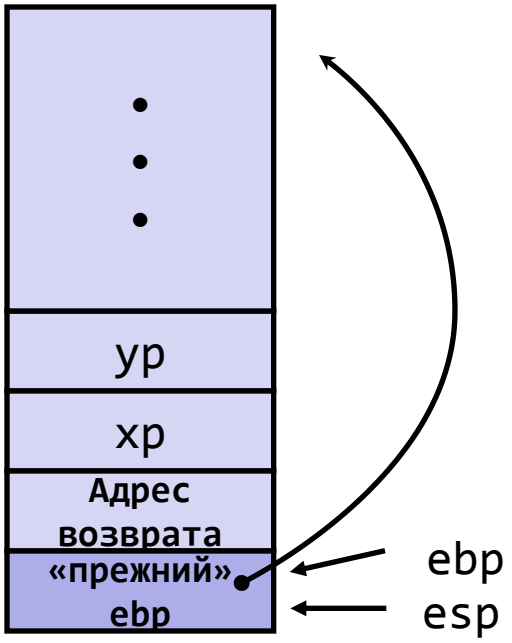
© 2014 МГУ/ВМК/СП

Swap: как работает пролог функции. №2

Начальное состояние стека



Текущее состояние стека

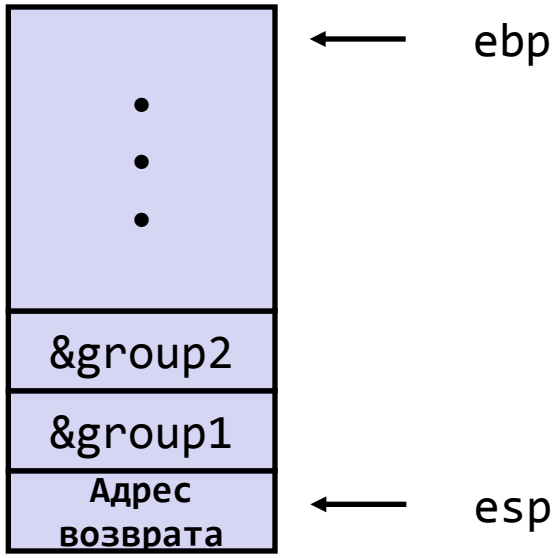



```
swap:
    push ebp        ; (1)
    mov  ebp, esp   ; (2)
    push ebx        ; (3)
```

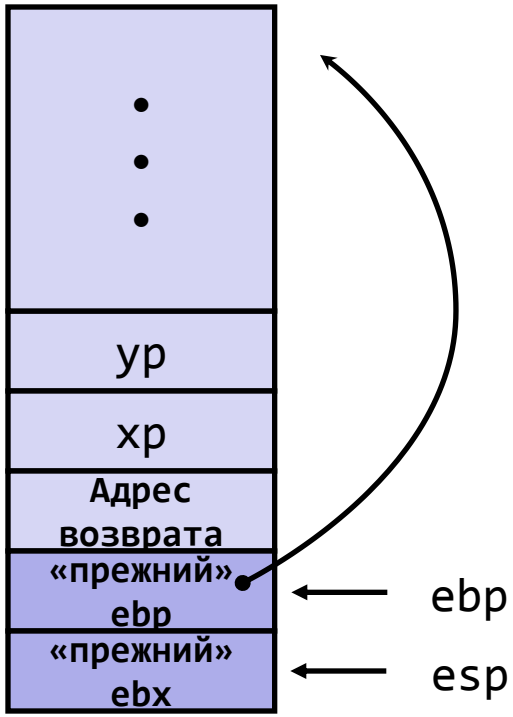
© 2014 МГУ/ВМК/СП

Swap: как работает пролог функции. №3

Начальное состояние стека



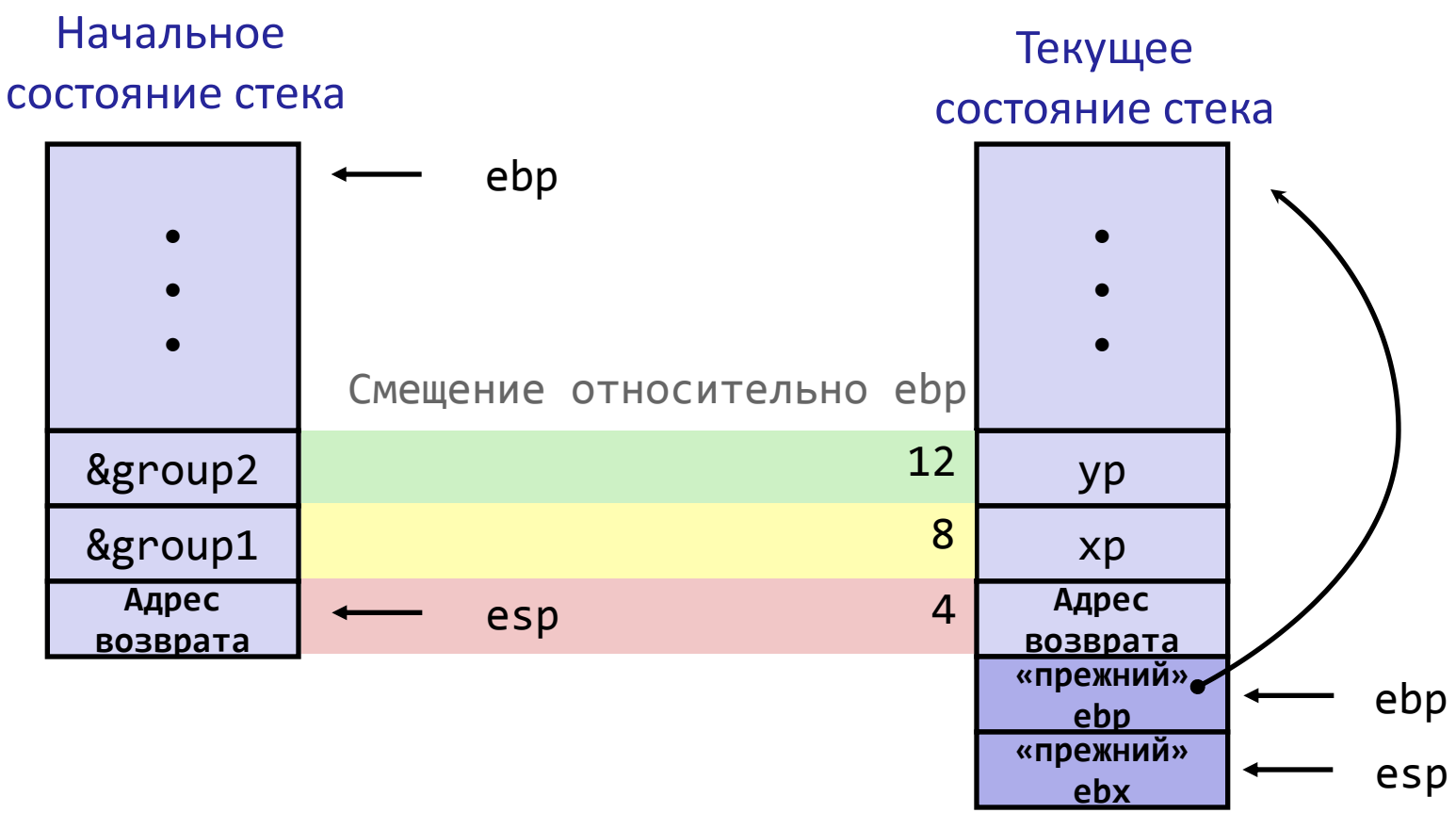
Текущее состояние стека




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

© 2014 МГУ/ВМК/СП

Тело функции swap

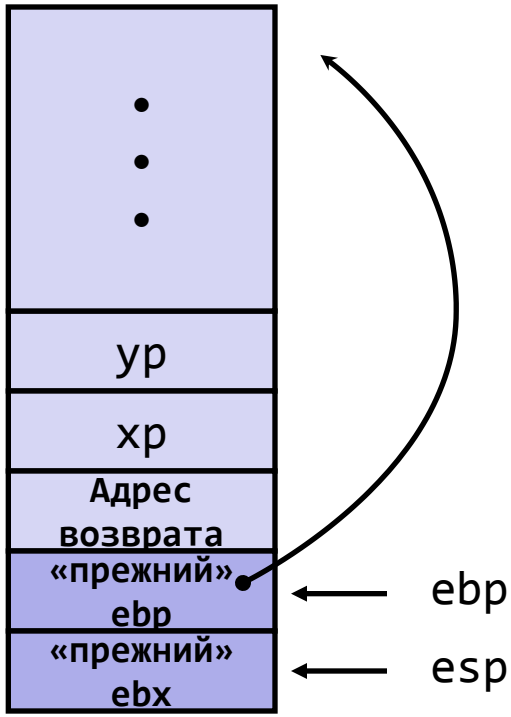



```
mov edx, dword [ebp + 8]  ; извлекаем параметр xp
mov ecx, dword [ebp + 12] ; извлекаем параметр yp
. . .
```

© 2014 МГУ/ВМК/СП

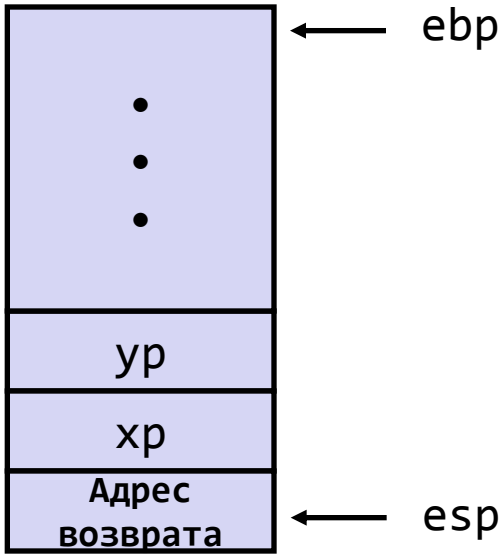
Swap: как работает эпилог функции

Стек до эпилога



pop ebx
pop ebp

Итоговое состояние стека

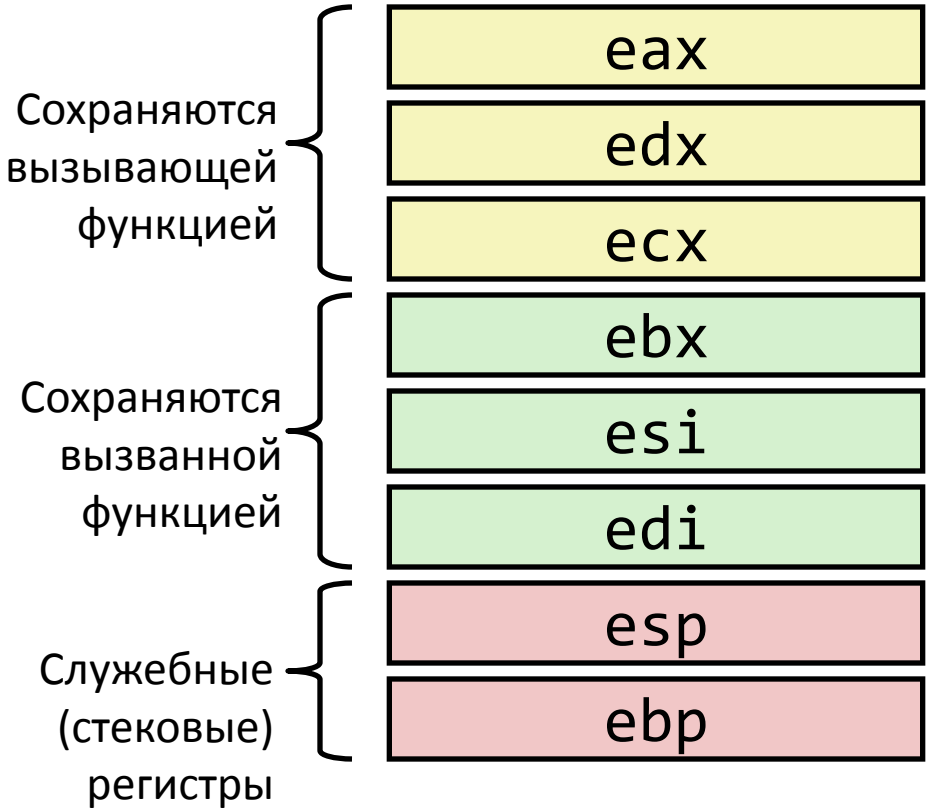


- Что происходит
  - Сохраняется и восстанавливается значение регистра ebx, …
  - но не eax, ecx, edx

© 2014 МГУ/ВМК/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



© 2014 МГУ/ВМК/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;
}
```

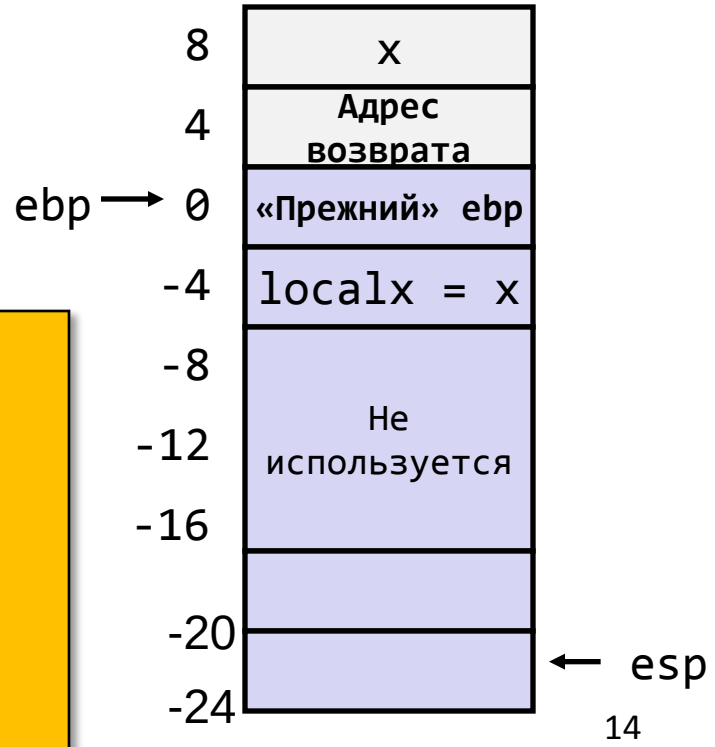
- add3 создает указатель и передает его в incrk

© 2014 МГУ/ВМК/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
```

© 2014 МГУ/ВМК/СП

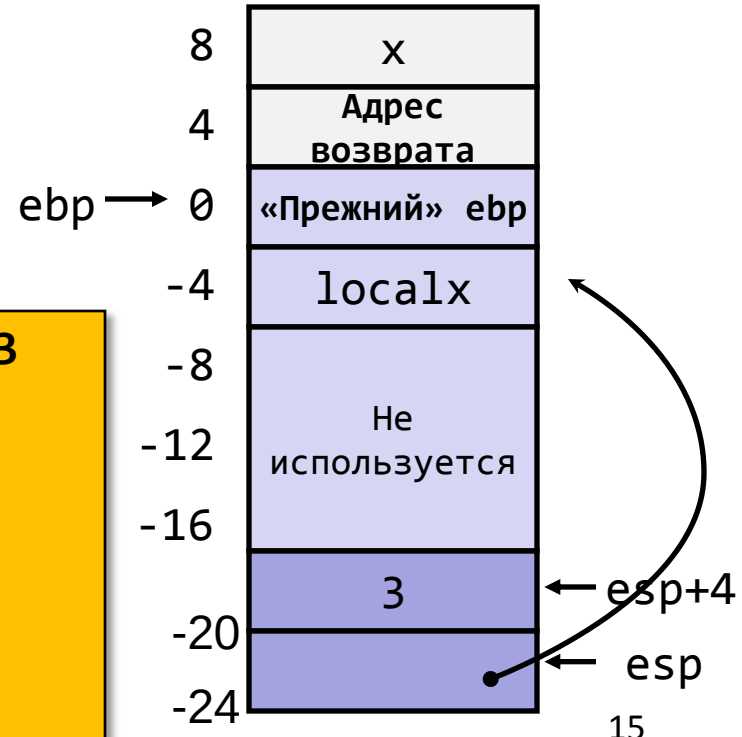
Создаем фактический параметр - указатель

```
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
```



© 2014 МГУ/ВМК/СП

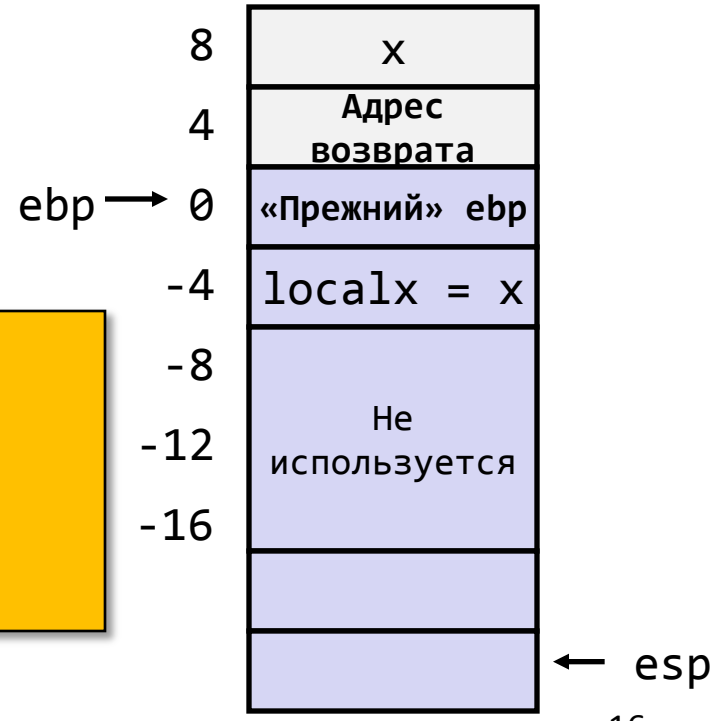
Возвращаемое значение – локальная переменная

```
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
```



© 2014 МГУ/ВМК/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
```

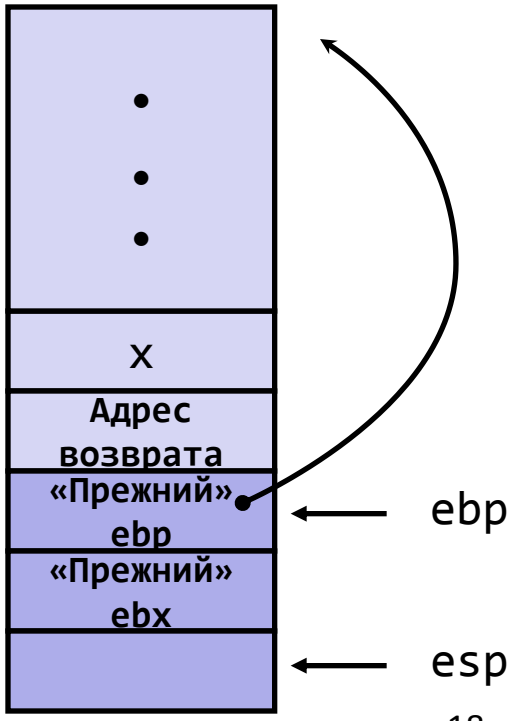
- Регистры
  - eax, edx используются без предварительного сохранения
  - ebx используется, сохраняется в начале и восстанавливается в конце

© 2014 МГУ/ВМК/СП

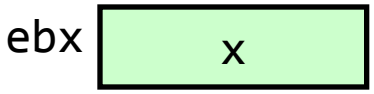
Рекурсивный вызов (1/5)

```
/* 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



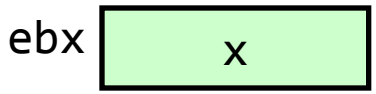
© 2014 МГУ/ВМК/СП

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



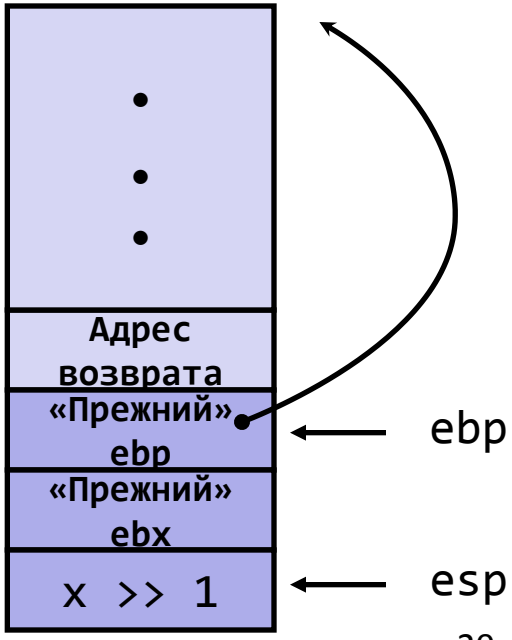
© 2014 МГУ/ВМК/СП

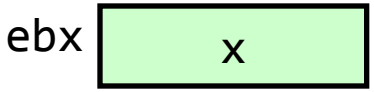
Рекурсивный вызов (3/5)

```
/* 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





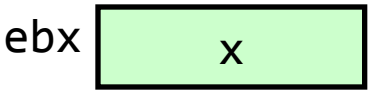
© 2014 МГУ/ВМК/СП

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



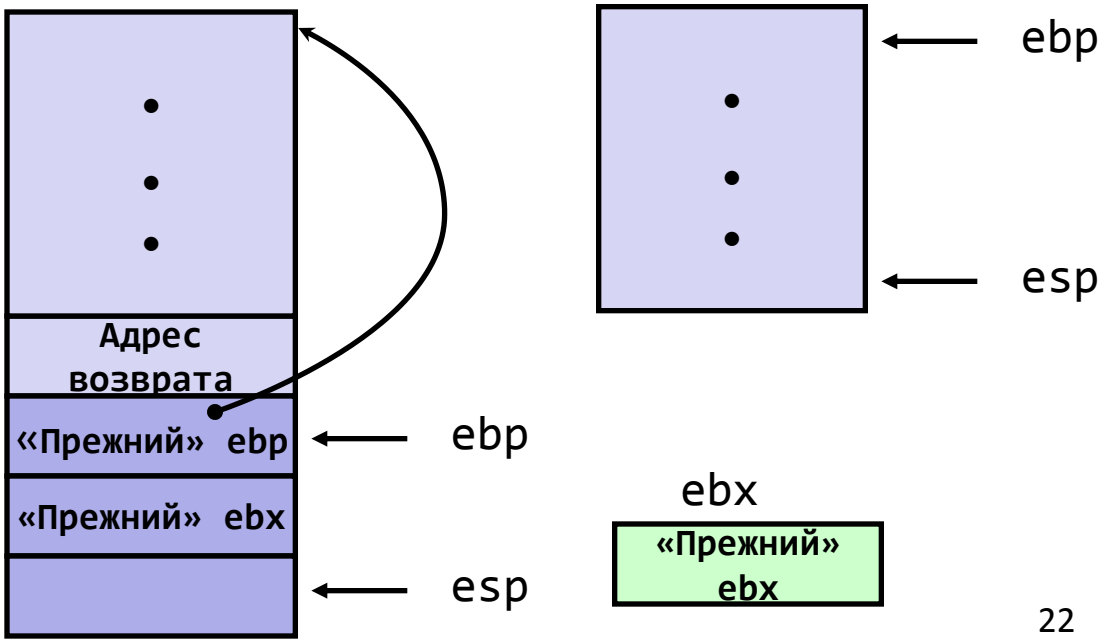
© 2014 МГУ/ВМК/СП

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



© 2014 МГУ/ВМК/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

© 2014 МГУ/ВМК/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

© 2014 МГУ/ВМК/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
    movsx 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

© 2014 МГУ/ВМК/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
  - ???

© 2014 МГУ/ВМК/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/

© 2014 МГУ/ВМК/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
```

© 2014 МГУ/ВМК/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)
```

© 2014 МГУ/ВМК/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)
```