Лекция 0xB

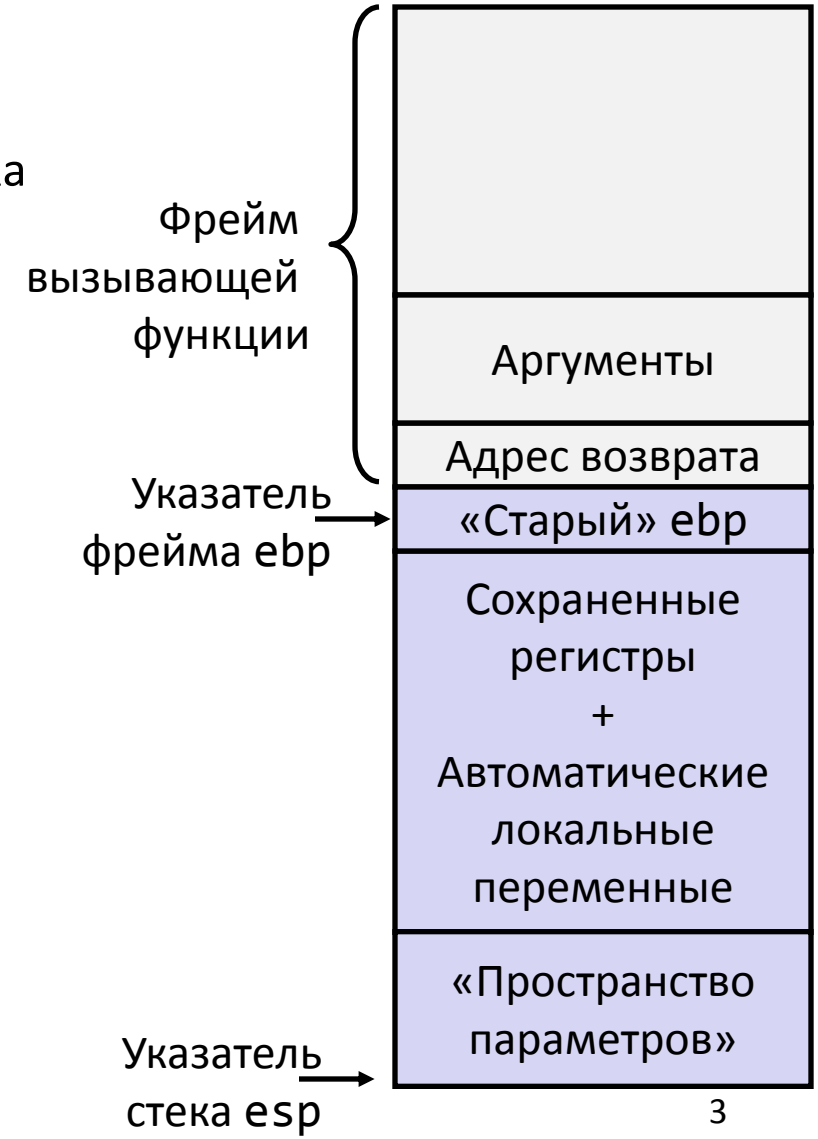
20 марта

Далее…

- ***Функции***
  - ***Соглашение CDECL***
    - ***Рекурсия***
  - ***Что происходит в Си-программе до и после функции main***
  - ***Выравнивание стека***
  - ***Различные соглашения о вызове функций***
    - ***cdecl/stdcall/fastcall, отказ от указателя фрейма***
    - ***Соглашение вызова для x86-64***
  - ***Переполнение буфера, эксплуатация ошибок, механизмы защиты***
- Организация динамической памяти
- Числа с плавающей точкой

Поддержка функций на уровне аппаратуры

- Аппаратный стек
  - Регистр ESP указывает на верхушку стека
  - Стек растет вниз
  - Команды PUSH и POP, сложение и вычитание констант из ESP
- Вызов/возврат
  - Команды CALL и RET
- Состояния выполняющихся функций
  - На стеке размещаются фреймы
  - Каждый фрейм хранит текущее состояние вызова функции
  - На верхнюю границу указывает EBP
  - Содержимое
    - Фактические аргументы/формальные параметры
    - Адрес возврата
    - Автоматические локальные переменные
    - Вспомогательные переменные



Поддержка функций на уровне соглашений: соглашение CDECL

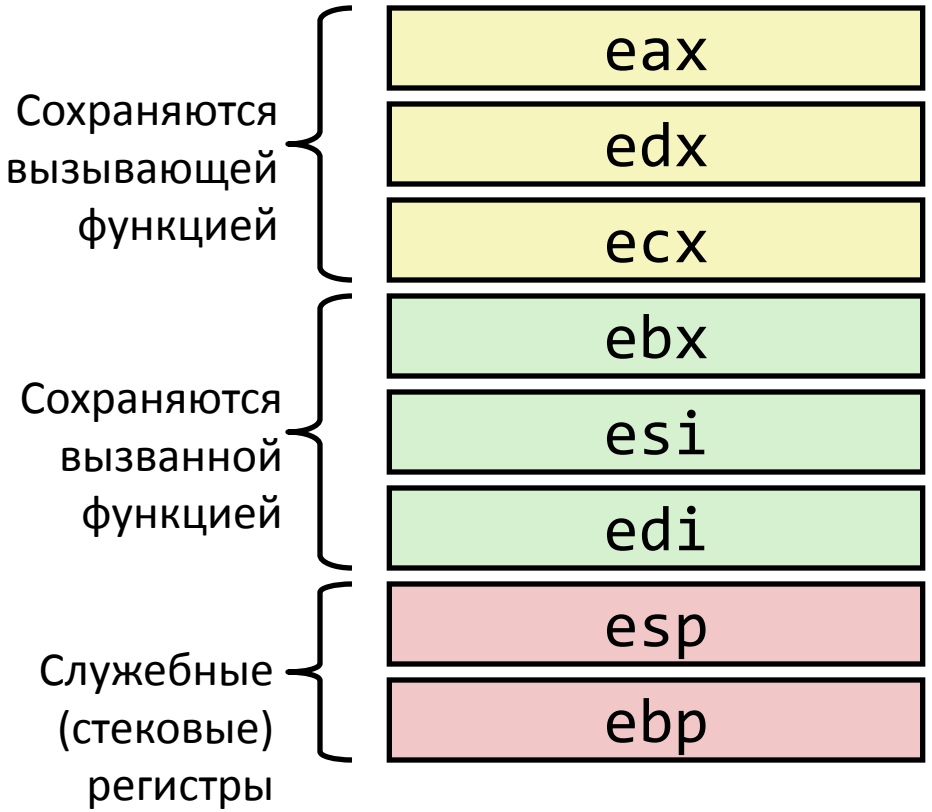
- Размещение параметров
  - На стеке, с обязательным выравниванием по 4-х байтной границе
- Порядок параметров
  - «Обратный», от «верхушки» стека ко «дну»: сразу над адресом возврата размещен первый параметр, затем второй, и т.д.
- Какие регистры могут быть использованы вызванной функцией без предварительного сохранения
  - EAX, EDX, ECX
- Очистка стека от аргументов после вызова
  - Очищает вызывающая функция
  - Поскольку cdecl в большинстве случаев не предполагает изменение границ фрейма во время работы функции, очистка после возврата в вызывающую функцию фактически не выполняется, а переносится в эпилог
- Возвращаемое функцией значение
  - EAX
  - EDX:EAX
  - Через память

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
  - `EDX:EAX`
- Массивы ≡ указатели
- Структуры и объединения
  - В gcc используется *гибридное* соглашение вызова, совмещающее `cdecl` и `stdcall`

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



Рекурсивная функция

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

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

```
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
```

Рекурсивный вызов (1/5)

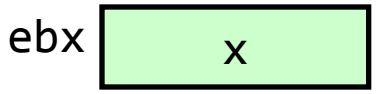
```
/* Рекурсивный popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

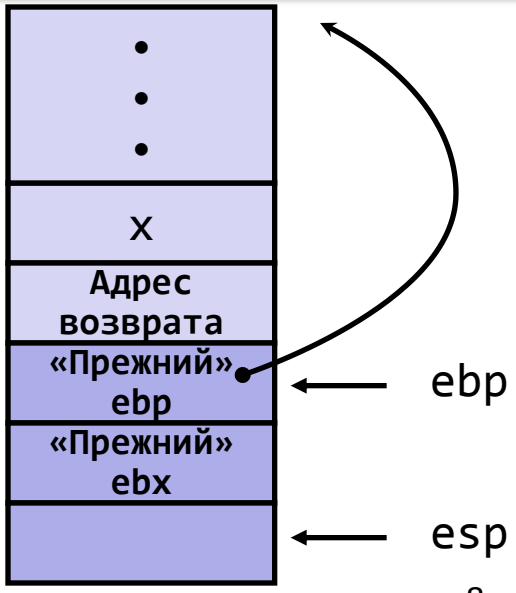
```
pcount_r:
    push   ebp
    mov    ebp, esp
    push   ebx
    sub    esp, 4

    mov    ebx, dword [ebp + 8]
    ...
```

Пролог функции

- Действия в прологе
  - Сохраняем и «переставляем» ebp
  - Сохраняем значение ebx на стеке
  - Расширяем фрейм: выделяем место для размещения аргумента рекурсивного вызова
- Размещаем значение x в ebx



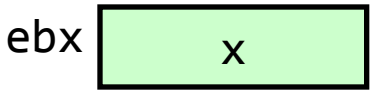


Рекурсивный вызов (2/5)

```
/* 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`

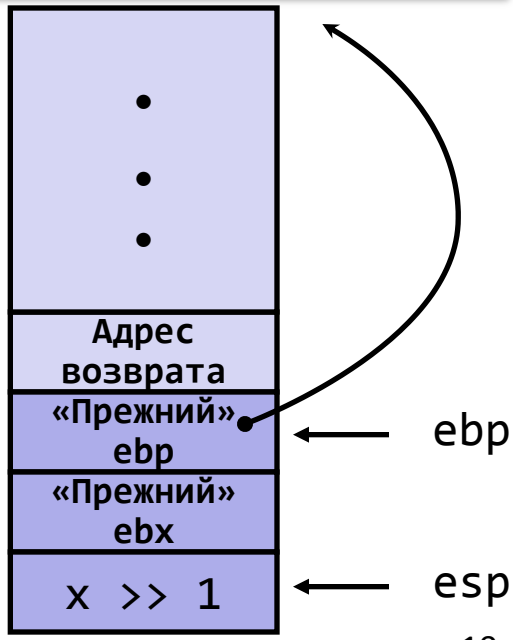


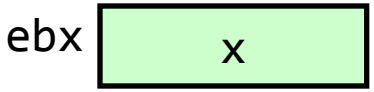
Рекурсивный вызов (3/5)

```
/* 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



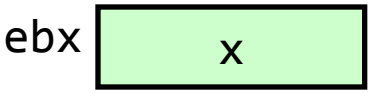


Рекурсивный вызов (4/5)

```
/* 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



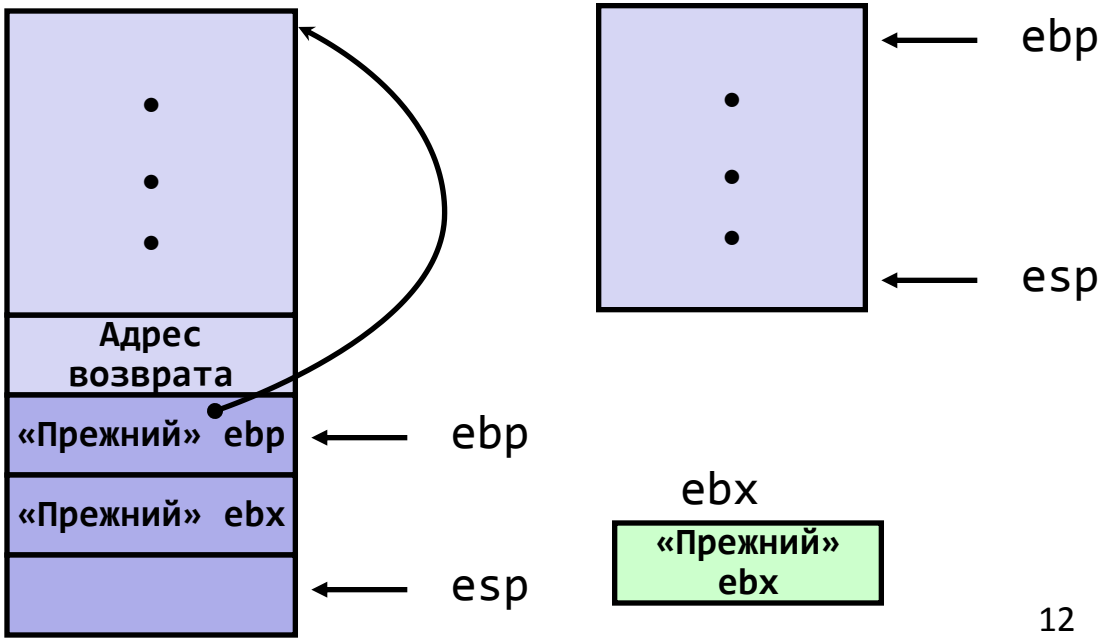
Рекурсивный вызов (5/5)

```
/* 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
```

Эпилог функции

- Действия в эпилоге
  - Восстанавливаем значения ebx и ebp
  - Восстанавливаем esp



Рекурсия – выводы

- 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

Обратная задача – восстанавливаем объявление функции (CDECL)

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