© 2018 МГУ/ВМК/СП

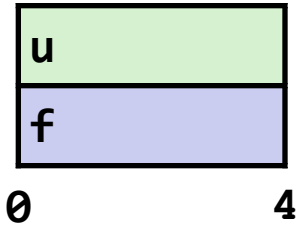
Лекция 0xB

17 марта

© 2018 МГУ/ВМК/СП

Использование объединений для доступа к отдельным битам

```
typedef union {
  float f;
  unsigned u;
} bit_float_t;
```




```
float bit2float(unsigned u) {
  bit_float_t arg;
  arg.u = u;
  return arg.f;
}
```

```
unsigned float2bit(float f) {
  bit_float_t arg;
  arg.f = f;
  return arg.u;
}
```

Тоже самое, что и `(float) u` **?**

Тоже самое, что и `(unsigned) f` **?**

© 2018 МГУ/ВМК/СП

Порядок байт

- Основная идея
  - short/long/double хранятся в памяти как последовательности из 2/4/8 байт
  - Где именно расположен старший (младший) байт?
  - Может являться проблемой при пересылке двоичных данных между машинами разной архитектуры
- Big-endian / от старшего к младшему
  - Старший байт имеет наименьший адрес
  - Sparc (до V8)
- Little-endian / от младшего к старшему
  - Младший байт имеет наименьший адрес
  - Intel x86 (IA-32)
- Переключаемый порядок байт
  - ARM, PowerPC, Alpha, SPARC V9, MIPS, PA-RISC и IA-64

© 2018 МГУ/ВМК/СП

Пример

```
union {
  unsigned char c[8];
  unsigned short s[4];
  unsigned int i[2];
  unsigned long l[1];
} dw;
```

32 бита

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
s[0]		s[1]		s[2]		s[3]	
i[0]				i[1]			
l[0]							

64 бита

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
s[0]		s[1]		s[2]		s[3]	
i[0]				i[1]			
l[0]							

© 2018 МГУ/ВМК/СП

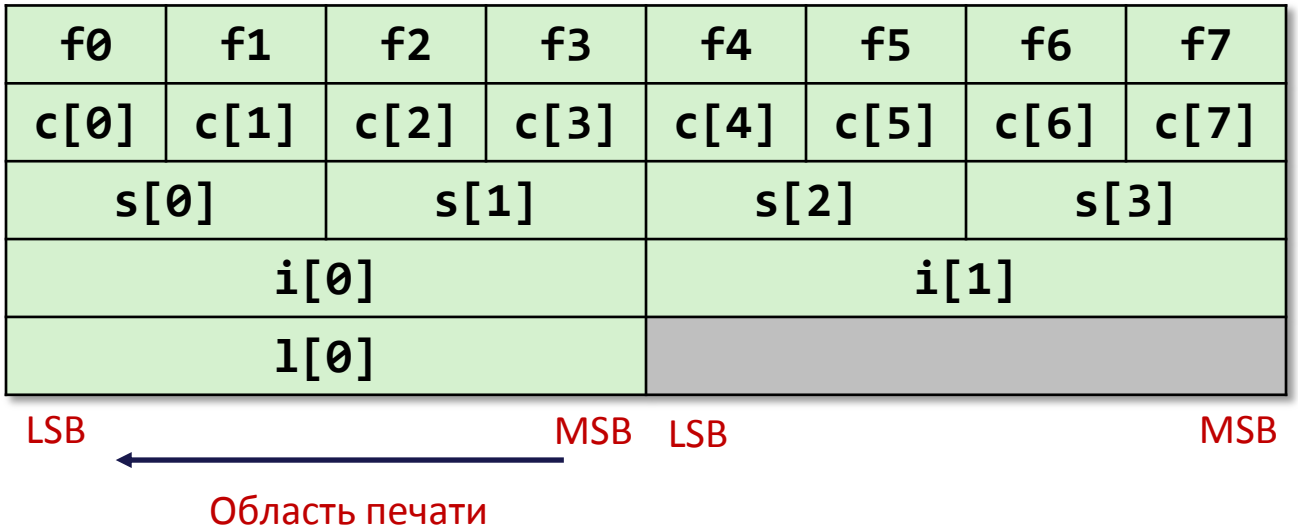
Пример (продолжение)

```
int j;
for (j = 0; j < 8; j++)
    dw.c[j] = 0xf0 + j;

printf("Characters 0-7 ==
[0x%x,0x%x,0x%x,0x%x,0x%x,0x%x,0x%x,0x%x]\n",
    dw.c[0], dw.c[1], dw.c[2], dw.c[3],
    dw.c[4], dw.c[5], dw.c[6], dw.c[7]);

printf("Shorts 0-3 == [0x%x,0x%x,0x%x,0x%x]\n",
    dw.s[0], dw.s[1], dw.s[2], dw.s[3]);

printf("Ints 0-1 == [0x%x,0x%x]\n",
    dw.i[0], dw.i[1]);

printf("Long 0 == [0x%lx]\n",
    dw.l[0]);
```

© 2018 МГУ/ВМК/СП

Порядок байт IA-32 и многих других архитектур

Порядок байт от младшего к старшему



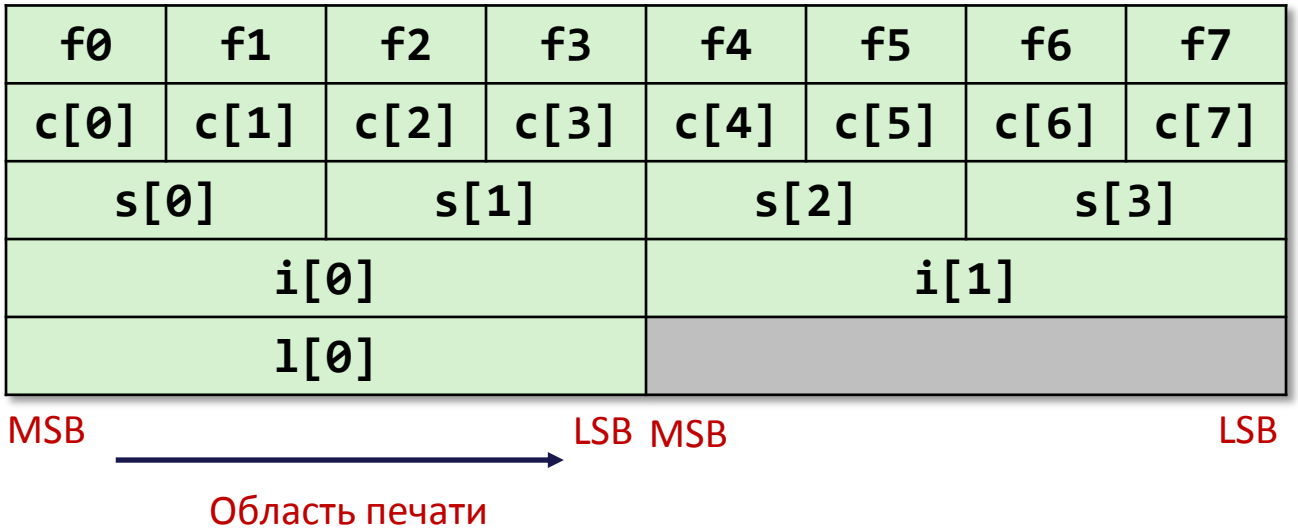
Вывод на консоль:

```
Characters 0-7 == [0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7]
Shorts     0-3 == [0xf1f0,0xf3f2,0xf5f4,0xf7f6]
Ints       0-1 == [0xf3f2f1f0,0xf7f6f5f4]
Long       0   == [0xf3f2f1f0]
```

© 2018 МГУ/ВМК/СП

Порядок байт SPARC, PPC, m68k и некоторых других архитектур

Порядок байт от старшего к младшему



Вывод на консоль:

```
Characters 0-7 == [0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7]
Shorts     0-3 == [0xf0f1,0xf2f3,0xf4f5,0xf6f7]
Ints       0-1 == [0xf0f1f2f3,0xf4f5f6f7]
Long       0   == [0xf0f1f2f3]
```

© 2018 МГУ/ВМК/СП

Порядок байт в x86-64

Порядок байт от младшего к старшему

Вывод на консоль:

```
Characters 0-7 == [0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7]
Shorts     0-3 == [0xf1f0,0xf3f2,0xf5f4,0xf7f6]
Ints       0-1 == [0xf3f2f1f0,0xf7f6f5f4]
Long       0   == [0xf7f6f5f4f3f2f1f0]
```

© 2018 МГУ/ВМК/СП

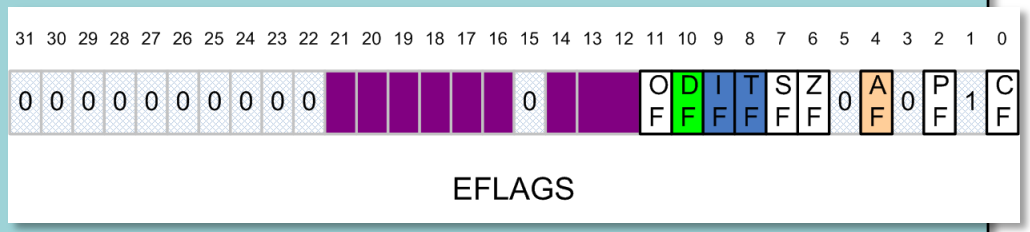
Битовые поля

- Размещение битовых полей зависит от реализации
- Битовые поля могут размещаться как справа налево, так и наоборот, в зависимости от реализации
- Битовые поля могут пересекать границы машинных слов
- Выравнивание битовых полей зависит от реализации
- К битовым полям неприменима операция & и оператор `sizeof`
- Код непереносим между различными системами

© 2018 МГУ/ВМК/СП

```
typedef union {
   unsigned int raw;
   struct {
      int CF   : 1;
      int gap1 : 1;
      int PF   : 1;
      int gap2 : 1;
      int AF   : 1;
      int gap3 : 1;
      int ZF   : 1;
...
      int gap5 : 10;
   } fields;
} t_eflags;

int checkEflagsState(t_eflags *sw) {
   t_eflags andMask = {0};
   andMask.fields.gap2 = 1;
   andMask.fields.gap3 = 1;
   andMask.fields.gap4 = 1;
   andMask.fields.gap5 = -1;
   return !(sw->raw & andMask.raw) && (sw->raw & 2);
}
```

```
checkEflagsState:
   push ebp
   mov  ebp, esp
   mov  eax, dword [ebp+8]
   mov  edx, dword [eax]
   xor  eax, eax
   test edx, -4161496
     ; 11111111_11000000_10000000_00101000b
   jne  .L3
   mov  eax, edx
   shr  eax
   and  eax, 1
.L3:
   pop     ebp
   ret
```



© 2018 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 с целью выравнивания данных
- Объединения
  - Объявленные поля перекрываются в памяти
  - Способ жестокого обмана системы типов языка Си

© 2018 МГУ/ВМК/СП

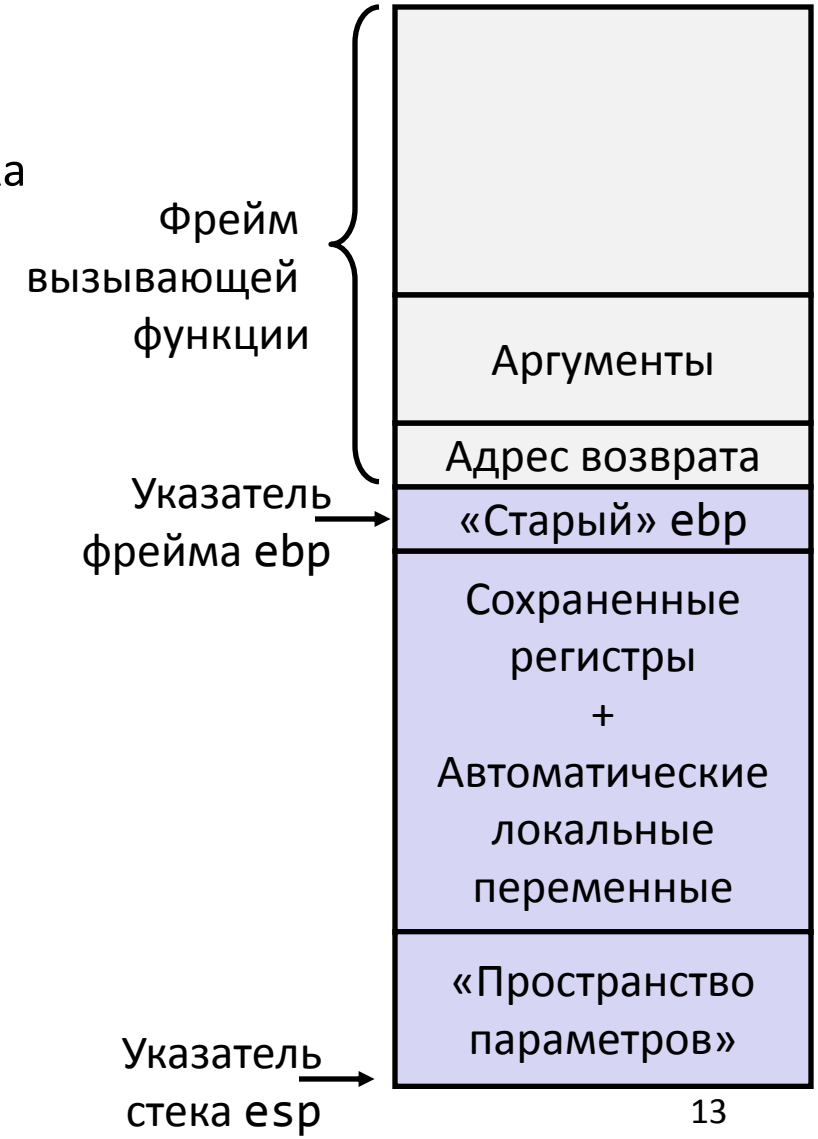
Далее…

- ***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

© 2018 МГУ/ВМК/СП

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



© 2018 МГУ/ВМК/СП

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

© 2018 МГУ/ВМК/СП

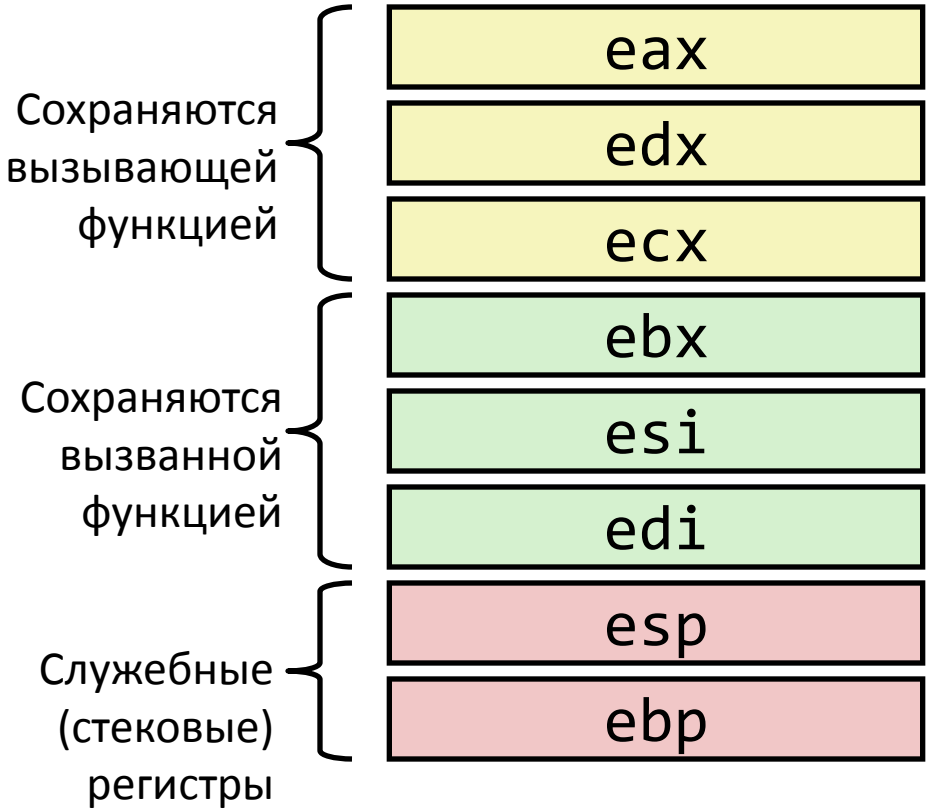
Размещение параметров и возвращаемого значения

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

© 2018 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



© 2018 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, но предварительно сохраняется в начале функции и восстанавливается в конце

© 2018 МГУ/ВМК/СП

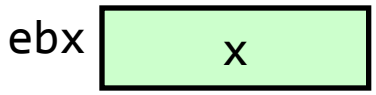
Рекурсивный вызов (1/5)

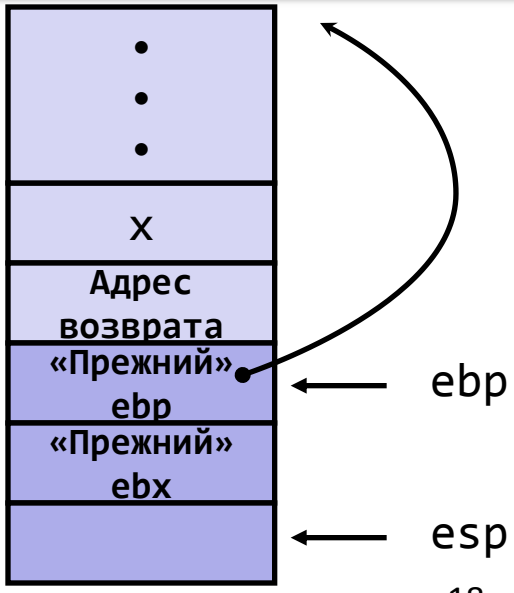
```
/* Рекурсивный popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

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

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

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





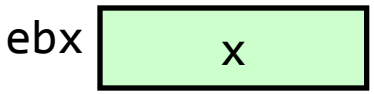
© 2018 МГУ/ВМК/СП

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`



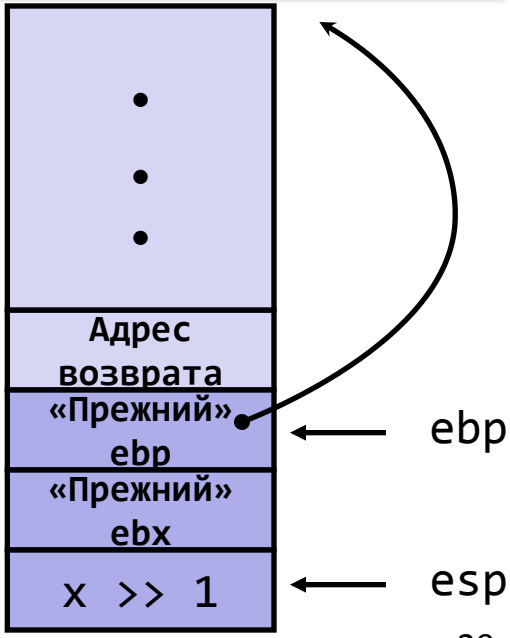
© 2018 МГУ/ВМК/СП

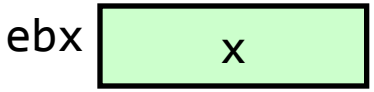
Рекурсивный вызов (3/5)

```
/* 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





© 2018 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

ebx | x

© 2018 МГУ/ВМК/СП

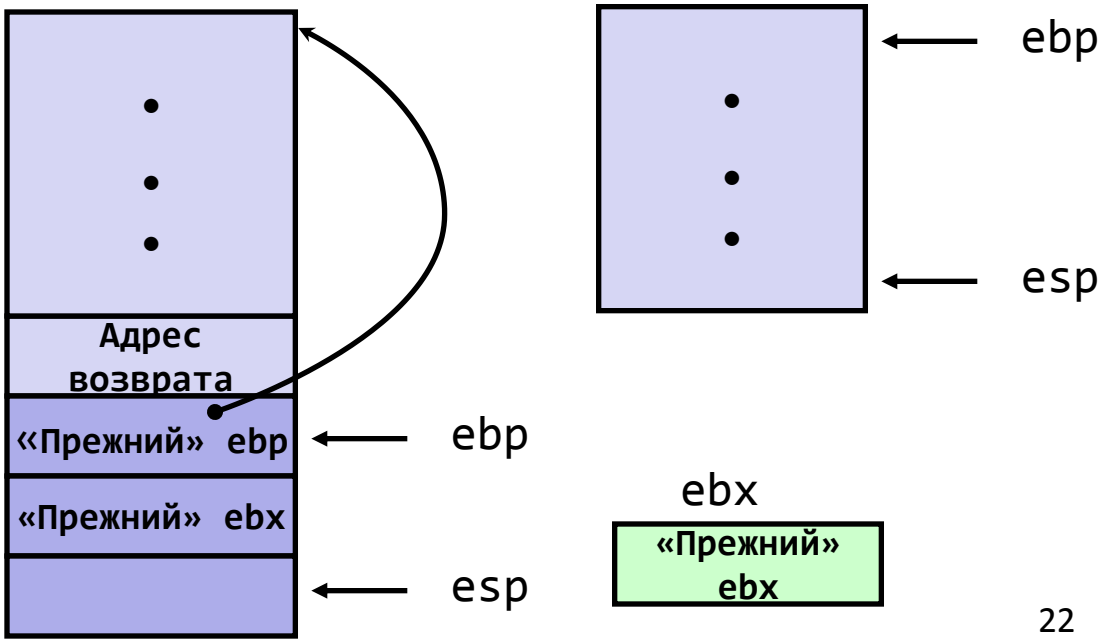
Рекурсивный вызов (5/5)

```
/* 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
```

Эпилог функции

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



© 2018 МГУ/ВМК/СП

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

© 2018 МГУ/ВМК/СП

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

© 2018 МГУ/ВМК/СП

ABI – двоичный интерфейс приложения

- Двоичный интерфейс приложения (application binary interface) задает правила взаимодействия между модулями программы на уровне исполняемого кода
  - Библиотеки и функции операционной системы (системные вызовы) также рассматриваются как модули, входящие в состав программы
- ABI определяет
  - Использование ISA и регистров процессора, организацию стека и т.п.
  - Соглашение вызова функций
  - Размещение данных в памяти, требования по выравниванию
  - Выполнение системных вызовов
  - Формат объектных файлов, состав библиотек (в случае, если ABI описывает всю среду выполнения программ)
- При соблюдении ABI можно разрабатывать модули программы на разных языках программирования

Пример ABI: http://www.sco.com/developers/devspecs/abi386-4.pdf

© 2018 МГУ/ВМК/СП

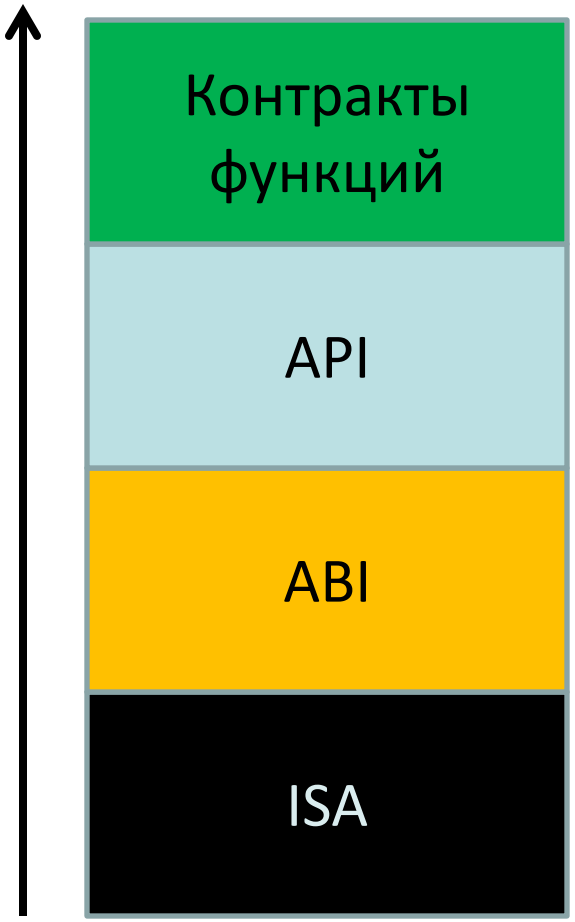
Вставляем элемент в непустой список,
Указатели на элемент и список не нулевые.

```
typedef struct chain chain;
struct chain {
   int    payload;
   chain *next;
};

chain* insert(chain* list, chain* elem) {
 if (0 == list->next) {
    ...
```

```
insert:
   push    ebx
   sub     esp, 8
   mov     ebx, dword [esp+16]
   mov     eax, dword [ebx+4]
   test    eax, eax
   je      .L22
   ...
```

```
esp   ← esp-4
[esp] ← ebx
...
```

Уровни интерфейсов



© 2018 МГУ/ВМК/СП

Функция main

```
#include <stdio.h>

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

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

- Есть ли отличия у функции main от остальных функций?
- Откуда берутся параметры argc и argv ?
- Что происходит со стеком в функции main ?

и наконец …

- Как можно воспользоваться стандартной библиотекой языка Си в ассемблерной программе?

© 2018 МГУ/ВМК/СП

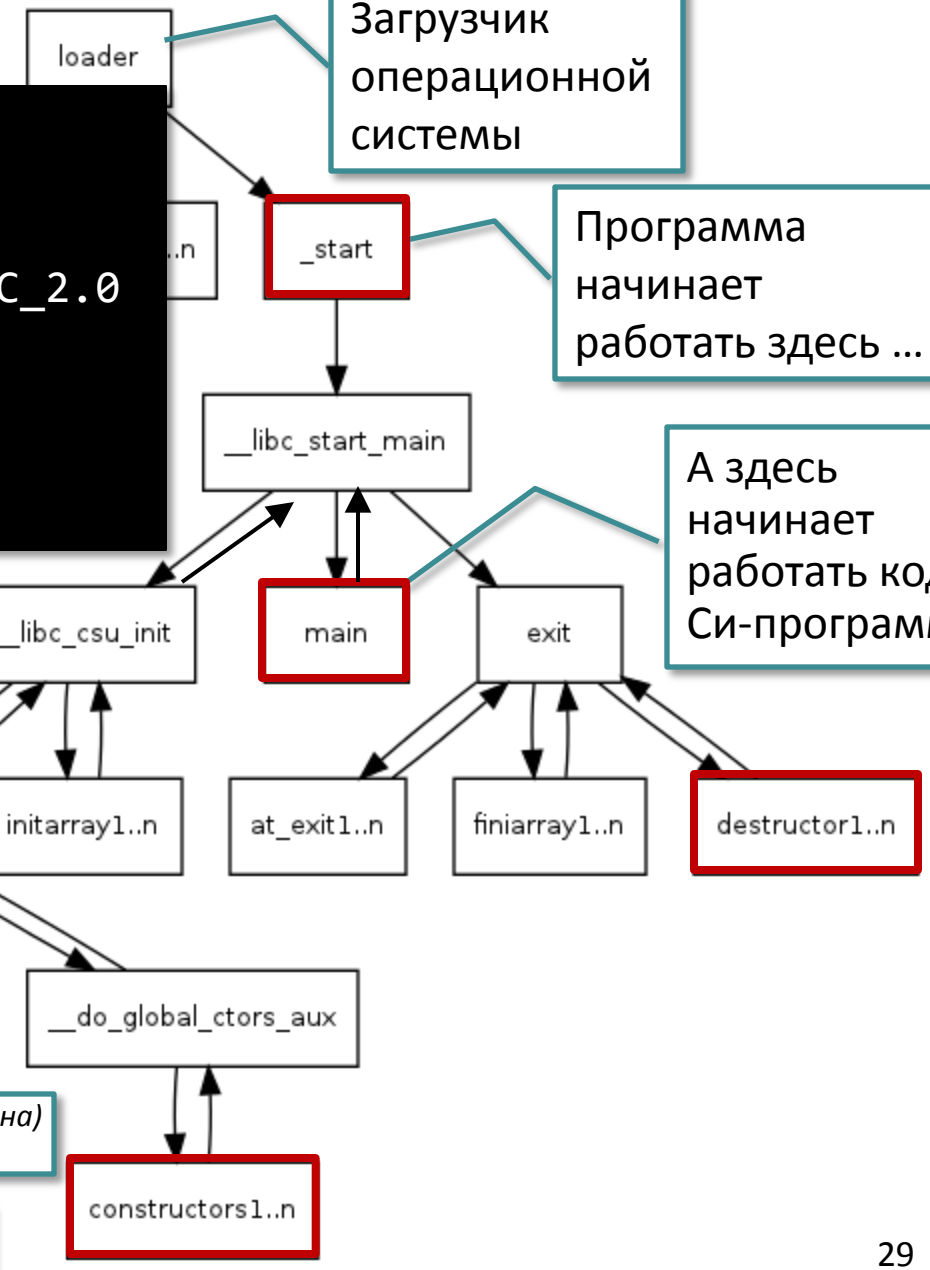
```
snoop@earth:~/samples$ gcc -g -o main main.c
snoop@earth:~/samples$ gdb main
```

```
(gdb) br main
Breakpoint 1 at 0x80483f3: file main.c, line 7.
(gdb) run
Starting program: /home/snoop/samples/main

Breakpoint 1, main (argc=1, argv=0xbffff184) at main.c:7
7           nullify(argc, argv);
(gdb) bt
#0  main (argc=1, argv=0xbffff184) at main.c:7
(gdb) set backtrace past-main on
(gdb) bt
#0  main (argc=1, argv=0xbffff184) at main.c:7
#1  0xb7e1972e in __libc_start_main (main=0x80483f0 <main>, argc=1,
argv=0xbffff184, init=0x8048410 <__libc_csu_init>,
    fini=0x8048470 <__libc_csu_fini>, rtld_fini=0xb7feb210
<_dl_fini>, stack_end=0xbffff17c) at libc-start.c:289
#2  0x08048311 in _start ()
(gdb)
```

```
snoop@jezek:~/samples$ gcc -v
…
Target: i686-linux-gnu
…
gcc version 4.9.2 (Ubuntu 4.9.2-10ubuntu13)
```

© 2018 МГУ/ВМК/СП

```
snoop@earth:~/samples$ nm main
...
08048470 T __libc_csu_fini
08048410 T __libc_csu_init
         U __libc_start_main@@GLIBC_2.0
080483f0 T main
080483eb T nullify
...
080482f0 T _start
...
```



Загрузчик операционной системы

Программа начинает работать здесь …

А здесь начинает работать код Си-программы

Компилятор (компоновщик) добавляет к программе код поддержки времени выполнения

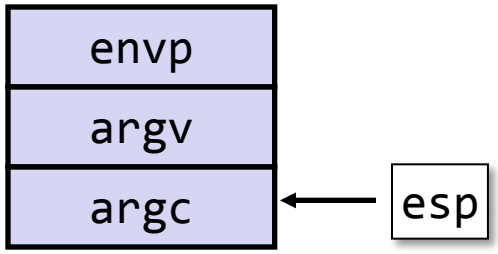
Запуск сбора профиля (если эта опция включена)
http://www.opennet.ru/docs/RUS/gprof/

```
crt1.o crti.o crtn.o crtbegin.o …
```

© 2018 МГУ/ВМК/СП

Функция _start

Начальное состояние стека




```
080482f0 <_start>:
 80482f0:   31 ed              xor    ebp,ebp
 80482f2:   5e                 pop    esi
 80482f3:   89 e1              mov    ecx,esp
 80482f5:   83 e4 f0           and    esp,0xfffffff0
 80482f8:   50                 push   eax
 80482f9:   54                 push   esp
 80482fa:   52                 push   edx
 80482fb:   68 70 84 04 08     push   0x8048470
 8048300:   68 10 84 04 08     push   0x8048410
 8048305:   51                 push   ecx
 8048306:   56                 push   esi
 8048307:   68 f0 83 04 08     push   0x80483f0
 804830c:   e8 cf ff ff ff     call   80482e0 <__libc_start_main@plt>
 8048311:   f4                 hlt
 8048312:   66 90              nop
 ...
```

Выравнивание стека (границ фрейма) по 16 байт

Адрес функции main

```
snoop@earth:~/samples$ objdump -M intel -d main
```

© 2018 МГУ/ВМК/СП

Функция _start

```
int __libc_start_main( int (*main) (int, char **, char **),
                                                  int argc,
                                          char ** ubp_av,
                                     void (*init) (void),
                                     void (*fini) (void),
                                void (*rtld_fini) (void),
                                     void (* stack_end));
```

```
08048300 <_start>:
 xor    ebp,ebp
 pop    esi
 mov    ecx,esp
 and    esp,0xfffffff0
 push   eax
 push   esp
 push   edx
 push   0x8048470
 push   0x8048410
 push   ecx
 push   esi
 push   0x80483f0
 call   80482e0 <__libc_start_main@plt>
 hlt
 nop
 ...
```