

Лекция 0xВ

14 марта

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

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

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

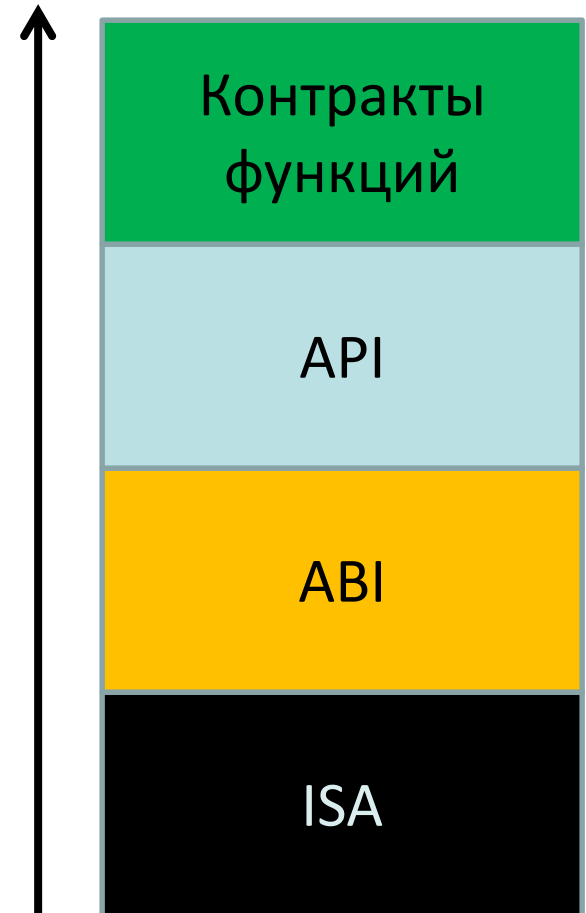
```
typedef struct chain chain;
struct chain {
    int    payload;
    chain *next;
};

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



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?

```
student@pc:~/asm$ gcc -m32 -g -no-pie -fno-pic -o main main.c
student@pc:~/asm$ gdb main
```

```
(gdb) br main
Breakpoint 1 at 0x80483ff: file main.c, line 7.
(gdb) run
Starting program: /home/student/asm/main

Breakpoint 1, main (argc=1, argv=0xffffd144) at main.c:7
7      nullify(argc, argv);
(gdb) bt
#0  main (argc=1, argv=0xffffd144) at main.c:7
(gdb) set backtrace past-main on
(gdb) bt
#0  main (argc=1, argv=0xffffd144) at main.c:7
#1  0xf7dfbe81 in __libc_start_main () from /lib32/libc.so.6
#2  0x08048312 in _start ()
(gdb)
```

```
student@pc:~/asm$ gcc -v
...
Target: x86_64-linux-gnu
...
gcc version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)
```


Начальное состояние стека

envp

argv

argc

esp ←

080482e0 <_start>:

```

80482e0:    31 ed                xor    ebp,ebp
80482e2:    5e                  pop    esi
80482e3:    89 e1                mov    ecx,esp
80482e5:    83 e4 f0            and    esp,0xfffffffff0
80482e8:    50                  push   eax
80482e9:    54                  push   esp
80482ea:    52                  push   edx
80482eb:    e8 23 00 00 00      call   8048313 <_start+0x33>
80482f0:    81 c3 10 1d 00 00    add    ebx,0x1d10
80482f6:    8d 83 80 e4 ff ff    lea   eax,[ebx-0x1b80]
80482fc:    50                  push   eax
80482fd:    8d 83 20 e4 ff ff    lea   eax,[ebx-0x1be0]
8048303:    50                  push   eax
8048304:    51                  push   ecx
8048305:    56                  push   esi
8048306:    c7 c0 fc 83 04 08    mov    eax,0x80483fc
804830c:    50                  push   eax
804830d:    e8 ae ff ff ff      call   80482c0 <__libc_start_main@plt>
8048312:    f4                  hlt
8048313:    8b 1c 24            mov    ebx,DWORD PTR [esp]
8048316:    c3                  ret

```

Выравнивание стека
(границ фрейма) по 16 байт

Адрес функции main

```
student@pc:~/asm$ objdump -M intel -d main
```



```
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));
```

```
080482e0 <_start>:
```

```
80482e0: xor    ebp,ebp
```

```
80482e2: pop   esi
```

```
80482e3: mov   ecx,esp
```

```
80482e5: and   esp,0xfffffffff0
```

```
80482e8: push  eax
```

```
80482e9: push  esp
```

```
80482ea: push  edx
```

```
80482eb: call  8048313 <_start+0x33>
```

```
80482f0: add   ebx,0x1d10
```

```
80482f6: lea   eax,[ebx-0x1b80]
```

```
80482fc: push  eax
```

```
80482fd: lea   eax,[ebx-0x1be0]
```

```
8048303: push  eax
```

```
8048304: push  ecx
```

```
8048305: push  esi
```

```
8048306: mov   eax,0x80483fc
```

```
804830c: push  eax
```

```
804830d: call  80482c0 <__libc_start_main@plt>
```

```
8048312: hlt
```

```
8048313: mov   ebx,DWORD PTR [esp]
```

```
8048316: ret
```

```
...
```

Функция main

ассемблерный листинг адаптирован под синтаксис nasm

080483fc <main>:

```
080483fc:      55          push    ebp
080483fd:      89 e5       mov     ebp,esp
080483ff:      ff 75 0c   push   dword [ebp+0xc]
08048402:      ff 75 08   push   dword [ebp+0x8]
08048405:      e8 ec ff ff call   nullify
0804840a:      83 c4 08   add    esp,0x8
0804840d:      b8 00 00 00 00 mov    eax,0x0
08048412:      c9        leave
08048413:      c3        ret
...
```

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

В зависимости от версии компилятора gcc выравнивание стека может выполняться в функции `_start` и/или `main`

```
#include <stdio.h>
```

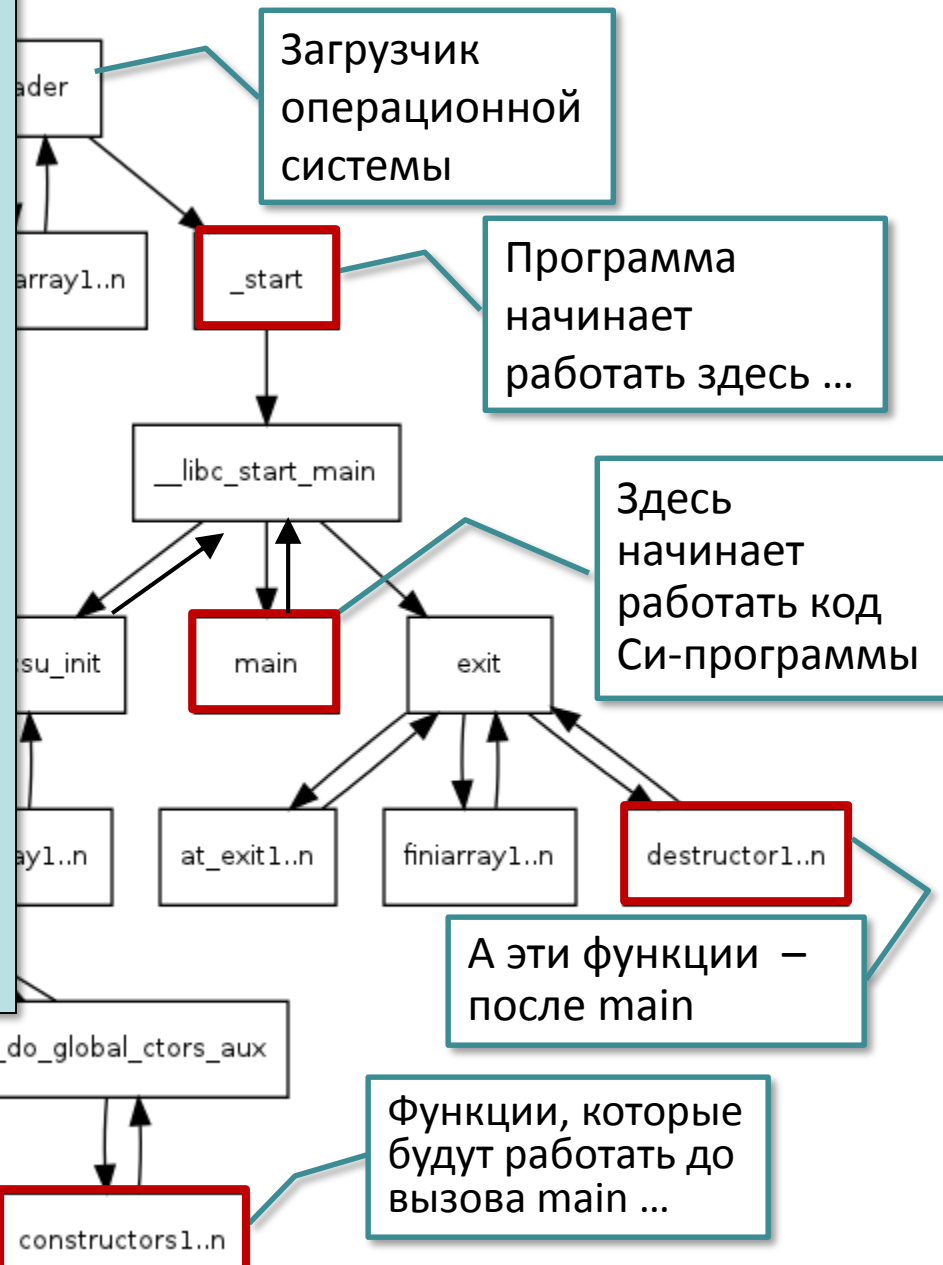
```
void __attribute__((constructor))
my_constructor() {
    printf("%s\n", __FUNCTION__);
}
```

```
void __attribute__((destructor))
my_destructor() {
    printf("%s\n", __FUNCTION__);
}
```

```
int main() {
    printf("%s\n", __FUNCTION__);
}
```

```
student@pc:~/asm$ ./main2
my_constructor
main
my_destructor
```

Идея метода/функции-конструктора не принадлежит Си++



```
student@pc:~/asm$ objdump -M intel -d main2
```

```
08048426 <my_constructor>:
```

```
8048426:    55                push   ebp
8048427:    89 e5             mov    ebp,esp
8048429:    83 ec 08          sub    esp,0x8
804842c:    83 ec 0c          sub    esp,0xc
804842f:    68 10 85 04 08    push  0x8048510
8048434:    e8 a7 fe ff ff    call  80482e0 <puts@plt>
8048439:    83 c4 10          add    esp,0x10
804843c:    90                nop
804843d:    c9                leave
804843e:    c3                ret
```

```
0804843f <my_destructor>:
```

```
804843f:    55                push   ebp
8048440:    89 e5             mov    ebp,esp
8048442:    83 ec 08          sub    esp,0x8
8048445:    83 ec 0c          sub    esp,0xc
8048448:    68 20 85 04 08    push  0x8048520
804844d:    e8 8e fe ff ff    call  80482e0 <puts@plt>
8048452:    83 c4 10          add    esp,0x10
8048455:    90                nop
8048456:    c9                leave
8048457:    c3                ret
```

gcc version 7.4.0

student@pc:~/asm\$ objdump -h main2

main2: file format elf32-i386

Sections:

Idx	Name	Size	VMA	LMA	File off	Align	
13	.text	000001e2	08048310	08048310	00000310	2**4	
			CONTENTS,	ALLOC,	LOAD,	READONLY,	CODE
18	.init_array	00000008	08049f04	08049f04	00000f04	2**2	
			CONTENTS,	ALLOC,	LOAD,	DATA	
19	.fini_array	00000008	08049f0c	08049f0c	00000f0c	2**2	
			CONTENTS,	ALLOC,	LOAD,	DATA	
...							

student@pc:~/asm\$ objdump -s -j .ctors -j .init_array -j .fini_array main2

main2: file format elf32-i386

Contents of section .init_array:

8049f04 20840408 26840408 ...&...

Contents of section .fini_array:

8049f0c f0830408 3f840408?...

Функция main с выравниванием стека

ассемблерный листинг адаптирован под синтаксис nasm

```
08048360 <main>:
8048360:      8d 4c 24 04      lea    ecx, [esp+0x4]
8048364:      83 e4 f0        and    esp, 0xffffffff0
8048367:      ff 71 fc        push   dword [ecx-0x4]
804836a:      55             push   ebp
804836b:      89 e5          mov    ebp, esp
804836d:      51            push   ecx
804836e:      83 ec 10       sub    esp, 0x10
8048371:      68 10 85 04 08  push   0x8048510
8048376:      e8 75 ff ff ff  call   80482f0 <puts@plt>
804837b:      8b 4d fc       mov    ecx, dword [ebp-0x4]
804837e:      83 c4 10       add    esp, 0x10
8048381:      c9           leave
8048382:      8d 61 fc       lea    esp, [ecx-0x4]
8048385:      c3           ret
```

Оболочка вокруг `main`

- **Выравнивание стека**
 - Каждый вызов функции происходит на выровненном стеке
 - Необходимо формировать каждый фрейм таким образом, чтобы однажды выполненное выравнивание сохранялось
- Поддержка повышенного уровня привилегий для данного запуска программы
- Поддержка многопоточного выполнения
- Запуск сбора профиля и запись в файл результатов профилирования после завершения работы деструкторов
 - Если программа была собрана с данной опцией
- Запуск конструкторов
- Запуск функции `main` с аргументами `argc` и `argv`
- Запуск деструкторов
- Передача результата функции `main` в функцию `exit`

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;
    }
}
```


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  
    ; ...
```

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]  
    ; ...
```

Пример вызова malloc

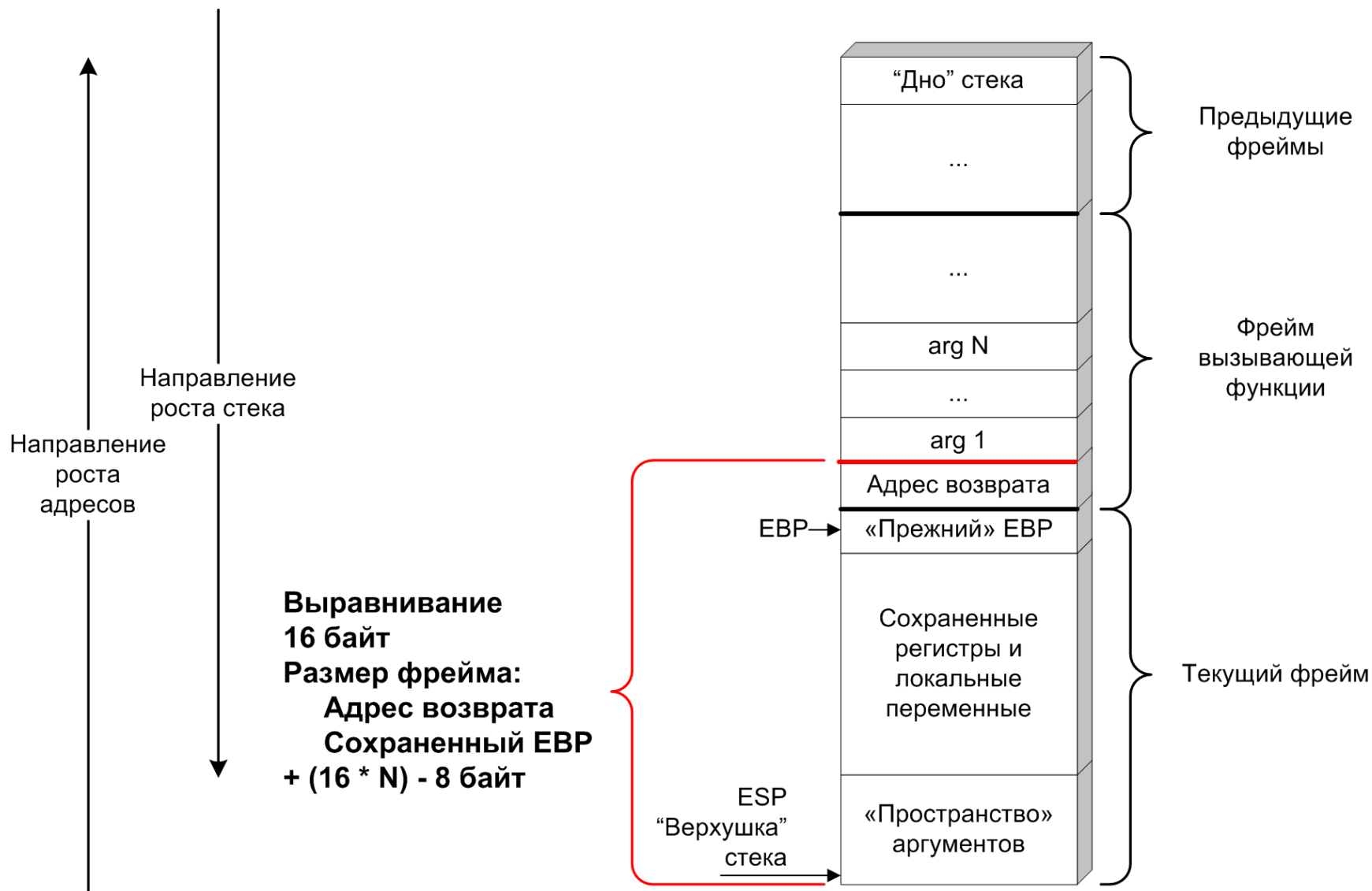
```
p_chain insert(p_chain p, int val) {  
    if ((0 == p) || (p->val > val)) {  
        p_chain np =  
            (p_chain)malloc(sizeof(t_chain));  
        np->val = val;  
        np->next = p;  
        return np;  
    } else {  
        p->next = insert(p, val);  
        return p;  
    }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

```
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:
```

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
```



Стандартная библиотека языка Си

- 24 заголовочных файла
- `stdlib.h`
 - Преобразование типов: `atoi`, `strtod`, ...
 - Генерация псевдослучайных последовательностей
 - Выделение и освобождение памяти
 - Сортировка и поиск
 - Математика
- `stdio.h`
 - Функции для файловых операций
 - Функции для операций ввода-вывода
- `string.h`
- ...

Выравнивание стека

- На стек помещаем только машинные слова (4 байта)
- Выравнивание стека по границе в 16 байт
IA-32/Linux/gcc
 - выполняется в функции `main`
 - остальные функции поддерживают выравнивание, формируя фрейм определенного размера
 - для листовых функций необязательно
- Не только производительность: команда `MOVDQA`
 - SSE2, Pentium4, 2001 г.
 - Быстрое копирование блока данных размером 16 байт
 - Аварийное завершение работы (`#GP`), если начальный адрес блока не выровнен по границе в 16 байт