© 2018 МГУ/ВМК/СП

# Лекция 0x11

7 апреля

© 2018 МГУ/ВМК/СП

# Непосредственное обновление EFLAGS

- F[U]COMI[P] St(*i*)
  - появились в P6

| Результат сравнения | ZF | PF | CF |
|---|---|---|---|
| St0 > St(*i*) | 0 | 0 | 0 |
| St0 < St(*i*) | 0 | 0 | 1 |
| St0 == St(*i*) | 1 | 0 | 0 |
| неупорядочены | 1 | 1 | 1 |

isLe(2.0, 3.0)

```
isLe:
   ...
   fld     dword [ebp+16]
   fld     qword [ebp+8]
   fucomip ; st0 vs. st1
   setbe   al
   ...
```

```
BE:
CF = 1 или ZF = 1
```

© 2018 МГУ/ВМК/СП

# Условная пересылка

- FCMOV*cc* St(*i*)
  - появились в P6

| Условие | ZF | PF | CF | Код |
|---|---|---|---|---|
| St0 > St(*i*) | 0 | 0 | 0 | A |
| St0 < St(*i*) | 0 | 0 | 1 | B |
| St0 == St(*i*) | 1 | 0 | 0 | E |
| неупорядочены | 1 | 1 | 1 | U |

```
f:
   push    ebp
   mov     ebp, esp
   fld     qword [ebp+16]
   fld     qword [ebp+8]
   fucomip st0, st1
   fld     qword [ebp+24]
   fcmovbe st1
   fstp    st1
   pop     ebp
   ret
```

```
double f(double a, double b, double c) {
   return a > b? c : b;
}
```

© 2018 МГУ/ВМК/СП

# Функции: возвращаемое значение – число с плавающей точкой

```
void caller(double *p) {
   *p = inverse(*p);
}
```

```
float inverse(double x) {
   return 1/x;
}
```

```
caller:
   push    ebp
   mov     ebp, esp
   sub     esp, 8
   mov     eax, dword [ebp+8]
   fld     dword [eax]
   fstp    qword [esp]
   call    inverse
   mov     eax, dword [ebp+8]
   fstp    qword [eax]
   leave
   ret
```

На входе в функцию регистры St0 – St7 должны быть пустыми

```
inverse:
   push    ebp
   mov     ebp, esp
   fld1
   fld     qword [ebp+8]
   fdivp
   pop ebp
   ret
```

На выходе из функции:

1. регистры St1 – St7 должны быть пустыми,
2. регистр St0 содержит возвращаемое значение

© 2018 МГУ/ВМК/СП

# Классификация соглашений вызова Си-функций для платформы gcc/Linux/IA-32

Передача параметров и очистка от них стека

FASTCALL – первые два параметра на регистрах ECX и EDX (если умещаются), остальные – на стеке
Очищает от стековых параметров вызванная функция

STDCALL – все параметры на стеке
Очищает стек вызванная функция

CDECL – все параметры на стеке
Очищает стек вызывающая функция

Структура (в gcc) через область памяти, адрес которой задан в первом, служебном, параметре

Числа с плавающей точкой St0

Целые числа и указатели EAX

long long EDX:EAX

Возвращаемое значение

EBP сохраняется вызванной функцией только при необходимости

EBP сохраняется на верхней границе фрейма для «удобства» работы отладчика

Использование регистра EBP

© 2018 МГУ/ВМК/СП

# Итоги второй части

- Процессор с точки зрения пользовательской программы
- Основные механизмы реализации языка Си
  - Соответствие типов
  - Указатели, адресная арифметика
  - Управляющие операторы: разные стратегии реализации
  - Размещение данных в памяти
    - Классы памяти: статическая, автоматическая, динамическая
    - Выравнивание
  - Соглашения о вызове функции
- ABI – Application Binary Interface
- IA-32: x86 + x87 + ...
  - Некоторые особенности устройства

© 2018 МГУ/ВМК/СП

# Далее ...

- Система программирования языка Си
- Формат объектных файлов ELF
- Статическая компоновка
- Статические библиотеки
- Загрузка исполняемого  файла в память
- Динамическая компоновка
- Динамические и разделяемые библиотеки

© 2018 МГУ/ВМК/СП

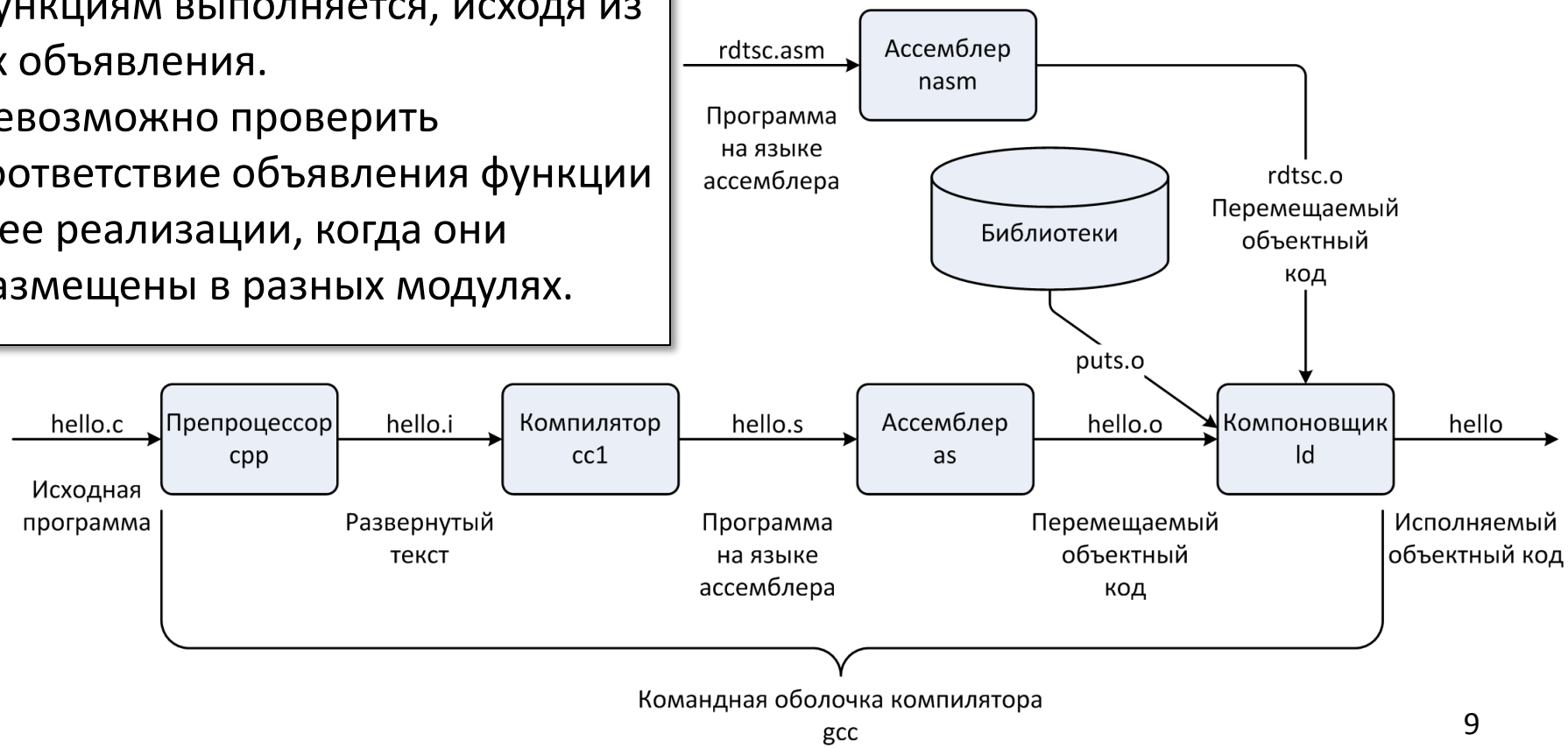
# Система программирования

- Системные/прикладные программы
  - Операционная система
  - Программные средства разработки
- Система программирования – комплекс средств
  - Язык программирования
    - Информационные ресурсы
    - Программные инструменты
    - Библиотеки
- Этапы жизненного цикла программы
  - Проектирование
    - Сбор и анализ требований к программе
    - Разработка
  - Реализация
    - Кодирование
    - Отладка
  - Сопровождение

© 2018 МГУ/ВМК/СП

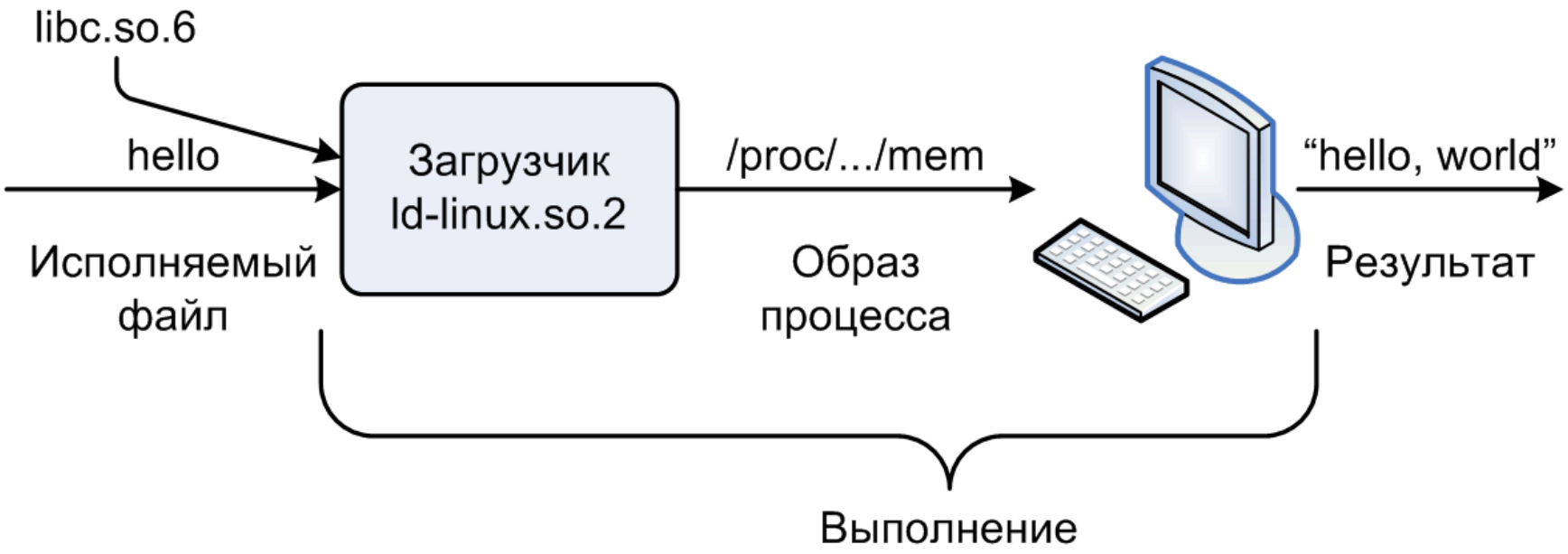
# Система программирования языка Си

- Язык Си использует независимую компиляцию. Обращение к глобальным переменным и функциям выполняется, исходя из их объявления.
- Невозможно проверить соответствие объявления функции и ее реализации, когда они размещены в разных модулях.



© 2018 МГУ/ВМК/СП

# Выполнение программы



© 2018 МГУ/ВМК/СП

# Компиляция

- Определения (и в некоторых случаях – объявления) глобальных, статических и локальных статических переменных
  - Создание помеченных областей статической памяти соответствующего размера
- Определение функции
  - Построение соответствующего кода
- Области видимости по умолчанию в Си и ассемблере отличаются
- Преобразование имен переменных в метки декорирование имен (name mangling)
  - Си@Linux: имена не меняются
  - Си@Windows

```
int __cdecl    f (int x) { return 0; }
int __stdcall  g (int y) { return 0; }
int __fastcall h (int z) { return 0; }
```

  - Си++, восстановление имен – утилита `c++filt`

```
int __cdecl Fx_i(int (__cdecl*)(int))
```

```
?Fx_i@@YAHP6AHH@Z@Z:
```

Приведены спецификаторы соглашения вызова, используемые в компиляторе MS Visual Studio

```
_f:
   xor  eax, eax
   ret

_g@4:
   xor  eax, eax,
   ret 4

@h@4:
   xor  eax, eax
   ret
```

© 2018 МГУ/ВМК/СП

# Схема работы ассемблера

- Проблема: опережающие ссылки
- Первый проход: частичная трансляция команд, составление таблицы символов, таблицы ссылок, таблицы строк, …
  - ***Символ***
    - Метка: адрес, с которым соотнесено имя (строка символов)
    - Атрибуты: размер, видимость из других модулей (глобальный/локальный)
  - Описание символов собрано в ***таблице символов***
  - Определяются размеры секций
  - Составляется таблица использований (ссылок) символов в операндах относительной и абсолютной адресации
- Второй проход: построение объектного кода
  - Заполнение ссылок на символы, когда символ и обращение к нему лежат в одной секции, а адресация – относительная
  - Файл с объектным кодом должен хранить содержимое секций и служебную информацию: где используются ссылки и на что они ссылаются, с какие места секций были помечены и какими именно метками

© 2018 МГУ/ВМК/СП

# Пример Си-программы

**main.c**

```
int buf[2] = {1, 2};
char str[] = "Hello";
char *p    = "world";

int swap(char*, char*);

int main()
{
  swap(str, p);
  return 0;
}
```

**swap.c**

```
extern int buf[];

int *bufp0 = &buf[0];
static int *bufp1;

void swap()
{
  int temp;

  bufp1 = &buf[1];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
```

*Код программы содержит намеренно внесенную ошибку.*

© 2018 МГУ/ВМК/СП

# Частичная трансляция команд

```
int buf[2] = {1, 2};
char str[] = "Hello";
char *p    = "world";

int swap(char*, char*);

int main()
{
  swap(str, p);
  return 0;
}
```

```
main:
   lea     ecx,[esp+0x4]
   and     esp,0xfffffff0
   push    dword [ecx-0x4]
   push    ebp
   mov     ebp,esp
   push    ecx
   sub     esp,0xc
   push    dword [p]
   push    str
   call    swap
   mov     ecx,dword [ebp-0x4]
   add     esp,0x10
   xor     eax,eax
   leave
   lea     esp,[ecx-0x4]
   ret
```

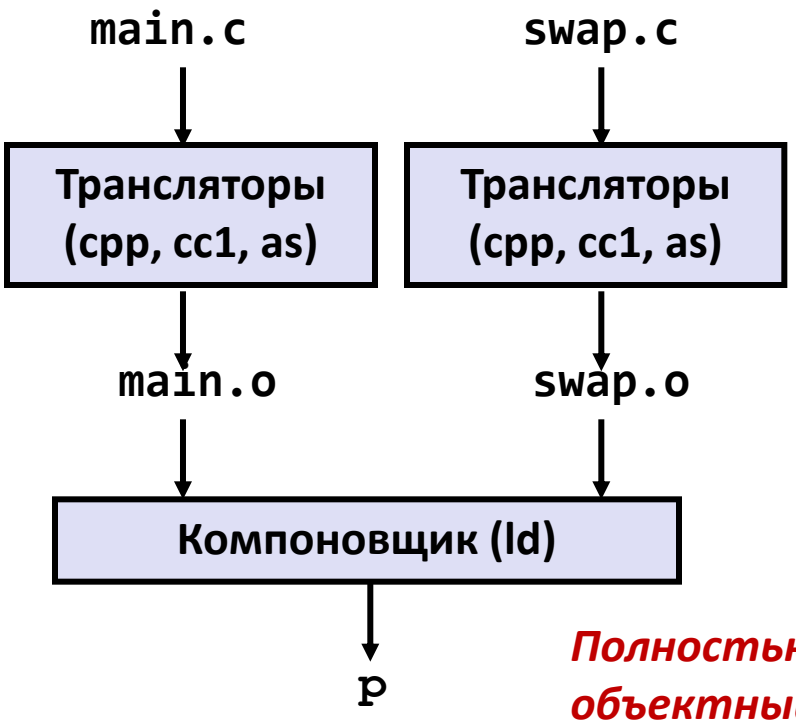
```
00000000 <main>:
   0:   8d 4c 24 04
   4:   83 e4 f0
   7:   ff 71 fc
   a:   55
   b:   89 e5
   d:   51
   e:   83 ec 0c
  11:   ff 35 ** ** ** **
  17:   68 ** ** ** **
  1c:   e8 ** ** ** **
  21:   8b 4d fc
  24:   83 c4 10
  27:   31 c0
  29:   c9
  2a:   8d 61 fc
  2d:   c3
```

Работоспособный код указанных команд не может быть построен ассемблером . Компоновщик окончательно заполнит в командах байты, относящееся к ссылкам.

© 2018 МГУ/ВМК/СП

# Статическая компоновка

- Программа транслируется и компонуется драйвером компилятора:
  - `unix> gcc -O2 -g -o p main.c swap.c`
  - `unix> ./p`




***Файлы с исходным кодом***

***Независимо друг от друга откомпилированные файлы с перемещаемым объектным кодом***

***Полностью скомпонованный исполняемый объектный файл (содержит код и данные всех функций определенные в `main.c` и `swap.c`)***

© 2018 МГУ/ВМК/СП

# Почему нужен компоновщик?

- Причина 1: Модульность программы
  - Программа может быть организована как набор небольших файлов с исходным кодом, а не один монолитный файл.
  - Есть возможность организовывать библиотеки функций, являющихся общими для разных программ
    - например, библиотека математических функций, стандартная библиотека языка Си

© 2018 МГУ/ВМК/СП

# Почему нужен компоновщик?

- Причина 2: Эффективность
  - Время: Раздельная компиляция
    - Меняем код в одном файле, компилируем только его, повторяем компоновку
    - Нет необходимости повторять компиляцию остальных файлов с исходным кодом.
  - Место на диске: Библиотеки
    - Общие функции можно объединить в одном файле...
    - Исполняемые файлы и образ программы в памяти содержит только те функции, которые действительно используются.

© 2018 МГУ/ВМК/СП

# Что делает компоновщик?

- Шаг 1. Разрешение символов
  - В ассемблерной программе определяют и используют *символы* (переменные и функции):
    - `void swap() {…}  /* определение символа swap */`
    - `swap(str, p);  /* ссылки на символы */`
    - `int *xp = &x;  /* определение символа xp, ссылка на  x */`
  - Определения символов сохраняются в таблице символов.
    - Таблица символов – массив структур
    - Каждая запись содержит имя, размер, позицию символа.
  - Компоновщик устанавливает связь каждой ссылки на символ с единственным определением символа.

```
typedef struct {
   int name;       /* Смещение в таблице строк */
   int value;      /* Смещение в секции или виртуальный адрес */
   int size;       /* Размер в байтах */
   char type:4,    /* Данные, код, секция, имя файла */
   binding:4;      /* Локальный или глобальный символ */
   char reserved;  /* Не используется */
   char section;   /* Номер заголовка секции, ABS, UNDEF, COMMON */
} Elf_Symbol;
```

© 2018 МГУ/ВМК/СП

# Что делает компоновщик?

- Шаг 2. Перемещение
  - Несколько объявлений секций кода и данных объединяются в единые секции
  - Символы перемещаются с их относительных позиций в `.o` файлах на абсолютные адреса в исполняемом файле.
  - Обновляются (перебазируются) все ссылки на символы, согласно их новым позициям.

```
typedef struct {
   int offset;    /* Смещение перебазируемой ссылки в секции */
   int symbol:24, /* Номер символа в таблице */
   type:8;        /* Тип перебазирования */
} Elf32_Rel;
```

© 2018 МГУ/ВМК/СП

# Три типа объектных файлов (модулей)

- Перемещаемые объектные файлы (`.o`-файлы)
  - Содержит код и данные в форме, позволяющей проводить компоновку с другими перемещаемым объектными файлами.
    - Каждый `.o`-файл производится из **одного** файла с исходным кодом (`.c`-файла)
- Исполняемые объектные файлы
  - Содержит код и данные в такой форме, что их можно напрямую копировать в память и запускать выполнение программы.

– Разделяемые объектные файлы (`.so`-файлы)
  - Особый вид перемещаемого объектного файла, который может быть загружен в память и скомпонован с программой динамически, во время ее загрузки и во время работы.
  - Windows - Dynamic Link Libraries (DLL)

© 2018 МГУ/ВМК/СП

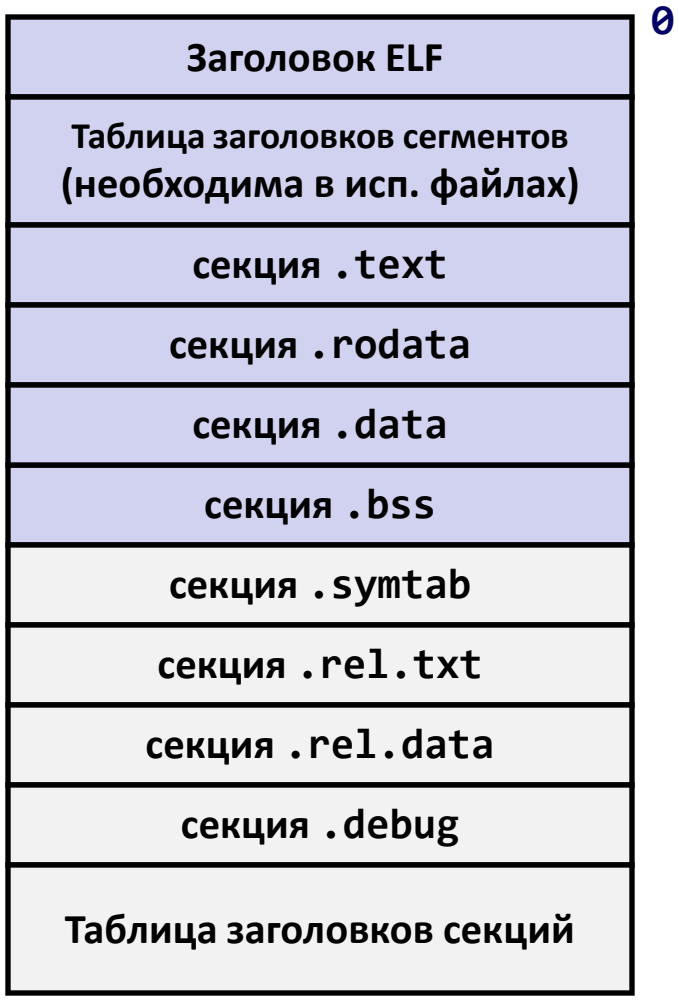
# Executable and Linkable Format (ELF)

- Стандартный бинарный формат объектных файлов
- Был предложен в AT&T System V Unix
  - Позже был поддержан в BSD и Linux
- Единый формат для
  - Перемещаемых объектных файлов (`.o`),
  - Исполняемых объектных файлов
  - Разделяемых объектных файлов (`.so`)

© 2018 МГУ/ВМК/СП

# Формат ELF файла

- Заголовок Elf
  - Размер машинного слова, порядок байт, тип файла (`.o`, исп., `.so`), и др.
- Таблица заголовков сегментов
  - Размер страницы, сегменты виртуальной памяти, размеры сегментов.
- Секция `.text`
  - код
- Секция `.rodata`
  - Данные, доступные только на чтение: таблицы переходов, константы
- Секция `.data`
  - Инициализированные глобальные переменные
- Секция `.bss`
  - Неинициализированные глобальные переменные
  - У секции есть заголовок, на сама секция не занимает места



© 2018 МГУ/ВМК/СП

# Формат ELF файла (продолжение)

- Секция `.symtab`
  - Таблица символов
  - Имена функций и статических переменных
  - Имена секций
- Секция `.rel.text`
  - Данные для перемещения ссылок в секции `.text`
  - Адреса обновляемых операндов в двоичном коде инструкций
- Секция `.rel.data`
  - Данные для перемещения ссылок в секции `.data`
  - Адреса глобальных переменных, инициализированных ссылками на внешние функции или глобальные переменные
- Секция `.debug`
  - Данные для символьного отладчика (`gcc -g`)
- Таблица заголовков секций
  - Смещения и размеры каждой секции