

# Лекция 0x11

13 апреля

# Порядок действий имеет значение

```
%include 'io.inc'  
  
section .data  
x dq 3.14  
y dq 1e50  
z dq -1e50  
  
section .bss  
r resq 1  
  
section .rodata  
lc db '%f', 10, 0  
  
section .text  
CEXTERN printf  
global CMAIN
```

```
CMAIN:  
; ... пролог функции  
sub     esp, 20  
  
fld     qword [x]  
fld     qword [y]  
fld     qword [z]  
faddp  
faddp  
fst     qword [r]  
fstp    qword [esp + 4]  
mov     dword [esp], lc  
call    printf  
  
add     esp, 20  
; ... эпилог функции
```

# Порядок действий имеет значение

```
СMAIN:  
; ... пролог функции  
sub     esp, 20  
  
fld     qword [y]  
fld     qword [z]  
fld     qword [x]  
faddp  
faddp  
fst     qword [r]  
fstp    qword [esp + 4]  
mov     dword [esp], 1c  
call    printf  
  
add     esp, 20  
; ... эпилог функции
```

```
СMAIN:  
; ... пролог функции  
sub     esp, 20  
  
fld     qword [x]  
fld     qword [y]  
fld     qword [z]  
faddp  
faddp  
fst     qword [r]  
fstp    qword [esp + 4]  
mov     dword [esp], 1c  
call    printf  
  
add     esp, 20  
; ... эпилог функции
```

# Распределение слагаемых

```
section .data
x dq 1e200
y dq 1e200
z dq 1e200

section .bss
r resq 1

section .rodata
lc db '%lf', 10, 0
```

```
CMAIN:
; ... пролог функции
sub     esp, 20

fld     qword [x]
fld     qword [y]
fsubp
fld     qword [z]
fmulp
fst     qword [r]
fstp    qword [esp + 4]
mov     dword [esp], lc
call    printf

add     esp, 20
; ... эпилог функции
```

# Распределение слагаемых

```
section .data
x dq 1e200
y dq 1e200
z dq 1e200

section .bss
r resq 1

section .rodata
lc db '%lf', 10, 0
```

```
CMAIN:
; ... пролог функции
sub     esp, 20

fld     qword [x]
fld     qword [z]
fmulp

fld     qword [y]
fld     qword [z]
fmulp
fsubp
; вызов printf

add     esp, 20
; ... эпилог функции
```

# Распределение слагаемых

```
section .data
x dq 1e200
y dq 1e200
z dq 1e200

section .bss
r resq 1
cw resw 1

section .rodata
lc db '%lf', 10, 0

CMAIN:
; ... пролог функции
sub     esp, 20
```

```
fstcw   word [cw]
and     word [cw], 11111111_11000000b
fldcw   word [cw]

fld     qword [x]
fld     qword [z]
fmulp

fld     qword [y]
fld     qword [z]
fmulp
fsubp
; вызов printf

add     esp, 20
; ... эпилог функции
```

# Предопределенные константы

- На «верхушку» стека регистров ( $St0$ ) помещается определенная константа
  - FLD1       $+1.0$
  - FLDL2T     $\log_2 10$
  - L2E         $\log_2 e$
  - FLDPI      $\pi$
  - FLDLG2     $\log_{10} 2$
  - FLDLN2     $\log_e 2$
  - FLDZ       $+0.0$

# Сравнение чисел

```
_Bool isLe(double x, float y) {
    return x <= y;
}
```

Результат сравнения	C3	C2	C0
St0 > St1	0	0	0
St0 < St1	0	0	1
St0 == St1	1	0	0
неопределенно	1	1	1

isLe:

```
    push    ebp
    mov     ebp, esp
    fld    dword [ebp+16]
    fld    qword [ebp+8]
    fucompp ; St0 vs. St1
    fnstsw ax
    sahf
    setbe  al
    pop    ebp
    ret
```



# Непосредственное обновление EFLAGS

- `F[U]COMI[P] St(i)`

– появились в P6

`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

Результат сравнения	ZF	PF	CF
<code>St0 &gt; St(i)</code>	0	0	0
<code>St0 &lt; St(i)</code>	0	0	1
<code>St0 == St(i)</code>	1	0	0
неупорядочены	1	1	1

# Условная пересылка

- `FCMOVcc St(i)`  
– появились в P6

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;
}

```

Условие	ZF	PF	CF	Код
<code>St0 &gt; St(i)</code>	0	0	0	A
<code>St0 &lt; St(i)</code>	0	0	1	B
<code>St0 == St(i)</code>	1	0	0	E
неупорядочены	1	1	1	U

# 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     qword [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

# 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 уместятся), 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

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

# 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

## Далее ...

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

# Система программирования

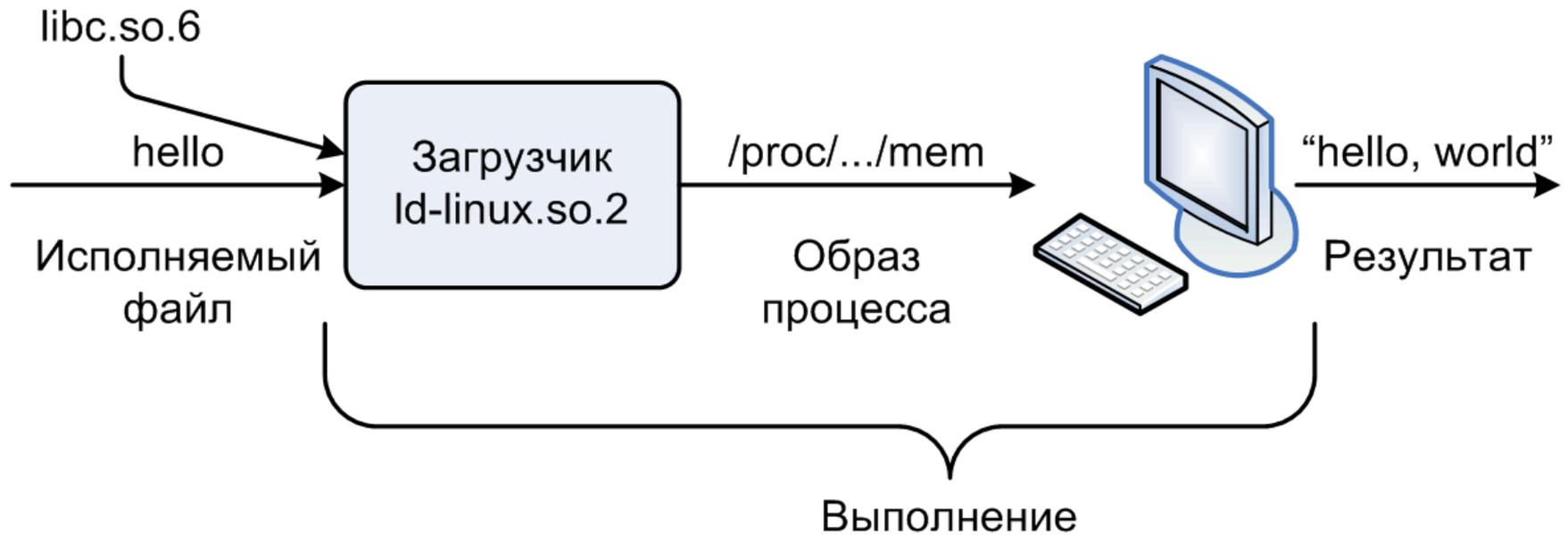
- Системные/прикладные программы
  - Операционная система
  - Программные средства разработки
- Система программирования – комплекс средств
  - Язык программирования
    - Информационные ресурсы
    - Программные инструменты
    - Библиотеки
- Этапы жизненного цикла программы
  - Проектирование
    -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

# 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.



# Выполнение программы



# 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@@YANP6ANH@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
```

# 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

# Пример Си-программы

## main.c

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

int swap(char*, char*);

int main(int argc, char* argv[]) {
    if (argc > 1) {
        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.*

# Частичная трансляция команд

**main:**

```

lea    ecx,[esp+0x4]
and    esp,0xffffffff0
push  dword [ecx-0x4]
push  ebp
mov    ebp,esp
push  ecx
sub    esp,0x4
cmp    dword [ecx],0x1
jle   .L2
push  eax
push  eax
push  dword [p]
push  str
call  swap
add   esp,0x10
.L2:
mov   ecx,dword [ebp-0x4]
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 04
11: 83 39 01
14: 7e (**)
16: 50
17: 50
18: ff 35 (** ** ** **)
1e: 68 (** ** ** **)
23: e8 (** ** ** **)
28: 83 c4 10

2b: 8b 4d fc
2e: 31 c0
30: c9
31: 8d 61 fc
34: c3

```

Кодируем в двоичном виде:

- коды операций;
- адресный код операндов, не содержащий меток.

main.L2

p

str

swap

# Построение объектного кода

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

int swap(char*, char*);

int main(int argc, char* argv[]) {
    if (argc > 1) {
        swap(str, p);
    }
    return 0;
}
```

```
push    ecx
push    eax
push    dword [p]
push    str
call    swap
add     esp, 0x10

.L2:
mov     ecx, dword [ebp-0x4]
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 04
11:     83 39 01
14:     7e 15
16:     50
17:     50
18:     ff 35 (** ** ** *)
1e:     68 (** ** ** *)
23:     e8 (** ** ** *)
28:     83 c4 10
2b:     8b 4d fc
2e:     31 c0
30:     c9
31:     8d 61 fc
34:     c3
```

Таблица  
символов

buf

p

str

swap

main

Таблица  
ссылок

№1

№2

№3

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