Лекция 15

31 марта

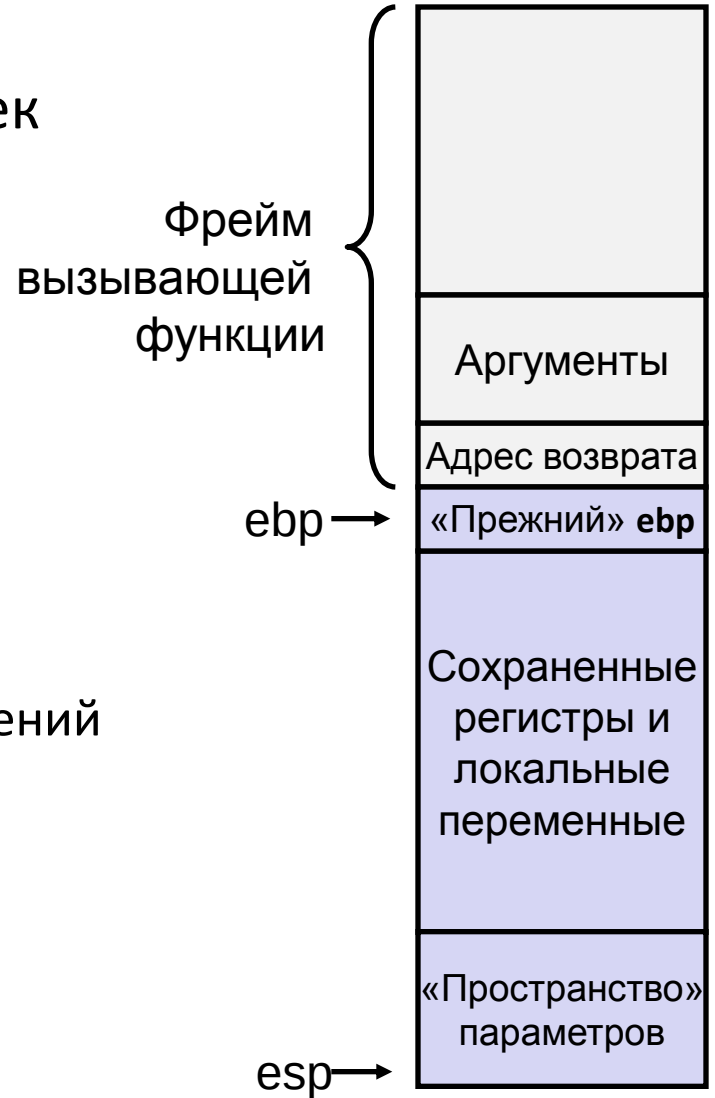
Ассемблерные вставки

- Нет единого стандарта
- Пример: gcc
  - Наиболее развитый механизм
  - Естественный синтаксис AT&T

```
int a=10, b;
asm (".intel_syntax noprefix\n"
     "mov %%eax, %1\n"
     "mov %0, %%eax\n"
     ".att_syntax  \n" /* ассемблерная вставка */
     :"=r"(b)          /* выходные операнды   */
     :"r"(a)           /* входные операнды    */
     :"%eax"           /* разрушаемые регистры */
     );
```

Вызов функций – заключение

- Порядок вызова функций образует стек (call / ret)
  - Если P вызывает Q, то Q завершается до завершения P
- Рекурсия (в том числе косвенная) корректно реализуется через общее соглашение о вызове функций
  - Фрейм используется для размещения локальных переменных и сохранения значений регистров
  - Аргументы для вызова очередной функции размещаются на «верхушке» стека
  - Результат возвращается через регистр eax
- Параметры передаются по значению



Сравнение строк

- CMPSB сравнение байт
- CMPSW сравнение 16-разрядных чисел
- CMPSD сравнение 32-разрядных чисел
- CLD/STD – очистить/установить флаг DF

```
Tmp = [ESI] – [EDI];
Установить статусные флаги согласно TMP;
If (DF == 0) {
   ESI += sizeof(операнд);
   EDI += sizeof(операнд);
} else { // DF == 1
   ESI -= sizeof(операнд);
   EDI -= sizeof(операнд);
}
```

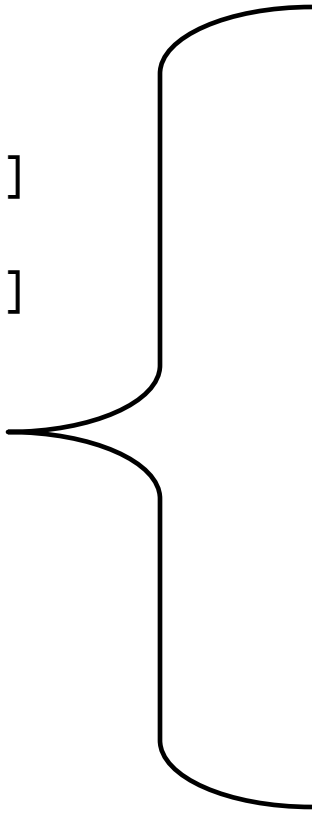
Сравнение строк равного размера

```
%include 'io.inc'

BUFSIZE equ 32

section .data
 s1 db 'some text'
    times BUFSIZE-$+s1 db 0

 s2 db 'some text...'
    times BUFSIZE-$+s2 db 0

section .text
global CMAIN
```

```
CMAIN:
    push ebp
    mov ebp, esp
    sub esp, 12
    mov dword [esp], s1
    mov dword [esp + 4], s2
    mov dword [esp + 8], BUFSIZE-1
    call my_strncmp ; возвращаем 0,
                    ; если строки равны
                    ; иначе – номер байта
                    ; в котором встретилось
                    ; различие (считаем с 1)
    PRINT_DEC 4, eax
    NEWLINE
    xor eax, eax
    mov esp, ebp
    pop ebp
    ret
```

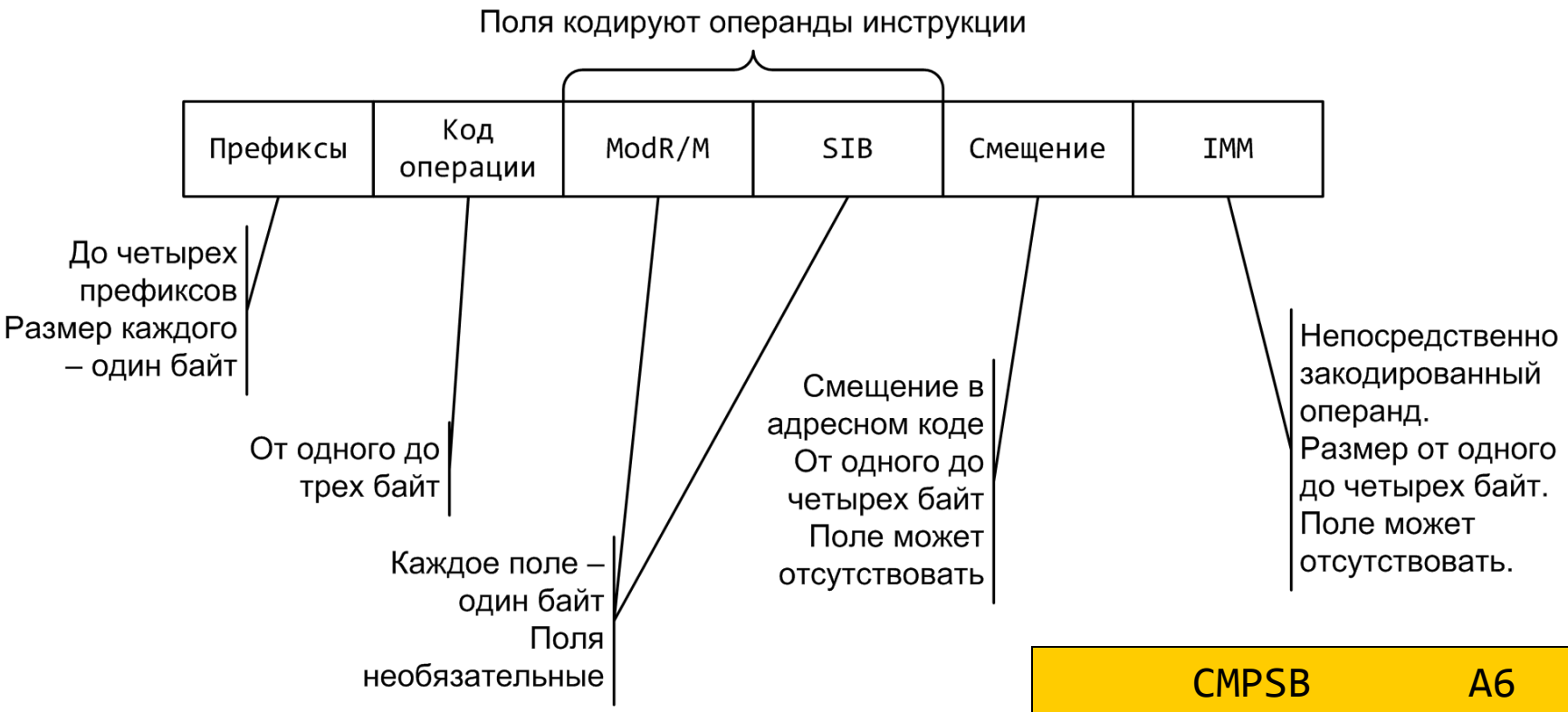
Сравнение строк равного размера

```
my_strncmp:
   push ebp
   mov ebp, esp
   push esi
   push edi
   xor eax, eax
   mov ecx, dword [ebp + 16]
   mov esi, dword [ebp + 8]
   mov edi, dword [ebp + 12]
   ;…


   ;…
.end:
   pop edi
   pop esi
   mov esp, ebp
   pop ebp
   ret
```



```
   ; …
   cld
   jecxz .end
.loop:
   cmpsb
   jne .ne
   loop .loop
   jmp .end
.ne:
   mov eax, dword [ebp + 16]
   sub eax, ecx
   inc eax
.end:
   ; …
```

Формат инструкции



	CMPSB		A6
REP	CMPSB	F3	A6
REPNE	CMPSB	F2	A6

Префикс повтора

- REPE = REPZ = REP
- REPNE = REPNZ

```
WHILE (ECX != 0) {
   Выполнить соответствующую строковую инструкцию;
   ECX = (ECX – 1);
   IF (ECX == 0)
       break;
  IF (((Префикс повтора == REPZ или REPE) и
       (ZF == 0)) или
       ((Префикс повтора == REPNZ) и (ZF == 1) ))) {
      break;
  }
```

Сравнение строк равного размера

```
my_strncmp:
   push ebp
   mov ebp, esp
   push esi
   push edi
   xor eax, eax
   mov ecx, dword [ebp + 16]
   mov esi, dword [ebp + 8]
   mov edi, dword [ebp + 12]
   ;…


   ;…
.end:
   pop edi
   pop esi
   mov esp, ebp
   pop ebp
   ret
```

```
   ; …
   cld
   repe cmpsb
   je .end
   mov eax, 1
   mov ecx, -1
   cmovl eax, ecx
.end:
   ; …
```

Приводим поведение функции к стандартному виду

- меньше -1
- равно 0
- больше 1

Остальные строковые команды

- SCAS(B|W|D)
  - Сравниваем аккумулятор со строкой [EDI]
- MOVS(B|W|D)
  - Копируем строки [EDI] ⇦ [ESI]
- STOS(B|W|D)
  - Выгружаем аккумулятор по адресу [EDI]
- LODS(B|W|D)
  - Загружаем в аккумулятор из адреса [ESI]
  - Комбинировать с префиксом повтора не целесообразно

strnlen

```
#include <string.h>
size_t strnlen(const char *s, size_t maxlen);

strnlen:
   push ebp
   mov ebp, esp
   push edi
   xor eax, eax
   mov ecx, dword [ebp + 12]
   mov edi, dword [ebp + 8]
   repne scasb
   mov eax, dword [ebp +12]
   sub eax, ecx
   dec eax
   pop edi
   mov esp, ebp
   pop ebp
   ret
```

memset

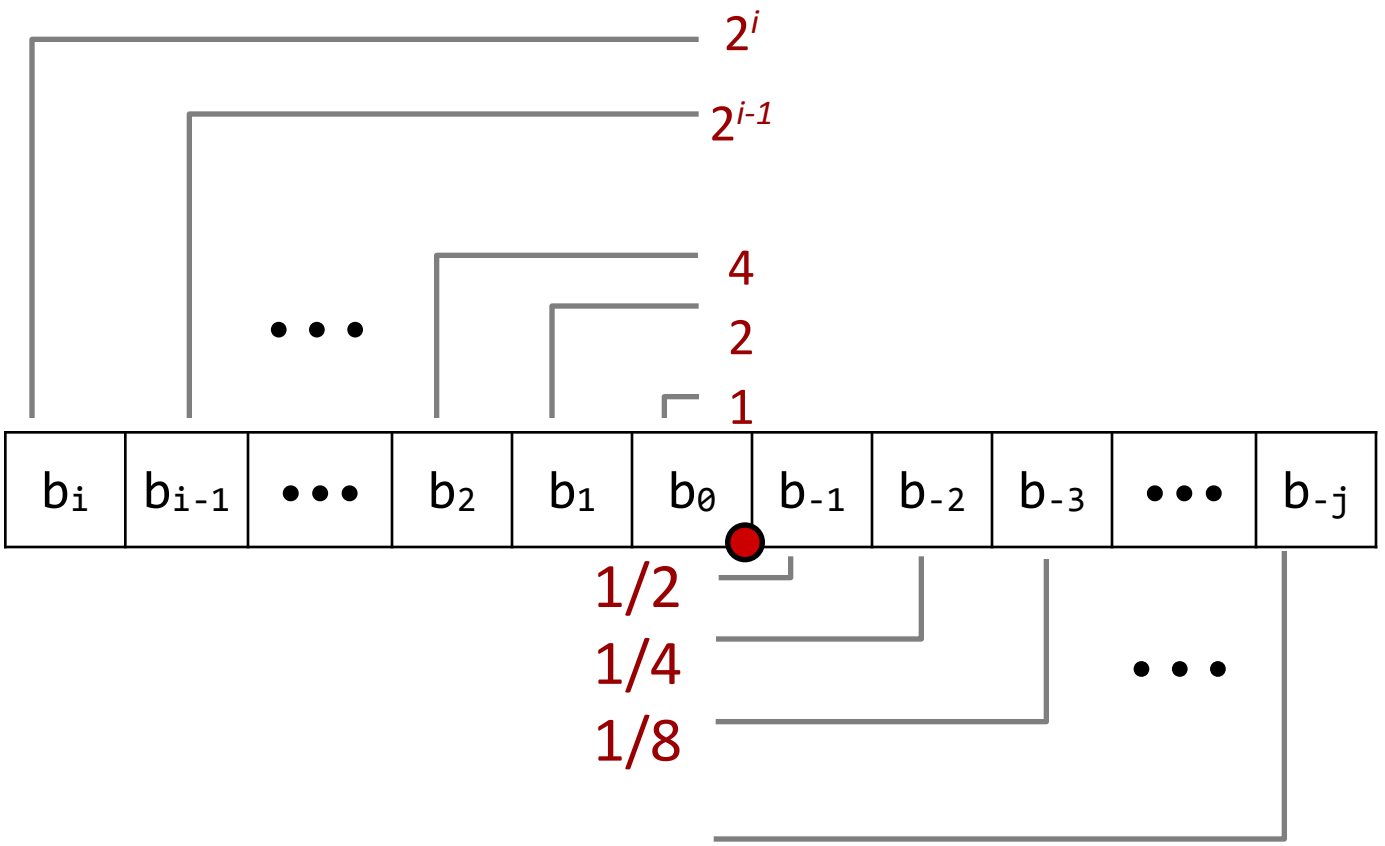
```
#include <string.h>
void *memset(void *s, int c, size_t n);

memset:
    push ebp
    mov ebp, esp
    push edi
    mov ecx, dword [ebp + 16]
    mov esi, dword [ebp + 8]
    mov al,  byte  [ebp + 12]
    rep stosb
    pop edi
    mov esp, ebp
    pop ebp
    ret
```

Компактные функции стандартной библиотеки компилятор может встроить непосредственно в место вызова

Далее

- Представления для вещественных чисел
  - Дробные двоичные числа
  - Числа с плавающей точкой
- Сопроцессор x87
  - Устройство
  - Примеры программ

Дробные двоичные числа



- Представление числа
  - Биты справа от “двоичной точки” представляют отрицательные степени 2
  - Точное представление для рациональных чисел вида : $\sum_{k=-j}^{i} b_k \times 2^k$

Примеры дробных двоичных чисел

Число	Представление
5 3/4	101.11_2
2 7/8	10.111_2
63/64	0.111111_2

- Деление на 2 может выполняться сдвигом вправо, …
- … а умножение на 2 – сдвигом влево
- Числа вида $0.11111\ldots_2$
  - На один «шаг» меньше чем 1.0
  - Используется специальное обозначение $1.0 - \varepsilon$

Представимые рациональные числа

- Ограничение
  - Можно представить рациональные числа только вида $x/2^k$
  - Другие рациональные числа представляются повторяющимися группами бит

- Число Представление

  1/3 0.0101010101[01]…$_2$

  1/5 0.00110011001[0011]…$_2$

  1/10 0.000110011001[0011]…$_2$

Представление чисел с плавающей точкой

- Численное представление

$$(-1)^s \times M \times 2^E$$

  - Знаковый бит s определяет, является число положительным или отрицательным
  - Мантисса M – дробное число в полуинтервале [1.0,2.0).
  - Порядок E определяет степень 2 в третьем множителе

- Кодировка
  - Наибольший значащий бит s – знаковый бит s
  - Поле exp кодирует порядок E
  - Поле frac кодирует мантиссу M

Размеры чисел

- Одинарная точность: 32 бита. Тип – `float`.
  - Знак s — 1 бит
  - Мантисса M — 23 бита
  - Порядок E — 8 битов
- Двойная точность: 64 бита. Тип – `double`.
  - Знак s — 1 бит
  - Мантисса M — 52 бита
  - Порядок E — 11 битов
- Нормализация чисел
  - Нормализованное значение – порядок не принимает «крайние» значения (одни нули или одни единицы)
  - Денормализованное значение – порядок либо ноль, либо 11…11

Нормализованное число

- Значение: `float f = 15213.0;`

  $15213_{10} = 11101101101101_2$

  $= 1.1101101101101_2 \times 2^{13}$

- Мантисса

  M = 1.1101101101101_2

  frac = $11011011011010000000000_2$

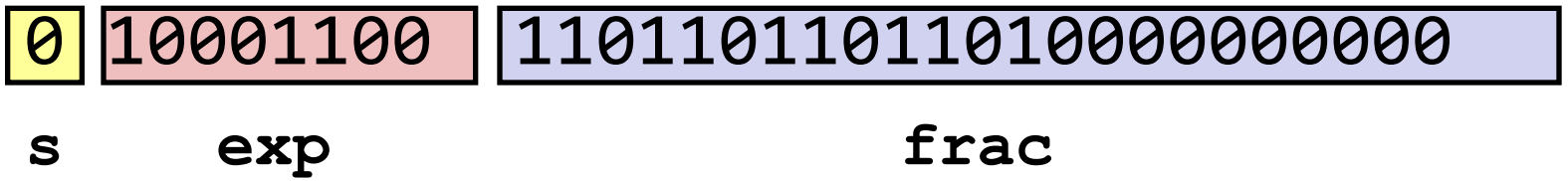
- Порядок

  E = 13

  Смещение = 127

  Exp = E + Смещение = 140 = 10001100_2

- Итого:



Денормализованное число

- Условие: exp = 000…0
- Значение порядка: `E = –Смещение + 1` (вместо `E = 0 – Смещение`)
- Мантисса кодируется с ведущим `0`: `M = 0.`xxx…x_2
  - xxx…x: биты поля `frac`
- Примеры
  - exp = 000…0, frac = 000…0
    - Представляет число ноль
    - Различные кодировки для +0 и –0
  - exp = 000…0, frac ≠ 000…0
    - Кодируются числа близкие к 0.0
    - Распределены по числовой прямой с равным шагом

Особые числа

- Условие: exp = 111…1
- Пример: exp = 111…1, frac = 000…0
  - Представляет бесконечно большое число ∞
    (как положительное, так и отрицательное)
  - Требуются для операций в которых может произойти переполнение

    `1.0/0.0 = –1.0/–0.0 = +∞`

    `1.0/–0.0 = –∞`
- Пример: exp = 111…1, frac ≠ 000…0
  - Not-a-Number (NaN)
  - Используется в ситуациях, когда значение операции не определено

    `sqrt(–1)`

    `∞ – ∞`

    `∞ × 0`

Диапазоны значений