Лекция 14

28 марта

Выравнивание стека - итоги

- 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
  - Быстрое копирование блока данных размером 16 байт
  - Аварийное завершение работы (#GP), если начальный адрес блока не выровнен по границе в 16 байт

Отказ от указателя фрейма

```
void f(int x, int y) {
    int numerator =
        (x + y) * (x - y);
    int denominator =
        x * x + y * y;
    if (0 == denominator) {
        denominator = 1;
    }
    return (100 * numerator) /
                denominator;
}
```

Регистр	Значение
esi	**y**
ecx	**X**

-fomit-frame-pointer

```
f:
    ; пролог
    sub     esp, 8
    mov     dword [esp+4], esi
    mov     esi, dword [esp+16]
    mov     ecx, dword [esp+12]
    mov     dword [esp], ebx
    ; …
```

Сохраненный регистр	адрес
esi	**[esp + 4]**
ebx	**[esp]**

Отказ от указателя фрейма

```
void f(int x, int y) {
    int numerator =
        (x + y) * (x - y);
    int denominator =
        x * x + y * y;
    if (0 == denominator) {
        denominator = 1;
    }
    return (100 * numerator) /
                denominator;
}
```

Регистр	Значение
esi	**y**
ecx	**X**

```
f:
    ; …
    mov     edx, esi
    imul    edx, esi ; edx = y^2
    mov     eax, ecx
    imul    eax, ecx ; eax = x^2
    mov     ebx, edx
    add     ebx, eax
                ; ebx = x^2 + y^2
    jne     .L2
    mov     ebx, 1
.L2
    ; …
```

Отказ от указателя фрейма

```
void f(int x, int y) {
   int numerator =
      (x + y) * (x - y);
   int denominator =
      x * x + y * y;
   if (0 == denominator) {
      denominator = 1;
   }
   return (100 * numerator) /
               denominator;
}
```

```
f:
   ; …
.L2
   lea      edx, [esi+ecx]
   sub      ecx, esi
   imul     edx, ecx
   ; …
```

Регистр	Значение
esi	**y**
ecx	**x**
ebx	x^2 + y^2

Отказ от указателя фрейма

```
void f(int x, int y) {
    int numerator =
        (x + y) * (x - y);
    int denominator =
        x * x + y * y;
    if (0 == denominator) {
        denominator = 1;
    }
    return (100 * numerator) /
                denominator;
}
```

```
f:
    ; …
    imul    edx, edx, 100
    mov     eax, edx
    sar     edx, 31
    idiv    ebx
    ; …
```

Регистр	Значение
esi	**y**
ecx	**x**
ebx	x^2 + y^2
edx	(x + y) * (x - y)

Отказ от указателя фрейма

```
void f(int x, int y) {
   int numerator =
      (x + y) * (x - y);
   int denominator =
      x * x + y * y;
   if (0 == denominator) {
      denominator = 1;
   }
   return (100 * numerator) /
             denominator;
}
```

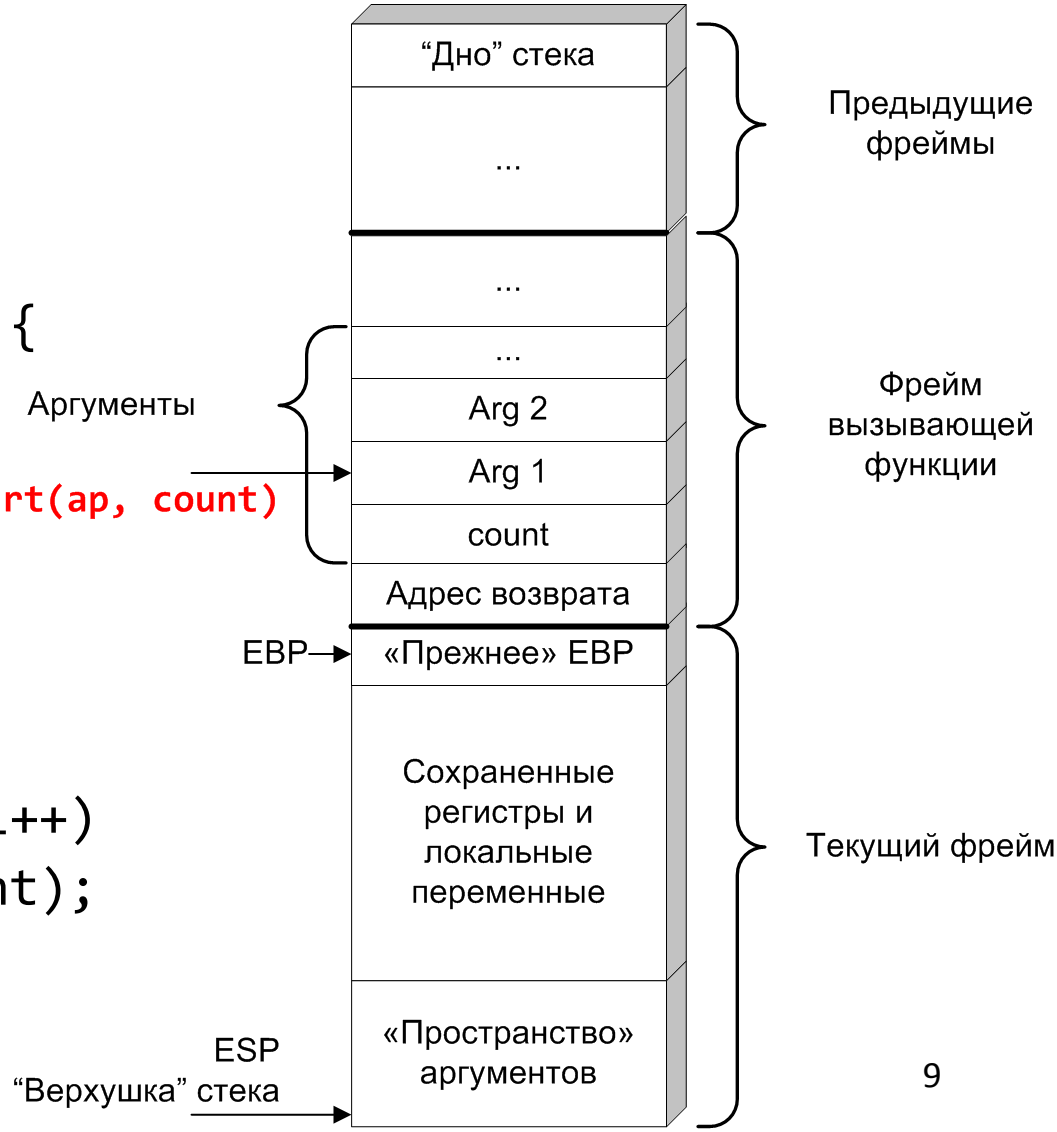
```
f:
   ; …
   ; эпилог
   mov     esi, dword [esp+4]
   mov     ebx, dword [esp]
   add     esp, 8
   ret
```

Переменное количество параметров

- Многоточие (...) помещается в конце списка параметров.
- Тип данных
  - va_list
- Макрокоманды
  - va_start(va_list, last fixed param)
  - va_arg(va_list, cast type)
  - va_end(va_list)

Переменное количество параметров

```
#include <stdarg.h>

int average(int count, ...) {
    va_list ap;
    int sum = 0;
    if (0 == count) {
       return -1;
    }
    va_start(ap, count);
    for (int i=0; i<count; i++)
        sum += va_arg(ap, int);
    va_end(ap);
    return sum/count;
}
```



Переменное количество параметров

```
#include <stdarg.h>

int average(int count, ...) {
    va_list ap;
    int sum = 0;
    if (0 == count) {
       return -1;
    }
    va_start(ap, count);
    for (int i=0; i<count; i++)
        sum += va_arg(ap, int);
    va_end(ap);
    return sum/count;
}
```

```
average:
  push ebp
  mov  ebp, esp
  push ebx
  mov  ecx, dword [ebp+8]
  test ecx, ecx
  jne  .L11
  mov  eax, -1
  pop  ebx
  pop  ebp
  ret
.L11:
; …
```

Переменное количество параметров

```
#include <stdarg.h>

int average(int count, ...) {
    va_list ap;
    int sum = 0;
    if (0 == count) {
       return -1;
    }
    va_start(ap, count);
    for (int i=0; i<count; i++)
        sum += va_arg(ap, int);
    va_end(ap);
    return sum/count;
}
```

```
average:
  ; …
.L11:
  xor  eax, eax
  xor  edx, edx
  test ecx, ecx
  lea  ebx, [ebp+12]
  jle  .L5
.L8:
  add  edx, dword [ebx+eax*4]
  add  eax, 1
  cmp  ecx, eax
  jg   .L8
.L5:
```

Переменное количество параметров

```
#include <stdarg.h>

int average(int count, ...) {
    va_list ap;
    int sum = 0;
    if (0 == count) {
       return -1;
    }
    va_start(ap, count);
    for (int i=0; i<count; i++)
        sum += va_arg(ap, int);
    va_end(ap);
    return sum/count;
}
```

```
average:
  ; …
.L11:
  xor  eax, eax
  xor  edx, edx
  test ecx, ecx
  lea  ebx, [ebp+12]
  jle  .L5
.L8:
  add  edx, dword [ebx+eax*4]
  add  eax, 1
  cmp  ecx, eax
  jg   .L8
.L5:
```

Переменное количество параметров

```
#include <stdarg.h>

int average(int count, ...) {
    va_list ap;
    int sum = 0;
    if (0 == count) {
       return -1;
    }
    va_start(ap, count);
    for (int i=0; i<count; i++)
        sum += va_arg(ap, int);
    va_end(ap);
    return sum/count;
}
```

```
average:
  ; …
.L5:
  mov  eax, edx
  sar  edx, 31
  idiv ecx
  pop  ebx
  pop  ebp
  ret
```

Вызов функции по указателю

```
#include <stdio.h>

int sum(int, int);
int sub(int, int);
int mul(int, int);
int div(int, int);

typedef int (*arith)(int, int);
int eval(arith pf, int x, int y);

int main() {
   int a = 1, b = 2, c;
   arith pf = sum;
   c = eval(pf, a, b);
   printf("%d\n", c);
   return 0;
}
```

```
int eval(arith pf, int x, int y) {
   return pf(x, y);
}

int sum(int x, int y) {
   return x + y;
}

int sub(int x, int y) {
   return x - y;
}

int mul(int x, int y) {
   return x * y;
}

int div(int x, int y) {
   return x / y;
}
```

Вызов функции по указателю

```
int sum(int x, int y) {
   return x + y;
}

int sub(int x, int y) {
   return x - y;
}
```

```
global sum
sum:
        push    ebp
        mov     ebp, esp
        mov     eax, dword [ebp+12]
        add     eax, dword [ebp+8]
        pop     ebp
        ret

global sub
sub:
        push    ebp
        mov     ebp, esp
        mov     eax, dword [ebp+8]
        sub     eax, dword [ebp+12]
        pop     ebp
        ret
```

Вызов функции по указателю

```
int mul(int x, int y) {
   return x * y;
}

int div(int x, int y) {
   return x / y;
}
```

```
global mul
mul:
        push    ebp
        mov     ebp, esp
        mov     eax, dword [ebp+12]
        imul    eax, dword [ebp+8]
        pop     ebp
        ret

global div
div:
        push    ebp
        mov     ebp, esp
        mov     edx, dword [ebp+8]
        mov     eax, edx   ; cdq
        sar     edx, 31    ;
        idiv    dword [ebp+12]
        pop     ebp
        ret
```

Вызов функции по указателю

```
int main() {
   int a = 1, b = 2, c;
   arith pf = sum;
   c = eval(pf, a, b);
   printf("%d\n", c);
   return 0;
}
```

```
%include ‘io.inc’
section .rodata
LC0: db '%d', 10, 0

CEXTERN printf

section .text
global CMAIN
CMAIN:
        lea     ecx, [esp+4]
        and     esp, -16
        push    dword [ecx-4]
```

```
push    ebp
mov     ebp, esp
push    ecx
sub     esp, 20
mov     dword [esp+8], 2
mov     dword [esp+4], 1
mov     dword [esp], sum
call    eval
mov     dword [esp+4], eax
mov     dword [esp], LC0
call    printf
mov     eax, 0
add     esp, 20
pop     ecx
pop     ebp
lea     esp, [ecx-4]
ret
```

Вызов функции по указателю

```
int eval(arith pf, int x, int y) {
   return pf(x, y);
}
```

```
global eval
eval:
        push    ebp
        mov     ebp, esp
        sub     esp, 8
        mov     edx, dword [ebp+12]
        mov     eax, dword [ebp+16]
        mov     ecx, dword [ebp+8]
        mov     dword [esp], edx
        mov     dword [esp+4], eax
        call    ecx
        mov     esp, ebp
        pop     ebp
        ret
```

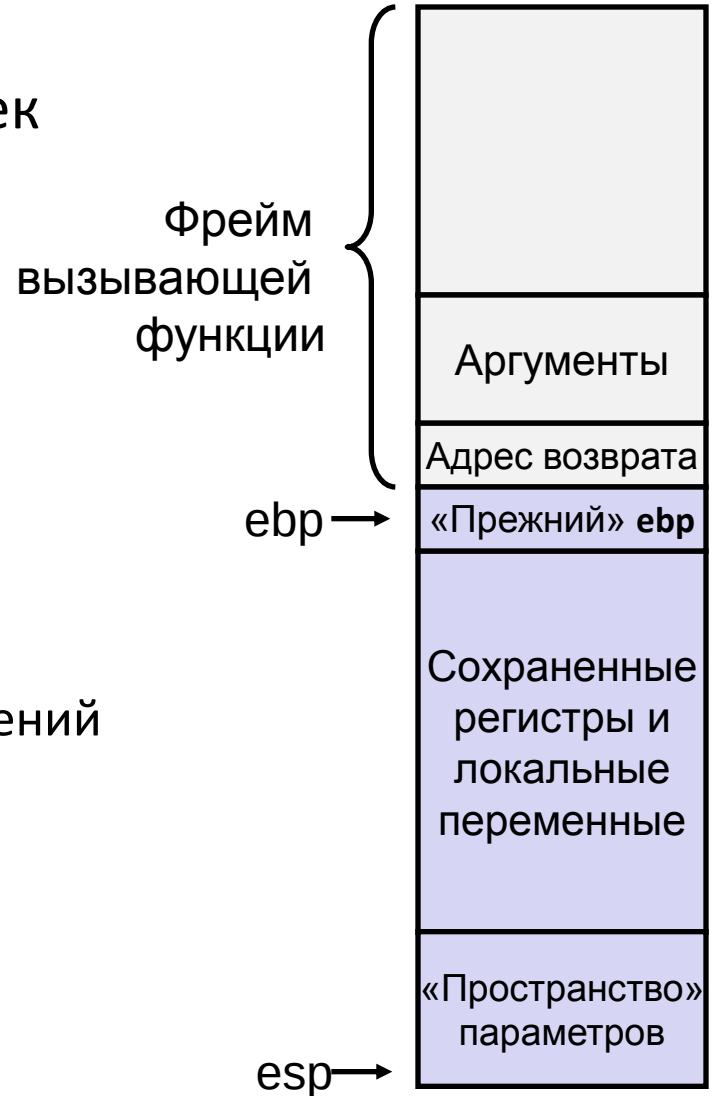
Ассемблерные вставки

- 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



Коллоквиум №1
Типовые ошибки

- **Нарушение синтаксиса языка Си**
- Неверное сопоставление типов языка Си и операндов машинных инструкций
  - Неверные операции: маскирование, сдвиги …
- Указатели
  - уровень косвенности
- Адресная арифметика
  - Си – смещение в количестве элементов типа
    Асм – смещение в байтах
- **Короткая логика**
  - **часть выражения может не вычисляться**
- Тип _Bool
  - true/false

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);
}
```

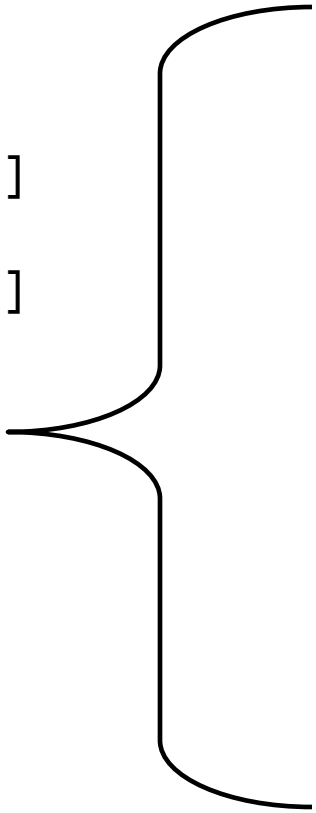
Сравнение строк равного размера

```
%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
```

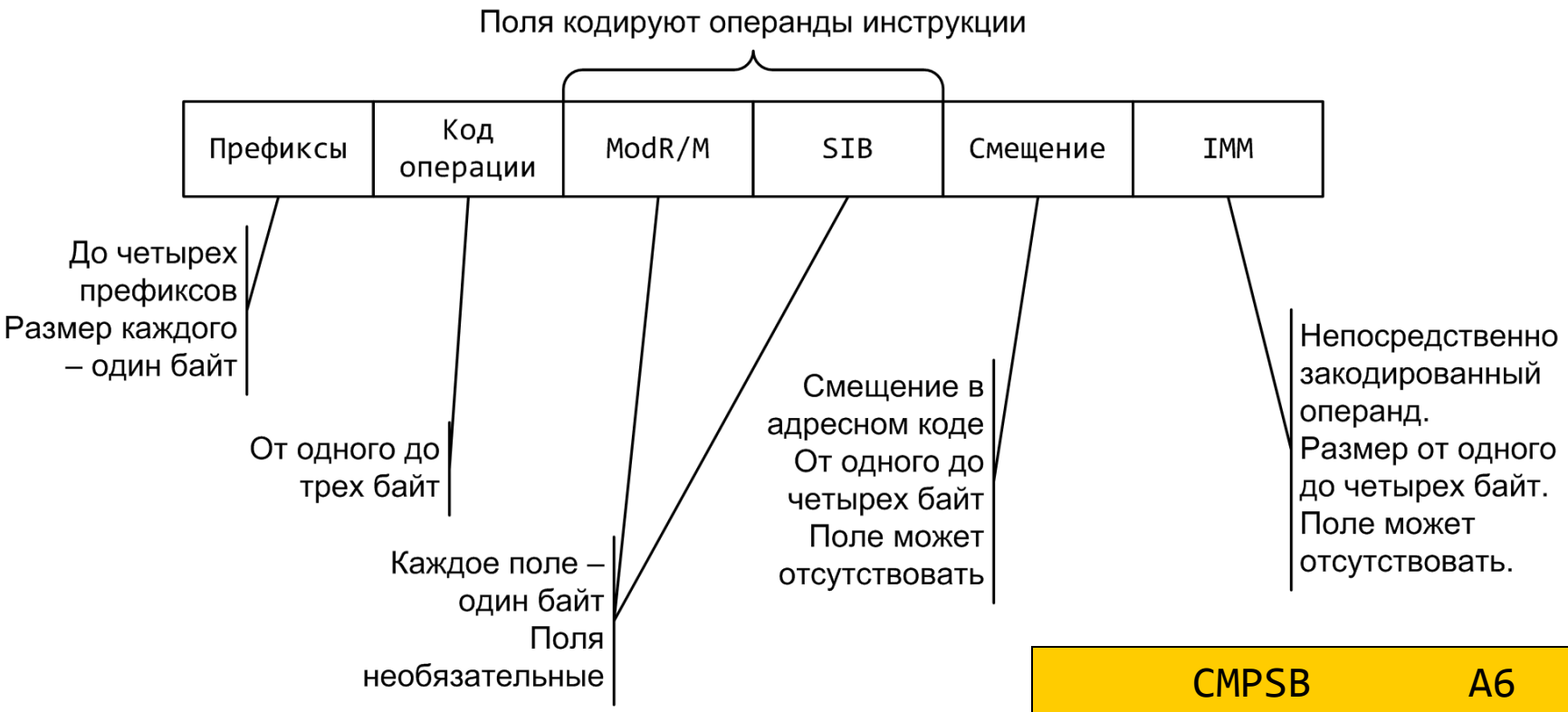
Сравнение строк равного размера

```
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[25]

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

Приводим поведение функции к стандартному виду
- меньше -1
- равно 0
- больше 1

```
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
```

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

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