Лекция 13

26 марта

Ассемблерные вставки

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

```
int a=10, b;
asm ("movl %1, %%eax;
      movl %%eax, %0;" /* ассемблерная вставка */
     :"=r"(b)          /* выходные операнды    */
     :"r"(a)           /* входные операнды     */
     :"%eax"           /* разрушаемые регистры */
     );
```

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

```
#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
```

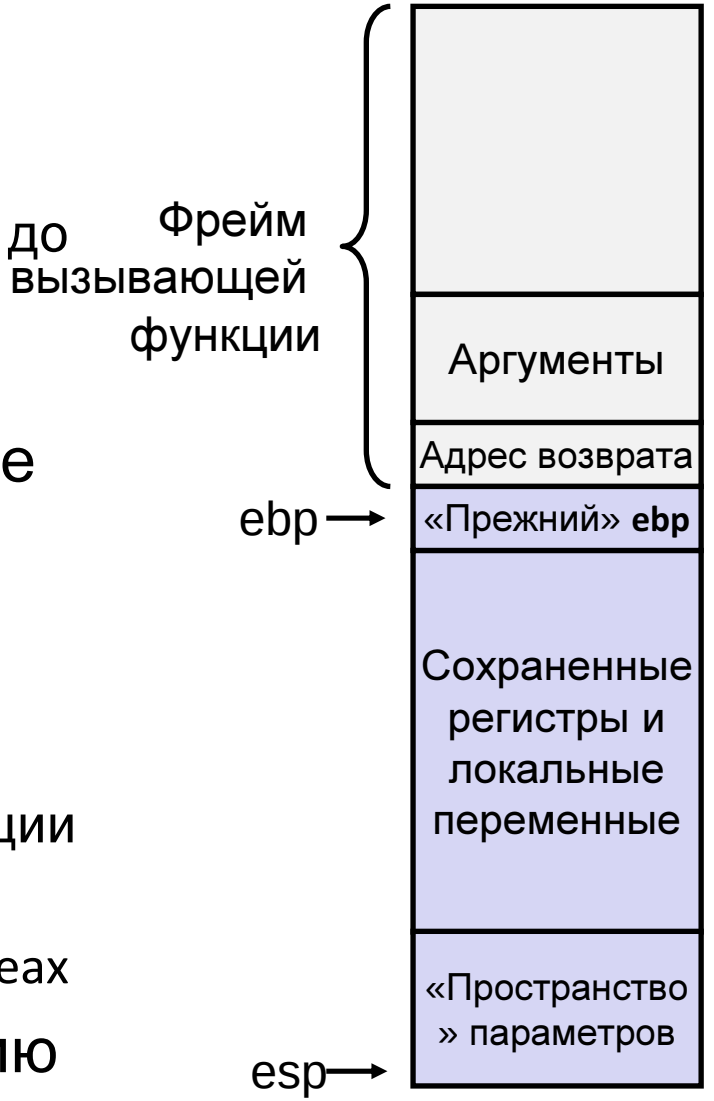
Вызов функции по указателю

```
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
```

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);
}
```

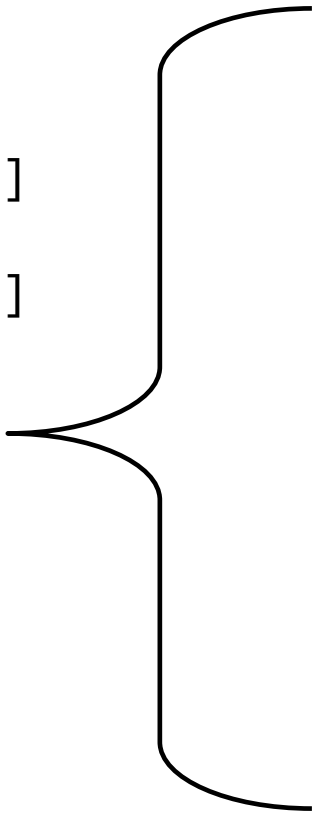
Сравнение строк

```
%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
```

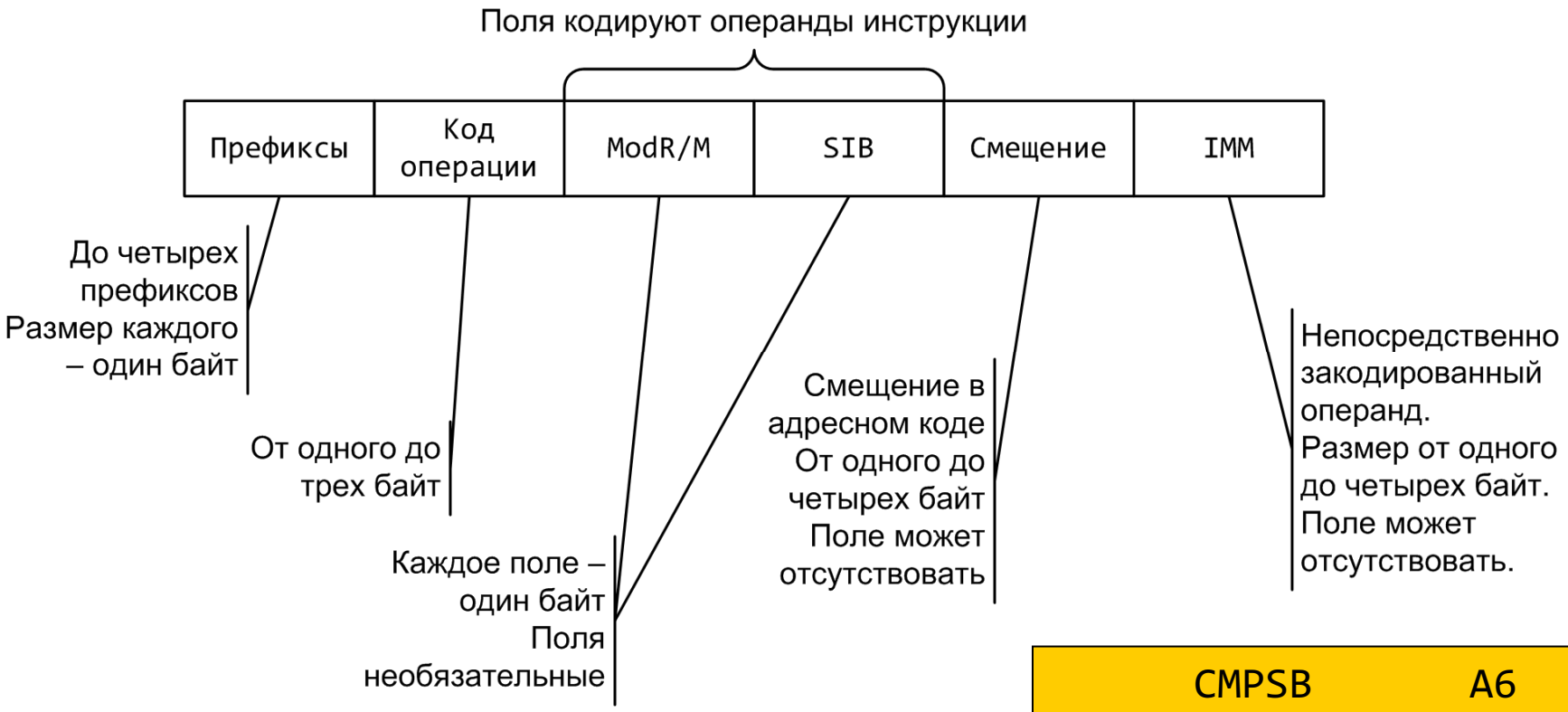
Сравнение строк

```
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
```

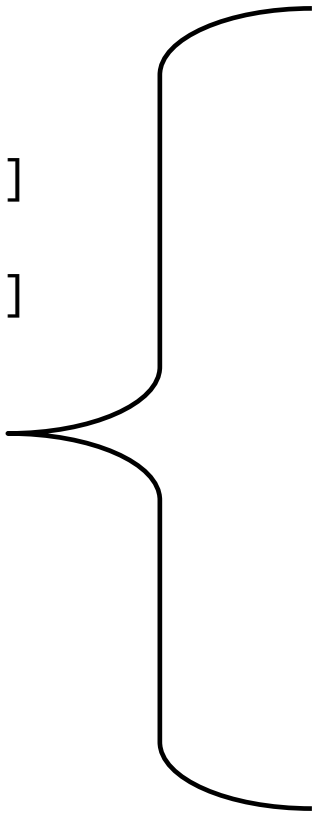
Префикс повтора

- 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

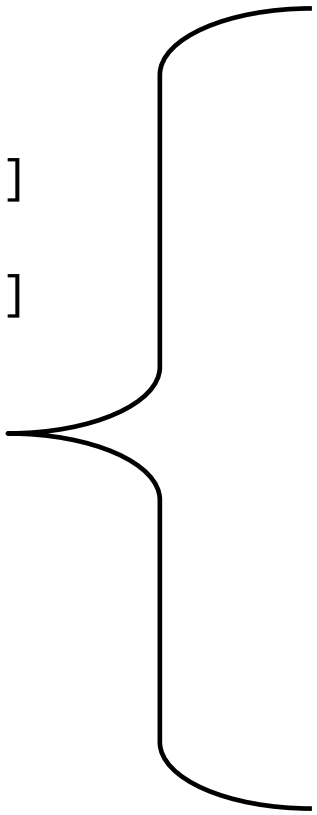
```
my_strncmp:
   push ebp
   mov ebp, esp
   push esi
   push edi
   xor eax, eax
   mov ecx, dword [ebp + 16]
   mov esi, dword [ebp + 8]
   mov edi, dword [ebp + 12]
   ;…


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



```
   ; …
   cld
   repe cmpsb
   je .end
   mov eax, dword [ebp + 16]
   sub eax, ecx
.end:
   ; …
```

Сравнение строк – приведение к стандартному поведению функции strncmp

```
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 ecx, -1
   mov eax, 1
   cmovb 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