# Архитектура ЭВМ и язык ассемблера

Семинар #32:

1. Организация цикла на условных переходах.
2. Пример кодогенерации: `if+else` и `switch`.
3. Компактное кодирование цикла, инструкция LOOP.
4. Игра: заблудившись в трёх массивах.

20.02.2026

# Организация цикла на условных переходах

# Организация цикла для чисел Фибоначчи

**Математический объект:**

$F_0 = 0$

$F_1 = 1$

$F_n = F_{n-1} + F_{n-2}$ для $n > 1$

**Корректность алгоритма**

**Инвариант цикла:**

$prev = F_i$

$cur = F_{i+1}$

**При выходе из цикла:**

$i = n$, $prev = F_n$

**Алгоритм на псевдокоде (почти C):**

```
if (n == 0)
{
    return 0;
}
i = 0, prev = 0, cur = 1;
do
{
    next = prev + cur;
    prev = cur; cur = next;
    i++;
}
while (i < n);
return prev;
```

# Организация цикла для чисел Фибоначчи

**Алгоритм на псевдокоде (почти C):**

```
if (n == 0)
{
    return 0;
}
i = 0, prev = 0, cur = 1;
do
{
    next = prev + cur;
    prev = cur; cur = next;
    i++;
}
while (i < n);
return prev;
```

**Реализация алгоритма**

1. Ввод-вывод.
   **ПРОВЕРКА ВВОДА!**
2. Размеры переменных и переполнение.
   **Ограничения на входы!**
3. Размещение переменных
   - Регистры
   - Ячейки памяти.

   **Вызовы функций по ABI!!!**
4. Выбор инструкций:
   - Ветвления
   - Арифметика

# Вызов функций по ABI

Table 2.3: Register Usage

| Register | Usage | Preserved across function calls |
|---|---|---|
| %eax | scratch register; also used to return integer and pointer values from functions; also stores the address of a returned struct or union | No |
| %ebx | callee-saved register; also used to hold the GOT pointer when making function calls via the PLT | Yes |
| %ecx | scratch register | No |
| %edx | scratch register; also used to return the upper 32bits of some 64bit return types | No |
| %esp | stack pointer | Yes |
| %ebp | callee-saved register; optionally used as frame pointer | Yes |
| %esi | callee-saved register | yes |
| %edi | callee-saved register | yes |

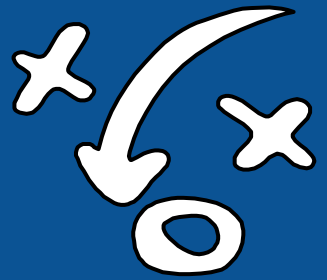
System V i386 ABI -> Function Calling Sequence -> 2.3.1. Registers

# Организация цикла для чисел Фибоначчи

```
    ; Выводим приветственную строку.
    mov     eax, inputstr
    call    io_print_string

    ; Считываем номер числа Фибоначчи для рассчёта.
    call    io_get_udec
    ; ESI - номер итогового числа Фибоначчи
    mov     esi, eax

    ; Выводим нулевое число Фибоначчи
    cmp     esi, 0
    jne     .init_fibs_computation

    mov     eax, 0
    call    io_print_udec
    call    io_newline
    jmp     .exit

.init_fibs_computation:
    ; ECX - номер текущего числа Фибоначчи
    mov     ecx, 0
    ; EDI - предыдущее число Фибоначчи
    mov     edi, 0
    ; EBX - текущее число Фибоначчи
    mov     ebx, 1
.compute_next_fibs:
```

```
.compute_next_fibs:
    ; EDI := EBX
    ; EBX := EDI + EBX
    add     edi, ebx
    xchg    edi, ebx

    ; Производим проверку на переполнение.
    jc      .detect_overflow

    inc     ecx
    cmp     ecx, esi
    jl      .compute_next_fibs

    ; Выводим результирующее число Фибоначчи.
    mov     eax, outputstr
    call    io_print_string

    mov     eax, edi
    call    io_print_udec
    call    io_newline
    jmp     .exit

.detect_overflow:
    mov     eax, overflowstr
    call    io_print_string

.exit:
    xor     eax, eax
    ret
```

# Пример кодогенерации: if+else и switch



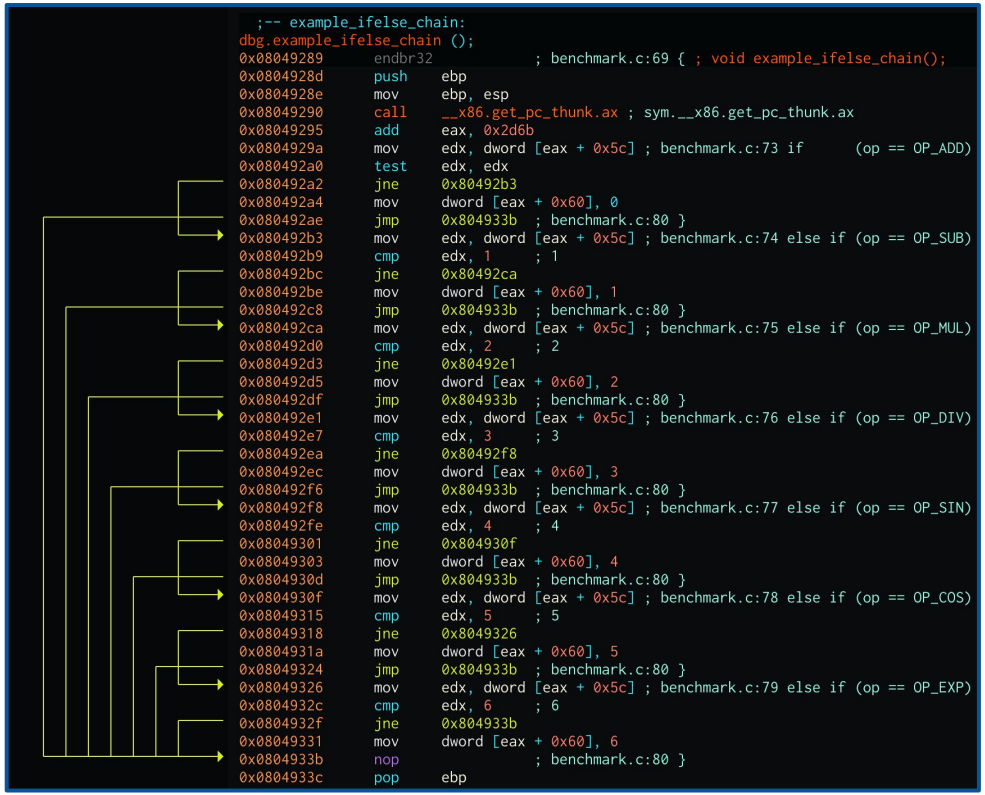
# Пример кодогенерации: цепочка if+else

```
typedef enum
{
    OP_ADD,
    OP_SUB,
    OP_MUL,
    OP_DIV,
    OP_SIN,
    OP_COS,
    OP_EXP
} OperationType;
```

```
void example_ifelse_chain(void)
{
    static OperationType op;
    static int result;

    if      (op == OP_ADD) { result = 0; }
    else if (op == OP_SUB) { result = 1; }
    else if (op == OP_MUL) { result = 2; }
    else if (op == OP_DIV) { result = 3; }
    else if (op == OP_SIN) { result = 4; }
    else if (op == OP_COS) { result = 5; }
    else if (op == OP_EXP) { result = 6; }
}
```

# Пример кодогенерации: цепочка if+else

```
void example_ifelse_chain(void)
{
    static OperationType op;
    static int result;

    if      (op == OP_ADD) { result = 0; }
    else if (op == OP_SUB) { result = 1; }
    else if (op == OP_MUL) { result = 2; }
    else if (op == OP_DIV) { result = 3; }
    else if (op == OP_SIN) { result = 4; }
    else if (op == OP_COS) { result = 5; }
    else if (op == OP_EXP) { result = 6; }
}
```

# Пример кодогенерации: switch

```
void example_switch(void)
{
    static OperationType type;
    static int result;

    switch (type)
    {
    case OP_ADD:
        result = 0; break;
    case OP_SUB:
        result = 1; break;
    case OP_MUL:
        result = 2; break;
    case OP_DIV:
        result = 3; break;
    case OP_SIN:
        result = 4; break;
    case OP_COS:
        result = 5; break;
    case OP_EXP:
        result = 6; break;
    }
}
```

# Пример кодогенерации: JUMP-таблица

```
0x08049345      call    __x86.get_pc_thunk.ax ; sym.__x8
0x0804934a      add     eax, 0x2cb6
0x0804934f      mov     edx, dword [eax + 0x64] ; benchm
0x08049355      cmp     edx, 6        ; 6
0x08049358      ja      0x80493ba
0x0804935a      shl     edx, 2
0x0804935d      mov     edx, dword [edx + eax - 0x1ff8]
0x08049364      add     edx, eax
0x08049366      jmp     edx
```

Таблица в секции `.rodata`

```
0x0804a008          .dword 0xffffd369
0x0804a00c          .dword 0xffffd375
0x0804a010          .dword 0xffffd381
0x0804a014          .dword 0xffffd38d
0x0804a018          .dword 0xffffd399
0x0804a01c          .dword 0xffffd3a5
0x0804a020          .dword 0xffffd3b1
```

EAX = 0x0804934A + 0x2CB6

**Адрес JMP-таблицы:**

ADDR = EAX - 0x1FF8 = 0x0804A008

**Смещение в таблице:**

EDX = type, OFFSET = 4 * EDX

**Адрес прыжка при type=0:**

EAX + 0xFFFFD369 = 0x08049369

**Адрес прыжка при type=1:**

EAX + 0xFFFFD375 = 0x08049375

```
0x08049366      jmp     edx
;-- .L33:
0x08049369      mov     dword [eax + 0x68], 0
0x08049373      jmp     0x80493bc  ; dbg.examp
;-- .L32:
0x08049375      mov     dword [eax + 0x68], 1
0x0804937f      jmp     0x80493bc  ; dbg.examp
```

# Компактное кодирование циклов, инструкция LOOP

# Инструкция LOOP

| Команда | Описание |
|---|---|
| JCXZ/JECXZ | Переход выполняется, если значение регистра CX/ECX равно нулю. |
| LOOP | Переход выполняется, если значение регистра ECX не равно нулю. |
| LOOPZ/LOOPE | Переход выполняется, если значение регистра ECX не равно нулю и флаг ZF установлен. |
| LOOPNZ/LOOPNE | Переход выполняется, если значение регистра ECX не равно нулю и флаг ZF сброшен. |

## LOOP/LOOP*cc*—Loop According to ECX Counter

| Opcode | Instruction | Op/ En | 64-Bit Mode | Compat/ Leg Mode | Description |
|---|---|---|---|---|---|
| E2 cb | LOOP rel8 | D | Valid | Valid | Decrement count; jump short if count ≠ 0. |
| E1 cb | LOOPE rel8 | D | Valid | Valid | Decrement count; jump short if count ≠ 0 and ZF = 1. |
| E0 cb | LOOPNE rel8 | D | Valid | Valid | Decrement count; jump short if count ≠ 0 and ZF = 0. |

Пример кода: поиск максимума + перевод числа в HEX

# Игра: заблудившись в трёх массивах

# Запись array[x][y] = 1, но в какой массив?

```
mov     ebx, dword [x]          (1)
mov     ecx, dword [y]
mov     edx, array
shl     ebx, 0x2
add     ecx, ebx
mov     dword [edx + 4*ecx], 0x1
```

```
mov     ecx, dword [x]          (2)
mov     edx, array
mov     edx, dword [edx + 4*ecx]
mov     ecx, dword [y]
shl     ecx, 0x2
add     edx, ecx
mov     dword [edx], 0x1
```

```
(A) int* array[4];
```

```
(B) int** array2;
```

```
(C) int array[4][4];
```

```
mov     edx, array              (3)
mov     edx, dword [edx]
mov     ecx, dword [x]
shl     ecx, 0x2
add     edx, ecx
mov     edx, dword [edx]
mov     eax, dword [y]
shl     eax, 0x2
add     eax, edx
mov     dword [eax], 0x1
```

# Запись array[x][y] = 1, но в какой массив?

```
mov     ebx, dword [x]          (1)
mov     ecx, dword [y]
mov     edx, array
shl     ebx, 0x2
add     ecx, ebx
mov     dword [edx + 4*ecx], 0x1
```

```
mov     ecx, dword [x]          (2)
mov     edx, array
mov     edx, dword [edx + 4*ecx]
mov     ecx, dword [y]
shl     ecx, 0x2
add     edx, ecx
mov     dword [edx], 0x1
```

```
(A) int* array[4];
```

```
(B) int** array2;
```

```
(C) int array[4][4];
```

```
mov     edx, array              (3)
mov     edx, dword [edx]
mov     ecx, dword [x]
shl     ecx, 0x2
add     edx, ecx
mov     edx, dword [edx]
mov     eax, dword [y]
shl     eax, 0x2
add     eax, edx
mov     dword [eax], 0x1
```

# Вопросы?

Красивые иконки взяты с сайта handdrawngoods.com