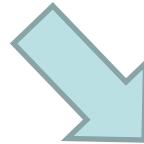


# Лекция 8

7 марта

# Оператор while

```
int pcount_while(unsigned x) {  
    int result = 0;  
    while (x) {  
        result += x & 0x1;  
        x >>= 1;  
    }  
    return result;  
}
```



```
int pcount_do(unsigned x) {  
    int result = 0;  
    if (!x) goto done;  
loop:  
    result += x & 0x1;  
    x >>= 1;  
    if (x)  
        goto loop;  
done:  
    return result;  
}
```

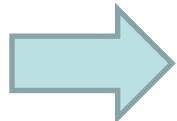


```
int pcount_do(unsigned x) {  
    int result = 0;  
loop:  
    if (!x) goto done;  
    result += x & 0x1;  
    x >>= 1;  
    goto loop;  
done:  
    return result;  
}
```

# Оператор for

```
#define WSIZE 8*sizeof(int)

int pcount_for(unsigned x) {
    int i;
    int result = 0;
    for (i = 0; i < WSIZE; i++) {
        unsigned mask = 1 << i;
        result += (x & mask) != 0;
    }
    return result;
}
```



```
int pcount_for_gt(unsigned x) {
    int i;
    int result = 0;
    i = 0;
    if (!(i < WSIZE))
        goto done;
loop:
{
    unsigned mask = 1 << i;
    result += (x & mask) != 0;
}
i++;
if (i < WSIZE)
    goto loop;
done:
    return result;
}
```

```

int fib(int x) { // x >= 1
    int i;
    int p_pred = 0;
    int pred = 1;
    int res = 1;
    x--;
    for (i = 0; i < x; i++) {
        res = p_pred + pred;
        p_pred = pred;
        pred = res;
    }
    return res;
}

```

Регистр	Значение
ecx	x
edx	p_pred
ebx	pred
eax	res

```

fib:
    push    ebp
    mov     ebp, esp
    push    ebx

    mov     ecx, dword [ebp + 8] ; x
    xor     edx, edx           ; p_pred
    mov     ebx, 1              ; pred
    mov     eax, 1              ; res
    dec     ecx

    jecxz .end

.loop:
    lea     eax, [edx + ebx]
    mov     edx, ebx
    mov     ebx, eax
    loop   .loop

.end:
    pop    ebx
    pop    ebp
    ret

```

```

int fib(int x) { // x >= 1
    int i;
    int p_pred = 0;
    int pred = 1;
    int res = 1;
    x--;
    for (i = 0; i < x; i++) {
        res = p_pred + pred;
        p_pred = pred;
        pred = res;
    }
    return res;
}

```

Регистр	Значение
ecx	x
edx	p_pred
ebx	pred
eax	res

```

fib:
    push    ebp
    mov     ebp, esp
    push    ebx

    mov     ecx, dword [ebp + 8] ; x
    xor     edx, edx           ; p_pred
    mov     ebx, 1              ; pred
    mov     eax, 1              ; res
    dec     ecx

    jecxz .end

.loop:
    lea     eax, [edx + ebx]
    mov     edx, ebx
    mov     ebx, eax
    loop   .loop

.end:
    pop    ebx
    pop    ebp
    ret

```

```

int fib(int x) { // x >= 1
    int i;
    int p_pred = 0;
    int pred = 1;
    int res = 1;
    x--;
    for (i = 0; i < x; i++) {
        res = p_pred + pred;
        p_pred = pred;
        pred = res;
    }
    return res;
}

```

Регистр	Значение
ecx	x
edx	p_pred
ebx	pred
eax	res

```

fib:
    push    ebp
    mov     ebp, esp
    push    ebx

    mov     ecx, dword [ebp + 8] ; x
    xor     edx, edx           ; p_pred
    mov     ebx, 1              ; pred
    mov     eax, 1              ; res
    dec     ecx

    jecxz .end

.loop:
    lea     eax, [edx + ebx]
    mov     edx, ebx
    mov     ebx, eax
    loop   .loop

.end:
    pop    ebx
    pop    ebp
    ret

```

```

int fib(int x) { // x >= 1
    int i;
    int p_pred = 0;
    int pred = 1;
    int res = 1;
    x--;
    for (i = 0; i < x; i++) {
        res = p_pred + pred;
        p_pred = pred;
        pred = res;
    }
    return res;
}

```

Регистр	Значение
ecx	x
edx	p_pred
ebx	pred
eax	res

```

fib:
    push    ebp
    mov     ebp, esp
    push    ebx

    mov     ecx, dword [ebp + 8] ; x
    xor     edx, edx           ; p_pred
    mov     ebx, 1              ; pred
    mov     eax, 1              ; res
    dec     ecx

    jecxz .end

.loop:
    lea     eax, [edx + ebx]
    mov     edx, ebx
    mov     ebx, eax
    loop   .loop

.end:
    pop    ebx
    pop    ebp
    ret

```

# Обратная задача

f:

```
...
mov edx, dword [ebp+8] ; (1)
mov eax, 0              ; (2)
test edx, edx           ; (3)
je .L7                  ; (4)
.L10:
                ; (5)
xor eax, edx           ; (5)
shr edx, 1              ; (6)
jne .L10                ; (7)
.L7:
                ; (8)
and eax, 1              ; (8)
...
```

```
int f(unsigned x) {
    int val = 0;
    while (_____) {
        _____;
        _____;
    }
    return _____;
}
```

# Передача управления

## Си

- if
- if-else
- **switch**
- do-while
- while
- for
- goto
- break
- continue
- return

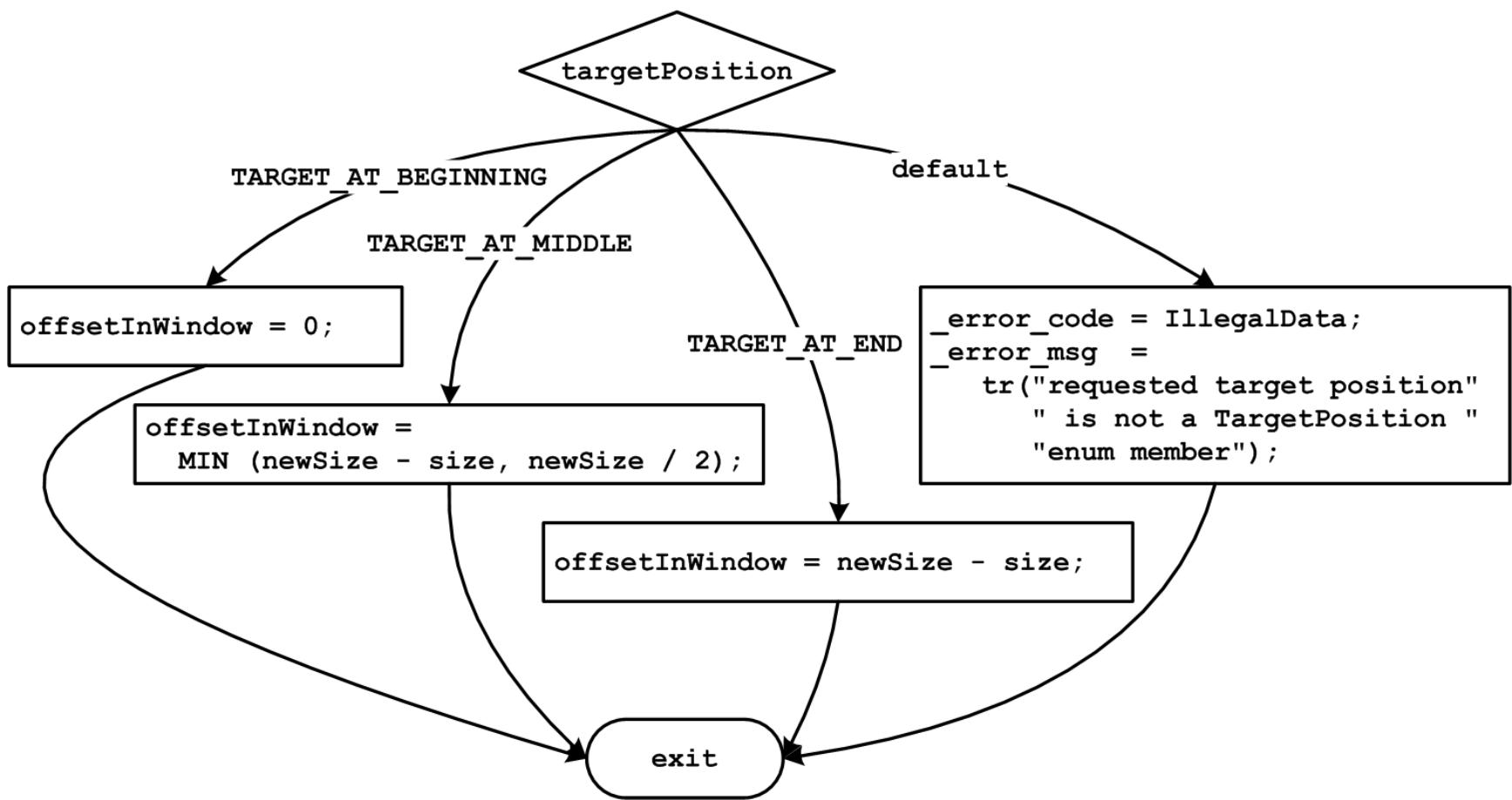
## Ассемблер

- JMP
- Jcc
- CALL
- RET
- CMOVcc

```
enum TargetPosition {
    TARGET_AT_BEGINNING,
    TARGET_AT_MIDDLE,
    TARGET_AT_END
};

switch (targetPosition){

case TARGET_AT_BEGINNING:
    offsetInWindow = 0;
    break;
case TARGET_AT_MIDDLE:
    offsetInWindow = MIN (newSize - size, newSize / 2);
    break;
case TARGET_AT_END:
    offsetInWindow = newSize - size;
    break;
default:
    _error_code = IllegalData;
    _error_msg  = tr("requested target position"
                    " is not a TargetPosition enum member");
}
```



```
enum TargetPosition {
    TARGET_AT_BEGINNING,
    TARGET_AT_MIDDLE,
    TARGET_AT_END
};

if (TARGET_AT_BEGINNING == targetPosition) {
    offsetInWindow = 0;
} else if (TARGET_AT_MIDDLE == targetPosition) {
    offsetInWindow = MIN (newSize - size, newSize / 2);
} else if (TARGET_AT_END == targetPosition) {
    offsetInWindow = newSize - size;
} else {
    _error_code = IllegalData;
    _error_msg = tr("requested target position"
                   " is not a TargetPosition "
                   " enum member");
}
```

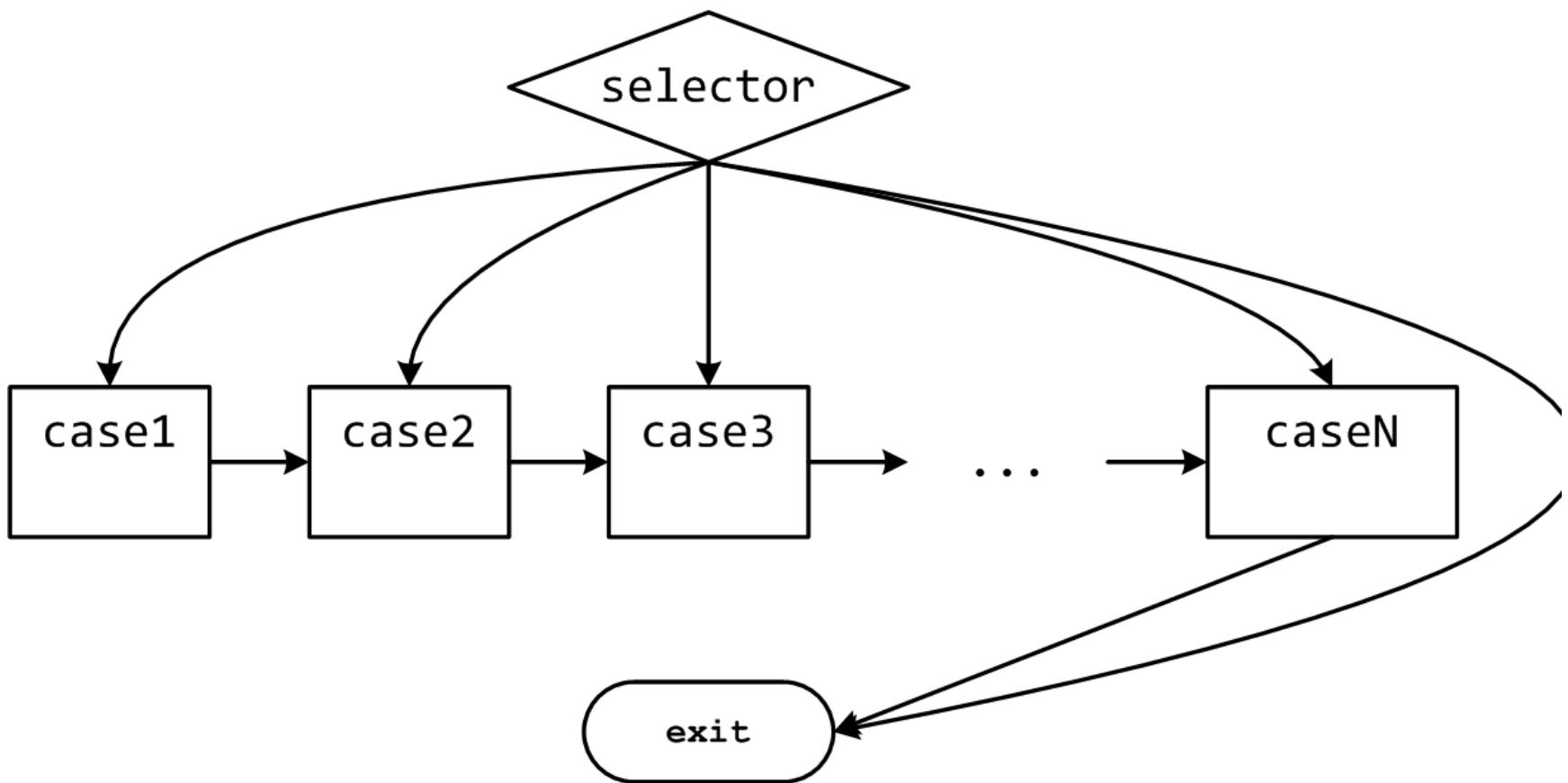
```
; в edx помещено значение управляющего выражения  
; т.е. targetPosition
```

```
cmp  edx, TARGET_AT_BEGINNING  
jne .comp2  
; код для case TARGET_AT_BEGINNING:  
jmp .switch_exit
```

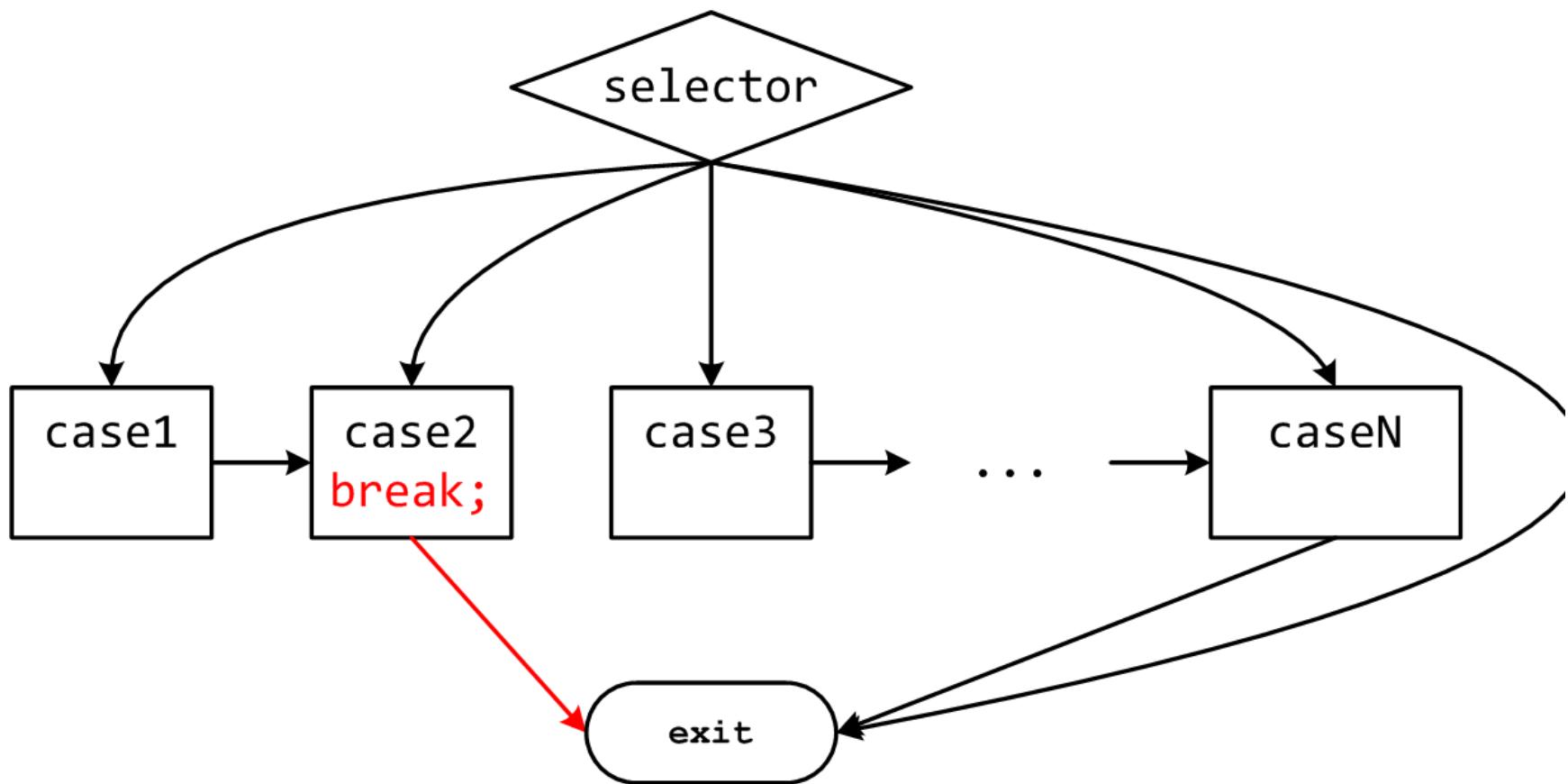
```
.comp2:  
cmp  edx, TARGET_AT_MIDDLE  
jne .comp3  
; код для case TARGET_AT_MIDDLE:  
jmp .switch_exit
```

```
.comp3:  
cmp  edx, TARGET_AT_END  
jne .default  
; код для case TARGET_AT_END:  
jmp .switch_exit
```

```
.default:  
; код для default:  
.switch_exit:
```



Вспоминаем пример из курса «АиАЯ»:  
подсчет количества дней, прошедших с первого января.



# Duff's Device

```
void duffs_device(char *to, char *from, int count) {  
  
    register n = (count + 7) / 8; /* count > 0 assumed */  
  
    switch (count % 8) {  
        case 0: do { *to = *from++;  
        case 7:           *to = *from++;  
        case 6:           *to = *from++;  
        case 5:           *to = *from++;  
        case 4:           *to = *from++;  
        case 3:           *to = *from++;  
        case 2:           *to = *from++;  
        case 1:           *to = *from++;  
        } while (--n > 0);  
    }  
}
```

```
long switch_eg
(long x, long y, long z)
{
    long w = 1;
    switch (x) {
        case 1:
            w = y*z;
            break;
        case 2:
            w = y/z;
            /* «проваливаемся» */
        case 3:
            w += z;
            break;
        case 5:
        case 6:
            w -= z;
            break;
        default:
            w = 2;
    }
    return w;
}
```

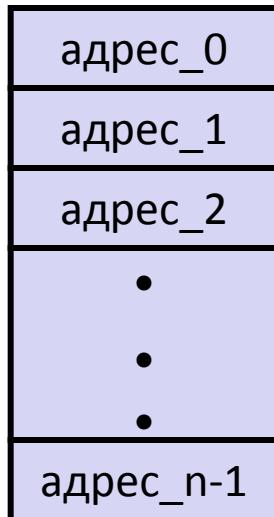
- Допустимо использовать несколько меток для одного блока
  - cases 5 и 6
- В отсутствии break управление «проводится» в следующий блок кода
  - case 2
- Некоторые значения могут быть пропущены
  - case 4

## Исходный оператор switch

```
switch (x) {
    case val_0:
        Блок 0
    case val_1:
        Блок 1
    • • •
    case val_n-1:
        Блок n-1
}
```

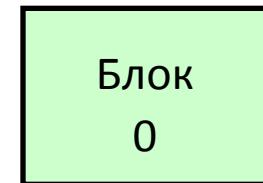
## Таблица переходов

JTab:



## Размещение кода

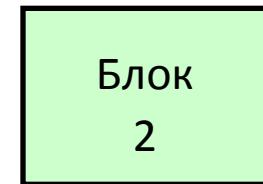
адрес\_0:



адрес\_1:



адрес\_2:



•

•

•

адрес\_n-1:



## Упрощенное отображение

```
target = JTab[x];
goto *target;
```

```
long switch_eg(long x, long y, long z) {
    long w = 1;
    switch (x) {
        . . .
    }
    return w;
}
```

**switch\_eg:**

push	ebp	;	Инициализацию переменной <b>w</b> отложили на потом ...
mov	ebp, esp	;	
mov	edx, dword [ebp + 8]	;	edx = x
mov	eax, dword [ebp + 12]	;	eax = y
mov	ecx, dword [ebp + 16]	;	ecx = z
dec	edx		
cmp	edx, 5	;	сравниваем (x-1) и 5
ja	.L8	;	если > <sub>u</sub> goto default
jmp	[.L4 + 4*edx]	;	goto *JTab[x]

```
long switch_eg(long x, long y, long z) {  
    long w = 1;  
    switch (x) {  
        . . .  
    }  
    return w;  
}
```

1. Вычисление индекса в таблице переходов

## Таблица переходов

```
section .rodata align=4
.L4:
    dd .L3 ; x = 1
    dd .L5 ; x = 2
    dd .L9 ; x = 3
    dd .L8 ; x = 4
    dd .L7 ; x = 5
    dd .L7 ; x = 6
```

1. Вычисление индекса в таблице переходов
2. Переход по адресу, взятым из таблицы

```
switch_eg:  
    push    ebp  
    mov     ebp, esp  
  
    mov     edx, dword [ebp + 8]      ; edx = x  
    mov     eax, dword [ebp + 12]      ; eax = y  
    mov     ecx, dword [ebp + 16]      ; ecx = z  
  
    dec     edx  
    cmp     edx, 5  
    ja     .L8  
    jmp    [.L4 + 4*edx]  
    .L8 :  
        ; взято из таблицы  
        dd .L8 ; x = 4  
        dd .L7 ; x = 5  
        dd .L7 ; x = 6  
  
    Косвенный  
    переход  
    ; сравниваем (x-1) и 5  
    ; если > goto default  
    ; goto *JTab[x]
```

# Косвенный переход

- Организация таблицы переходов

- Каждый элемент занимает 4 байта
- Базовый адрес - .L4

- Переходы

- Прямые: `jmp .L2`
- Для обозначения цели перехода используется метка `.L2`
- Косвенные: `jmp [.L4 + 4*edx]`
- Начало таблицы переходов `.L4`
- Коэффициент масштабирования должен быть 4  
(в IA-32 метка содержит 32 бита = 4 байта)
- Выбираем цель перехода через исполнительный адрес `.L4 + edx*4`
  - Только для  $x: 0 \leq x-1 \leq 5$

Таблица переходов

```
section .rodata align=4
.L4:
    dd .L3 ; x = 1
    dd .L5 ; x = 2
    dd .L9 ; x = 3
    dd .L8 ; x = 4
    dd .L7 ; x = 5
    dd .L7 ; x = 6
```

## Таблица переходов

```
section .rodata align=4
.L4:
    dd .L3 ; x = 1
    dd .L5 ; x = 2
    dd .L9 ; x = 3
    dd .L8 ; x = 4
    dd .L7 ; x = 5
    dd .L7 ; x = 6
```

```
switch(x) {
    case 1:          // .L3
        w = y*z;
        break;
    case 2:          // .L5
        w = y/z;
        /* «проваливаемся» */
    case 3:          // .L9
        w += z;
        break;
    case 5:
    case 6:          // .L7
        w -= z;
        break;
    default:         // .L8
        w = 2;
}
```

```
long w = 1;  
. . .  
switch(x) {  
    . . .  
    case 2:  
        w = y/z;  
        /* «проваливаемся» */  
    case 3:  
        w += z;  
        break;  
    . . .  
}
```

```
case 2:  
    w = y/z;  
    goto merge;
```

```
case 3:  
    w = 1;  
merge:  
    w += z;
```

# Начало оператора switch

```
switch(x) {  
    case 1:          // .L3  
        w = y*z;  
        break;  
  
    . . .  
  
    case 5:  
    case 6:          // .L7  
        w -= z;  
        break;  
    default:         // .L8  
        w = 2;  
}
```

```
.L3:           ; x == 1  
    imul eax, ecx ; w = y*z;  
    jmp  .L2       ; goto switch_end
```

# Продолжение оператора switch

```
switch(x) {  
    . . .  
    case 2:    // .L5  
        w = y/z;  
    /* «проваливаемся»  
     на точку слияния  
     .L6 */  
    case 3:    // .L9  
        w += z;  
        break;  
    . . .  
}
```

```
.L5:          ; x == 2  
    cdq          ;  
    idiv ecx    ; w = y/z;  
    jmp .L6      ; goto merge (.L6)  
.L9:          ; x == 3  
    mov eax, 1   ; w = 1;  
.L6:          ; точка слияния  
    add eax, ecx ; w += z;  
    jmp .L2      ; goto switch_end
```

# Окончание оператора switch

```
switch(x) {  
    case 1:          // .L3  
        w = y*z;  
        break;  
  
    . . .  
  
    case 5:  
    case 6:          // .L7  
        w -= z;  
        break;  
    default:         // .L8  
        w = 2;  
}
```

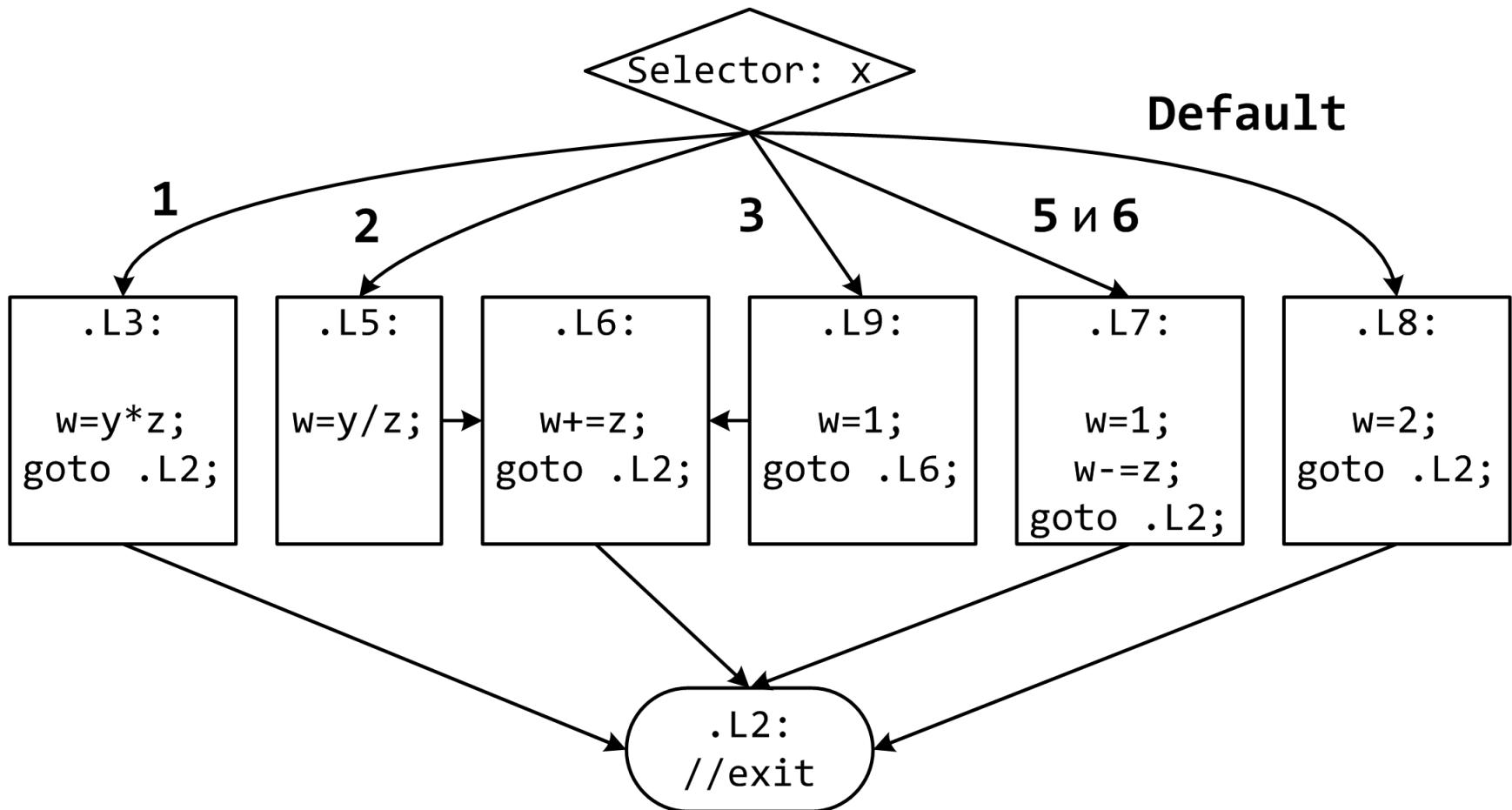
```
.L7:                 ; x == 5 или x == 6  
    mov    eax, 1      ; w = 1;  
    sub    eax, ecx    ; w -= z;  
    jmp    .L2         ; goto switch_end  
.L8:                 ; default  
    mov    eax, 2      ; w = 2  
.L2:                 ; выход из switch
```

# Завершение функции

```
return w;
```

```
pop ebp  
ret      ; возвращаемое  
          ; значение уже  
          ; размещено в ЕАХ
```

- Преимущества таблицы переходов
  - Применение таблицы переходов позволяет избежать последовательного перебора значений меток
    - Фиксированное время работы
  - Позволяет учитывать «дыры» и повторяющиеся метки
  - Код располагается упорядоченно, удобно обрабатывать «пропуски»
  - Инициализация  $w = 1$  не проводилась до тех пор пока не потребовалась
- В качестве меток используем значения типа enum



```
int f(int n, int *p) {  
    int _res;  
    switch (n) {  
    default:  
        _res = 0;  
        /* «проваливаемся» */  
    case 1:  
        *p = _res;  
        break;  
    case 64:  
        _res = 1;  
        break;  
    case 63:  
        _res = 2;  
        *p = _res;  
        /* «проваливаемся» */  
    case 256:  
        _res = 3;  
        break;  
    case 65536:  
        _res = 4;  
    }  
    return _res;  
}
```

- Случай `default` расположен первым
- Управление «проводится» в случаях `default` и 63
- Таблица переходов получается неприемлемо большой

```
...
push ebp
mov ebp, esp
mov eax, 1
mov edx, dword [ebp+8] ; n
cmp edx, 64
je .L9
jle .L13
...
```

