

Лекция 8

4 марта

Обратная задача

f:

```

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

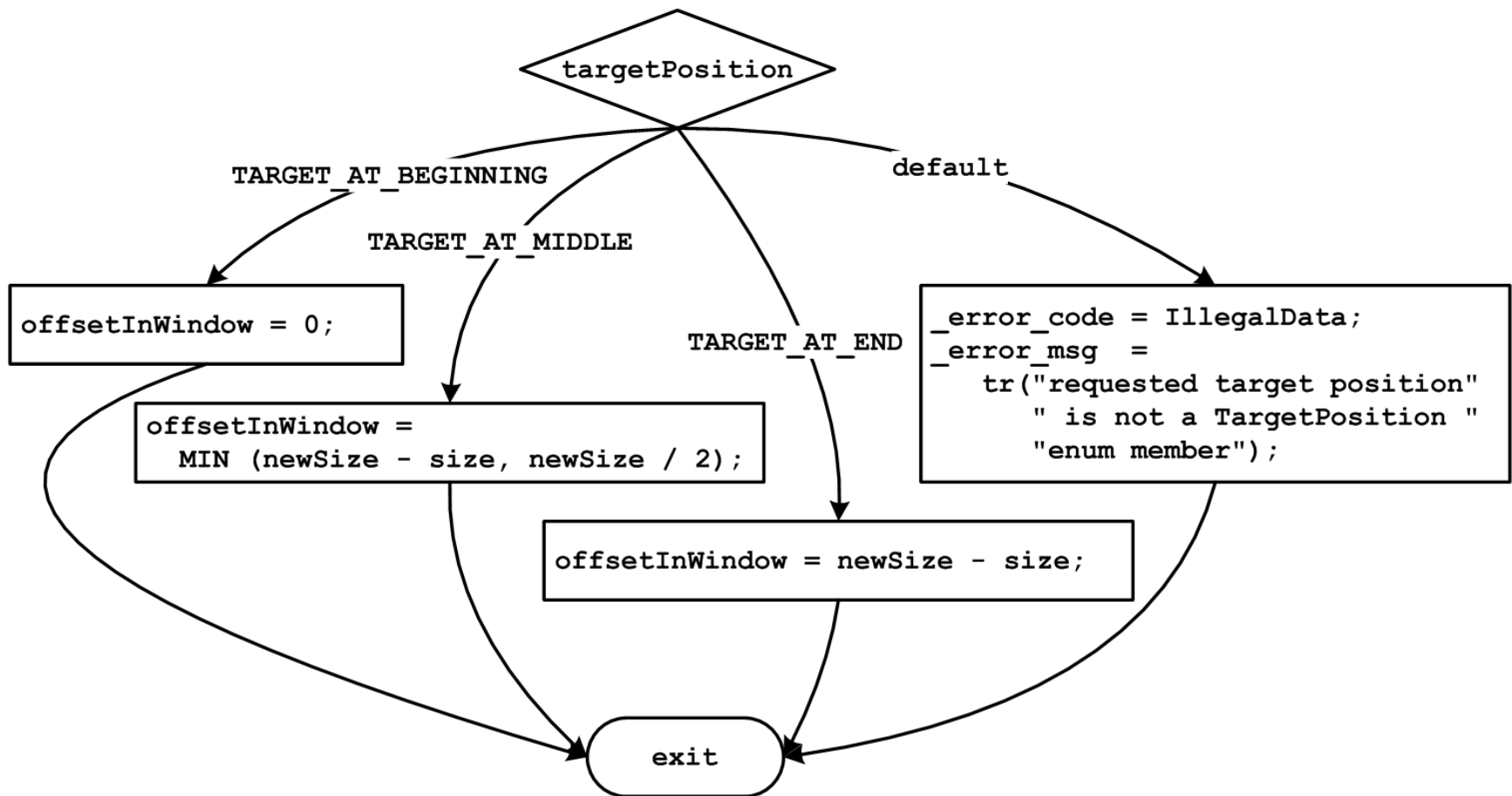
```

```

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

```

```
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:

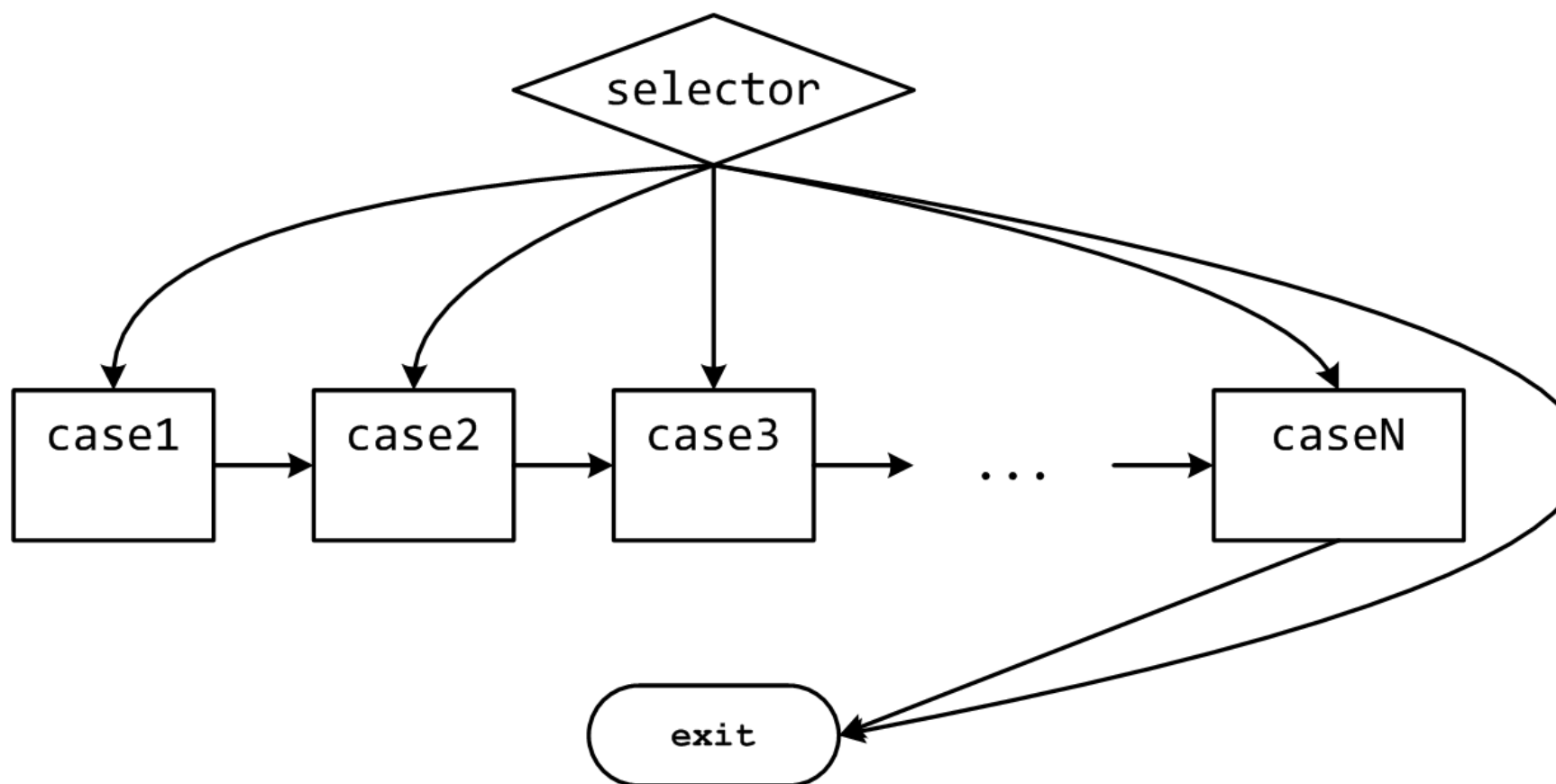
```
cmp  edx, TARGET_AT_END  
jne  .default
```

; код для case TARGET_AT_END:
jmp .switch_exit

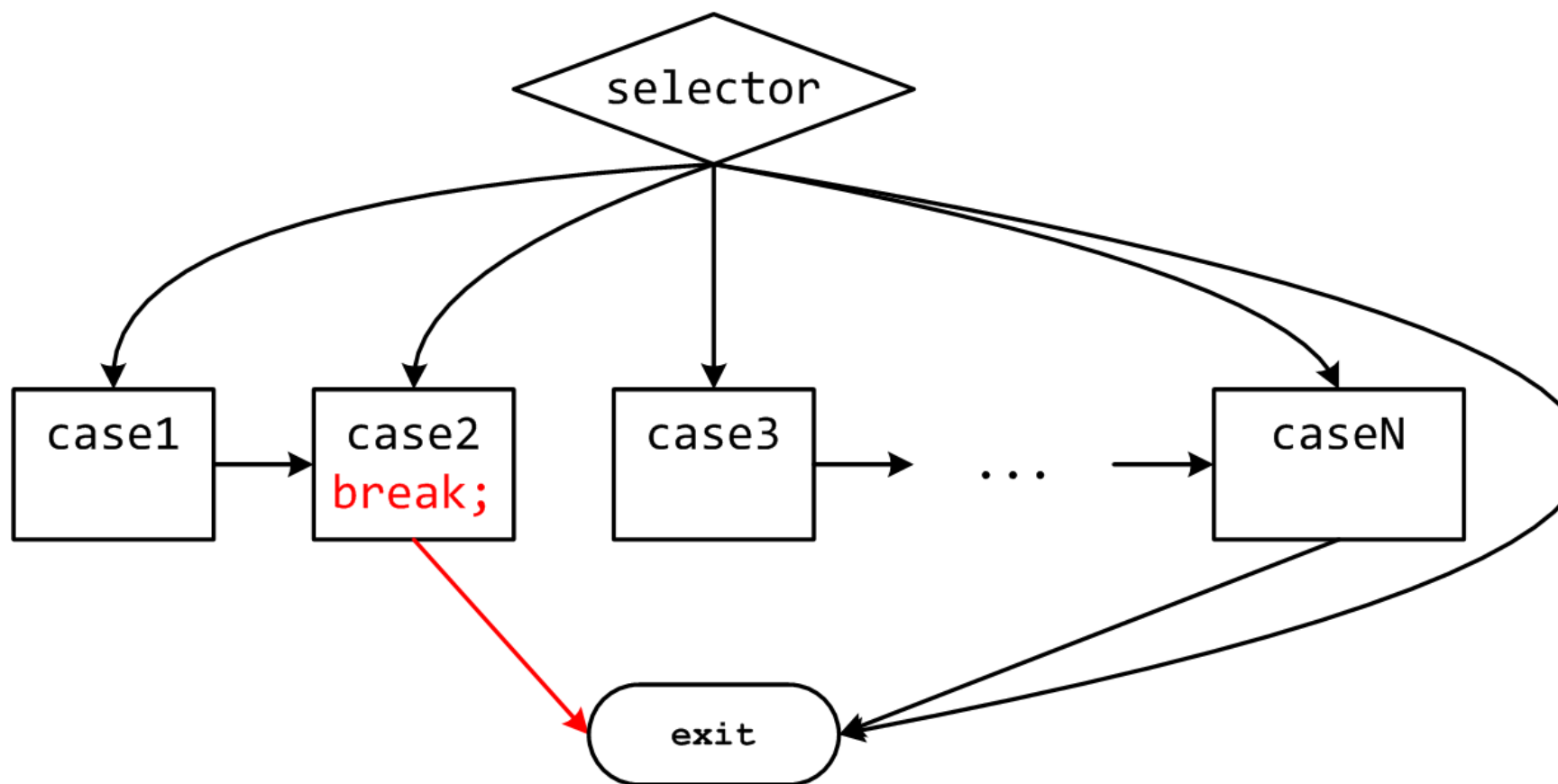
.default:

; 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 «проваливается» 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:

адрес_0
адрес_1
адрес_2
•
•
•
адрес_n-1

Размещение кода

адрес_0:

Блок
0

адрес_1:

Блок
1

адрес_2:

Блок
2

•

•

•

адрес_n-1:

Блок
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                ;
    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 >u goto default
    jmp     [.L4 + 4*edx]       ; goto *JTab[x]

```

Инициализацию переменной **w**
отложили на потом ...

```

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

```

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, взятому 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              ; сравниваем (x-1) и 5
ja      .L8                 ; если >_u goto default
jmp     [.L4 + 4*edx]        ; goto *JTab[x]

```

Косвенный
переход



- Организация таблицы переходов

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

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
```

- 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
```

```
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;  
}
```

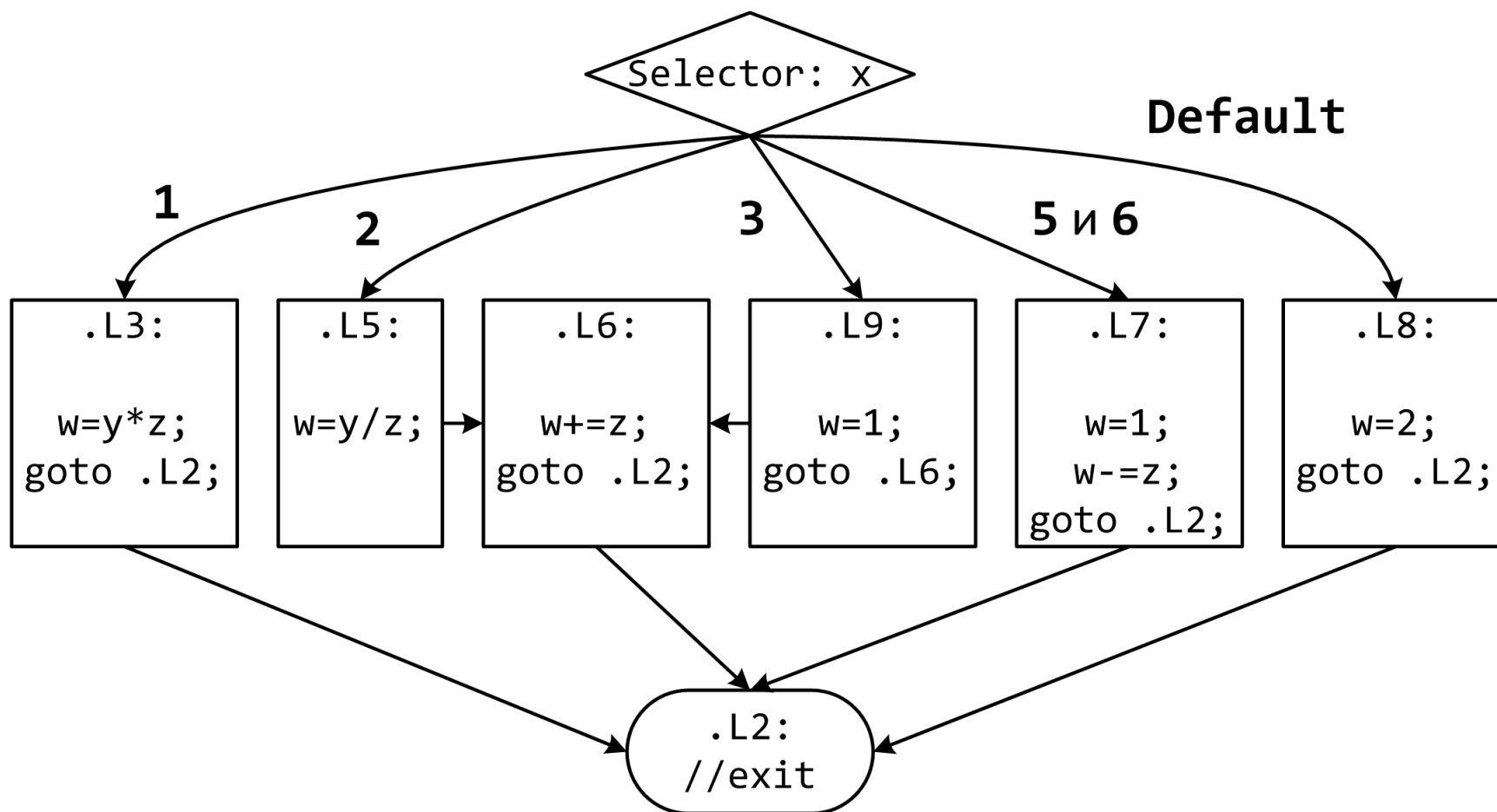
```
.L7:                ; x == 5 или x == 6  
    mov    eax, 1    ; w = 1;  
    sub    eax, ecx   ; w -= z;  
    jmp    .L2        ; goto switch_end  
.L8:                ; default  
    mov    eax, 2    ; w = 2  
.L2:                ; ВЫХОД ИЗ switch
```

Завершение функции

```
return w;
```

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

- 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`



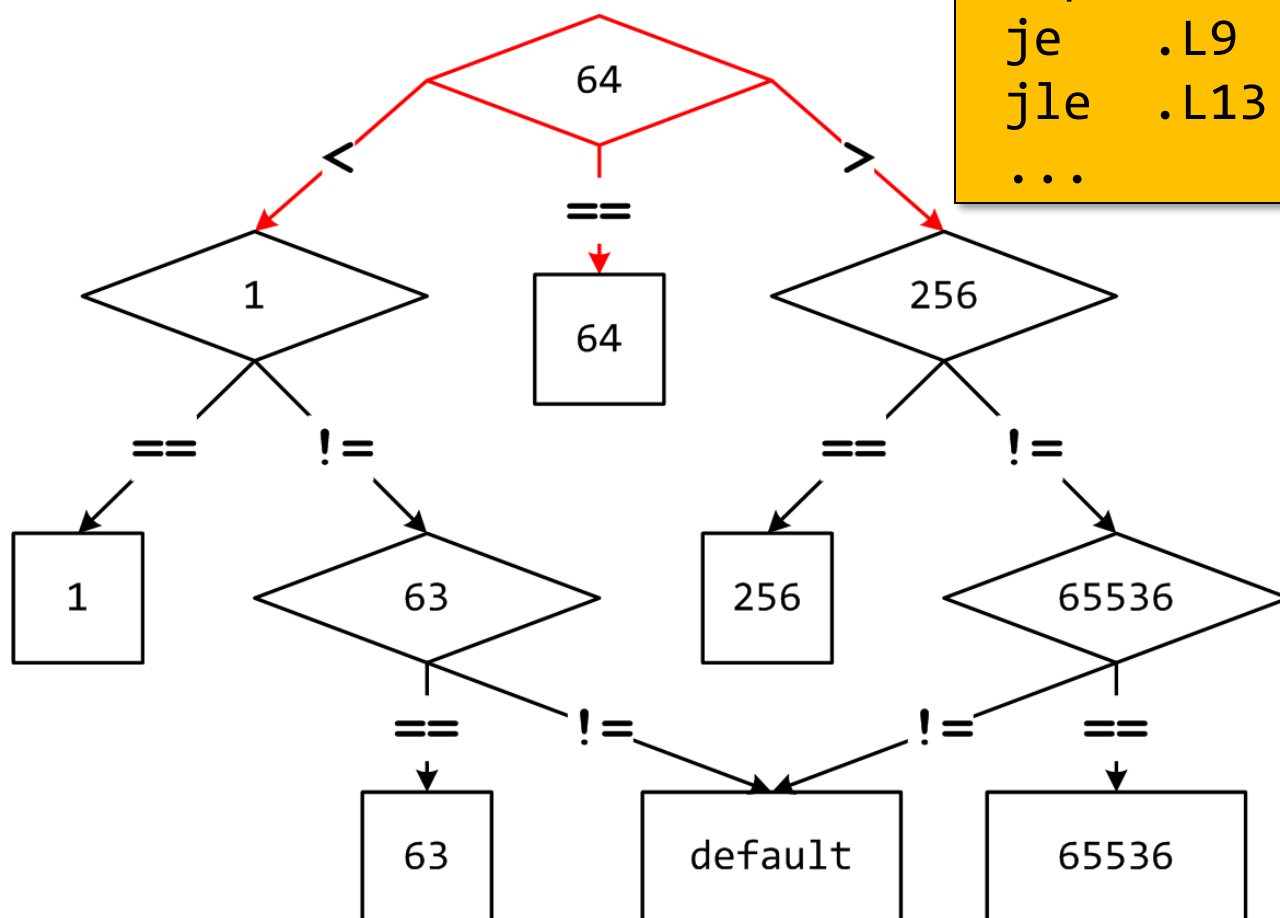
```
int f(int n, int *p) {  
    int _res;  
    switch (n) {  
    default:  
        _res = 0;  
        /* «проваливаемся» */  
    case 1:  
        *p = _res;  
        break;  
    case 64:  
        _res = 1;  
        break;  
    case 63:  
        _res = 2;  
        *p = _res;  
        /*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

- Случай default расположен первым
- Управление «проваливается» 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
...

```



Обратная задача

```
int switchMeOnce(int x) {  
    int result = 0;  
  
    switch (x) {  
        . . .  
    }  
  
    return result;  
}
```

```
section .text  
    . . .  
    mov eax, dword [ebp-8]  
    add eax, 2  
    cmp eax, 6  
    ja .L2  
    jmp [.L8 + 4*eax]  
    . . .
```

```
section .rodata  
    .L8 dd .L3, .L2, .L4, .L5, .L6, .L6, .L7
```

1. Сколько раз было использовано ключевое слово case?
2. Какие константы использовались?
3. Какие ветки выполнения были объединены?
4. Что помечено .L2?

Промежуточные итоги: передача управления

- Язык Си
 - if, if-else
 - do-while
 - while, for
 - switch
- Язык ассемблера
 - Условная передача управления
 - Условная передача данных
 - Косвенные переходы
- Стандартные приемы
 - Преобразования циклов к виду do-while
 - Использование таблицы переходов для операторов switch
 - Операторы switch с «разреженным» набором значений меток реализуются деревом решений
- Следующая тема: указатели и агрегатные типы данных

Типы данных языка Си

- char
- Стандартные знаковые целочисленные типы
 - signed char
 - short int
 - int
 - long int
 - long long int
- Стандартные беззнаковые целочисленные типы
 - _Bool
- Перечисление
- Типы чисел с плавающей точкой
 - float
 - double
 - long double
 - _Complex
- Производные типы
 - **Массивы**
 - **Структуры**
 - **Объединения**
 - **Указатели**
 - Указатели на функции

Регистры и типы данных

• Целые числа

- Размещаются и обрабатываются в регистрах общего назначения
- Знаковые/беззнаковые числа

• Intel	Асм.	байты	Си
• byte	b	1	[unsigned] char
• word	w	2	[unsigned] short
• double word	d	4	[unsigned] int
• quad word	q	8	[unsigned] long long int

• Указатели

• Числа с плавающей точкой

- Размещаются и обрабатываются в специализированных регистрах для чисел с плавающей точкой

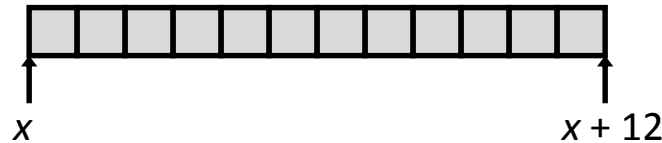
• Intel	Асм.	байты	Си
• Single	d	4	float
• Double	q	8	double

Массивы – размещение в памяти

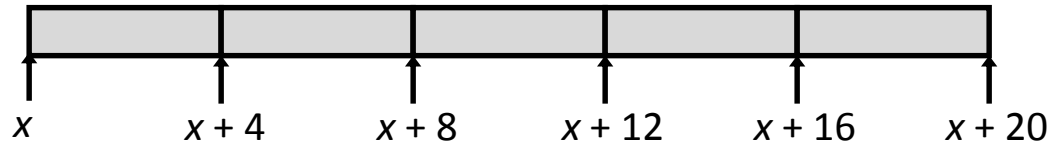
T A[L];

- Массив элементов типа T, размер массива – L
- Массив располагается в непрерывном блоке памяти размером $L * \text{sizeof}(T)$ байт

`char string[12];`



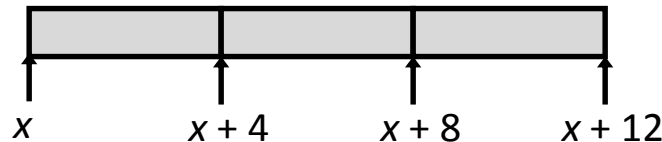
`int val[5];`



`double a[3];`



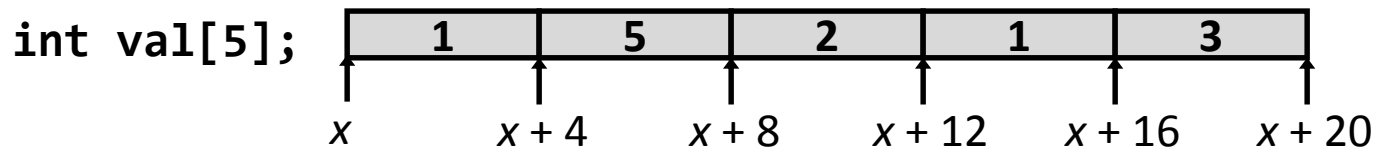
`char *p[3];`



Доступ к элементам массива

$T \ A[L];$

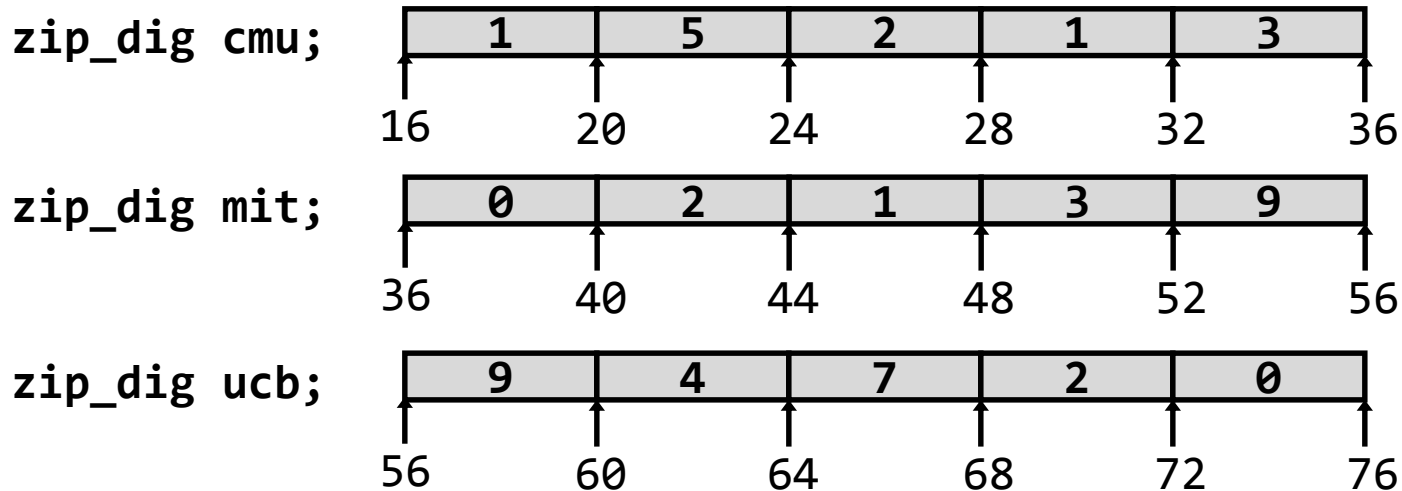
- Массив элементов типа T , размер массива – L
- Идентификатор A может использоваться как указатель на элемент массива с индексом 0 . Тип указателя – T^*



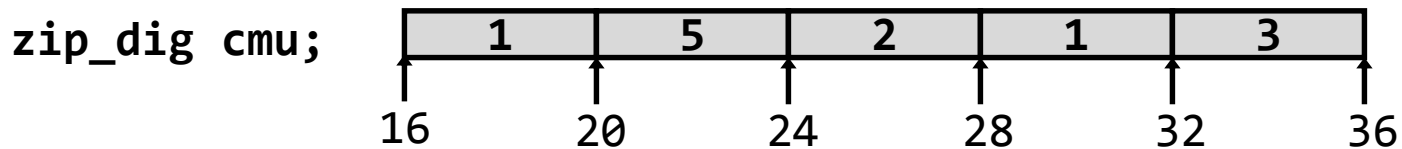
- Задачи ...

```
#define ZLEN 5
typedef int zip_dig[ZLEN];

zip_dig cmu = { 1, 5, 2, 1, 3 };
zip_dig mit = { 0, 2, 1, 3, 9 };
zip_dig ucb = { 9, 4, 7, 2, 0 };
```



- Объявление переменной “**zip_dig cmu**” эквивалентно “**int cmu[5]**”
- Массивы были размещены в последовательно идущих блоках памяти размером 20 байт каждый
 - В общем случае не гарантируется, что массивы будут размещены непрерывно



```
int get_digit (zip_dig z, int dig) {
    return z[dig];
}
```

```
; edx = z
; eax = dig
mov eax, dword [edx+4*eax] ; z[dig]
```

- Регистр `edx` содержит начальный (базовый) адрес массива
- Регистр `eax` содержит индекс элемента в массиве
- Адрес элемента $edx + 4 * eax$

```
void zincr(zip_dig z) {
    int i;
    for (i = 0; i < ZLEN; i++)
        z[i]++;
}
```



```
void zincr_v(zip_dig z) {
    void *vz = z;
    int i = 0;
    do {
        (*((int *) (vz + i))) += 1;
        i += ISIZE;
    } while (i != ISIZE * ZLEN);
}
```

```

                                ;   edx = z
                                ;   eax = i
    mov eax, 0                    ; loop:
.L4:                             ;   z[i]++
    add dword [edx + 4 * eax], 1 ;   i++
    add eax, 1                    ;   i vs. 5
    cmp eax, 5                    ;   if (!=) goto loop
    jne .L4
```