Лекция 8

3 марта

```
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
```

Передача управления

Си

- `if`
- `if-else`
- **`switch`**
- `do-while`
- `while`
- `for`
- **`goto`**
- `break`
- `contunue`
- `return`

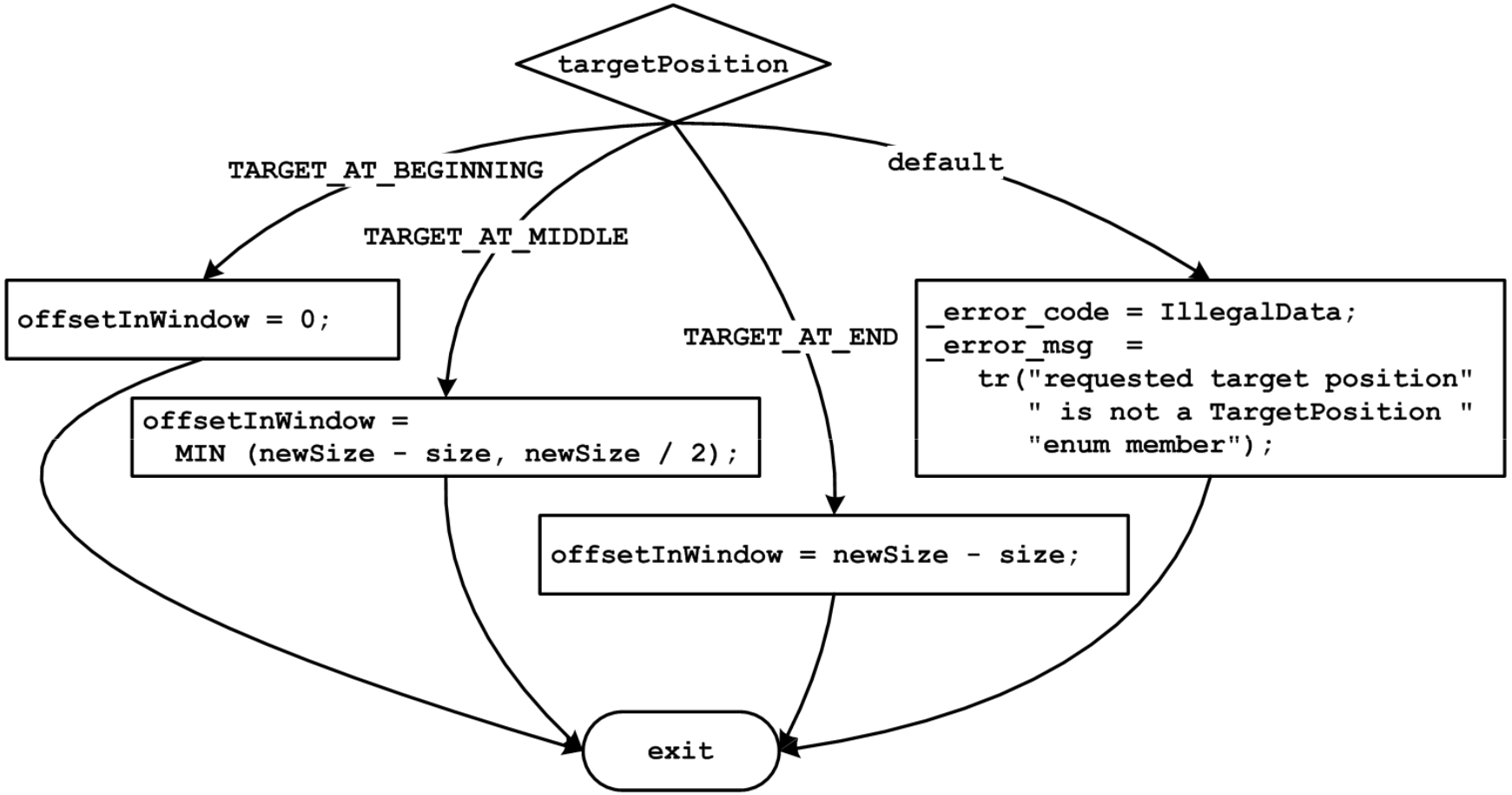
Ассемблер

- JMP
- J*cc*

- CALL
- RET

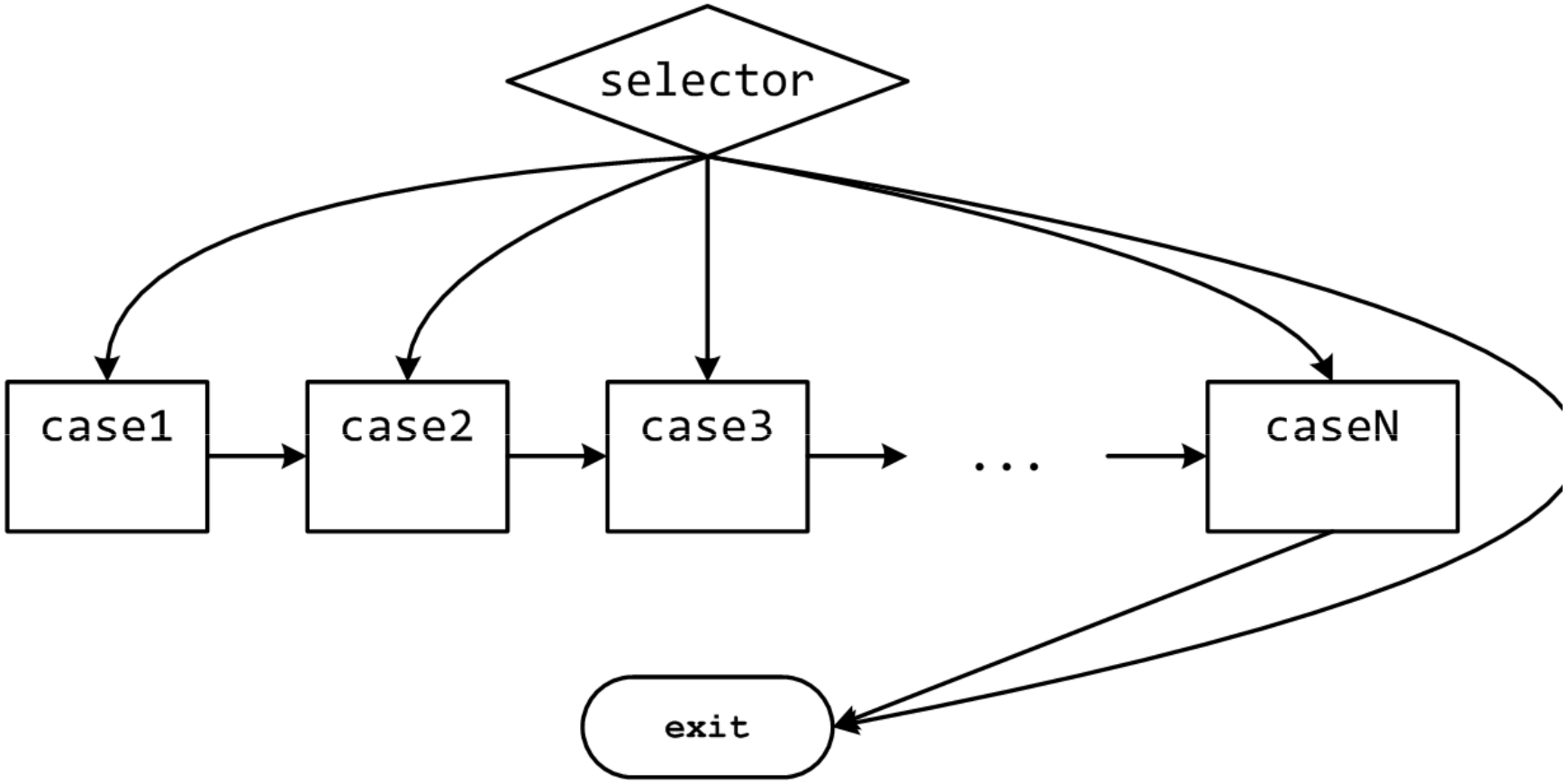
- CMOV*cc*

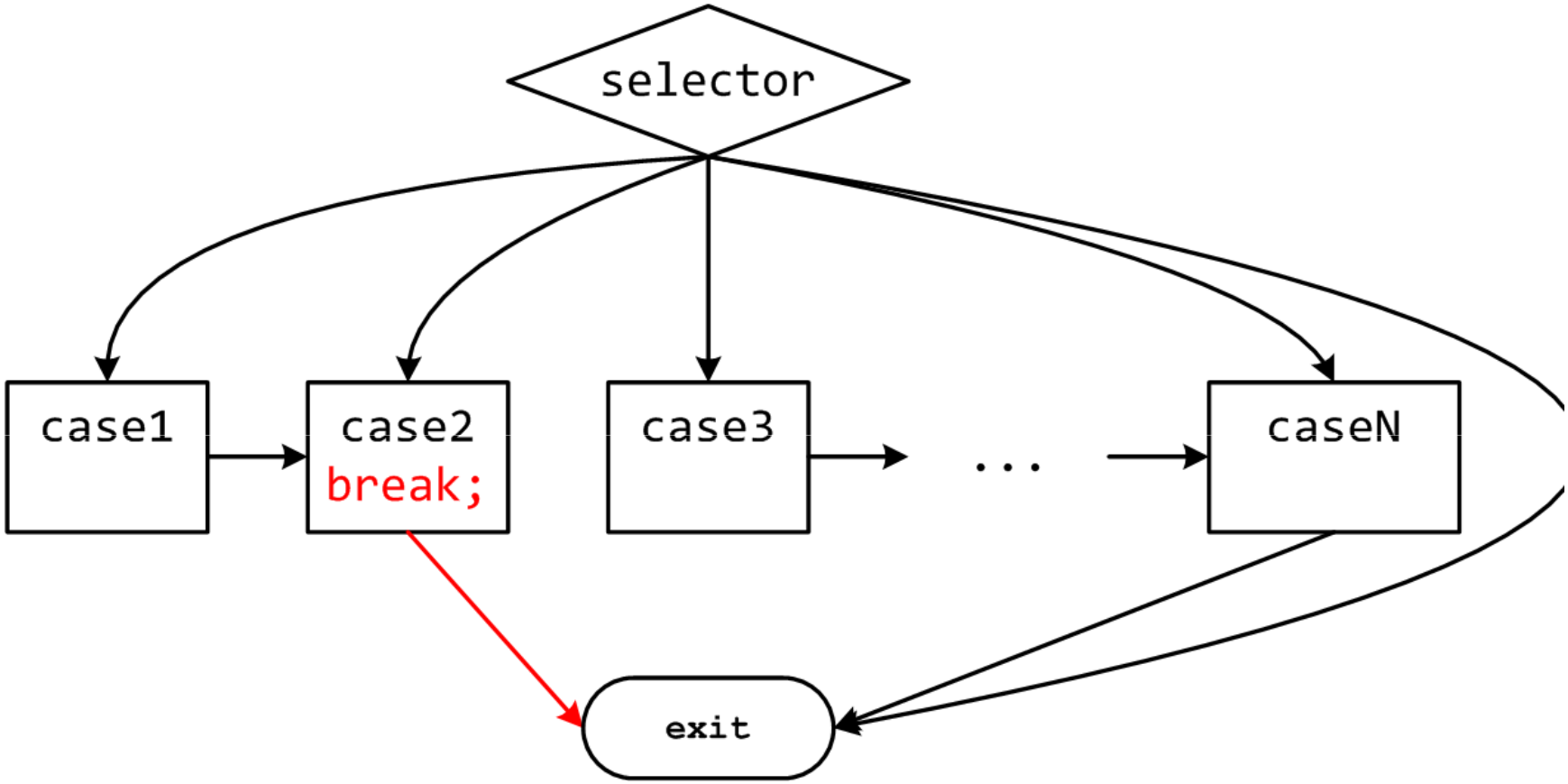
```
enum TargetPosition {
   TARGET_AT_BEGINNING,
   TARGET_AT_MIDDLE,
   TARGET_AT_END
};

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:
```





```
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

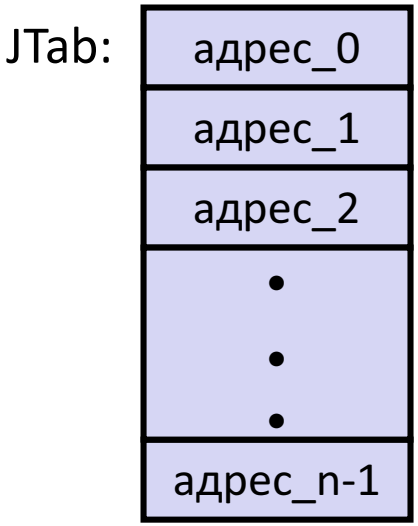
Исходный оператор switch

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

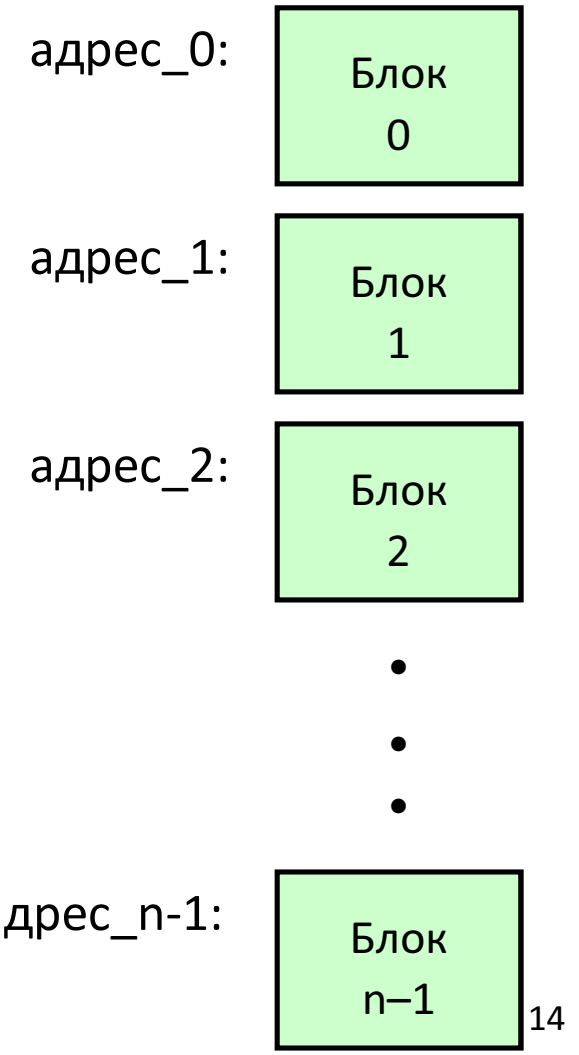
Упрощенное отображение

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

Таблица переходов



Размещение кода



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

Переменная **w** еще не была инициализирована

```
switch_eg:
   push    ebp                          ;
   mov     ebp, esp                     ;
   mov     eax, dword [ebp + 8]         ; eax = x
   cmp     eax, 6                       ; сравниваем x и 6
   ja      .L2                          ; если >u goto default
   jmp     [.L7 + 4*eax]                ; goto *JTab[x]
```

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

Таблица переходов

```
section     .rodata align=4
.L7:
  dd        .L2; x = 0
  dd        .L3; x = 1
  dd        .L4; x = 2
  dd        .L5; x = 3
  dd        .L2; x = 4
  dd        .L6; x = 5
  dd        .L6; x = 6
```

Вычисление индекса в таблице переходов

```
switch_eg:
   push    ebp                          ;
   mov     ebp, esp                     ;
   mov     eax, dword [ebp + 8]         ; eax = x
   cmp     eax, 6                       ; сравниваем x и 6
   ja      .L2                          ; если >u goto default
   jmp     [.L7 + 4*eax]                ; goto *JTab[x]
```

Косвенный переход

- Организация таблицы переходов
  - Каждый элемент занимает 4 байта
  - Базовый адрес - .L7

Таблица переходов

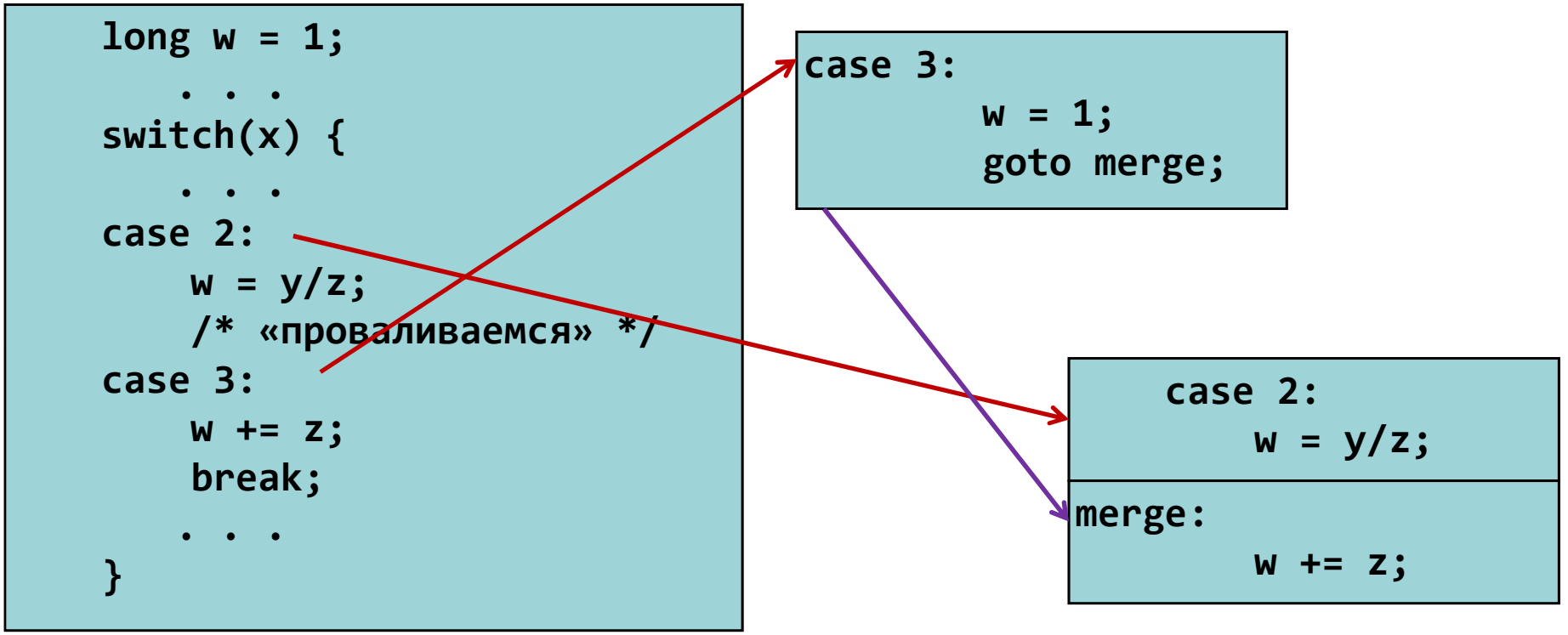
```
section     .rodata align=4
.L7:
  dd        .L2; x = 0
  dd        .L3; x = 1
  dd        .L4; x = 2
  dd        .L5; x = 3
  dd        .L2; x = 4
  dd        .L6; x = 5
  dd        .L6; x = 6
```

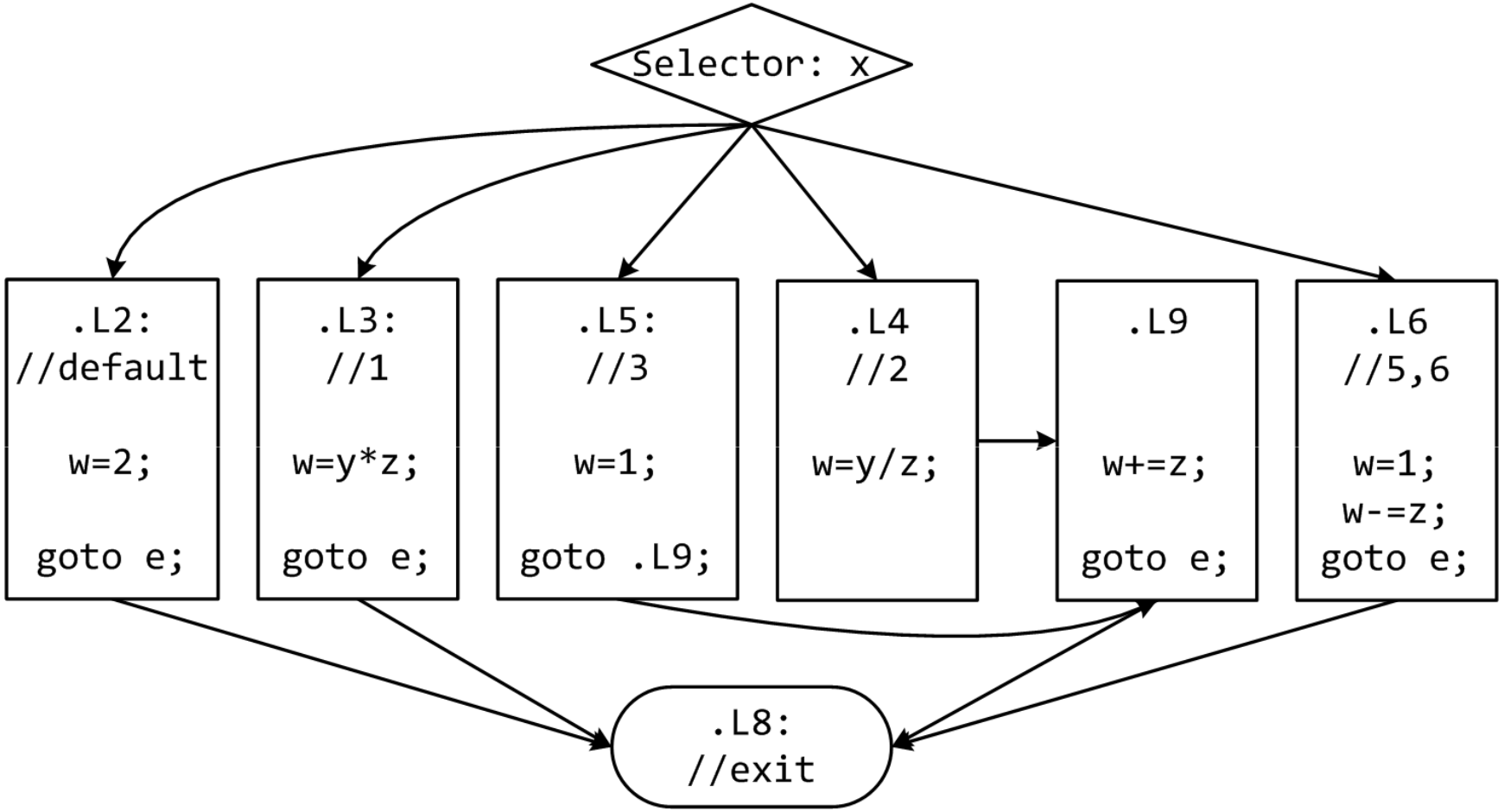
- Переходы
  - Прямые: **jmp .L2**
  - Для обозначения цели перехода используется метка .L2
  - Косвенные: **jmp [.L7 + 4*eax]**
  - Начало таблицы переходов .L7
  - Коэффициент масштабирования должен быть 4 (в IA32 метка содержит 32 бита = 4 байта)
  - Выбираем цель перехода через исполнительный адрес .L7 + eax*4
    - Только для x: 0 ≤ x ≤ 6

Таблица переходов

```
section     .rodata align=4
.L7:
  dd        .L2 ; x = 0
  dd        .L3 ; x = 1
  dd        .L4 ; x = 2
  dd        .L5 ; x = 3
  dd        .L2 ; x = 4
  dd        .L6 ; x = 5
  dd        .L6 ; x = 6
```

```
    switch(x) {
    case 1:       // .L3
        w = y*z;
        break;
    case 2:       // .L4
        w = y/z;
        /* «проваливаемся» */
    case 3:       // .L5
        w += z;
        break;
    case 5:
    case 6:       // .L6
        w -= z;
        break;
    default:      // .L2
        w = 2;
    }
```





Начало оператора switch

```
switch(x) {
case 1:          // .L3
      w = y*z;
      break;
 . . .
case 3:          // .L5
    w += z;
    break;
  . . .
default:         // .L2
    w = 2;
}
```

```
.L2:                    ; default
   mov   eax, 2         ; w = 2
   jmp   .L8            ; goto done

.L5:                    ; x == 3
   mov   eax, 1         ; w = 1
   jmp   .L9            ; goto merge

.L3:                    ; x == 1
   mov   eax, dword [ebp + 16]
                        ; z
   imul  eax, dword [ebp + 12]
                        ; w = y*z
   jmp   .L8            ; goto done
```

…продолжение…

```
switch(x) {
  . . .
  case 2:  // .L4
      w = y/z;
  /* «проваливаемся» */
  merge:    // .L9
      w += z;
      break;
  case 5:
  case 6:  // .L6
      w -= z;
      break;
}
```

```
.L4:                          ; x == 2
   mov   edx, dword [ebp + 12]
   mov   eax, edx
   sar   edx, 31
   idiv  dword [ebp + 16]; w = y/z

.L9:                          ; merge:
   add   eax, dword [ebp + 16]
                              ; w += z
   jmp   .L8                  ; goto done

.L6:                          ; x == 5, 6
   mov   eax, 1               ; w = 1
   sub   eax, dword [ebp + 16]
                              ; w = 1-z
```

...Окончание

```
return w;
```

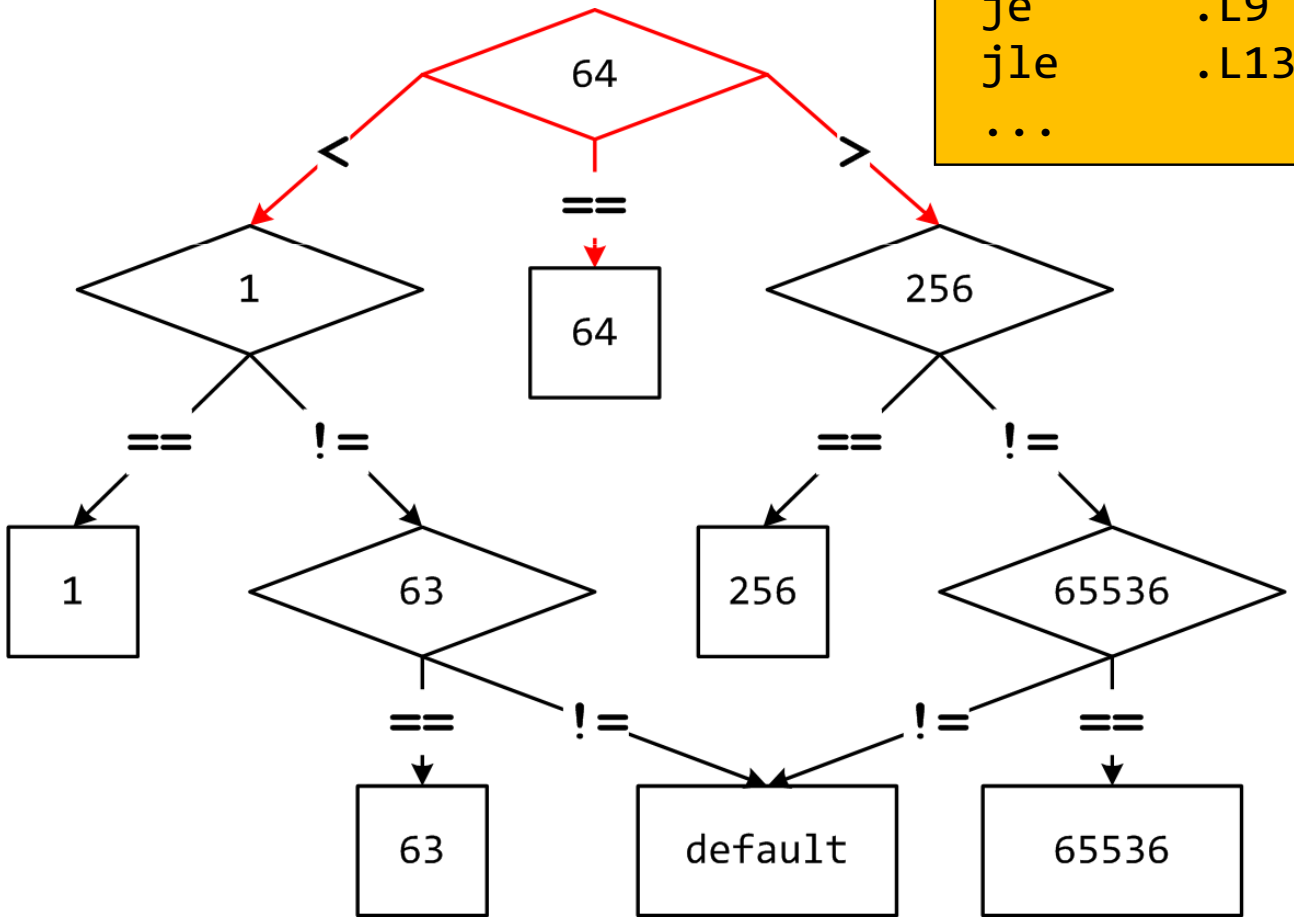
```
.L8:                    ; done:
   pop   ebp
   ret
```

- 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 элементы 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
...
```



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