

Машинный уровень 2: Управление

Основы информатики

Компьютерные основы программирования

goo.gl/X7evF

На основе CMU 15-213/18-243:
Introduction to Computer Systems

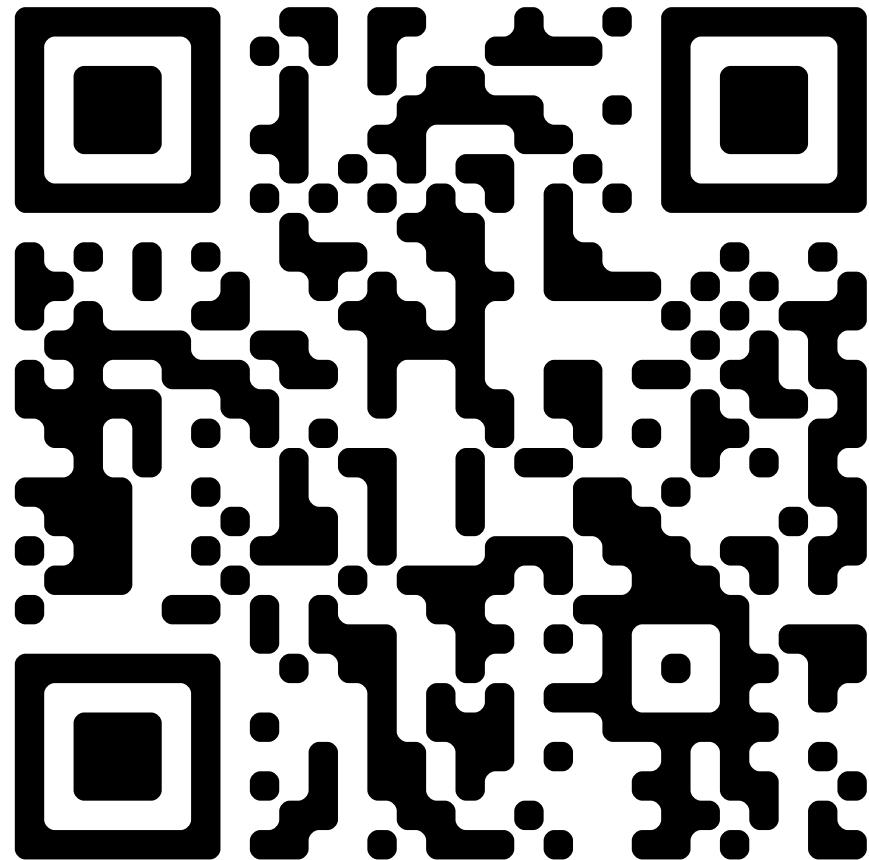
goo.gl/Q7vgWw

Лекция 5, 05 марта, 2017

Лектор:

Дмитрий Северов, кафедра информатики 608 КПМ

dseverov@mail.mipt.ru



w27001.vdi.mipt.ru/wp/?page_id=346

Машинный уровень 2: Управление

- Управление: флаги условий
- Условные переходы и пересылки
- Циклы
- Операторы переключения
- Процедуры IA 32
 - Структура стека
 - Соглашения вызова процедур
 - Рекурсия и указатели

Состояние процессора (x86-64, частично)

■ Информация о непосредственно исполняемой программе

- Промежуточные данные (`%rax`, ...)
- Адрес вершины стека (`%rsp`)
- Адрес текущей команды (`%rip`, ...)
- Результаты последних проверок (`CF`, `ZF`, `SF`, `OF`)

Текущая вершина стека

Регистры

<code>%rax</code>	<code>%r8</code>
<code>%rbx</code>	<code>%r9</code>
<code>%rcx</code>	<code>%r10</code>
<code>%rdx</code>	<code>%r11</code>
<code>%rsi</code>	<code>%r12</code>
<code>%rdi</code>	<code>%r13</code>
<code>%rsp</code>	<code>%r14</code>
<code>%rbp</code>	<code>%r15</code>

`%rip`

Указатель команды

CF

ZF

SF

OF

Флаги условий

Флаги условий (неявная установка)

■ Однобитные регистры- флаги

- Устанавливаются арифметическими операциями неявно (как побочный результат)

Пример: $\text{addq } Src, Dest \leftrightarrow t = a+b$

CF=1, если перенос в старший бит или заём из него (беззнаковое переполнение)

ZF=1, если t == 0

SF=1, если $t < 0$ (как знаковое), SF == MSB

OF=1, если переполнился дополнительный код (знаковый)
($a>0 \ \&\& \ b>0 \ \&\& \ t<0$) || ($a<0 \ \&\& \ b<0 \ \&\& \ t\geq 0$)

■ Не устанавливаются командой `lea`!

Флаги условий (явная установка сравнением)

■ Явная установка командами сравнения

- `cmpq Src2, Src1`
- `cmpq b, a` как вычисление $a - b$ без сохранения результата
- **CF=1, если перенос из старшего бита или заём в него**
(используется при беззнаковых сравнениях)
- **ZF=1, если $a == b$**
- **SF=1, если $(a-b) < 0$ (как знаковые)**
- **OF=1, переполнение в дополнительном коде (знаковое)**
$$(a>0 \ \&\& \ b<0 \ \&\& \ (a-b)<0) \ \mid\mid \ (a<0 \ \&\& \ b>0 \ \&\& \ (a-b)>0)$$

Флаги условий (явная установка проверкой бит)

■ Явная установка командой test

- `testq Src2, Src1`
 - `testq b, a` как вычисление `a&b` без сохранения результата
 - Устанавливает флаги в зависимости от `Src1 & Src2`
 - Удобно, если один из операндов - маска
- **ZF=1, если $a \& b == 0$**
- **SF=1, если $a \& b < 0$ (старший бит == 1)**

Чтение флагов условий IA32 - 1

■ Команды set*

- Устанавливают один байт в 1 или 0 в зависимости от флагов
- Не изменяют остальные 7 байт

Команда	Условие	Описание
sete	ZF	Равно / Ноль
setne	~ZF	Неравно / Не ноль
sets	SF	Отрицательно
setns	~SF	Неотрицательно
setg	~ (SF^OF) & ~ZF	Больше (знаковое)
setge	~ (SF^OF)	Больше или равно (знаковое)
setl	(SF^OF)	Меньше (знаковое)
setle	(SF^OF) ZF	Меньше или равно (знаковое)
seta	~CF & ~ZF	Выше (беззнаковое)
setb	CF	Ниже (unsigned)

x86-64 Целочисленные регистры

%rax	%al	%r8	%r8b
%rbx	%bl	%r9	%r9b
%rcx	%cl	%r10	%r10b
%rdx	%dl	%r11	%r11b
%rsi	%sil	%r12	%r12b
%rdi	%dil	%r13	%r13b
%rsp	%spl	%r14	%r14b
%rbp	%bp1	%r15	%r15b

- Можно обращаться к младшим байтам

Чтение флагов условий - 2

■ Команды set*

- Устанавливают один байт в 1 или 0 в зависимости от флагов

■ Один из 8 байтовых регистров

- Не изменяются остальные байты регистра
- Зануление остатка обычно – **movzbl**
 - 32-битные команды зануляют старшие 32 бита

```
int gt (long x, long y)
{
    return x > y;
}
```

Регистр	Применение
%rdi	аргумент x
%rsi	аргумент y
%rax	результат

```
cmpq  %rsi, %rdi    # Сравнение x:y
setg  %al           # Установка младшего байта %rax в 1 если >
movzbl %al, %eax   # Зануление остальных байт %rax
ret
```

Машинный уровень 2: Управление

- Управление: флаги условий
- Условные переходы и пересылки
- Циклы
- Операторы переключения
- Процедуры IA 32
 - Структура стека
 - Соглашения вызова процедур
 - Рекурсия и указатели

Переходы

■ Команды j*

- Передача управления по адресу в зависимости от флагов

j*	Условия	Описание
<code>jmp</code>	1	Безусловный
<code>je</code>	<code>ZF</code>	Равно / Ноль
<code>jne</code>	<code>~ZF</code>	Неравно / Не ноль
<code>js</code>	<code>SF</code>	Отрицательно
<code>jns</code>	<code>~SF</code>	Неотрицательно
<code>jg</code>	<code>~(SF^OF) & ~ZF</code>	Больше (знаковое)
<code>jge</code>	<code>~(SF^OF)</code>	Больше или равно (знаковое)
<code>jl</code>	<code>(SF^OF)</code>	Меньше (знаковое)
<code>jle</code>	<code>(SF^OF) ZF</code>	Меньше или равно (знаковое)
<code>ja</code>	<code>~CF & ~ZF</code>	Выше (беззнаковое)
<code>jb</code>	<code>CF</code>	Ниже (беззнаковое)

Пример условного ветвления (по старинке)

■ Создание

```
> gcc -Og -S -fno-if-conversion control.c
```

```
long absdiff
  (long x, long y)
{
    long result;
    if (x > y)
        result = x-y;
    else
        result = y-x;
    return result;
}
```

```
absdiff:
    cmpq    %rsi, %rdi    # x:y
    jle     .L4
    movq    %rdi, %rax
    subq    %rsi, %rax
    ret
.L4:      # x <= y
    movq    %rsi, %rax
    subq    %rdi, %rax
    ret
```

Регистр	Применение
%rdi	аргумент x
%rsi	аргумент y
%rax	результат

Представление «go to» кодом

- Си допускает оператор `goto`
- Переход в точку, обозначенную меткой

```
long absdiff
(long x, long y)
{
    long result;
    if (x > y)
        result = x-y;
    else
        result = y-x;
    return result;
}
```

```
long absdiff_j
(long x, long y)
{
    long result;
    int ntest = x <= y;
    if (ntest) goto Else;
    result = x-y;
    goto Done;
Else:
    result = y-x;
Done:
    return result;
}
```

Трансляция условного выражения в общем (используя ветвление)

Си код

```
val = Test ? Then_Expr : Else_Expr;
```

```
val = x>y ? x-y : y-x;
```

“*goto*” версия

```
nt = !Test;
if (nt) goto Else;
val = Then_Expr;
goto Done;
Else:
    val = Else_Expr;
Done:
    . . .
```

- Test – целочисленное выражение
 - = 0 интерпретируется как ложь
 - ≠ 0 интерпретируется как истина
- Создаёт раздельные фрагменты кода для **Then_Expr** и **Else_Expr**
- Исполняется только один из двух

Использование условной пересылки

■ Команды условной пересылки

- Команды поддерживают:

if (Test) Dest \leftarrow Src

- Есть в x86 процессорах после 1995г.
- GCC пытается использовать их
 - Но, только в безопасных случаях!

■ А зачем?

- Переходы разрушают конвейерное исполнение инструкций
- Условная пересылка не вызывает передачи управления

Си код

```
val = Test  
? Then_Expr  
: Else_Expr;
```

“goto” версия

```
result = Then_Expr;  
eval = Else_Expr;  
nt = !Test;  
if (nt) result = eval;  
return result;
```

Пример условной пересылки

```
long absdiff
    (long x, long y)
{
    long result;
    if (x > y)
        result = x-y;
    else
        result = y-x;
    return result;
}
```

Регистр	Применение
%rdi	аргумент x
%rsi	аргумент y
%rax	результат

absdiff:

```
    movq    %rdi, %rax    # x
    subq    %rsi, %rax    # result = x-y
    movq    %rsi, %rdx
    subq    %rdi, %rdx    # eval = y-x
    cmpq    %rsi, %rdi    # x:y
    cmovle %rdx, %rax    # if <=, result = eval
    ret
```

Неудачные применения усл. пересылки

Ресурсоёмкие вычисления

```
val = Test(x) ? Hard1(x) : Hard2(x);
```

- Оба значения вычисляются
- Имеет смысл только для очень простых выражений

Рискованные вычисления

```
val = p ? *p : 0;
```

- Оба значения вычисляются
- Возможен нежелательный эффект

Вычисления с побочным эффектом

```
val = x > 0 ? x*=7 : x+=3;
```

- Оба значения вычисляются
- Побочные эффекты должны исключаться

Машинный уровень 2: Управление

- Управление: флаги условий
- Условные переходы и пересылки

- Циклы

- Операторы переключения
- Процедуры IA 32

- Структура стека
- Соглашения вызова процедур
- Рекурсия и указатели

Пример цикла “do-while”

Си код

```
long pcount_do
(unsigned long x) {
    long result = 0;
    do {
        result += x & 0x1;
        x >>= 1;
    } while (x);
    return result;
}
```

“goto” версия

```
long pcount_goto
(unsigned long x) {
    long result = 0;
loop:
    result += x & 0x1;
    x >>= 1;
    if(x) goto loop;
    return result;
}
```

- Подсчитывает количество единичных бит в аргументе x (подсчёт выталкиванием)
- Использует условный переход для зацикливания или выхода из цикла

Компиляция цикла “do-while”

“goto” версия

```
long pcount_goto
(unsigned long x) {
    long result = 0;
loop:
    result += x & 0x1;
    x >>= 1;
    if(x) goto loop;
    return result;
}
```

Регистры	Применение
%rdi	аргумент x
%rax	результат

```
        movl    $0, %eax      # result = 0
.L2:                           # loop:
        movq    %rdi, %rdx
        andl    $1, %edx      # t = x & 0x1
        addq    %rdx, %rax    # result += t
        shrq    %rdi          # x >>= 1
        jne     .L2          # if (x) goto loop
        rep; ret
```

Компиляция “do-while” в общем

Си код

```
do  
    Тело  
    while (Условие);
```

■ Тело: {

оператор₁;
 оператор₂;
 ...
 оператор_n;

}

“goto” версия

```
loop:  
    Тело  
    if (Условие)  
        goto loop
```

Трансляция “while” в общем №1

- Трансляция “переход-в-середину”
- Используется при -Og

“while” версия

```
while (Условие)
      Тело
```



“goto” версия

```
goto test;
loop:
Тело
test:
if (Условие)
    goto loop;
done:
```

Пример цикла “while” №1

Си код

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

Переход в середину

```
long pcount_goto_jtm  
  (unsigned long x) {  
    long result = 0;  
    goto test;  
loop:  
    result += x & 0x1;  
    x >>= 1;  
test:  
    if(x) goto loop;  
    return result;  
}
```

■ Отличие от do-while

- Начальный goto начинает цикл с проверки

Компиляция “while” в общем №2

“while” версия

```
while (Условие)  
    Тело
```



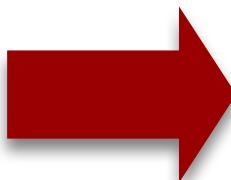
“do-while” версия

```
if (!Условие)  
    goto done;  
do  
    Тело  
    while(Условие);  
done:
```

- сведение к “do-while”
- Используется при -O1

“goto” версия

```
if (!Условие)  
    goto done;  
loop:  
    Тело  
    if (Условие)  
        goto loop;  
done:
```



Пример цикла “while” №2

Си код

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

“do-while” версия

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

■ Отличие от “do-while”

- Начальное условие защищает вход в цикл

Общая форма цикла “for”

Общая форма

```
for (Начало; Условие; Изменение)  
    Тело
```

Начало

i = 0

Условие

i < WSIZE

Изменение

i++

Тело

```
{  
    unsigned bit =  
        (x >> i) & 0x1;  
    result += bit;  
}
```

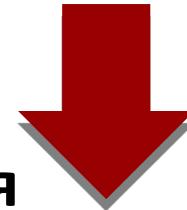
```
#define WSIZE 8*sizeof(int)  
long pcount_for  
(unsigned long x)  
{  
    size_t i;  
    long result = 0;  
    for (i = 0; i < WSIZE; i++)  
    {  
        unsigned bit =  
            (x >> i) & 0x1;  
        result += bit;  
    }  
    return result;  
}
```

Цикл “for” → цикл “while”

“for” версия

```
for (Начало; Условие; Изменение )
```

Тело



“while” версия

```
Начало ;
```

```
while (Условие) {
```

Тело

Изменение ;

```
}
```

Преобразование for-while

Начало

```
i = 0
```

Условие

```
i < WSIZE
```

Изменение

```
i++
```

Тело

```
{  
    unsigned bit =  
        (x >> i) & 0x1;  
    result += bit;  
}
```

```
long pcount_for_while  
(unsigned long x)  
{  
    size_t i;  
    long result = 0;  
    i = 0;  
    while (i < WSIZE)  
    {  
        unsigned bit =  
            (x >> i) & 0x1;  
        result += bit;  
        i++;  
    }  
    return result;  
}
```

Преобразование цикла “for” - “do-while”

“goto” версия

Си код

```
long pcount_for
(unsigned long x)
{
    size_t i;
    long result = 0;
    for (i = 0; i < WSIZE; i++)
    {
        unsigned bit =
            (x >> i) & 0x1;
        result += bit;
    }
    return result;
}
```

■ Начальное условие может быть оптимизировано

```
long pcount_for_goto_dw
(unsigned long x) {
```

size_t i;

long result = 0;

i = 0;

~~if (! (i < WSIZE))~~

~~goto done;~~

loop:

{

unsigned bit =
 (x >> i) & 0x1;

result += bit;

}

i++;

Изменение

if (i < WSIZE)

goto loop;

Начало

Условие

Тело

done:

return result;

}

Условие

Машинный уровень 2: Управление

- Управление: флаги условий
- Условные переходы и пересылки
- Циклы
- Операторы переключения
- Процедуры IA 32
 - Структура стека
 - Соглашения вызова процедур
 - Рекурсия и указатели

```
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;
            /* Переход к другому */
        case 3:
            w += z;
            break;
        case 5:
        case 6:
            w -= z;
            break;
        default:
            w = 2;
    }
    return w;
}
```

Пример оператора switch

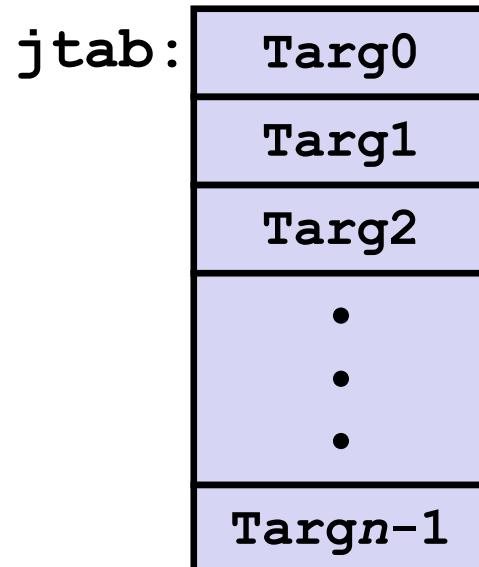
- Совмещённые варианты
 - case 5 и case 6
- Переход к другому варианту
 - Из case 2 в case 3
- Отсутствующие варианты
 - case 4

Структура таблицы переходов

В виде switch

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

Таблица переходов



Цели переходов

Targ0:

Блок кода
0

Targ1:

Блок кода
1

Targ2:

Блок кода
2

•
•
•

Targ{n-1}:

Блок кода
n-1

Приблизительный перевод

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

Пример оператора перехода

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

Пролог:

switch_eg:

```
movq    %rdx, %rcx
cmpq    $6, %rdi    # x:6
ja      .L8
jmp    * .L4(,%rdi,8)
```

Какой диапазон значений определён местом default?

Регистры	Использование
%rdi	аргумент x
%rsi	аргумент y
%rdx	аргумент z
%rax	результат

w инициализирован не здесь !

Пример оператора перехода

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

Пролог:

```
switch_eg:
    movq    %rdx, %rcx
    cmpq    $6, %rdi      # x:6
    ja     .L8           # переход к default
    jmp    * .L4(,%rdi,8) # goto *JTab[x]
```

Косвенный переход

Таблица переходов

.section	.rodata
.align	8
.L4:	
.quad	.L8 # x = 0
.quad	.L3 # x = 1
.quad	.L5 # x = 2
.quad	.L9 # x = 3
.quad	.L8 # x = 4
.quad	.L7 # x = 5
.quad	.L7 # x = 6

Пояснения к ассемблерному прологу

■ Структура таблицы

- Каждый переход требует 8 байт
- Базовый адрес (начало) .L4

■ Переход

- **Прямой:** `jmp .L8`
- Цель перехода обозначена меткой .L8
- **Косвенный:** `jmp * .L4(,%rdi,8)`
- Начало таблицы переходов: .L4
- Масштабный множитель 8 (адреса - 8 байт)
- Адрес перехода – в ячейке по эффективному адресу .L4 + x*8
 - Только для $0 \leq x \leq 6$

Таблица переходов

```
.section    .rodata
.align 8
.L4:
    .quad     .L8    # x = 0
    .quad     .L3    # x = 1
    .quad     .L5    # x = 2
    .quad     .L9    # x = 3
    .quad     .L8    # x = 4
    .quad     .L7    # x = 5
    .quad     .L7    # x = 6
```

Таблица переходов

```
.section .rodata
.align 8
.L4:
.quad .L8 # x = 0
.quad .L3 # x = 1
.quad .L5 # x = 2
.quad .L9 # x = 3
.quad .L8 # x = 4
.quad .L7 # x = 5
.quad .L7 # x = 6
```

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

Блоки кода ($x == 1$)

```
switch(x) {  
    case 1:          // .L3  
        w = y*z;  
        break;  
    . . .  
}
```

```
.L3:  
    movq    %rsi, %rax # y  
    imulq   %rdx, %rax # y*z  
    ret
```

Регистры	Использование
%rdi	аргумент x
%rsi	аргумент y
%rdx	аргумент z
%rax	результат

Рализация перехода к другому

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

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

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

Блоки кода ($x == 2$, $x == 3$)

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

```
.L5:          # Case 2  
    movq    %rsi, %rax  
    cqto  
    idivq   %rcx      # y/z  
    jmp     .L6       # goto merge  
.L9:          # Case 3  
    movl    $1, %eax  # w = 1  
.L6:          # merge:  
    addq    %rcx, %rax # w += z  
    ret
```

Регистры	Использование
%rdi	аргумент x
%rsi	аргумент y
%rdx	аргумент z
%rax	результат

Блоки кода ($x == 5$, $x == 6$, default)

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

```
.L7:                      # Case 5, 6  
    movl $1, %eax      # w = 1  
    subq %rdx, %rax   # w -= z  
    ret  
.L8:                      # Default:  
    movl $2, %eax      # 2  
    ret
```

Регистры	Использование
%rdi	аргумент x
%rsi	аргумент y
%rdx	аргумент z
%rax	результат

Промежуточный итог

■ Управление в Си

- if-then-else
- do-while
- while, for
- switch

■ Управление в ассемблере

- Условный переход
- Условная пересылка
- Косвенный переход (по таблице переходов)
- Компилятор создаёт код для более сложного управления

■ Стандартные приёмы

- Циклы преобразуются в форму do-while или переход-в-середину
- Большие switch используют таблицы переходов
- Разреженные switch могут использовать решающие деревья