

Команды управления потоком вычислений

Команды управления потоком

Это команды переходов, которые позволяют изменять в программе адрес следующей выполняемой машинной команды.

Команды безусловного перехода

call адрес

Вызов процедуры

jmp адрес

Безусловный переход

ret значение

Возврат из процедуры

Команды условного перехода

ja /jnbe адрес	Переход, если выше/не ниже или равно
jae /jnb адрес	Переход, если выше или равно/не ниже
jb /jnae адрес	Переход, если ниже/не выше или равно
jbe /jna адрес	Переход, если ниже или равно/не выше
jc адрес	Переход, если был перенос
je /jz адрес	Переход, если равно/нуль
jg /jnie адрес	Переход, если больше/не меньше или равно
jge /jnl адрес	Переход, если больше или равно/не меньше
jl /jnge адрес	Переход, если меньше/не больше или равно
jle /jng адрес	Переход, если меньше или равно/не больше
jnc адрес	Переход, если нет переноса
jne /jnz адрес	Переход, если не равно/не нуль

jno адрес

Переход, если нет переполнения

jnp/jpo адрес

Переход, если нет паритета/паритет нечетн.

jns адрес

Переход, если нет знака

jo адрес

Переход по переполнению

jp/jpe адрес

Переход, если есть паритет/ паритет четный

js адрес

Переход, если есть знак

Команды цикла

jcxz адрес

Переход, если cx равно нулю

loop адрес

Цикл пока cx $\neq 0$

Безусловные переходы

Безусловный переход изменяет *адрес следующей исполняемой команды*.

При безусловных переходах в регистр **(e)ip**, а в некоторых случаях также в регистр кодового сегмента **cs** загружаются значения новых адресов. Вместе **cs:(e)ip** определяют адрес следующей исполняемой команды.

Изменение одного или обоих регистров приводит к изменению адреса следующей исполняемой команды

Безусловный переход

команда	dst
jmp dst	<ul style="list-style-type: none">- непосредственный операнд (<i>имя метки</i>)- регистр- память

Типы переходов

- **short** (короткий) – относительный в пределах -128...+127 байт от команды **jmp**
- **near** (ближний) – в том же сегменте (относительный или абсолютный)
- **far** (дальний) – переход в другой сегмент с тем же уровнем привилегий (абсолютный)
- **task switch** - переход с переключением задачи (абсолютный, только в защищенном режиме)

Типы переходов

- По умолчанию – *short* (или по выбору компилятора)
- Синтаксис для принудительного задания типа:
 - `jmp near ptr метка`
 - `jmp far ptr метка`
- *near* переход разрешен начиная с 80386 (± 16736 байт от команды)
- *far* переход разрешен начиная с 80586

Пример

```
mov ax, 1
jmp skip
mov ax, 2
skip:  mov bx, ax ; bx = 1
```

Условные переходы

Условные переходы требуют целевого адреса - метки, обозначающей *место в программе*, с которого она продолжит исполняться в случае выполнения заданного условия.

При НЕ-выполнении условия будет выполнена следующая за командой условного перехода команда

Например:

<code>mov cx, 1</code>	; Записать 1 в cx
<code>cmp ax, bx</code>	; Сравнение ax с bx
<code>je Continue</code>	; Переход, если ax=bx
<code>xor cx, cx</code>	; Иначе, установить cx в 0
<code>Continue: ret</code>	; Возврат в программу

Команды условного перехода

- Имеют синтаксис

Jcc адрес

- где **cc** может быть:

- *буквой*, обозначающей **флаг** (например, **jс** – перейти если **CF**)
- *аббревиатурой*, обозначающей **результат сравнения** (например, **jne** – перейти, если **not equal**). Может проверять 1..3 флага.
- несколько синонимов для одной комбинации флагов

Условные обозначения

- n – not, не
- e – equal, равно
- z - ноль
- g – greater, ">" с учетом знака
- l – less, "<" с учетом знака
- a - above, ">" без учета знака
- b - below, "<" без учета знака

Условный переход

код	условие	расшифровка	пояснение
ja jbe	$CF = 0$ и $ZF = 0$	above not below	$>$ (без знака)
jae jnb jnc	$CF = 0$	above equal not below no carry	\geq (без знака) нет переноса
jb jnae jc	$CF = 1$	below not above equal carry	$<$ (без знака) есть перенос
jbe jna	$CF = 1$ или $ZF = 1$	below equal not above	\leq (без знака)
je jz	$ZF = 1$	equal zero	$=$ ноль

Условный переход

КОД	УСЛОВИЕ	расшифровка	Пояснение
jg jnl	$ZF = 0$ и $SF = OF$	greater not less equal	$>$ (со знаком)
jge jnl	$SF = OF$	greater equal not less	\geq (со знаком)
jl jnge	$SF \neq OF$	less not greater equal	$<$ (со знаком)
jle jng	$ZF = 0$ или $SF \neq OF$	less equal not greater	\leq (со знаком)
jne jnz	$ZF = 0$	not equal not zero	\neq НЕ НОЛЬ

Условный переход

код	условие	расшифровка	пояснение
jno	$OF = 0$	no overflow	нет переполнения
jo	$OF = 1$	overflow	переполнение
jnp jpo	$PF = 0$	no parity	нечетное число единиц
jp jpe	$PF = 1$	parity	четное число единиц
jns	$SF = 0$	no sign	≥ 0
js	$SF = 1$	sign	< 0

Использование условных переходов

Запомните два следующих положения:

1. Используйте переходы выше-ниже, такие как **ja** или **jbe**, для беззнаковых значений
2. Используйте переходы меньше-больше, такие как **jle** и **jg**, для чисел со знаком.

Ограничения при использовании условных переходов

Все условные переходы имеют одно существенное ограничение: они могут передавать управление только на короткие расстояния строго до 128 байт назад (в нижние адреса) или до 127 байт вперед (в верхние адреса) от первого байта команды, непосредственно следующей за командой перехода.

Для перехода более чем на 127 байт необходимо комбинировать условные и безусловные переходы вот так:

*Используйте условный переход, *противоположный тому*, которым вы могли применять, если бы целевая точка попадала в допустимый диапазон. После этого переходите в нужное место с помощью команды *jmp*.

Перед условным переходом необходимо
выполнить:

- либо команду сравнения (*стр*),
- либо в программе должна присутствовать некоторая арифметическая (логическая) команда, непосредственно предшествующая команде условного перехода, (т.к. эти команды устанавливают флаги, которые и проверяются на самом деле командами условных переходов).

Сравнение

команда	dst	src
cmp dst, src	регистр память	регистр память число

- Вычисляет ($\text{dst} - \text{src}$), но не сохраняет результат
- Устанавливает флаги так же, как **sub**

Пример

cmp ax, bx ; ax ? bx

jg Greater

j1 Less

... ; ax==bx

jmp Continue

Greater:

... ; ax>bx

jmp Continue

Less:

... ; ax<bx

Continue: ... ; продолжаем

Сравнение с обменом

команда	dst	src
cmpxchg dst, src	регистр память	регистр

- Сравнивает **AL/AX/EAX** с **dst**
(т.е. устанавливает флаги как и **cmp**)
- Выполняет пересылку:
 если $AL/AX/EAX == dst$
 то $dst := src$
 иначе $AL/AX/EAX := src$

Сравнение с обменом

команда	dst	src
cmpxchg8b dst	память (8 байт)	регистр

- Сравнивает **EDX:EAX** с **dst** (64 бита) и:
если $EDX:EAX == dst$
то $dst := ECX:EBX$
иначе $EDX:EAX := src$

Пример первой программы:

$$Y = \begin{cases} a * X + b, & \text{если } x \geq a; \\ - (a * X / (a - b)) & \text{иначе} \end{cases}$$

```
#include "pch.h"
#include <iostream>
using namespace std;
int main()
```

```
{
    setlocale(LC_ALL, "Rus");
    int a, b, x, y, DivZ=0;           // данные DD для ASM

    cout << "Введите число a\n";
    cin >> a;
    cout << "Введите число b\n";
    cin >> b;
    cout << "Введите число x\n";
    cin >> x;
```

$$Y = \begin{cases} a * X + b, & \text{если } x \geq a; \text{ или} \\ - (a * X / (a - b)) & \text{иначе} \end{cases}$$

$Y = a * X + b,$ $- (a * X / (a - b))$	если $x \geq a$; или иначе
---	--------------------------------

_asm

{

mov eax, a	; a -> eax
imul x	; eax * x -> edx:eax -> (a * x) ИЛИ можно ; так: imul eax, x (как лучше?)
mov ebx, x	; x -> ebx
cmp ebx, a	; ebx ? a -> (x ? a)
jl Less	; if (x < a) then (goto <i>Less</i>) else (<i>Continue</i>)
add eax, b	; eax + b -> eax -> (a * x + b)
jmp OK	; goto <i>OK</i> -> (y is ready in eax)
Less: mov ebx, a	; a -> ebx
sub ebx, b	; ebx - b -> ebx -> (a - b)
jz Zero	; if (==0) then (goto <i>Zero</i>) else (<i>Continue</i>)

```

neg    eax    ; - eax -> eax -> (- a*x)
cdq                      ; подготовка к делению: eax -> edx:eax
idiv   ebx    ; edx:eax / ebx -> edx:eax -> (-a*x)/(a-b)
jmp    OK     ; goto OK -> (y is ready in eax)
Zero:  mov    DivZ,1 ; 1 -> DivZ -> (признак деления на 0)
OK:    mov    y, eax ; eax -> y -> (y=a*x+b) || (-a*x)/(a-b)
                      ; закончили

```

```

}
system("cls");

```

$Y = a * X + b,$	если $x \geq a$, или
$- (a * X / (a - b))$	иначе

```

cout << "Divizion by Zero -> " << DivZ << endl << endl;
cout << "y = " << y << endl << endl;

```

```

return 0;

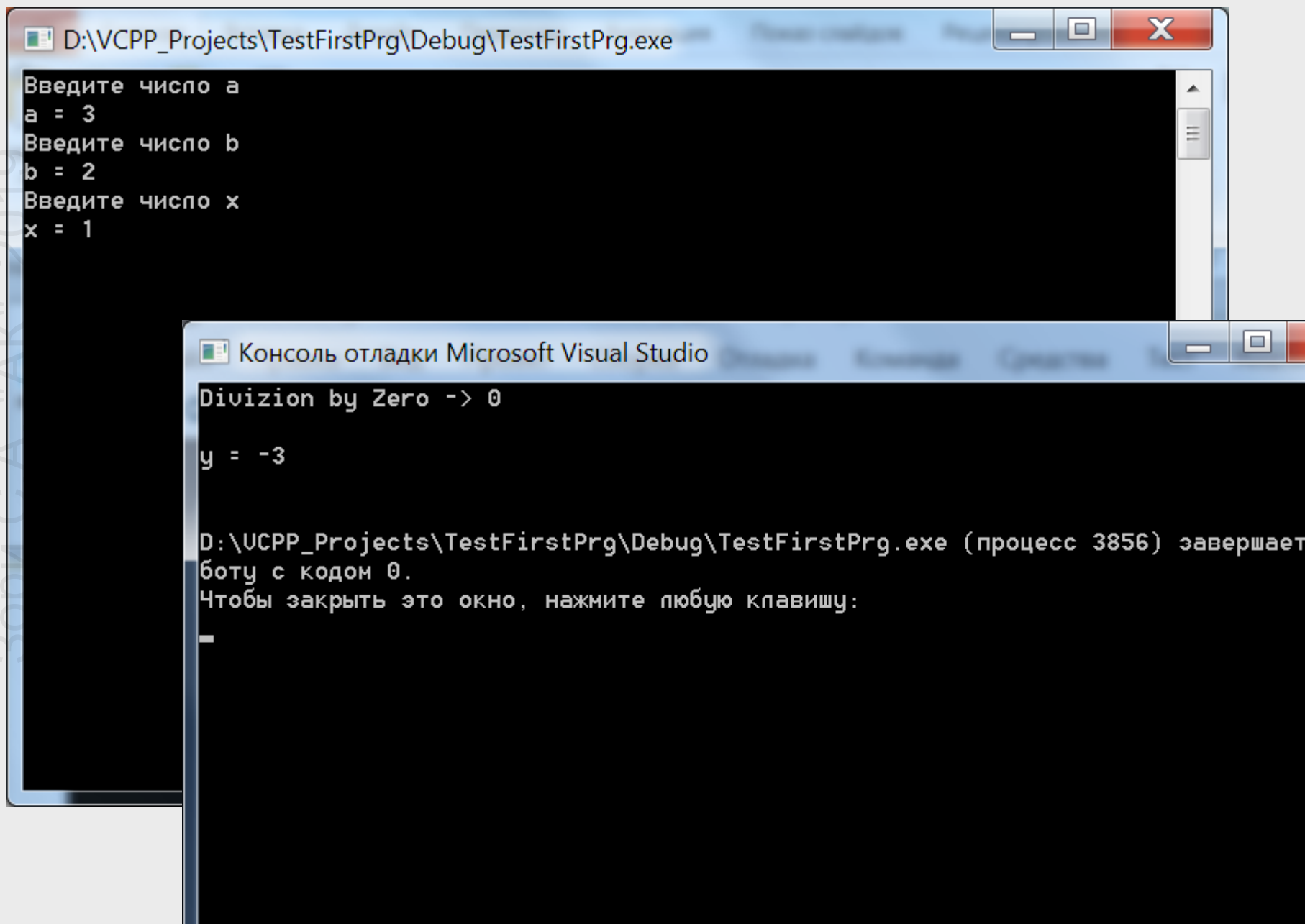
```

```

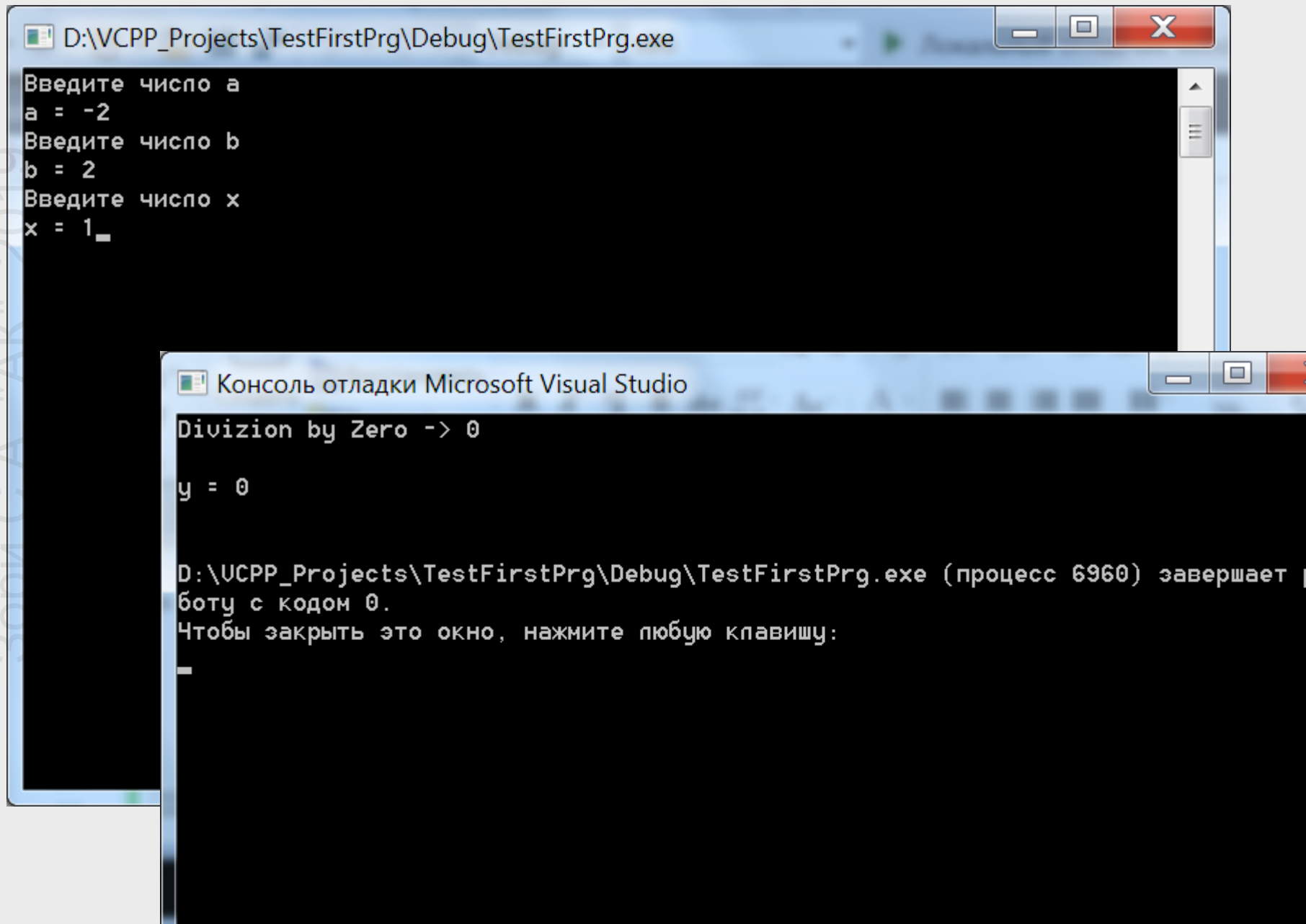
}

```

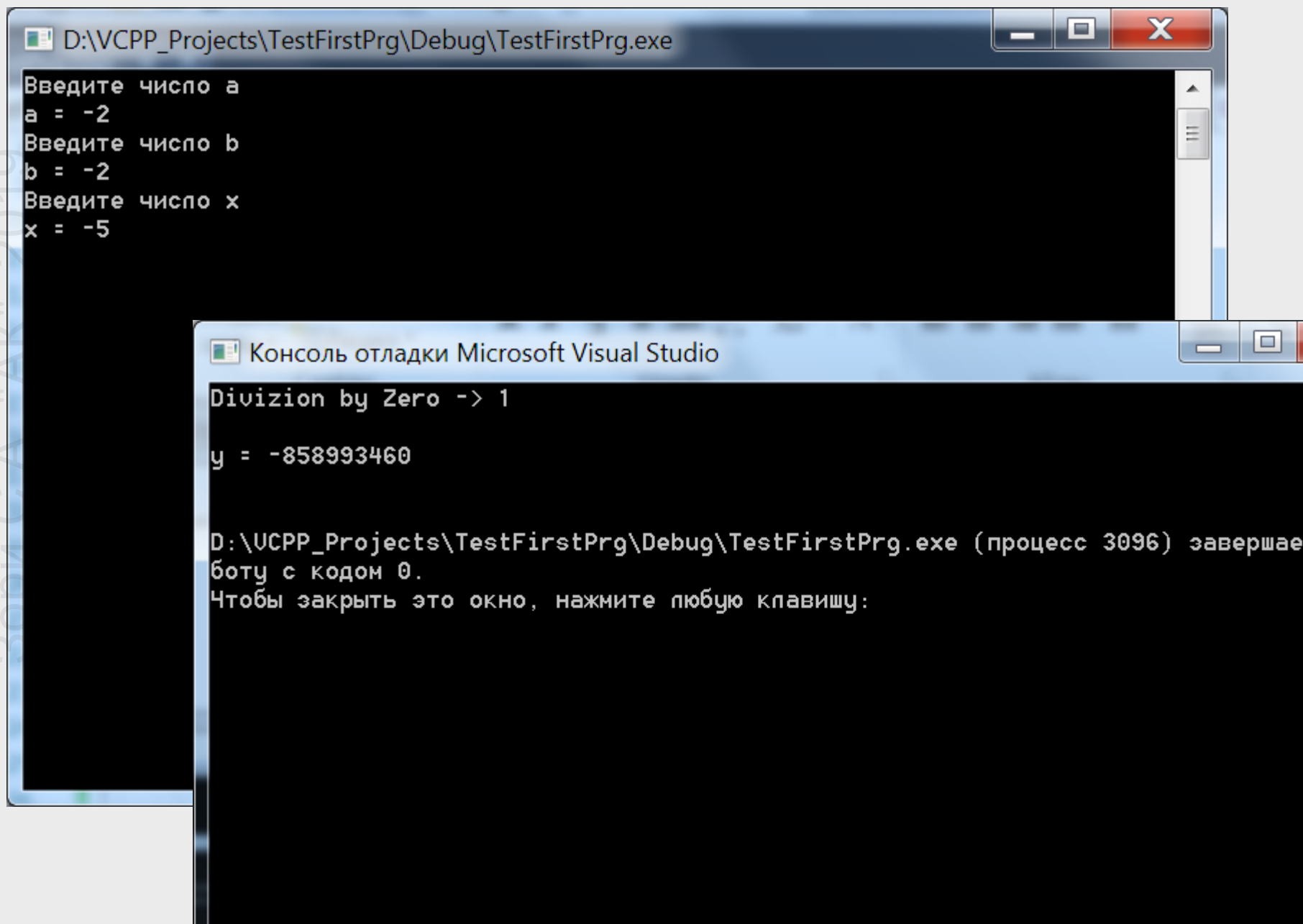
Проверяем все ветки программы!



Проверяем все ветки программы!



Проверяем все ветки программы!



The image shows two overlapping windows. The top window is titled "D:\VCpp_Projects\TestFirstPrg\Debug\TestFirstPrg.exe" and contains a simple text-based program in Russian. The bottom window is the "Консоль отладки Microsoft Visual Studio" (Visual Studio Debug Console), which shows the execution output of the program, including a division by zero error and the final exit code.

```
D:\VCpp_Projects\TestFirstPrg\Debug\TestFirstPrg.exe
Введите число a
a = -2
Введите число b
b = -2
Введите число x
x = -5
```

```
Консоль отладки Microsoft Visual Studio
Divizion by Zero -> 1
y = -858993460
D:\VCpp_Projects\TestFirstPrg\Debug\TestFirstPrg.exe (процесс 3096) завершае
боту с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу:
```