

Наверняка, у каждого есть "любимая книга, с которой все началось".

Для меня по данному направлению такой книгой стала:

- Архитектура ЭВМ. Задания и примеры выполнения лабораторных работ. Методические указания/ сост.: А.Е. Докторов, Е. А. Докторова. - Ульяновск : УлГТУ, 2008. - 32 с.

Чем примечательны эти методические указания?

- В них предлагается короткий путь вхождения в Ассемблер. Всего 32 с., из них 18 страниц посвящено теории написания ассемблерных вставок на Pascal 2.6. Остальное же - инструкции и задания к лабораторным работам.
- Подобный подвиг компактного изложения материала не удалось повторить ни одному автору, чьи книги мне попадались.
- Хотя Pascal 2.6. потерял свою актуальность, но теория и задания переносимы и на другие среды.

# Архитектура ЭВМ

## **МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ ДЛЯ СТУДЕНТОВ СПЕЦИАЛЬНОСТИ ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ**

Сост.: А. Е. ДОКТОРОВ  
Е. А. ДОКТОРОВА

Эта книга содержит таблицы команд Ассемблера, по которым вполне можно строить дальнейшие планы по изучению языка. Только не все из этих команд ныне работают...

<b>ПЕРЕСЫЛКА ДАННЫХ</b>						
MOV	PUSH	POP	XCHG	PUSHF	POPF	
XLAT	LEA	LDS	LES	LAHF	SAHF	
<b>АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ</b>						
ADD	ADC	INC	SUB	SBB	DEC	CMP
MUL	IMUL	DIV	IDIV	NEG	CBW	CWD
<b>ЛОГИЧЕСКИЕ ОПЕРАЦИИ</b>						
NOT	SHL/SAL	SHR	SAR	ROL	ROR	
RCL	RCR	AND	TEST	OR	XOR	
<b>ОБРАБОТКА БЛОКОВ ДАННЫХ</b>						
REP	REPE	REPNE	REPZ	REPNZ		
CMPSB	LODSB	MOVSB	SCASB	STOSB		
CMPSW	LODSW	MOVSW	SCASW	STOSW		
<b>КОМАНДЫ ПЕРЕДАЧИ УПРАВЛЕНИЯ</b>						
CALL	JMP	RET				
<b>КОМАНДЫ УСЛОВНОГО ПЕРЕХОДА</b>						
JZ	JO	JP	JS	JC	JA	JB
JNZ	JNO	JNP	JNS	JNC	JNA	JNB
JE		JPE			JNAE	JBE
JNE		JPO			JAЕ	JNBE
			LOOP	JCXZ		
JL	JG		LOOPE			
JNL	JGE		LOOPNE			
JLE	JNGE		LOOPZ			
JNLE			LOOPNZ			
<b>УПРАВЛЕНИЕ СОСТОЯНИЕМ ПРОЦЕССОРА</b>						
CLC	CMC	STC	CLD	STD	NOP	

## Команды пересылки данных

Итак, приступим. Первая серьёзная лабораторная работа. Если материал изучается на том или ином этапе, знайте, что позже вернуться к нему возможности не будет.

Почему? Потому что изучение глубин Ассемблера перспективами уходит в бесконечность.

Материала много, самостоятельно его почти не изучить, малейшая безалаберность на уроке приводит к почти полному выпаданию из последующих тем курса. Дабы это предотвратить выполняется по 2 задания на человека. Разрешено командное выполнение, при условии - по 2 задачи на каждого члена команды.

Почему так? Если этого не сделать, то можно можно почти сразу получить стадию бесконечного повторения предыдущего материала.

Если на данной лабораторной работе в группе это не предотвратить, то до конца курса добросовестно доходят единицы.

Выполняется на Visual Studio/c++

# Лабораторная работа

Варианты заданий.

- 1) Обменять значения в переменных `int x`; и `int yy`;; где `int yy` - указатель.
- 2) Обменять значения в переменных `x[4]` и `yy[3]`;; где `yy` - указатель на элемент массива.
- 3) Обменять значения в переменных `x[4]` и `yy[3]`;; где `yy` - указатель на элемент массива.  
Используйте команды `PUSH` и `POP` для временного хранения элементов массива в стеке.
- 4) Сделать то же самое с использованием команды `LEA`.
- 5) Используя команды пересылок, покажите, как работает команда `CMC`.
- 6) Содержимое регистра флагов поместить в переменную `int x`.
- 7) Обменять значения в переменных `int x`; и `int yy`;; где `int yy` - указатель. При этом использовать команду `XCHG`.

Перед лабораторной работой рекомендуется дать задание студентам на самостоятельный поиск информации по командам пересылки данных в Интернете или других источниках.

## ПЕРЕСЫЛКА ДАННЫХ

MOV	PUSH	POP	XCHG	PUSHF	POPF
XLAT	LEA	LDS	LES	LAHF	SAHF

## А теперь теория

Предком современного Ассемблера является Intel 8086. Поэтому регистры общего назначения называются А, В, С, D. При этом они хранят машинные слова Byte (8 бит), Word (16 бит), DWord (32 бит).

Вот более полная таблица, взятая из "Tomasz Grysztar. Flat assembler 1.73 Programmer's Manual."

Operator	Bits	Bytes
byte	8	1
word	16	2
dword	32	4
fword	48	6
pword	48	6
qword	64	8
tbyte	80	10
tword	80	10
dqword	128	16
xword	128	16
qqword	256	32
yword	256	32
dqqword	512	64
zword	512	64

Table 1.8: Size operators.

[Официальный сайт.](#)

Ассемблерные команды могут быть без операндов, могут обладать одним операндом, могут обладать двумя операндами.

*mov*[приемник], [источник]

.

[приемник] - операнд, в который помещаются машинное слово.

[приемник] - может быть только регистром.

[источник] - операнд, из которого изымается машинное слово.

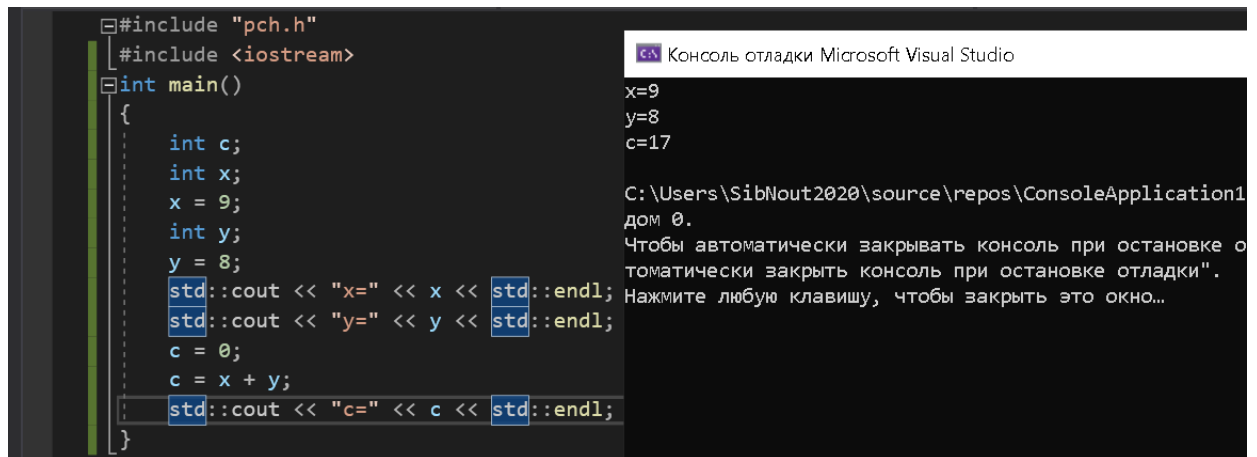
Рассмотрим регистр А.

- RAX - 64 битный регистр
- EAX - 32 битный регистр. Нижняя половина регистра RAX.
- AX - 16 битный регистр. Нижняя половина регистра EAX.
- AL - 8 битный регистр. Нижняя половина регистра AX.
- AH - 8 битный регистр. Верхняя половина регистра AX.

С остальными регистрами общего назначения - по аналогии.

Ожидается, что учащийся обладает навыками C++.

## Пример первый



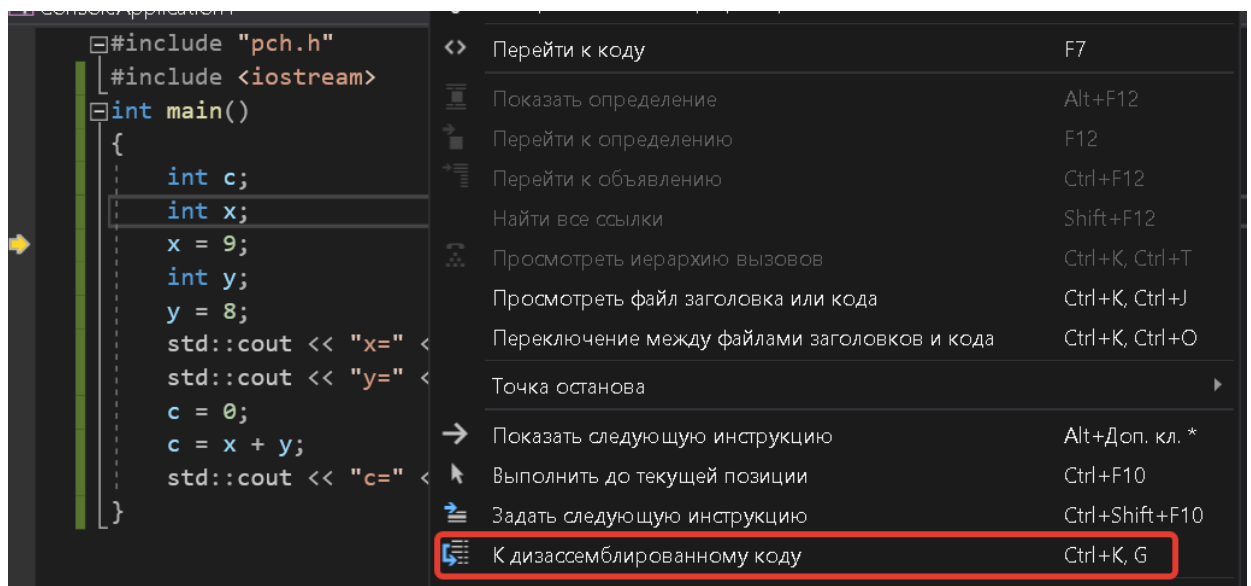
```
#include "pch.h"
#include <iostream>
int main()
{
    int c;
    int x;
    x = 9;
    int y;
    y = 8;
    std::cout << "x=" << x << std::endl;
    std::cout << "y=" << y << std::endl;
    c = 0;
    c = x + y;
    std::cout << "c=" << c << std::endl;
}
```

Консоль отладки Microsoft Visual Studio

```
x=9
y=8
c=17

C:\Users\SibNout2020\source\repos\ConsoleApplication1
дом 0.
Чтобы автоматически закрывать консоль при остановке о
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно...
```

Этот программный код обменивает местами значения двух обычных переменных. Поскольку на начальных этапах у любого юного ассемблера возникают многочисленные проблемы, мы пойдем по пути дизассемблера. Если не знаете как писать что-то, то напишите это на с++ (как мы сделали сейчас). Нажмите F10 (пошаговое исполнение программного кода), правой кнопкой мыши на самом программном коде вызываем контекстное меню, в котором выбираем "Перейти к дизассемблированному коду".



Откроется вкладка "Дизассемблированный код".

```
In [ ]: #include "pch.h"
#include <iostream>
int main()
{
    int c=0;
012428C8 push     ebp
012428C9 mov      ebp,esp
012428CB sub      esp,30h
012428CE cmp      dword ptr ds:[11B42F0h],0
012428D5 je      <Module>.main()+014h (012428DCh)
012428D7 call    72DEFD80
012428DC xor      edx,edx
012428DE mov     dword ptr [ebp-4],edx
012428E1 xor      edx,edx
```

```

012428E3 mov     dword ptr [ebp-0Ch],edx
012428E6 xor     edx,edx
012428E8 mov     dword ptr [ebp-8],edx
012428EB xor     edx,edx
012428ED mov     dword ptr [ebp-4],edx
        int x;
        x = 9;
012428F0 mov     dword ptr [ebp-0Ch],9
        int y;
        y = 8;
012428F7 mov     dword ptr [ebp-8],8
        std::cout << "x=" << x << std::endl;
012428FE mov     ecx,dword ptr [__imp_std::cout (0767084h)]
01242904 mov     edx,767450h
01242909 call    dword ptr [Указатель на CLRStub[MethodDescPrestub]@352c5db101240851]
0124290F mov     dword ptr [ebp-10h],eax
01242912 mov     ecx,dword ptr [ebp-10h]
01242915 mov     edx,dword ptr [ebp-0Ch]
01242918 call    <Module>.std.basic_ostream<char,std::char_traits<char> >.<<(std.bas
0124291D mov     dword ptr [ebp-14h],eax
01242920 mov     edx,dword ptr [__unep@??$endl@DU?$char_traits@D@std@@@std@@$FYAAAV
01242926 mov     ecx,dword ptr [ebp-14h]
01242929 call    <Module>.std.basic_ostream<char,std::char_traits<char> >.<<(std.bas
0124292E mov     dword ptr [ebp-18h],eax
01242931 nop
        std::cout << "y=" << y << std::endl;
01242932 mov     ecx,dword ptr [__imp_std::cout (0767084h)]
01242938 mov     edx,767454h
0124293D call    dword ptr [Указатель на CLRStub[MethodDescPrestub]@352c5db101240851]
01242943 mov     dword ptr [ebp-1Ch],eax
01242946 mov     ecx,dword ptr [ebp-1Ch]
01242949 mov     edx,dword ptr [ebp-8]
0124294C call    <Module>.std.basic_ostream<char,std::char_traits<char> >.<<(std.bas
01242951 mov     dword ptr [ebp-20h],eax
01242954 mov     edx,dword ptr [__unep@??$endl@DU?$char_traits@D@std@@@std@@$FYAAAV
0124295A mov     ecx,dword ptr [ebp-20h]
0124295D call    <Module>.std.basic_ostream<char,std::char_traits<char> >.<<(std.bas
01242962 mov     dword ptr [ebp-24h],eax
01242965 nop
        c = 0;
01242966 xor     edx,edx
01242968 mov     dword ptr [ebp-4],edx
        c = x + y;
0124296B mov     eax,dword ptr [ebp-0Ch]
0124296E add     eax,dword ptr [ebp-8]
01242971 mov     dword ptr [ebp-4],eax
        std::cout << "c=" << c << std::endl;
01242974 mov     ecx,dword ptr [__imp_std::cout (0767084h)]
0124297A mov     edx,767458h
0124297F call    dword ptr [Указатель на CLRStub[MethodDescPrestub]@352c5db101240851]
01242985 mov     dword ptr [ebp-28h],eax
01242988 mov     ecx,dword ptr [ebp-28h]
0124298B mov     edx,dword ptr [ebp-4]
0124298E call    <Module>.std.basic_ostream<char,std::char_traits<char> >.<<(std.bas
01242993 mov     dword ptr [ebp-2Ch],eax
01242996 mov     edx,dword ptr [__unep@??$endl@DU?$char_traits@D@std@@@std@@$FYAAAV
0124299C mov     ecx,dword ptr [ebp-2Ch]
0124299F call    <Module>.std.basic_ostream<char,std::char_traits<char> >.<<(std.bas
012429A4 mov     dword ptr [ebp-30h],eax
012429A7 nop
}

```

```

012429A8 xor     eax,eax
012429AA mov     esp,ebp
012429AC pop     ebp
012429AD ret
012429AE add     byte ptr [eax],al
012429B0 mov     ah,63h
012429B2 add     dword ptr ds:[eax],eax
012429B5 add     byte ptr [eax],al
012429B7 add     byte ptr [eax+28013E63h],ch
012429BD test    al,1Bh
012429BF ?? ??????

```

Из всего этого нам нужен лишь вот этот кусок программного кода.

```

In [ ]: int x;
        x = 9;
012428F0 mov     dword ptr [ebp-0Ch],9
        int y;
        y = 8;
012428F7 mov     dword ptr [ebp-8],8
        c = 0;
01242966 xor     edx,edx
01242968 mov     dword ptr [ebp-4],edx
        c = x + y;
0124296B mov     eax,dword ptr [ebp-0Ch]
0124296E add     eax,dword ptr [ebp-8]
01242971 mov     dword ptr [ebp-4],eax

```

```

In [ ]: # Мы видим, что int в этой системе 32 битный
# Это следствие оптимизации компилятора...
        int x;
        x = 9;
012428F0 mov     dword ptr [ebp-0Ch],9
# [ ] - обращение к ячейке памяти по указателю
# dword ptr [ebp-0Ch] - обращение к ячейке памяти по указателю,
# адрес ebp(вершина стека) -0Ch (смещение до ячейки памяти переменной x)
        int y;
        y = 8;
012428F7 mov     dword ptr [ebp-8],8
# dword ptr [ebp-8] - обращение к переменной y
        c = 0;
01242966 xor     edx,edx
# очистка регистра edx, а как по другому быстро получить ноль?
01242968 mov     dword ptr [ebp-4],edx
        c = x + y;
0124296B mov     eax,dword ptr [ebp-0Ch]
# Кладем машинное слово из переменной X в регистр eax
0124296E add     eax,dword ptr [ebp-8]
# Прибавляем к регистру машинное слово из переменной Y
01242971 mov     dword ptr [ebp-4],eax
# Возвращаем результат в переменную C.

```

Как видите, машина в одной ветви программного кода старается работать через один регистр общего назначения.

```
#include "pch.h"
#include <iostream>
int main()
{
    int c=0;
    int x;
    x = 9;
    int y;
    y = 8;
    std::cout << "x=" << x << std::endl;
    std::cout << "y=" << y << std::endl;
    c = 0;
    __asm
    {
        mov     eax, dword ptr[ebp - 0Ch]
        add     eax, dword ptr[ebp - 8]
        mov     dword ptr[ebp - 4], eax
    }
    std::cout << "c=" << c << std::endl;
}
```

Консоль отладки Microsoft Visual Studio

x=9  
y=8  
c=17

C:\Users\SibNout2020\source\repos\ConsoleApplication1  
дом 2061142408.  
Чтобы автоматически закрывать консоль при остановке отладки, нажмите любую клавишу, чтобы закрыть это окно...

## Пример второй

```
#include "pch.h"
#include <iostream>
int main()
{
    int x= 9;
    int y= 8;
    std::cout << "x=" << x << std::endl;
    std::cout << "y=" << y << std::endl;
    int c = 0;
    c = x;
    x = y;
    y = c;
    std::cout << "x=" << x << std::endl;
    std::cout << "y=" << y << std::endl;
}
```

Консоль отладки Microsoft Visual Studio

x=9  
y=8  
x=8  
y=9

C:\Users\SibNout2020\source\repos\ConsoleApplication1  
дом 0.  
Чтобы автоматически закрывать консоль при остановке отладки, нажмите любую клавишу, чтобы закрыть это окно...

```
In [ ]: c = x;
01522966 mov     eax,dword ptr [ebp-8]
01522969 mov     dword ptr [ebp-0Ch],eax
        x = y;
0152296C mov     eax,dword ptr [ebp-4]
0152296F mov     dword ptr [ebp-8],eax
        y = c;
01522972 mov     eax,dword ptr [ebp-0Ch]
01522975 mov     dword ptr [ebp-4],eax
```



```

int x= 9;
int y= 8;
std::cout << "x=" << x << std::endl;
std::cout << "y=" << y << std::endl;
int c = 0;
__asm {
    mov     eax, dword ptr[ebp - 8]
    mov     dword ptr[ebp - 0Ch], eax
    mov     eax, dword ptr[ebp - 4]
    mov     dword ptr[ebp - 8], eax
    mov     eax, dword ptr[ebp - 0Ch]
    mov     dword ptr[ebp - 4], eax
}
std::cout << "x=" << x << std::endl;
std::cout << "y=" << y << std::endl;

```

Консоль отладки Microsoft Visual Studio

```

x=9
y=8
x=8
y=9
C:\Users\SibNout2020\source\repos\ConsoleApplica
дом 2061142408.
Чтобы автоматически закрывать консоль при остано
томатически закрыть консоль при остановке отладк
Нажмите любую клавишу, чтобы закрыть это окно...

```

Можно и короче

```

int x= 9;
int y= 8;
std::cout << "x=" << x << std::endl;
std::cout << "y=" << y << std::endl;
int c = 0;
__asm {
    mov     eax, dword ptr[ebp - 8]
    mov     ebx, dword ptr[ebp - 4]
    mov     dword ptr[ebp - 8], ebx
    mov     dword ptr[ebp - 4], eax
}
std::cout << "x=" << x << std::endl;
std::cout << "y=" << y << std::endl;

```

Консоль отладки Microsoft Visual Studio

```

x=9
y=8
x=8
y=9
C:\Users\SibNout2020\source\repos\ConsoleApplica
дом 2061142408.
Чтобы автоматически закрывать консоль при остан
томатически закрыть консоль при остановке отлад
Нажмите любую клавишу, чтобы закрыть это окно...

```

```

int x= 9;int y= 8;
std::cout << "<" << x<<";"<< y << ">" << std::endl;
int c = 0;
__asm {
    mov     eax, dword ptr[ebp - 8]
    mov     ebx, dword ptr[ebp - 4]
    XCHG    eax, ebx
    mov     dword ptr[ebp - 8], eax
    mov     dword ptr[ebp - 4], ebx
}
std::cout << "<" << x << ";" << y << ">" << std::endl;

```

Консоль отладки Microsoft Visual Studio

```

<9;8>
<8;9>
C:\Users\SibNout2020\source\repos\ConsoleApplica
дом 2061142408.
Чтобы автоматически закрывать консоль при остано
томатически закрыть консоль при остановке отлад
Нажмите любую клавишу, чтобы закрыть это окно...

```

```

int x= 9;int y= 8;
std::cout << "<" << x<<";"<< y << ">" << std::endl;
int c = 0;
__asm {
    mov     eax, dword ptr[ebp - 8]
    XCHG    eax, dword ptr[ebp - 4]
    mov     dword ptr[ebp - 8], eax
}
std::cout << "<" << x << ";" << y << ">" << std::endl;

```

Консоль отладки Microsoft Visual Studio

```

<9;8>
<8;9>
C:\Users\SibNout2020\source\repos\ConsoleApplica
дом 2061142408.
Чтобы автоматически закрывать консоль при остано
томатически закрыть консоль при остановке отлад
Нажмите любую клавишу, чтобы закрыть это окно...

```

Совет. Если пишете программный код, по-началу очищайте используемые регистры. **XOR EAX, EAX**. Это избавит от мусора в регистрах...

## Пример третий

```
In [ ]: #include "pch.h"
#include <iostream>
int main()
{
    //переменная x
    int x = 9;
    //переменная y
    int y = 8;
    //Указатель на переменную yy
    int* yy = &y;
    std::cout << "x=" << x << std::endl;
    std::cout << "y=" << y << std::endl;
    //Обращение к адресу оперативной памяти переменной
    std::cout << "&y=" << &y << std::endl;
    //Просто посмотреть - что внутри указателя
    std::cout << "yy=" << yy << std::endl;
    //Обращение к значению по адресу из указателя
    std::cout << "*yy=" << *yy << std::endl;
    _asm
    {
        xor eax, eax
        xor ebx, ebx
        mov     eax, dword ptr[x]
        mov     ecx, dword ptr[yy]
        // Обращаемся к переменной указателя
        // Получаем содержимое переменной указателя в регистр ecx
        mov     ebx, dword ptr[ecx]
        //Теперь ecx содержит указатель на переменную y
        //Получаем значение переменной y в ebx
        XCHG eax, ebx
        mov     dword ptr[x], eax
        mov     dword ptr[ecx], ebx
        //Кладем значение в переменную y
        //ecx содержит указатель на переменную y
    }
    std::cout << "x=" << x << std::endl;
    std::cout << "y=" << y << std::endl;
}
```

```

//переменная x
int x = 9;
//переменная y
int y = 8;
//Указатель на переменную yy
int* yy = &y;
std::cout << "x=" << x << std::endl;
std::cout << "y=" << y << std::endl;
//Обращение к адресу оперативной памяти по указателю
std::cout << "&y=" << &y << std::endl;
//Просто посмотреть что внутри указателя
std::cout << "yy=" << yy << std::endl;
//Обращение к значению по адресу из указателя
std::cout << "*yy=" << *yy << std::endl;

```

Внимание, при копировании текста в Visual Studio 2019 некоторые символы вставляются как символы других кодов, но с тем же внешним видом. В таких случаях рекомендуется либо создать новый проект, либо в Visual Studio набрать код с клавиатуры.

## Пример четвертый

Работа с массивами на Ассемблере ведется по тем же принципам, что и работа с массивами в ++ через арифметику указателей. Вот пример.

```

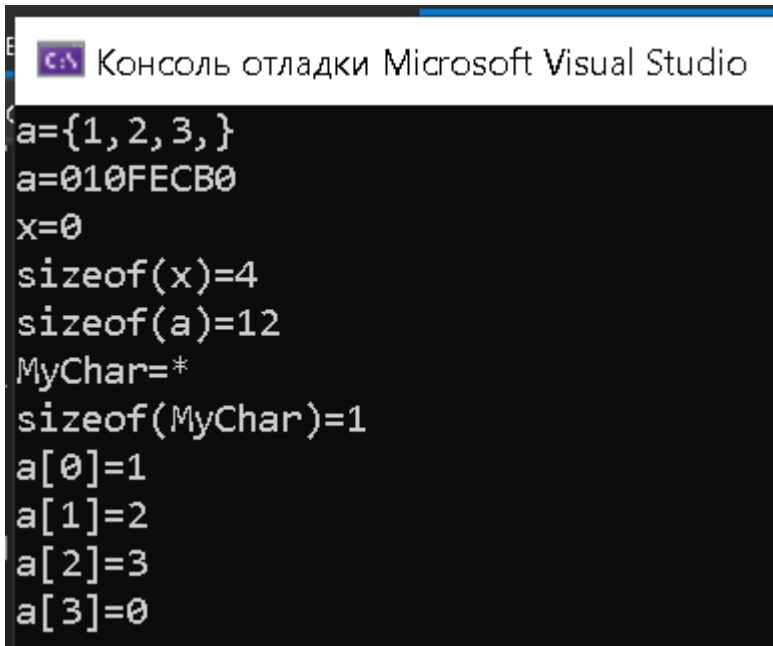
In [ ]: #include "pch.h"
#include <iostream>
int main()
{
    //переменная x
    int x = 0;
    //Создали одномерный массив из 3 элементов
    int a[3] = { 1, 2, 3 };
    //Создали символ
    char MyChar = '*';
    //Вывели наш массив на экран в стиле аля питон
    std::cout << "a={" << a[0] << "," << a[1] << "," << a[2] << "," << "}" << std::endl;
    //Вывели на экран весь массив, и вдруг выяснилось, что это указатель
    std::cout << "a=" << a << std::endl;
    std::cout << "x=" << x << std::endl;
    //Получили размер переменной x, которая int
    std::cout << "sizeof(x)=" << sizeof(x) << std::endl; //4
    //Получили размер всего нашего массива (он 12) из 3 элементов размером по 4
    std::cout << "sizeof(a)=" << sizeof(a) << std::endl;
    std::cout << "MyChar=" << MyChar << std::endl;
    //Вывели размер символа
    std::cout << "sizeof(MyChar)=" << sizeof(MyChar) << std::endl;
    //Вывели на экран первый элемент массива, обратившись к нему через Ассемблер
    std::cout << "a[0]=";
    _asm
    {
        xor eax, eax
        mov eax, dword ptr[a]
    }
}

```

```

        mov dword ptr[x], eax
    }
    std::cout << x << std::endl;
    //Вывели на экран первый элемент массива, обратившись к нему через Ассемблер
    //НЕ ЗАБЫЛИ У INT ПРО РАЗМЕР И ДОПИСАЛИ СМЕЩЕНИЕ В ОПЕРАТИВНОЙ ПАМЯТИ +4
    std::cout << "a[1]=";
    _asm
    {
        xor eax, eax
        mov eax, dword ptr[a + 4 * 1]
        mov dword ptr[x], eax
    }
    std::cout << x << std::endl;
    //+4 2 раза
    std::cout << "a[2]=";
    _asm
    {
        xor eax, eax
        mov eax, dword ptr[a + 4 * 2]
        mov dword ptr[x], eax
    }
    std::cout << x << std::endl;
    //Обратились к 4, несуществующему элементу массива
    //Получили мусор
    std::cout << "a[3]=";
    _asm
    {
        xor eax, eax
        mov eax, dword ptr[a + 4 * 3]
        mov dword ptr[x], eax
    }
    std::cout << x << std::endl;
}

```



Консоль отладки Microsoft Visual Studio

```

a={1, 2, 3, }
a=010FECB0
x=0
sizeof(x)=4
sizeof(a)=12
MyChar=*
sizeof(MyChar)=1
a[0]=1
a[1]=2
a[2]=3
a[3]=0

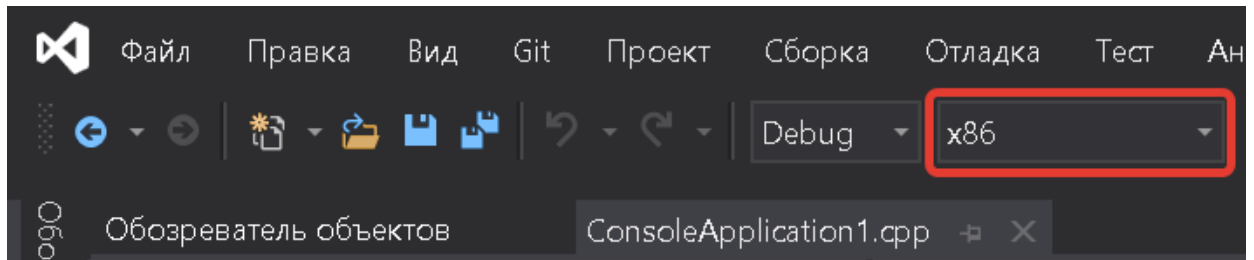
```

## Лабораторная работа

Задание взято из "Архитектура ЭВМ. Задания и примеры выполнения лабораторных работ.

Методические указания/ сост. : А.Е. Докторов, Е. А. Докторова. - Ульяновск : УлГТУ, 2008. - 32 с".

**Тема лабораторной работы. 64 битные операции на 32 битной машине** Выставляем в Visual Studio "Debug x86". После этого программный код будет генерироваться 32 битным даже на 64 битной машине.



Далее рекомендуется дать на самостоятельный поиск по источникам из Интернета следующие команды. Также их краткие описания можно посмотреть в [А. Е. Докторов.2008]

### АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

ADD	ADC	INC	SUB	SBB	DEC	CMP
MUL	IMUL	DIV	IDIV	NEG	CBW	CWD

Требуется написать ассемблерную вставку сложения (вычитания) двух 64 битных чисел.

```
In [ ]: #include "pch.h"
#include <iostream>
int main()
{
    __int64 c=0;
    __int64 x;
    x = 4;
    __int64 y;
    y = 6;
    c = x + y;
    std::cout << "x=" << x << std::endl;
    std::cout << "y=" << y << std::endl;
    std::cout << "c=" << c << std::endl;
}
```

```
In [ ]: __int64 x;
x = 4;
0068290F mov     eax,4
00682914 cdq
00682915 mov     dword ptr [ebp-10h],eax
00682918 mov     dword ptr [ebp-0Ch],edx
__int64 y;
y = 6;
0068291B mov     eax,6
00682920 cdq
00682921 mov     dword ptr [ebp-8],eax
00682924 mov     dword ptr [ebp-4],edx
__int64 c=0;
c = x + y;
00682927 mov     eax,dword ptr [ebp-10h]
0068292A mov     edx,dword ptr [ebp-0Ch]
0068292D add     eax,dword ptr [ebp-8]
00682930 adc     edx,dword ptr [ebp-4]
00682933 mov     dword ptr [ebp-18h],eax
00682936 mov     dword ptr [ebp-14h],edx
c = x - y;
00EE2927 mov     eax,dword ptr [ebp-10h]
```

```
00EE292A  mov     edx,dword ptr [ebp-0Ch]
00EE292D  sub     eax,dword ptr [ebp-8]
00EE2930  sbb     edx,dword ptr [ebp-4]
00EE2933  mov     dword ptr [ebp-18h],eax
00EE2936  mov     dword ptr [ebp-14h],edx
```

Задим студентам вопрос: "Почему сложение 64 битных чисел?" Возможно, кто-то из них догадается или найдет ответ в презентациях...

Ответ ожидается примерно следующий: 32 битная система оперирует машинными словами Word 32 бита.

Единственный способ эмулировать 64 битный код, это использовать два 32 битных слова.

Ранее подобный прием использовался на 16 битных машинах с 32 битным кодом.

Как бы выглядел 64 битный код на 8 битной машине?

Производительность в этом случае была бы просто "фантастическая".

Поддержка кода осуществляется за счет транслирования с макро-ассемблера в микро-ассемблер, родной для конкретной машины. Чтобы это осуществить, требуется написать множество подпрограмм, заменяющих базовые операции или другие подпрограммы.

Ранее базовые команды микропроцессора реализовывались на уровне железа. Сейчас некоторые базовые команды реализуются на программном уровне. Это приводит к снижению производительности, но позволяет крос-платформенность.

Последующие темы по методическим указаниям [Докторов 2008]:

- Изучение логических команд и команд сдвигов.
- Изучение команд обработки блоков данных. Цикл LOOP. Обработка текстов.
- Изучение команд условного перехода.
- Изучение команд передачи управления.

Но не так все просто.

Команда LOOP работает только в Pascal ранних версий и DosBox.

Некоторых команд обработки данных нет в современном ассемблере. Их заменяют своими процедурами и работой через указатели.

Потому, с этого момента [Докторов 2008] будет использован нами как дополнительный материал.

Мы плавно переключаемся на программирование под DosBox на FlatAssembler1.71.

Сайт [FasmWorld](http://FasmWorld.com).

Продолжение следует...

In [ ]: