

# Машинно-зависимые языки программирования

## Лабораторная работа №7

### “Программирование для процессоров x86/x86-64. Обработка строк”

#### Справочная информация

##### Особенности 32- и 64-разрядных процессоров семейства x86

В процессорах присутствуют все те же самые регистры, но в расширенном виде - в 32-разрядном исполнении они имеют приставку E, в 64-разрядном - R. Таким образом, регистр RAX включает в себя младшую часть, к которой можно обратиться по имени EAX. В свою очередь, к младшей части EAX можно обратиться по имени AX. С сегментными регистрами при реализации ассемблерных вставок или линкуемых модулей работать не требуется.

##### Соглашения о вызовах

**Соглашение о вызове** — формализация правил вызова подпрограмм, которое должно включать:

- способ передачи параметров;
- способ возврата результата из функции;
- способ возврата управления.

Соглашения о вызовах определяются в рамках отдельных языков высокого уровня, а также - различных программных API, в т. ч. API операционных систем.

##### Соглашение о вызове Си

**cdecl32** — соглашение о вызовах, используемое компиляторами для языка Си на 32-разрядных системах.

1. Аргументы функций передаются через стек, справа налево.
2. Аргументы, размер которых меньше 4-х байт, расширяются до 4-х байт.
3. Очистку стека производит вызывающая программа.
4. Возврат параметров 1, 2, 4 байта (целые числа, указатели) - через eax.
5. Возврат больших структур, массивов, строк - указателем через eax.

Перед вызовом функции вставляется код, выполняющий следующие действия:

- сохранение значений регистров, используемых внутри функции;
- запись в стек аргументов функции.

После вызова функции вставляется код, выполняющий следующие действия:

- очистка стека;

- восстановление значений регистров.

В 64-разрядных системах могут применяться другие соглашения.

### Разработка фрагментов кода на ассемблере для ЯВУ

В языках высокого уровня, в частности C/C++, часто присутствует 2 возможности встраивания низкоуровневого кода:

1. Ассемблерные вставки - специальный оператор ЯВУ, внутри которого пишется код на ассемблере.
2. Компиляция и линковка отдельных модулей, полностью написанных на ассемблере.

Составной оператор языка высокого уровня, телом которого является код на языке ассемблера.

Пример вставки на C++ для Visual Studio 2019:

```
#include <iostream>

int main()
{
    int i;
    __asm {
        mov eax, 5;
        mov i, eax;
    }
    std::cout << i;
    return 0;
}
```

### Возможности Visual Studio

В процессе отладки программы доступна возможность получения дизассемблированного кода программы (изначально написанного как на ассемблере, так и на C++):

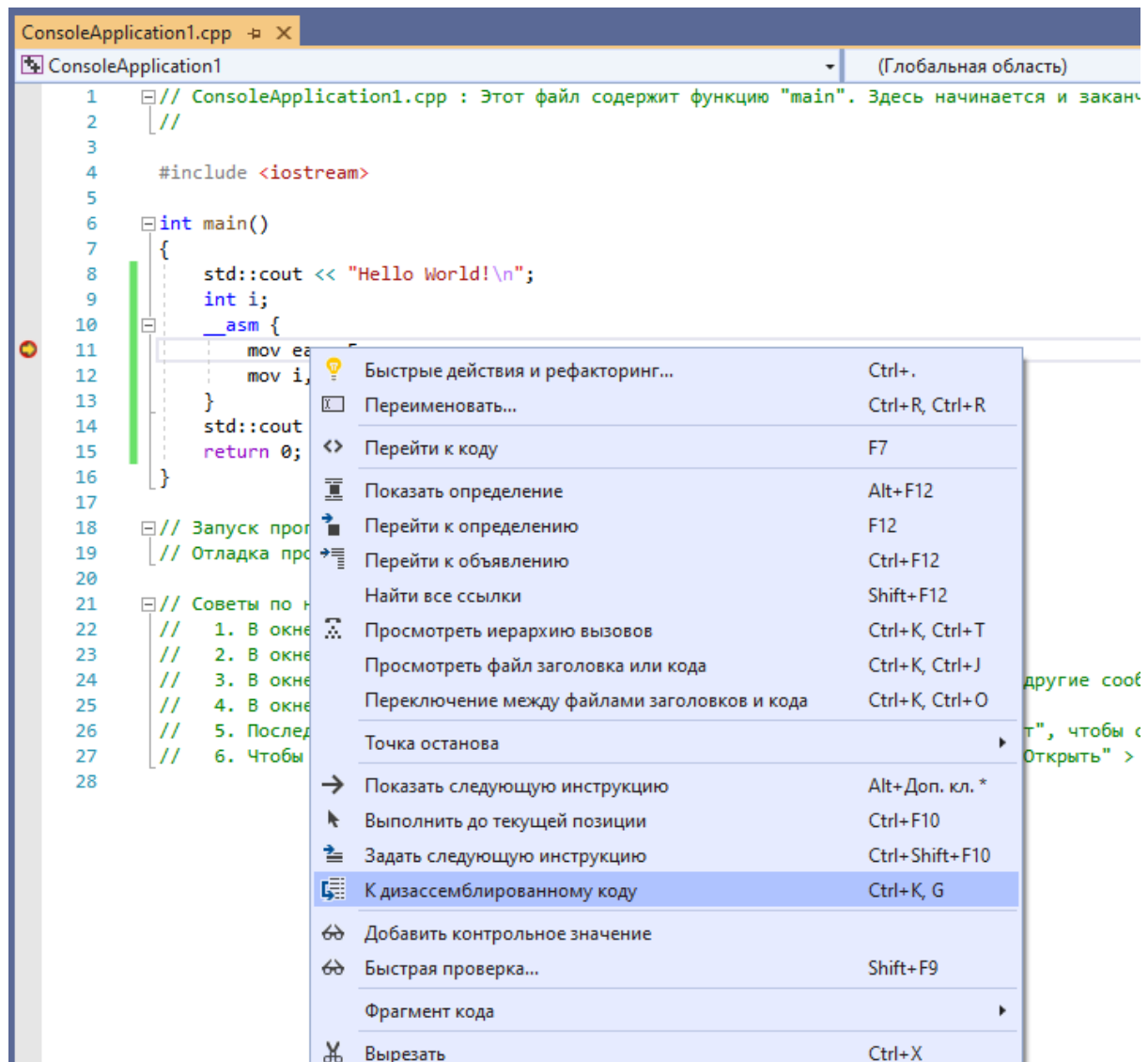


Рисунок 1. Дизассемблирование

Дизассемблированный код - X ConsoleApplication1.cpp

Адрес: main(void)

Параметры просмотра

```

{
008624B0 push     ebp
008624B1 mov      ebp,esp
008624B3 sub      esp,0D0h
008624B9 push     ebx
008624BA push     esi
008624BB push     edi
008624BC lea      edi,[ebp-0D0h]
008624C2 mov      ecx,34h
008624C7 mov      eax,0CCCCCCCCh
008624CC rep stos dword ptr es:[edi]
008624CE mov      eax,dword ptr [__security_cookie (086C004h)]
008624D3 xor      eax,ebp
008624D5 mov      dword ptr [ebp-4],eax
    std::cout << "Hello World!\n";
008624D8 push     offset string "Hello World!\n" (0869B30h)
008624DD mov      eax,dword ptr [__imp__cout@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A (086D0CCh)]
008624E2 push     eax
008624E3 call     std::operator<<<std::char_traits<char> > (086120Dh)
008624E8 add      esp,8
    int i;
    __asm {
        mov eax, 5;
008624EB mov      eax,5
        mov i, eax;
008624F0 mov      dword ptr [i],eax
    }
    std::cout << i;
008624F3 mov      esi,esp
008624F5 mov      eax,dword ptr [i]
008624F8 push     eax
008624F9 mov      ecx,dword ptr [__imp__cout@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A (086D0CCh)]
008624FF call     dword ptr [__imp_std::basic_ostream<char,std::char_traits<char> >::operator<< (086D09Ch)]
00862505 cmp      esi,esp
00862507 call     __RTC_CheckEsp (086127Bh)

```

Рисунок 2. Дизассемблированный код программы

Под каждой строкой кода на Си(Си++) показан дизассемблированный машинный код, в который эта строка была скомпилирована. Например, на рис. 2 видно, что оператору << соответствует 5-7 машинных команд, включая передачу параметров через стек и вызов библиотечной функции.

*Примечание: если пункт “К дизассемблированному коду” недоступен, может потребоваться выбрать пункт “Параметры...” в меню “Отладка” и установить флажок “Включить отладку на уровне адреса.”*

Также в Visual Studio существует возможность включения в проект asm-файлов. Для этого требуется:

1. в Обозревателе решений открыть контекстное меню проекта и выбрать “Зависимости сборки” -> “Настройки сборки...” и в открывшемся окне установить флажок напротив строки `masm`
2. через Проводник (или другой файловый менеджер) создать файл `.asm` в каталоге с исходным кодом
3. в Обозревателе решений добавить созданный файл в группу “Исходные файлы”

Например, создадим проект с кодом на Си++:

```
#include <iostream>
```

```
extern "C"
```

```

{
    void testAsm(); // подключение в код на Си/Си++ функции
                    // на другом языке программирования,
                    // выполненной в соответствии с соглашениями
                    // о вызовах Си
}

int main()
{
    int i;
    __asm {
        mov eax, 5;
        mov i, eax;
    }
    std::cout << i;
    testAsm();
    __asm {
        mov i, eax;
    }
    std::cout << i;
    return 0;
}

```

и файлом test.asm

```

.686
.MODEL FLAT, C
.STACK

.CODE

testAsm PROC
    mov eax, 7
    ret
testAsm ENDP
END

```

Такая программа выведет в консоль символы 57: первая цифра задана в ассемблерной вставке, вторая - в отдельном файле.

## Практическое задание

Написать программу на Си/Си++, которая вызывает 2 подпрограммы на ассемблере:

- первая принимает 1 параметр - указатель на строку, определяет длину строки и выполнена в виде ассемблерной вставки;
- вторая копирует строку с адреса, заданного одним указателем, по адресу, заданному другим указателем, и реализована в отдельном asm-файле. Функция должна принимать 3 параметра: два указателя и длину строки. Про

расположение указателей в памяти и расстояние между ними заранее ничего не известно (первая строка может начинаться раньше второй или наоборот; строки могут перекрываться).

Подпрограммы должны соответствовать соглашению о вызовах языка Си и использовать команды обработки строк с префиксом повторения.

Лабораторная работа должна быть выполнена под 32- или 64-разрядной системой. Допускается использование Visual Studio, Code::Blocks или любой другой среды под Windows либо gcc/g++ под Linux.