

# Лабораторная работа №1

## Программирование на ассемблере (основы)

**Цель** – получить сведения об особенностях языка программирования, ознакомиться с инструментами разработки и их настройке.

## Ход работы

Язык ассемблера — это символическое представление машинного языка. Все процессы в персональном компьютере (ПК) на самом низком, аппаратном уровне приводятся в действие только командами (инструкциями) машинного языка. По-настоящему решить проблемы, связанные с аппаратурой (или даже, более того, зависящие от аппаратуры как, к примеру, повышение быстродействия программы), невозможно без знания ассемблера.

Ассемблер представляет собой удобную форму команд непосредственно для компонент ПК и требует знание свойств и возможностей интегральной микросхемы, содержащей эти компоненты, а именно микропроцессора ПК. Таким образом, язык ассемблера непосредственно связан с внутренней организацией ПК. И не случайно практически все компиляторы языков высокого уровня поддерживают выход на ассемблерный уровень программирования.

Элементом подготовки программиста-профессионала обязательно является изучение ассемблера. Это связано с тем, что программирование на ассемблере требует знание архитектуры ПК, что позволяет создавать более эффективные программы на других языках и объединять их с программами на ассемблере.

Само слово **ассемблер** (assembler) переводится с английского как «сборщик». На самом деле так называется программа-транслятор, принимающая на входе текст, содержащий условные обозначения машинных команд, удобные для человека, и переводящая эти обозначения в последовательность соответствующих кодов машинных команд, понятных процессору. В отличие от машинных команд, их условные обозначения, называемые также **мнемониками**, запомнить сравнительно легко, так как они представляют собой сокращения от английских слов. В дальнейшем мы будем для простоты именовать мнемоники ассемблерными командами. Язык условных обозначений и называется **языком ассемблера**.

На заре компьютерной эры первые ЭВМ занимали целые комнаты и весили не одну тонну, имея объем памяти с воробьиный мозг, а то и того меньше. Единственным способом программирования в те времена было вбивать программу в память компьютера непосредственно в цифровом виде, переключая тумблеры, проводки и кнопки. Число таких переключений могло достигать нескольких сотен и росло по мере усложнения программ. Встал вопрос об экономии времени и денег. Поэтому следующим шагом в развитии стало появление в конце сороковых годов прошлого века первого транслятора-ассемблера, позволяющего удобно и просто писать машинные команды на человеческом языке и в результате автоматизировать весь процесс программирования, упростить, ускорить разработку программ и их отладку. Затем появились языки высокого уровня и **компиляторы** (более интеллектуальные генераторы кода с более понятного человеку языка) и **интерпретаторы** (исполнители написанной человеком программы на лету). Они совершенствовались, совершенствовались — и, наконец, дошло до того, что можно просто программировать мышкой.

Таким образом, ассемблер — это **машинно ориентированный язык** программирования, позволяющий работать с компьютером напрямую, один на один. Отсюда и его полная формулировка — язык программирования низкого уровня второго поколения (после машинного кода). Команды ассемблера один в один соответствуют командам процессора, но поскольку существуют различные модели процессоров со своим собственным набором команд, то, соответственно, существуют и разновидности, или диалекты, языка ассемблера. Поэтому использование термина «язык ассемблера» может вызвать ошибочное мнение о существовании единого языка низкого уровня или хотя бы стандарта на такие языки. Его не существует. Поэтому при именовании языка, на котором написана конкретная программа, необходимо уточнять, для какой архитектуры она предназначена и на каком диалекте языка написана. Поскольку ассемблер привязан к устройству процессора, а тип процессора жестко определяет набор доступных команд машинного языка, то программы на ассемблере не переносимы на иную компьютерную архитектуру.

Поскольку ассемблер всего лишь программа, написанная человеком, ничто не мешает другому программисту написать свой собственный ассемблер, что часто и происходит. На самом деле не так уж важно, язык какого именно ассемблера изучать. Главное — понять сам принцип работы на уровне команд процессора, и тогда не составит труда освоить не только другой ассемблер, но и любой другой процессор со своим набором команд.

## Какие разновидности ассемблеров нашли применение?

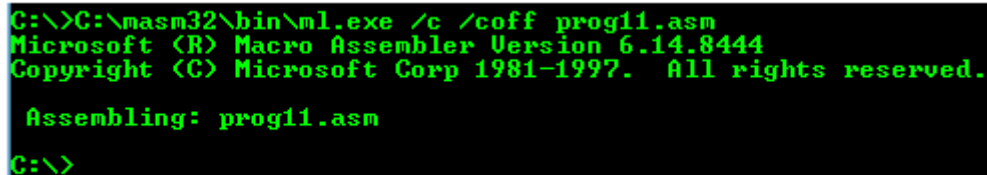
### MASM

Используется для создания драйверов под Windows. По [ссылке](http://www.masm32.com/) ( <http://www.masm32.com/> ) переходим на сайт и скачиваем пакет (**masm32v11r.zip**). После инсталляции программы на диске создается папка с нашим пакетом Диск и Директория установки \masm32. Создадим программу prog11.asm, которая ничего не делает.

```
.586P
.model flat, stdcall
_data segment
_data ends
_text segment
start:
ret
_text ends
end start
```

**(проделайте этот шаг!)**

Произведём ассемблирование (трансляцию) файла prog11.asm, используя установленный ассемблер.



```
C:\>C:\masm32\bin\ml.exe /c /coff prog11.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

Assembling: prog11.asm
C:\>
```

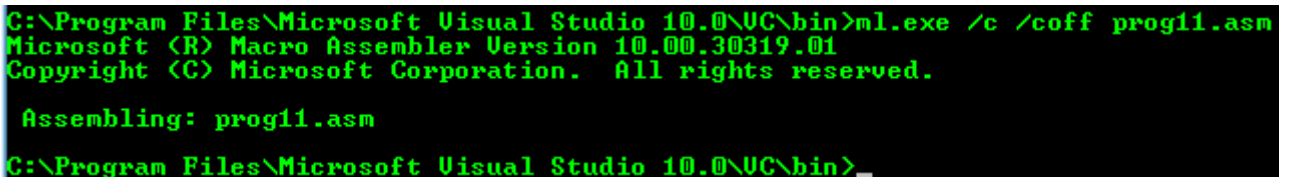
Ключ /coff используется здесь для трансляции 32-битных программ. Линковка производится командой `link /subsystem:windows prog11.obj` (`link /subsystem:console prog11.obj`)

Как [сказано](#) в Википедии MASM — один из немногих инструментов разработки Microsoft, для которых не было отдельных 16- и 32-битных версий.

---

### MASM в Visual Studio

Также MASM можно найти в папке с Visual Studio (для VS 10) вот здесь: **C:\Program Files\Microsoft Visual Studio 10.0\VC\bin\ml.exe**. (папка выделенная жирным зависит от версии Visual Studio и места ее установки).



```
C:\Program Files\Microsoft Visual Studio 10.0\VC\bin>ml.exe /c /coff prog11.asm
Microsoft (R) Macro Assembler Version 10.00.30319.01
Copyright (C) Microsoft Corporation. All rights reserved.

Assembling: prog11.asm
C:\Program Files\Microsoft Visual Studio 10.0\VC\bin>_
```

Для того, чтобы запустить на 32- или 64-разрядной системе и создавать программы, работающие как под 32-, так и под 64-разрядной Windows, подходит MASM32 (ml.exe, fl\_ml.exe). Для того, чтобы работать на 32- и 64-разрядных системах и создавать программы, работающие под 64-разрядной Windows, но не работающие под 32-разрядной нужен ассемблер ml64.exe. Лежит в папке C:\Program Files\Microsoft Visual Studio

10.0\VC\bin\amd64 и вот здесь — C:\Program Files\Microsoft Visual Studio 10.0\VC\bin\x86\_amd64.

## TASM

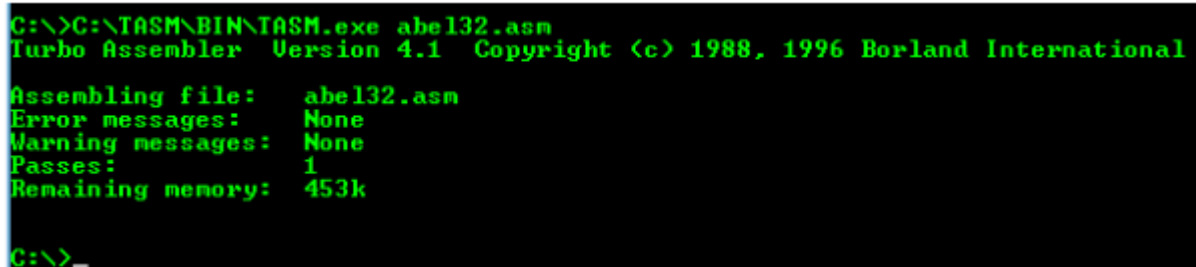
Программный пакет компании Borland, предназначенный для разработки программ на языке ассемблера для архитектуры x86. В настоящее время Borland прекратила распространение своего ассемблера.

**(проделайте этот шаг!)**

Скачать можно, например, [здесь](http://stilus-doctus.narod.ru/tasm/tasm_2.html) ([http://stilus-doctus.narod.ru/tasm/tasm\\_2.html](http://stilus-doctus.narod.ru/tasm/tasm_2.html)). Инсталлятора нет, просто извлекаем программу. Вот исходник из книги Питера Абея «Язык Ассемблера для IBM PC и программирования».

```
stacksg segment para stack 'stack'
    db 12 dup ('stackseg')
stacksg ends
codesg segment para 'code'
begin proc far
    assume ss:stacksg, cs:codesg, ds:nothing
    push ds
    sub ax, ax
    push ax
    mov ax, 0123h
    add ax, 0025h
    mov bx, ax
    add bx, ax
    mov cx, bx
    sub cx, ax
    sub ax, ax
    nop
    ret
begin endp
codesg ends
end begin
```

Выполним ассемблирование (трансляцию) файла abel32.asm.



```
C:\>C:\TASM\BIN\TASM.exe abel32.asm
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

Assembling file:    abel32.asm
Error messages:     None
Warning messages:   None
Passes:             1
Remaining memory:   453k

C:\>
```

Корректность работы программы можно проверить, произведя линковку (tlink.exe) объектного файла и запустив полученный файл в отладчике. Как было сказано выше, MASM можно использовать для работы с 16-битными программами. Выполним ассемблирование (трансляцию) программы abel32.asm с помощью ассемблера MASM:

```

C:\>C:\masm32\bin\nl.exe /c abel32.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

Assembling: abel32.asm
C:\>

```

Ключ */coff* здесь не используется. Линковка производится файлом *link16.exe*

## FASM

В статье Криса Касперски «Сравнение ассемблерных трансляторов» написано, что «FASM — неординарный и весьма самобытный, но увы, игрушечный ассемблер. Пригоден для мелких задач типа „hello, world“, вирусов, демоков и прочих произведений хакерского творчества.»

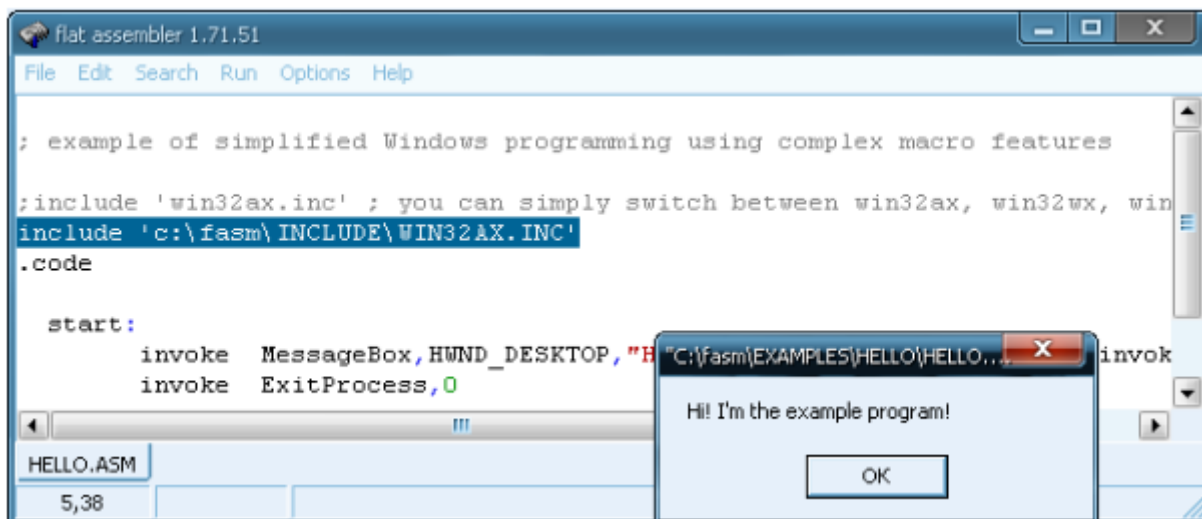
Скачать FASM можно с официального [сайта](http://flatassembler.net/) (<http://flatassembler.net/>). Инсталлятора нет, просто извлекаем программу. Откроем fasm editor — C:\fasm\fasmw.exe. В папке C:\fasm\EXAMPLES\HELLO есть файл HELLO.asm.

```

include 'win32ax.inc'
.code
start:
    invoke     MessageBox,HWND_DESKTOP,"Hi! I'm the example program!",invoke
GetCommandLine,MB_OK
    invoke     ExitProcess,0
.end start

```

Откроем файл HELLO.asm из fasmw.exe. Изменим строку `include 'win32ax.inc'` на строку `include 'c:\fasm\INCLUDE\WIN32AX.INC'`. Запускаем из меню Run → Run.



Вот ссылки на ресурсы, посвященные FASM:

- [FASM на Cyberforum'e](https://www.cyberforum.ru/fasm/) (<https://www.cyberforum.ru/fasm/>)
- [Цикл статей «Ассемблер под Windows для чайников»](https://nestor.minsk.by/kg/abc/) (<https://nestor.minsk.by/kg/abc/>)
- [Сайт на narod'e](http://flatassembler.narod.ru/fasm.htm) (<http://flatassembler.narod.ru/fasm.htm>)

## FASM в Linux

Для того, использовать FASM в Linux, необходимо скачать соответствующий дистрибутив (fasm-1.71.60.tgz), распаковать его, в папку. Бинарный файл fasm, копируем этот файл в /usr/local/bin для того, чтобы можно было запускать его из консоли, как любую другую команду. Выполняется ассемблирование программы например hello.asm также, как и в версиях под Windows, из папки fasm/examples/elfexe/hello.asm.

```
~$ cd fasm/examples/elfexe
~/fasm/examples/elfexe$ fasm hello.asm
flat assembler version 1.71.59 (16384 kilobytes memory)
3 passes, 160 bytes.
~/fasm/examples/elfexe$
```

---

## NASM для Windows

NASM для Windows можно установить, скачав соответствующий дистрибутив с [соответствующего сайта](https://www.nasm.us/) (https://www.nasm.us/).

Ассемблирование:

```
nasm -f bin имя_файла.asm -o имя_файла.com
```

Ссылки на ресурсы, посвященные Nasm:

- [Сайт А.В. Столярова](http://www.stolyarov.info/books/asm_unix) (http://www.stolyarov.info/books/asm\_unix)
- [Сайт, на котором лежит электронный учебник](#) (в архиве)
- [То же самое](http://www.opennet.ru/docs/RUS/nasm/) (http://www.opennet.ru/docs/RUS/nasm/)

---

## AS

Стандартный ассемблер практически во всех разновидностях UNIX, в том числе Linux и BSD. Свободная версия этого ассемблера называется GAS (GNU assembler). Позволяет транслировать программы с помощью компилятора GCC.

Из учебников удалось найти только книгу на английском «Programming from the ground up». На русском удалось найти только одну главу из книги С. Зубкова «Assembler для DOS, Windows и UNIX».

Возьмем пример программы, которая ничего не делает, с [сайта](#). Создадим программу gas.s

```
.section .text
.globl _start
_start:
    movl $1, %eax
    movl $2, %ebx
    int $0x80
```

Ассемблирование (трансляцию), линковку и запуск программы выполняется следующим образом:

```
$ as -o gas.o gas.s
$ ld -o gas gas.o
$ ./gas
```

Если в данной программе изменить `_start` на `main`, то можно выполнить ассемблирование (трансляцию) и линковку компилятором `gcc`.

```
.section .text
.globl main
main:
    movl    $1, %eax
    movl    $2, %ebx
    int     $0x80
```

Ассемблирование (трансляцию), линковку и запуск программы выполняется следующим образом:

```
$ gcc gas.s -o gas
$ ./gas
```

---

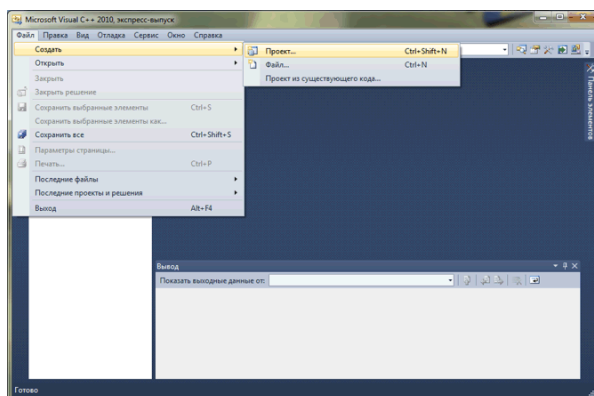
## Работа с MASM в Visual studio

(Выполните полностью!)

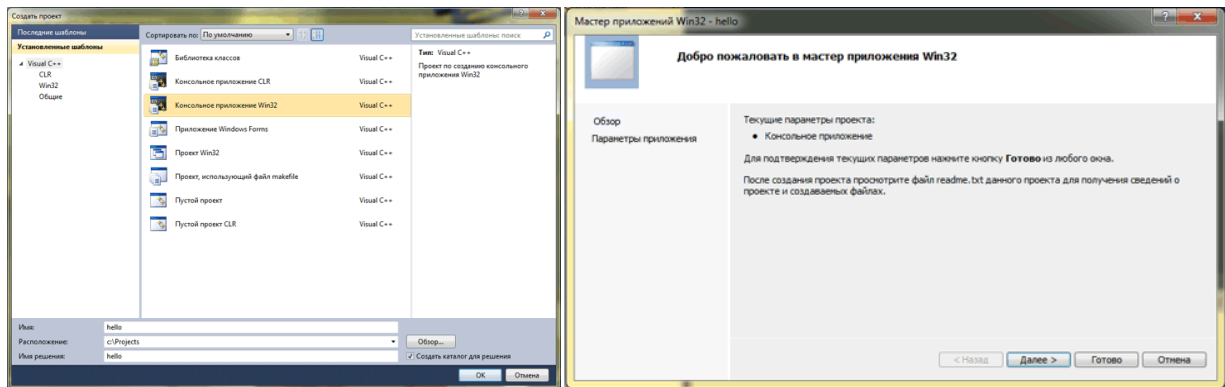
Создание проекта консольного или оконного Windows-приложения не отличается от рассмотренного для языков программирования Си и C++. Для создания Windows-приложений на C++ можно использовать среду разработки Microsoft Visual Studio 2010 Express с пакетом обновления SP1 или любую другую ее версию.

Описание действий приведено для VS 2010, для других версий пункты меню и внешний облик диалоговых окон может несколько отличаться (самостоятельно найдите этот пункт если используете новую версию VS).

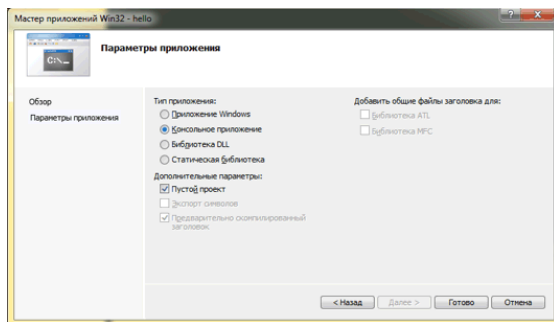
Для создания нового консольного приложения запускаем Microsoft Visual Studio 2010 Express и переходим в меню **Файл->Создать->Проект**



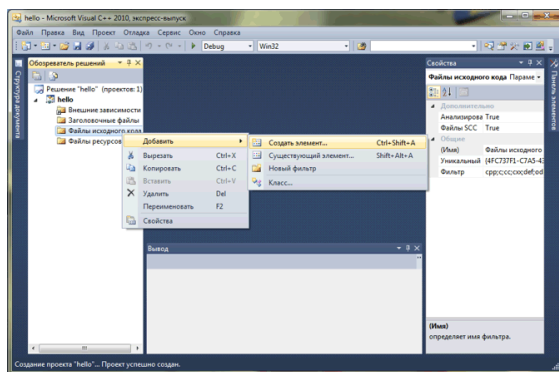
В появившемся окне выбираем **Консольное приложение Win32** и задаем имя проекта и нажимаем кнопку **ОК**.



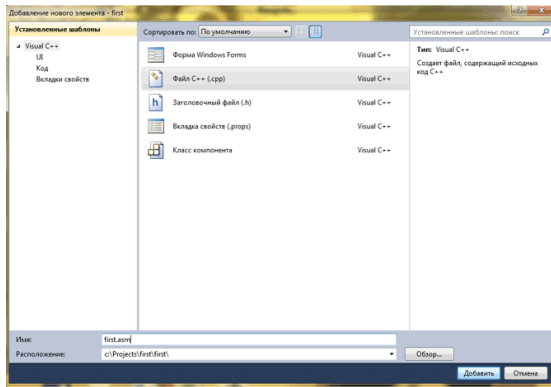
В появившемся окне нажимаем кнопку **Далее**. В следующем окне отмечаем галочку **Дополнительные параметры: Пустой проект** и нажимаем кнопку **Далее**.



После того, как в Visual Studio появилось окно проекта (в левой части появившегося окна отображается **Обозреватель решений**), для добавления нового файла программы в проект выбираем по правой кнопке мыши на папке **Файлы исходного кода** меню **Добавить->Создать элемент**.

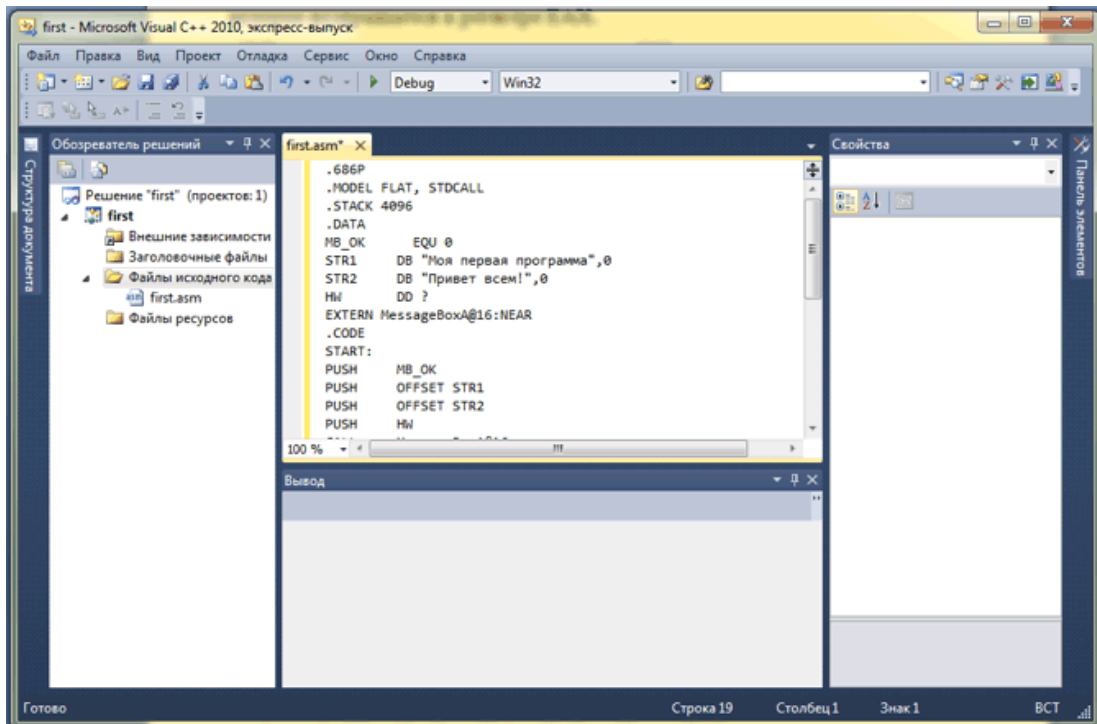


В появившемся окне выбираем **Файл C++ (.cpp)**, задаем имя файла и вручную добавляем к нему расширение **asm**. Нажимаем кнопку **Добавить**.



В появившемся окне набираем текст программы. В качестве примера можно использовать следующий текст:

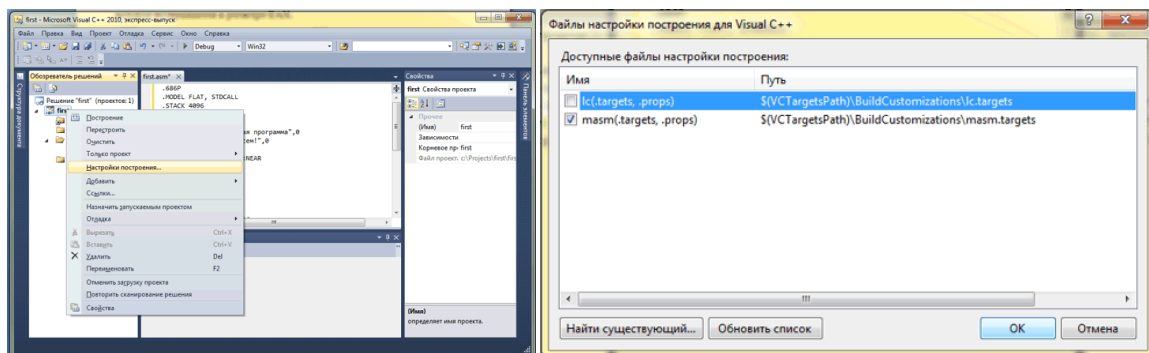
```
.686P
.MODEL FLAT, STDCALL
.STACK 4096
.DATA
MB_OK EQU 0
STR1 DB "Моя первая программа",0
STR2 DB "Привет всем!",0
HW DD ?
EXTERN MessageBoxA@16:NEAR
.CODE
START:
PUSH MB_OK
PUSH OFFSET STR1
PUSH OFFSET STR2
PUSH HW
CALL MessageBoxA@16
RET
END START
```



Далее необходимо сообщить среде разработки, что данный файл является программой на языке ассемблера, и для корректного включения его в проект требуется использовать

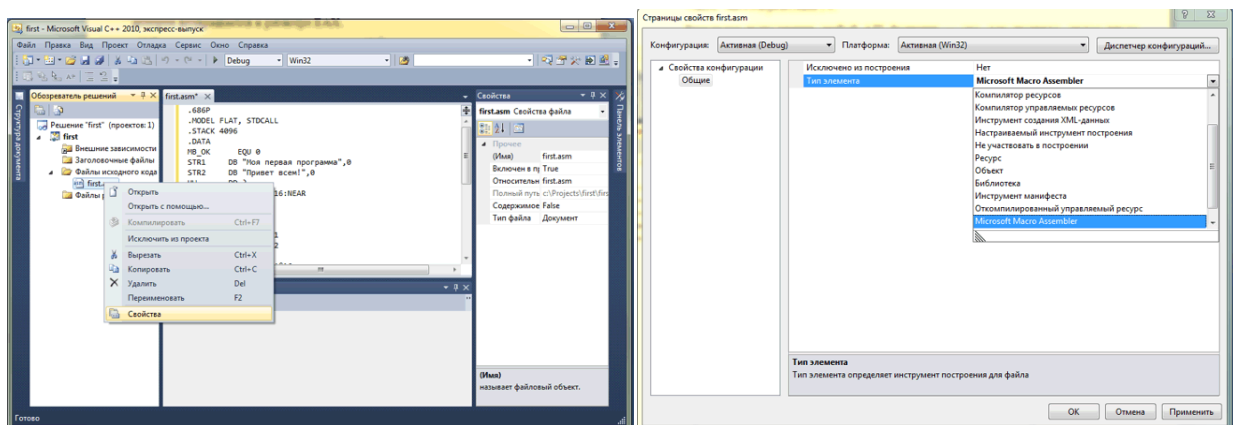


Microsoft Macro Assembler. Для этого выбираем для проекта (по правой клавише мыши) опцию **Настройки построения**.



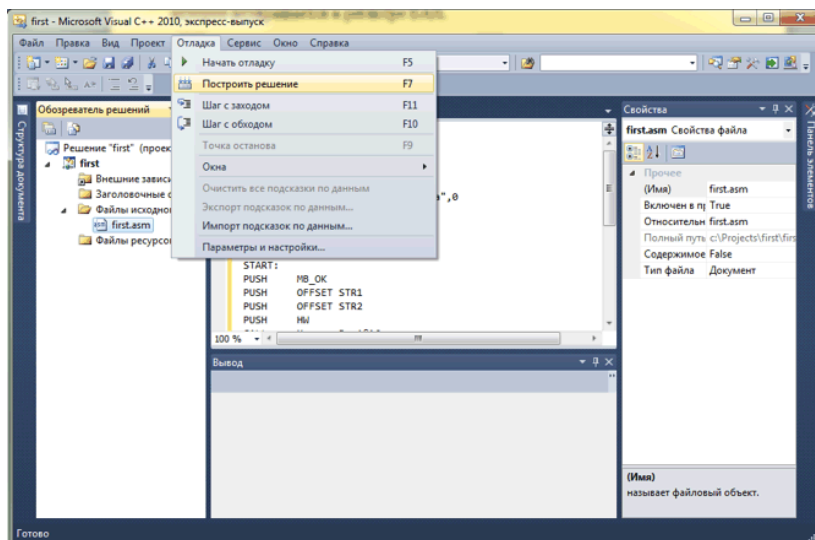
В появившемся окне ставим галочку для **masm** (Microsoft Macro Assembler) и нажимаем **ОК**.

Теперь нужно проверить, что для файла на языке ассемблера установлен соответствующий инструмент сборки. По правой кнопке мыши для файла с расширением **.asm** выбираем опцию **Свойства**.

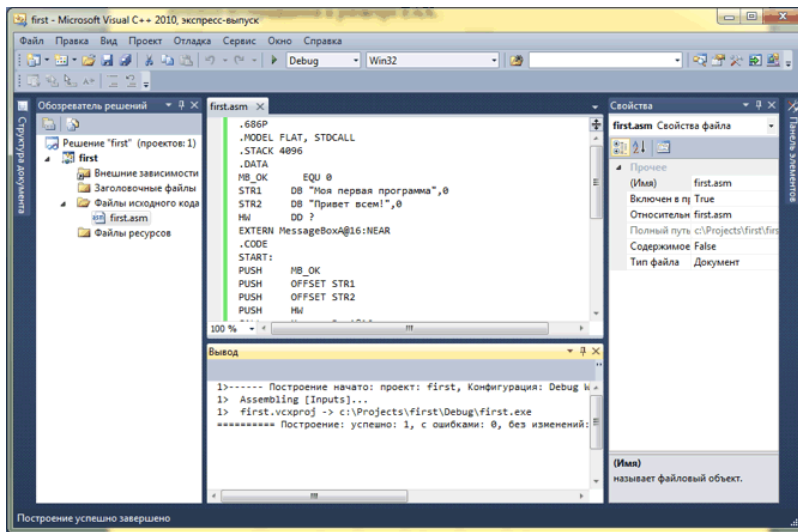


В появившемся окне для выбранного файла отмечаем инструмент сборки **Microsoft Macro Assembler**.

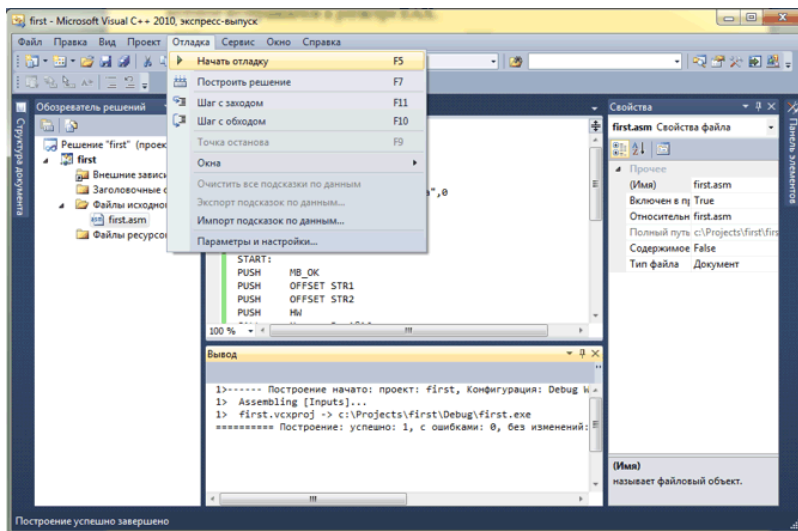
Для построения проекта выбираем меню **Отладка->Построить решение**.



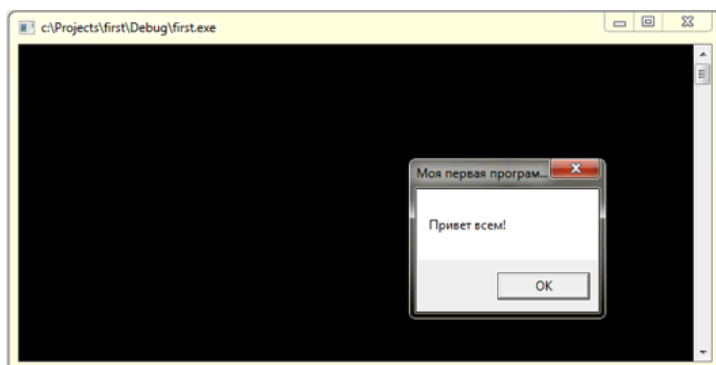
В случае успешного построения в нижней части окна отображается **Построение: успешно** 1.



Для запуска приложения выбираем меню **Отладка->Начать отладку**.

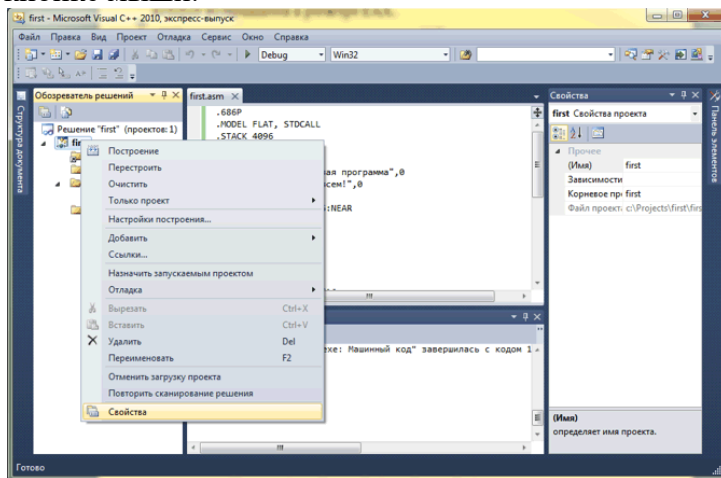


Результат выполнения программы:

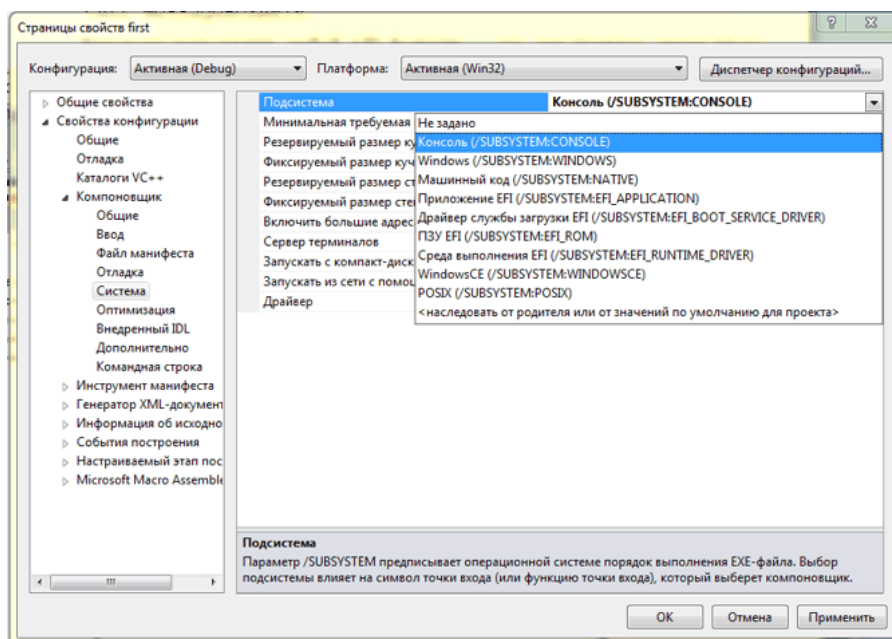


## Изменить тип приложения с консольного на оконное

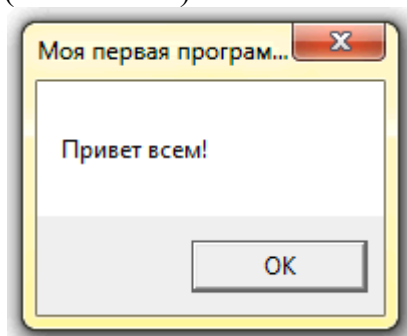
Чтобы убрать консоль (поменять тип приложения с консольного на оконное, или наоборот) необходимо обратиться к меню Свойства проекта, вызванного по правой кнопке мыши.



В появившемся окне выбрать раздел **Компоновщик->Система**, и в разделе **Подсистема** поменять тип с **Консоль** на **Windows** (или наоборот).



Повторная сборка и запуск программы на выполнения выдадут следующий результат (консоли нет):



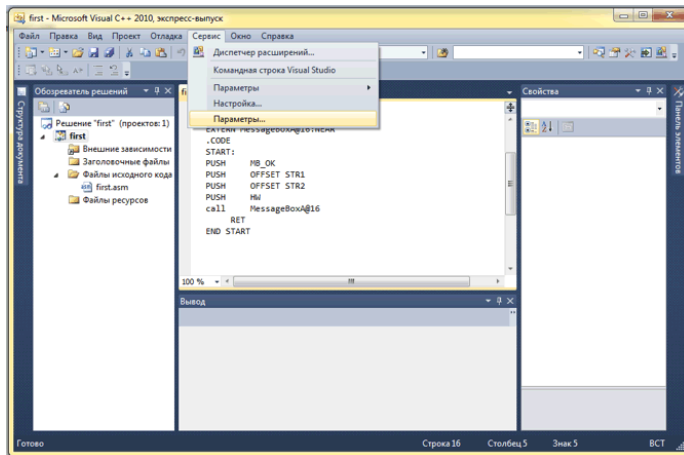
## Подсветка синтаксиса языка ассемблера

(Этот шаг для информации, выполнять не обязательно!)

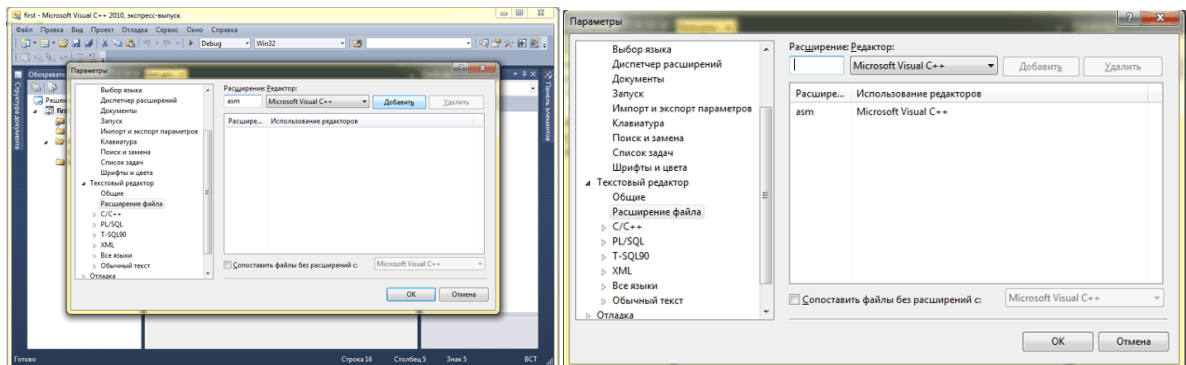
Для того, чтобы включить подсветку синтаксиса языка ассемблера в Microsoft Visual Studio Express 2010 необходимо загрузить файл [usertype](#) и распаковать его в папку

**C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE**

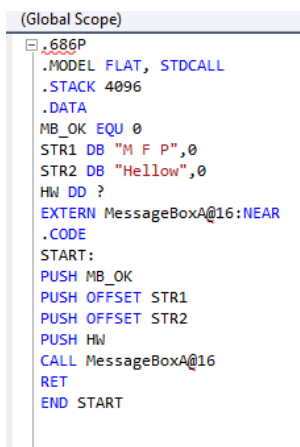
Для подключения подсветки синтаксиса выбираем меню **Сервис->Параметры**



В появившемся окне выбрать **Текстовый редактор->Расширение файла** и вручную добавляем расширение **asm**. Нажимаем кнопку **Добавить**, затем — **ОК**.



После перезапуска Microsoft Visual Studio Express 2010 подсветка синтаксиса языка ассемблера будет активна.



## Использование дизассемблера

Дизассемблер выполняет обратное преобразование исполняемого (машинного) кода в текст программы на языке ассемблера. Самый простой дизассемблер под windows (W32DSM89).

Используйте его и дизассемблируйте все скомпилированные вами исполняемые модули.

Определите соответствие исходного текста – тому что выдал дизассемблер.

Пример для проекта откомпилированного в Visual studio.

The screenshot displays the W32DSM89 disassembler interface. The main window shows the disassembly of a file named 'D:\vs\sss\as5\Debug\as5.exe'. The code listing includes menu information, dialog information, imported functions, and assembly code. A specific line of assembly code is highlighted in blue: `:00401007 681D404000 push 0040401D`. To the right, a 'Data Object Size = 512 Bytes, Page 1 of 1' window is open, showing a hex display of data. The data is organized into columns of hex values and their corresponding ASCII string representations. The strings include 'My First', 'assembl', 'er progr', 'amm! .Hel', 'lo All!', and several null-terminated strings. Below the hex display, a 'W32Dasm List of String Data Items' window is open, showing a list of string data items, including 'Hello All!'.

Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

Disassembly of File: D:\vs\sss\as5\Debug\as5.exe  
Code Offset = 00000400, Code Size = 00001200  
Data Offset = 00001A00, Data Size = 00000200

Number of Objects = 0006 (dec), Imagebase = 00400000h

Object01: .text RVA: 00001000 Offset: 00000400 Size: 0000  
Object02: .rdata RVA: 00003000 Offset: 00001600 Size: 0000  
Object03: .data RVA: 00004000 Offset: 00001A00 Size: 0000  
Object04: .idata RVA: 00005000 Offset: 00001C00 Size: 0000  
Object05: .rsrc RVA: 00006000 Offset: 00001E00 Size: 0000  
Object06: .reloc RVA: 00007000 Offset: 00002400 Size: 0000

+++++ MENU INFORMATION +++++  
There Are No Menu Resources in This Application

+++++ DIALOG INFORMATION +++++  
There Are No Dialog Resources in This Application

+++++ IMPORTED FUNCTIONS +++++  
Number of Imported Modules = 1 (decimal)  
Import Module 001: USER32.dll

+++++ IMPORT MODULE DETAILS +++++  
Import Module 001: USER32.dll  
Addr:00005088 hint(020E) Name: MessageBoxA

+++++ EXPORTED FUNCTIONS +++++  
Number of Exported Functions = 0000 (decimal)

+++++ ASSEMBLY CODE LISTING +++++  
//\*\*\*\*\* Start of Code in Object .text \*\*\*\*\*  
Program Entry Point = 00401000 (D:\vs\sss\as5\Debug\as5.exe File Offset:00002600)

//\*\*\*\*\* Program Entry Point \*\*\*\*\*  
:00401000 6A00 push 00000000  
:00401002 6800404000 push 00404000

\* Possible StringData Ref from Data Obj ->"Hello All!"  
:00401007 681D404000 push 0040401D  
:0040100C FF3528404000 push dword ptr [00404028]

\* Reference To: USER32.MessageBoxA, Ord:020Eh  
:00401012 E807000000 Call 0040101E  
:00401017 C3 ret

:00401018 CC int 03  
:00401019 CC int 03  
:0040101A CC int 03  
:0040101B CC int 03  
:0040101C CC int 03  
:0040101D CC int 03

\* Reference To: USER32.MessageBoxA, Ord:020Eh  
:0040101E FF2558504000 jmp dword ptr [00405058]  
:00401024 CC int 03  
:00401025 CC int 03

Data Object Size = 512 Bytes, Page 1 of 1  
Close Hex Display of Data Object Copy All Copy View

:00404000 4D 79 20 46 E9 72 73 74 My First  
:00404008 20 61 73 73 E5 6D 62 6C assembl  
:00404010 65 72 20 70 72 6F 67 72 er progr  
:00404018 61 6D 6D 21 00 48 65 6C amm! .Hel  
:00404020 6C 6F 20 41 6C 6C 21 00 lo All!  
:00404028 00 00 00 00 00 00 00 00 .....  
:00404030 00 00 00 00 00 00 00 00 .....  
:00404038 00 00 00 00 00 00 00 00 .....

W32Dasm List of String Data Items  
To Search Disassembly for String Data, Double Click on Text Cancel S

"Hello All!"  
Close Copy All C