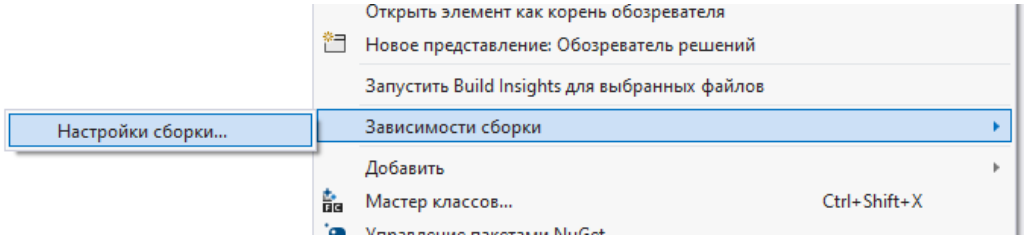
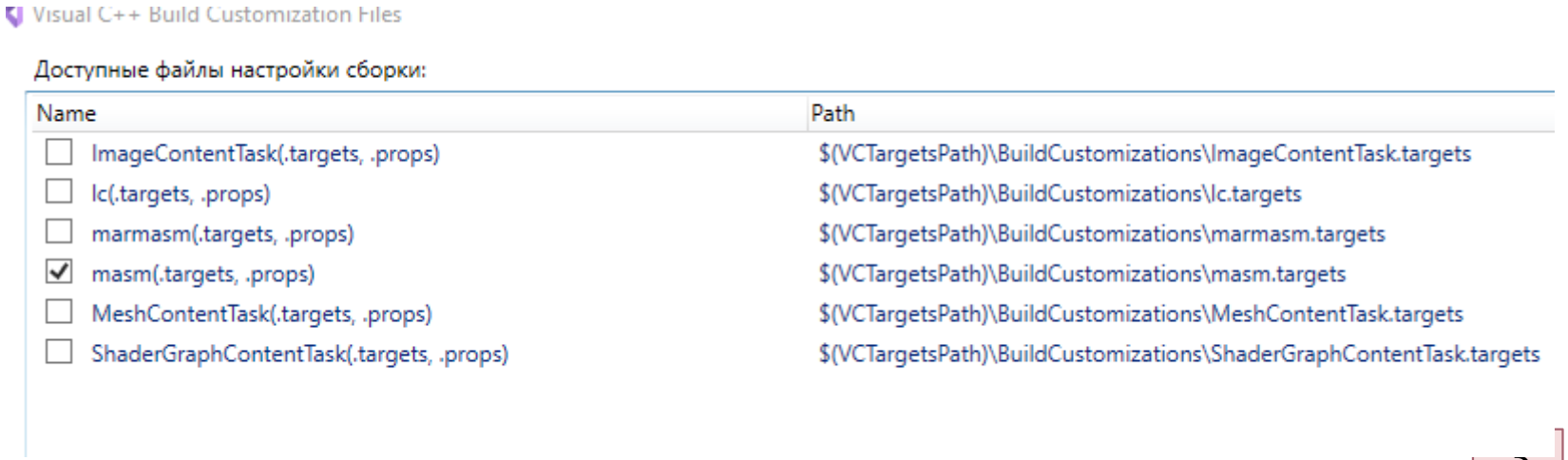
Модули

Модуль

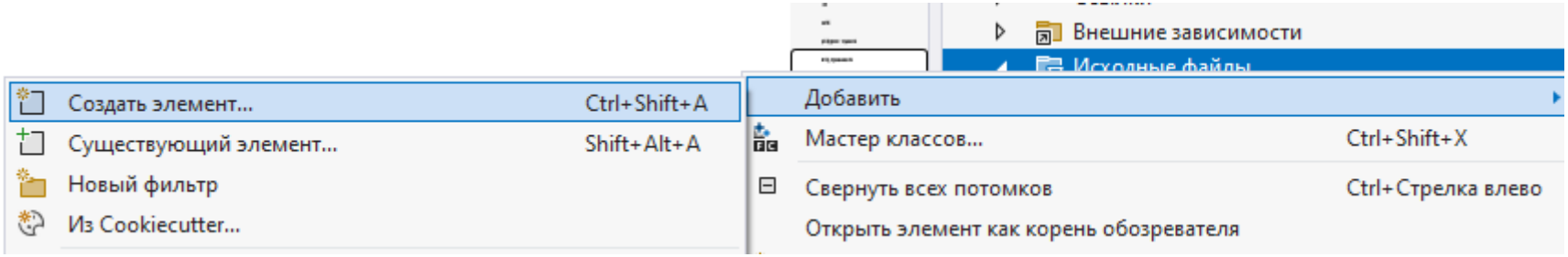
Модульное программирование — организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам.

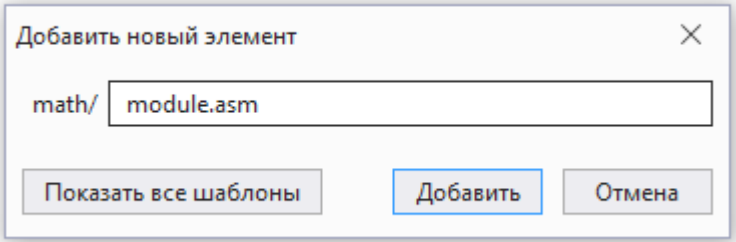
Создание ассемблерного модуля



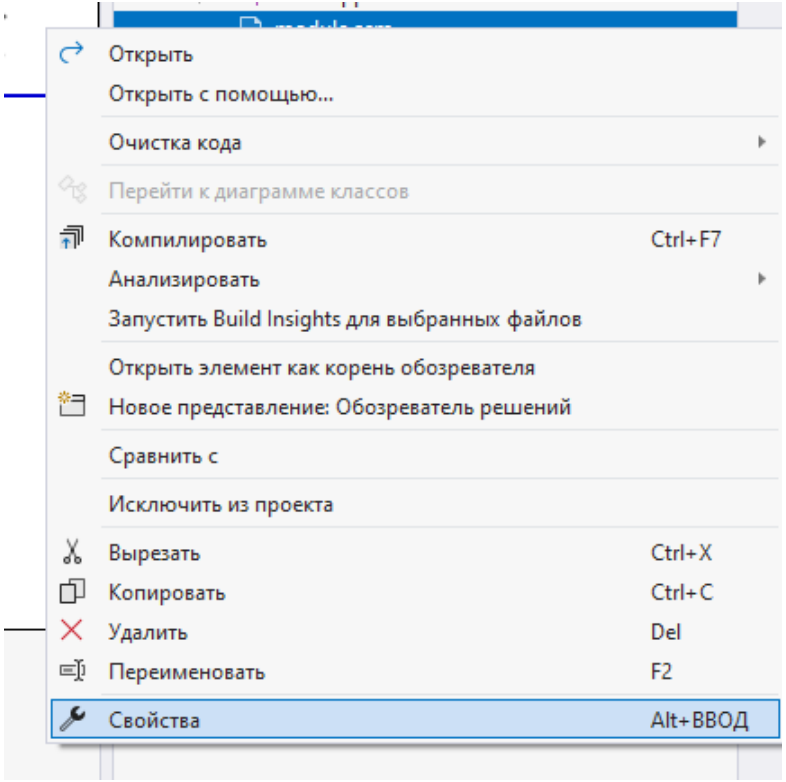


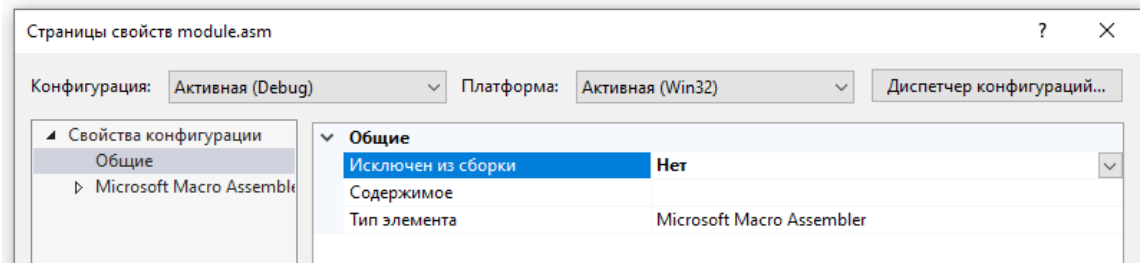
Создание ассемблерного модуля





Создание ассемблерного модуля





Создание ассемблерного модуля

- Разрешить masm
- Добавить файл .asm
- Тип файла Microsoft Assembler

module.asm

```
.686P
.MODEL FLAT, C

.DATA
    EXTERN myprint:NEAR
.CODE

some PROC
    push ebp;пролог - сохранение ебп
    mov ebp, esp;инициализация ебп
    mov ecx, [ebp + 12];доступ к первому аргументу
    mov eax, [ebp +8];доступ ко второму аргументу
    sub ecx, 1
    mov ebx, eax
    jcxz EXIT
    SYCLE:
        mul ebx
    loop SYCLE
    EXIT:
    push eax; добавление результата в стек
    call myprint; вызов функции вывода
    pop eax; очистка стека
    ret 8;восстановление контекста программы 4 * кол-во аргументов
some ENDP

END
```

math.cpp | module.asm

math

```cpp
// math.cpp : Этот файл содержит
//

#include <iostream>
using namespace std;
extern "C" {
    int some(int a, int b);

    void myprint(int a)
    {
        cout << a;
    }
}

int main()
{
    cout << some(5, 3);
}
```

Структура ассемблерного модуля

Типичная программа на MASM содержит одну или несколько секций, которые определяют, как содержимое программы будет располагаться в памяти. Эти секции начинаются с таких директив MASM, как .code или .data. Данные, используемые в программе, обычно определяются в секции .data. Инструкции ассембра определяются в секции .code.

Структура ассемблерного модуля

В общем случае программа на ассемблере MASM имеет следующий вид:

```
.data
.code
main proc
 ;код
 ret
main endp
end
```

.code

Директива .code указывает MASM сгруппировать операторы, следующие за ней, в специальный раздел памяти, зарезервированный для машинных инструкций.

Ассемблер преобразует каждую машинную инструкцию в последовательность из одного или нескольких байт. CPU интерпретирует эти значения байт как машинные инструкции во время выполнения программы.

.code

Далее с помощью операторов main proc определяется процедура main. Операторы main endp указывают на конец функции main. Между main proc и main endp располагаются выполняемые инструкции ассемблера. Причем в самом конце функции идет инструкция ret, с помощью которой выполнение возвращается в окружение, в котором была вызвана данная процедура. В конце файла кода идет инструкция end

Структура ассемблерного модуля

Программа может содержать комментарии, которые располагаются после точки с запятой:

```
.code   ; начало секции с кодом программы

main proc   ; Функция main

    ret ; возвращаемся в вызывающий код

main endp ; окончание функции main

end ; конец файла кода
```

Структура ассемблерного модуля

При создании программы на ассемблере стоит понимать, что это не высокоуровневый язык. В ассемблере, чтобы выполнить довольно простые вещи, придется писать много инструкций. И здесь есть разные подходы: мы можем написать весь код только на ассемблере - вариант, который в реальности втречается редко, либо мы можем какие-то части писать на ассемблере, а какие-то на языке высокого уровня, например, на C++.

Пример

Рассмотрим следу

module.asm

```
.686P
.MODEL FLAT, C

.DATA
    EXTERN myprint:NEAR
.CODE

some PROC
    mov eax, 5
    mov ebx, 4
    add eax, ebx
    push eax
    call myprint
    pop eax

some ENDP

END
```

ConsoleApplication1.cpp

```
#include "stdafx.h"
#include <iostream>
using namespace std;
extern "C"{

    void print(int a){//функция для вывода числа на кносоль
        cout << a<<endl;
    }
    int some(int a, int b);//прототип функции
}


int _tmain(int argc, _TCHAR* argv[])
{
    cout << some(3, 5);

    return 0;
}
```

Пример

.686 – означает поддержку инструкций процессоров Intel Pentium Pro и выше (включает MMX)

Это набор инструкций 6-го поколения

Пример

.model flat

Определяет плоскую модель памяти (x32), где все сегменты используют одно и то же 32-битное адресное пространство.

Альтернативы (устар.): tiny, small, compact, medium, large, huge

Пример

,C:

Определяет соглашение о вызовах. C-style — аргументы передаются через стек справа налево, вызывающая сторона очищает стек.

Пример

.DATA:

Здесь мы подключаем функцию из C++ для вывода числа на консоль

В зависимости от используемой модели организации программы: tiny, small, large и т.д. компилятор считает, что у него по умолчанию все переходы/вызовы: либо ближние(near-один сегмент), либо дальние(far-разные сегменты).

Программист может явно указать, что нужно делать компилятору и у того не остается выбора.

Пример

.CODE

Здесь находится основная функция some, складывающая содержимое регистров eax и ebx. Результат сложения выводится на консоль посредством вызова внешней функции myprint.

Пример

END

Завершение программы

Пример

extern «C»{функции}:

Это указывает компилятору C++ не применять искажение имен к этим функциям. Использовать C-style соглашения о вызовах.

Пример 2

Изменим код программы таким образом, чтоб процедура принимала параметры при вызове из функции main.

Пример 2

```
.686P
.MODEL FLAT, C
.DATA
    EXTERN print:NEAR
.CODE
some PROC

    push ebp
    mov ebp, esp
    mov eax, [ebp + 12]; доступ к первому параметру
    mov ebx, [ebp + 8]; доступ ко второму параметру

    sub eax, ebx
    push eax
    call print
    pop eax

some ENDP
END
```

```
#include "stdafx.h"
#include <iostream>
using namespace std;
extern "C"{

    void print(int a){//функция для вывода числа на кно
        cout << a<<endl;
    }
    int some(int a, int b);//прототип функции
}


int _tmain(int argc, _TCHAR* argv[])
{
    cout << some(3, 5);

    return 0;
}
```

Пример 2

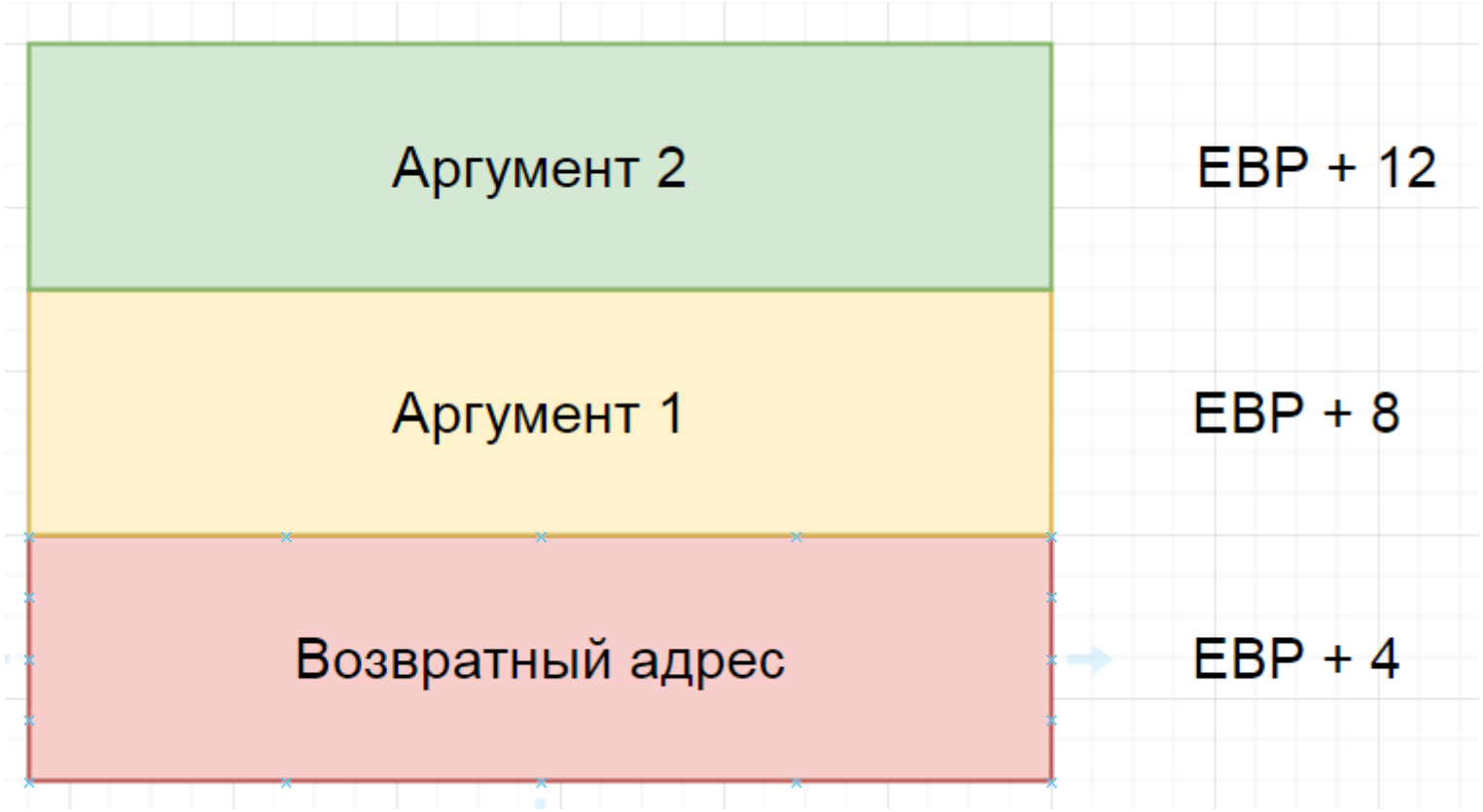
Инструкции push ebp и mov ebp, esp образуют стандартный пролог функции в ассемблере x86.

Пример 2

Это позволяет:

Легко обращаться к аргументам функции и локальным переменным, сохранять состояние стека перед выполнением функции, корректно вернуться из функции.

Пример 2



Задание

Написать функцию на языке ассемблера (MASM) для вычитания двух чисел, введенных пользователем. Вызвать функцию в коде на c++ и вывести результат в консоль.