

Тема 10

Связь Ассемблера и C++

© 2011-2018

Связь ассемблера с языками высокого уровня

Существуют следующие формы комбинирования программ на языках высокого уровня с ассемблером:

- использование ассемблерных вставок в виде встраиваемого ассемблерного кода;
- использование внешних процедур и функций. Это более универсальная форма комбинирования. У нее есть ряд преимуществ:
 - написание и отладку программ можно производить независимо;
 - написанные подпрограммы можно использовать в других проектах;
 - облегчаются модификация и сопровождение подпрограмм в течение жизненного цикла проекта.

Встраиваемый ассемблерный код

Можно:

- не передавать параметры, как в случае с внешней ассемблерной процедурой;
- иметь непосредственный доступ к командам и регистрам процессора;
- ссылаться на метки и переменные вне текущего блока, находящиеся в пределах видимости ассемблерной вставки;
- вызывать функции вне пределов ассемблерной вставки, причем эти функции должны быть ранее объявлены в программе (на уровне прототипа);
- использовать описание констант как в стиле Ассемблера, так и C++;
- использовать операторы **PTR**, **LENGTH**, **SIZE**, **TYPE** и директивы **EVEN** и **ALIGN**.

Встраиваемый ассемблерный код

Нельзя:

- использовать директивы определения данных простых (**DB** и **DD**) и сложных типов (**STRUC**, **RECORD**);
- описывать функции в пределах ассемблерной вставки;
- использовать в командах большинство операторов ассемблера типа **OFFSET**, **SEG** (вместо **OFFSET** нужно использовать **LEA**);
- использовать любые директивы макроопределений;
- обращаться к полям структур и объединений.

Внешний ассемблерный код. Порядок передачи параметров

Вопрос о порядке передачи параметров связан с тем, что в большинстве языков программирования высокого уровня используется понятие «список параметров» и сопоставление фактических и формальных параметров выполняется согласно месту в этом списке.

Очевидно, первый параметр из списка может быть занесен в стек первым (и тогда он окажется ниже всех остальных параметров) либо последним (и тогда он окажется наверху).

Назначение регистра ЕВР

Все соглашения по передаче параметров предполагают, что в регистре **ЕВР** сохраняется адрес вершины стека при начале работы процедуры. Для доступа к параметрам процедуры используется выражения вида **[ЕВР+n]** .

Однако при этом необходимо обеспечить сохранность этого регистра при цепочке вызовов. Это может быть достигнуто, например, включением в начало процедуры т.н. *пролога* из двух команд

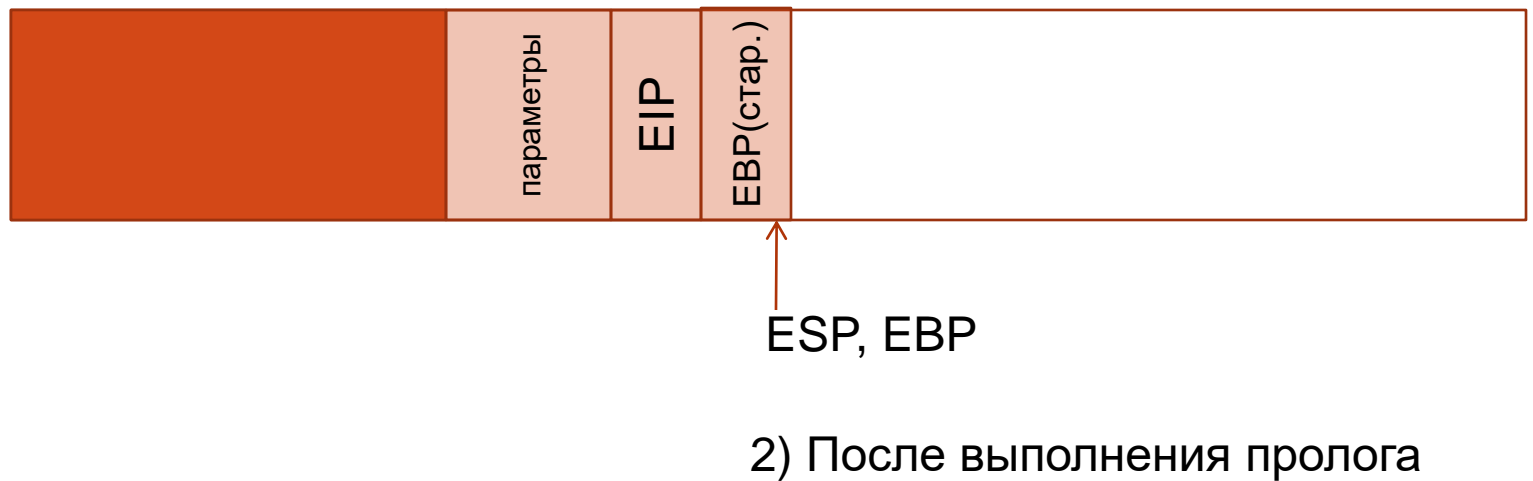
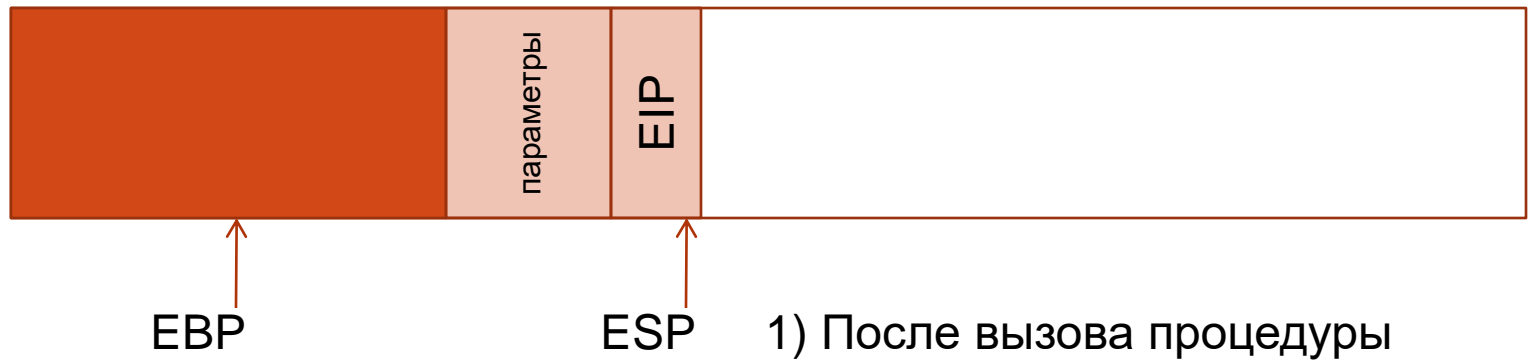
```
push ebp  
mov  ebp, esp
```

В конец процедуры рекомендуется добавлять *эпилог*, восстанавливающий содержимое **ЕВР**:

```
mov  esp, ebp  
pop  ebp
```

Первая команда эпилога очищает «мусор», который мог остаться в стеке!

Сохранение значения EBP в стеке



Соглашения по передаче параметров

Ассемблер допускает различную комбинацию механизмов передачи параметров, формируя т.н. *соглашения по передаче параметров*.

Наиболее часто используются следующие соглашения:

- - первый параметр заносится в стек первым, очистка стека выполняется процедурой (соглашения **PASCAL**);
- - первый параметр заносится в стек последним, очистка стека выполняется головной программой (соглашения **CDECL**);
- - первый параметр заносится в стек последним, очистка стека выполняется процедурой (соглашения **STDCALL**).

Соглашения по передаче параметров

Мы можем явно указать используемые соглашения, указав параметр **PASCAL**, **C** (**CPP**, **CDECL**) или **STDCALL** либо в директиве **.MODEL**, либо в директиве **PROC**, например:

```
.model    small, pascal
```

или

```
my_proc  proc c
```

Параметры директивы PROC

В директиве **PROC** можно задать:

- тип соглашения о связях (**C**, **STDCALL**, **PASCAL**);
- описания формальных параметров в виде
имя : длина

При описании формальных параметров пролог, эпилог, а также форма команды **RET** формируются АВТОМАТИЧЕСКИ!

Пример:

Процедура

```
s2w proc Pascal, StringAddr:word, StringLen:byte  
; передавать выходные параметры нельзя!
```

...

Вызов

```
call s2w, offset s1, 6
```

Использование регистра EBP для доступа к параметрам

Для доступа к первому параметру (в соглашениях **CDECL** и **STDCALL**) необходимо задать адрес **[EBP+8]**.

Для доступа к последующим параметрам в смещении относительно **EBP** должен быть учтен размер всех предыдущих параметров.

Пример использования регистра ЕВР для доступа к параметрам

Задача: написать функцию, получающую два целочисленных параметра a и b длиной 4 байта каждый и возвращающую значение $2a+b$. Соглашения о связях – **STDCALL**.

```
SumAB      proc
            push ebp
            mov  ebp, esp
            mov  eax, dword ptr[ebp+8]  ; a
            shl  eax, 1
            add  eax, [ebp+12]          ; b
            ; dword ptr можно не писать
            mov  esp, ebp
            pop  ebp
            ret  8
SumAB      endp
```

Пример использования формальных параметров

Задача: написать функцию, получающую два целочисленных параметра a и b длиной 4 байта каждый и возвращающую значение $2a+b$. Соглашения о связях – **STDCALL**.

```
SumAB      proc stdcall, a:dword, b:dword
            mov     eax, dword ptr a
            shl     eax, 1
            add     eax, b
            ; dword ptr можно не писать
            ret
SumAB      endp
```

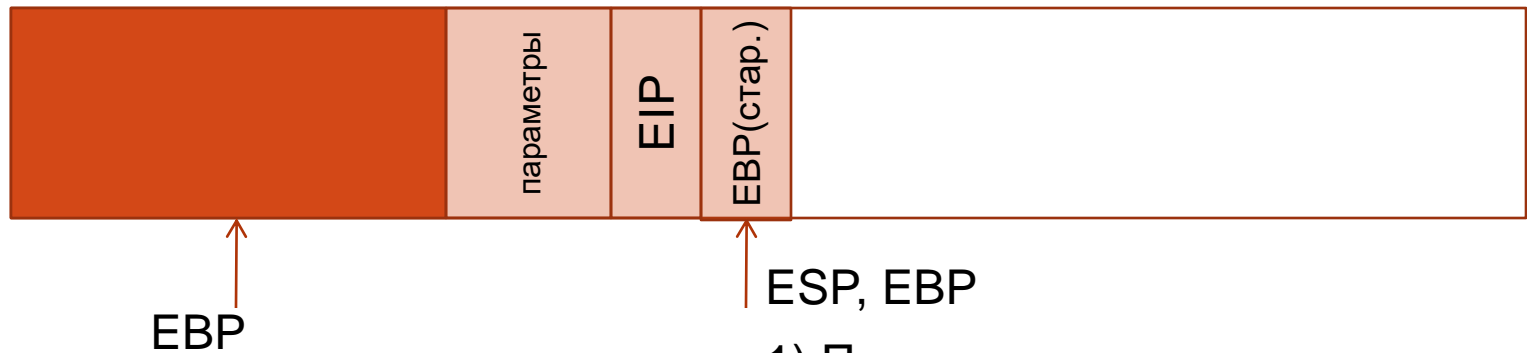
Использование локальных переменных в процедуре

Директива **LOCAL**, которая следует сразу за директивой **PROC**, позволяет описать локальные переменные в виде

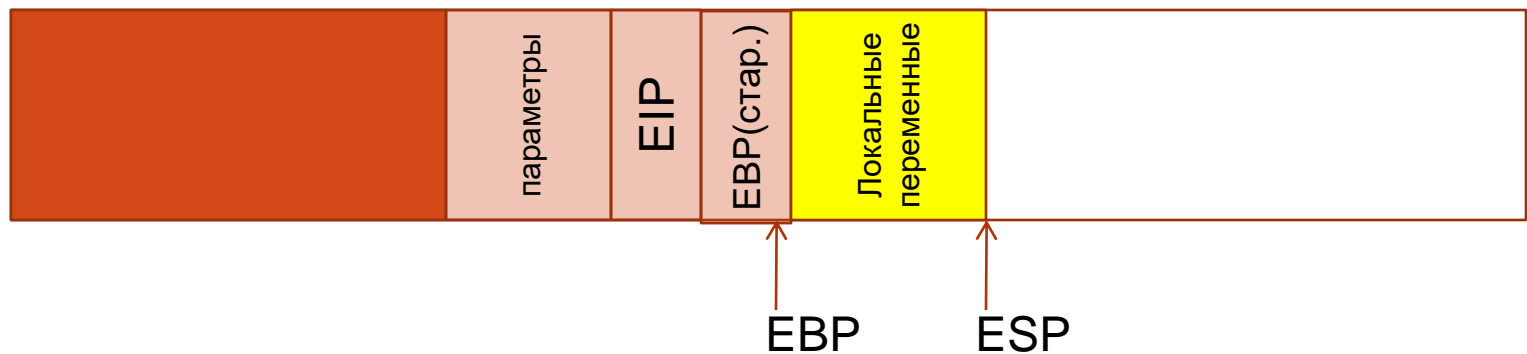
имя : длина

Пролог и эпилог в этом случае формируются автоматически!

Выделение в стеке места для локальных переменных



1) После вызова процедуры



2) После выделения памяти под локальные переменные

Пример использования локальных переменных

Задача: написать функцию, получающую два целочисленных параметра a и b длиной 4 байта каждый и возвращающую значение $3a+4b$. Никакие регистры, кроме **EAX**, использовать нельзя. Соглашения о связях – **STDCALL**.

```
SumAB    proc stdcall, a:dword, b:dword
          local tmp:dword
          mov     eax, a
          shl     eax, 2
          sub     eax, a
          mov     tmp, eax
          mov     eax, b
          shl     eax, 2
          add     eax, tmp
          ret
SumAB    endp
```


Внешний ассемблерный код

Основные правила:

- Всегда нужно сохранять (и перед выходом из процедуры восстанавливать) содержимое регистров **EBP** и **ESP**.
- Следует сохранять и восстанавливать все регистры, которые подвергаются изменению внутри процедуры.
- Передача аргументов в процедуру на ассемблере из программы на C/C++ осуществляется через стек, возвращаемое значение – через регистр **EAX**.

Внешний ассемблерный код

Для обмена информацией между модулями используются директивы **EXTRN** и **PUBLIC**:

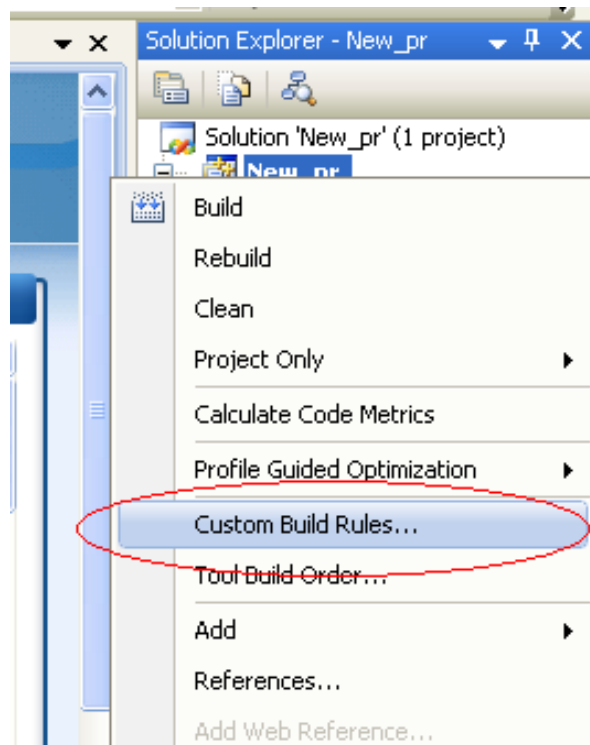
- **EXTRN** предназначена для объявления некоторого имени внешним по отношению к данному модулю;
- **PUBLIC** предназначена для объявления некоторого имени, определенного в этом модуле и видимого в других модулях.

Вызов процедур на Ассемблере из программ на C++

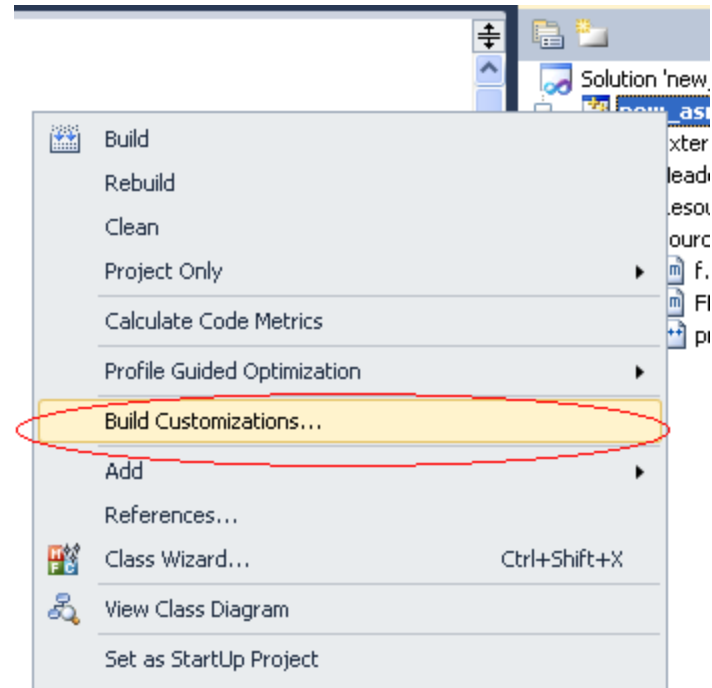
Для того, чтобы процедуры на Ассемблере вызывались из программ на C++, необходимо:

1. Создать пустой проект среде MS Visual Studio.
2. Подключить компилятор MASM к проекту

VS 2008

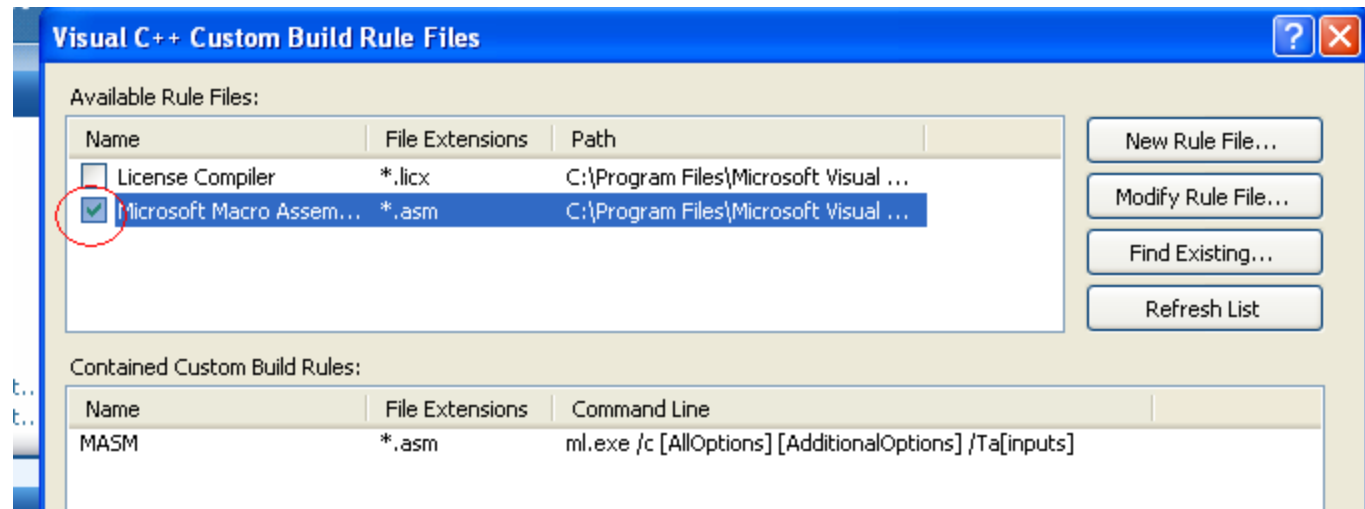


VS 2010

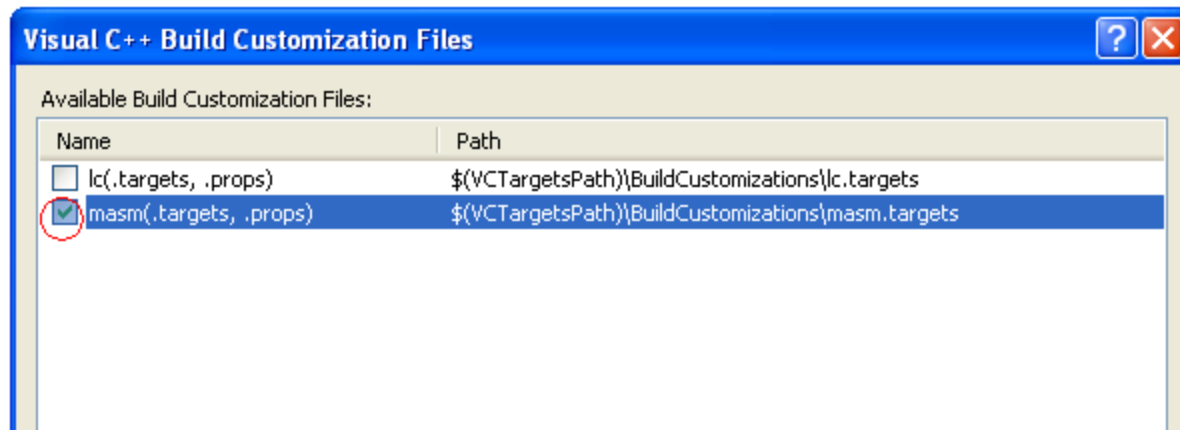


Вызов процедур на Ассемблере из программ на C++

VS 2008

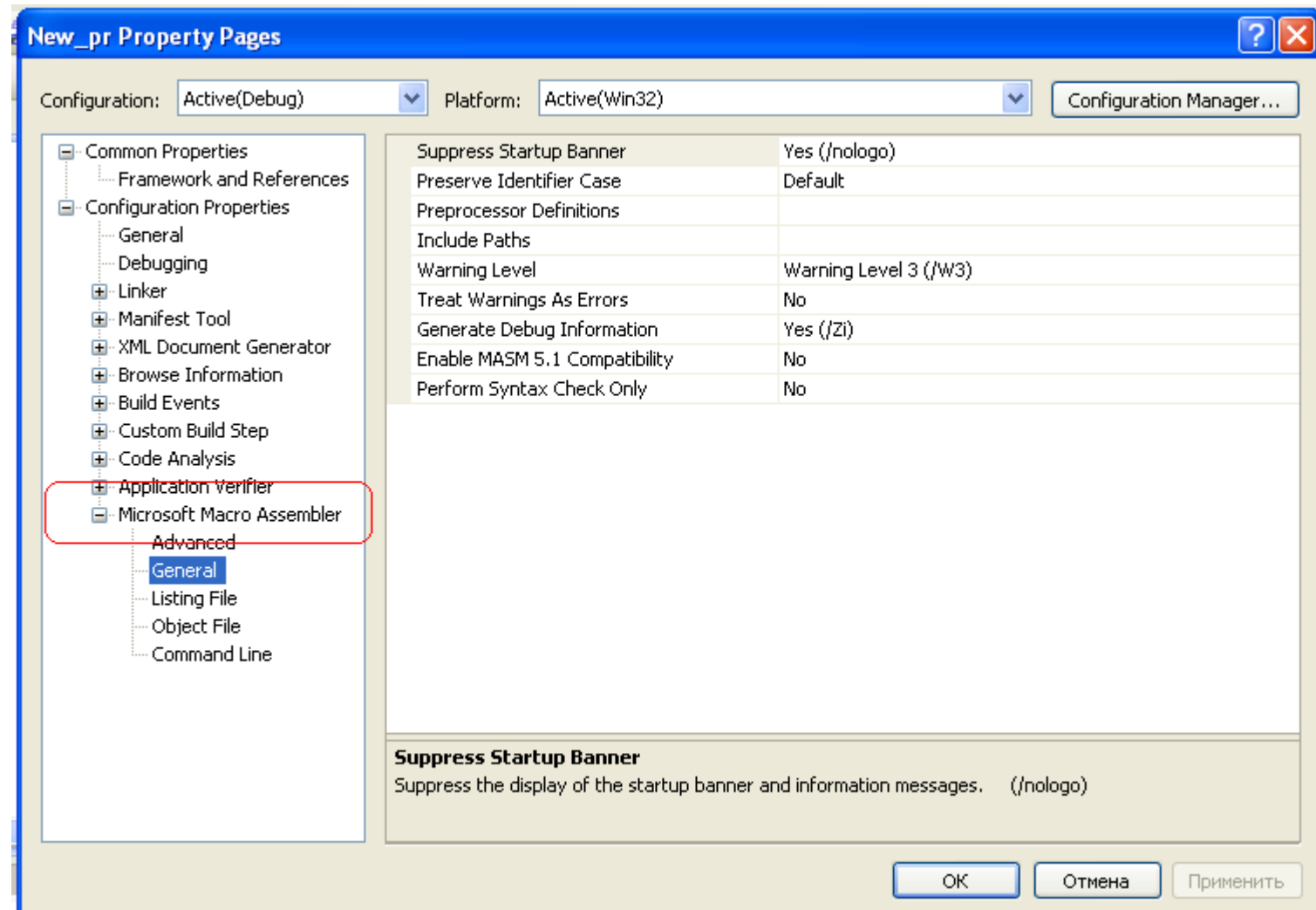


VS 2010



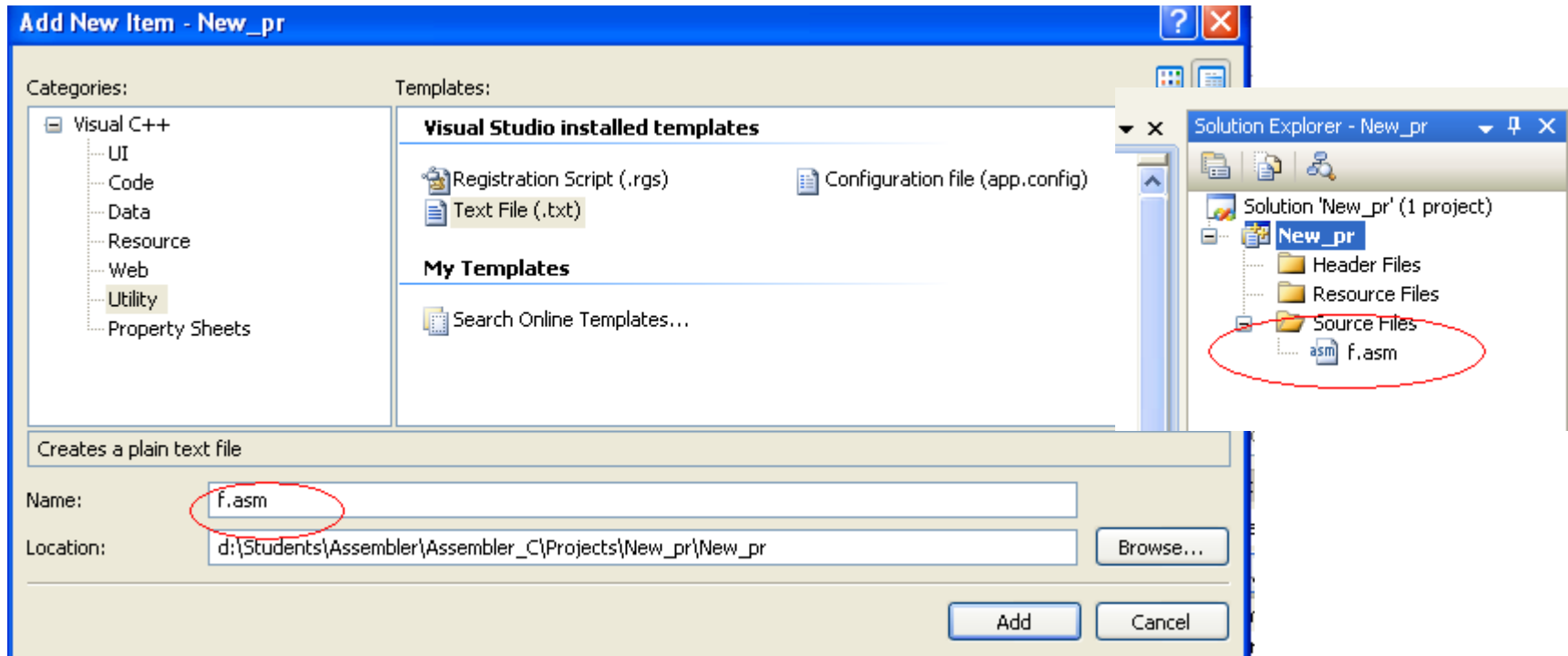
и нажать на **Ok**

Вызов процедур на Ассемблере из программ на C++



Вызов процедур на Ассемблере из программ на C++

3. Создать пустой файл с расширением **.asm**



4. Составить текст процедуры на ассемблере. Процедура должна обрамляться директивами **PROC** и **ENDP**.

Вызов процедур на Ассемблере из программ на C++

В файле ***.asm** до директивы **PROC** необходимо :

- включить директиву выбора CPU для распознавания отдельных команд, например:
 .386 или **.486** для адресации по 32-битным регистрам (всех команд вплоть до указанного процессора).
- объявить имя процедуры с директивой **PUBLIC** ;
- включить модель памяти **flat** (**.model flat**);

При написании текста процедуры соблюдать требования:

- параметры являются 32-разрядными;
- при доступе к параметрам учитывать их порядок;
- при возврате управления при необходимости очищать стек от параметров.

Вызов процедур на Ассемблере из программ на C++

5. Составить текст программы на C++.

Описать прототип функции, реализованной на Ассемблере, в виде

`extern "C" тип_возврата тип_соглашения имя(параметры) ;`

Важно:

При стыковке модулей C/C++ и Ассемблера следует учитывать, что:

- соглашение вызова (***calling convention***) определяет, как передаются параметры в подпрограмму, осуществляется возврат из подпрограммы и возвращается результат;
- компиляторы языков C и C++ искажают имена функций (***name decoration***).

Наиболее распространенные соглашения о вызовах

Соглашение (для Ассемблера)	Передача параметров	Очистка стека	Использование регистров для параметров
fastcall	слева направо (1-й внизу стека)	вызывающая программа	ecx, edx
stdcall	справа налево (1-й вверху стека)	Процедура (ret число)	нет
cdecl	справа налево	вызывающая программа	нет

name decoration

- *"name decoration"* или *"name mangling"* – механизм генерации компилятором строки, содержащей, кроме собственно имени функции, символы, используемые компилятором или компоновщиком для получения информации о типах параметров;
- *"decorated name"* – сгенерированная таким образом строка.

decorated name используется для разрешения внешних ссылок на этапе линковки (с учетом возможных перегрузок).

В 64-разрядной версии имена не декорируются!

Упрощенное name decoration

При указании, что внешняя функция написана в стиле C, а не C++ (**extern "C"**) – получение **"name decoration "** идет по упрощенным правилам:

- для соглашений **CDECL** *decorated name* получается добавлением символа **"_"** к началу имени функции;
- для соглашений **STDCALL** *decorated name* получается добавлением символа **"_"** к началу имени функции, символа **"@"** и глубины стека для параметров;
- для соглашений **FASTCALL** *decorated name* получается добавлением символа **"@"** к началу имени функции, символа **"@"** и глубины стека для параметров.

Правила декорирования имен в Ассемблере

Описатель	Соглашение о вызовах (для C)	Декорирование имени в Ассемблере
<code>extern "C"</code>	<code>__fastcall</code>	@имя@число
<code>extern "C"</code>	<code>__stdcall</code>	_имя@число
<code>extern "C"</code>	<code>__cdecl</code>	_имя

По умолчанию тип соглашения `__cdecl`

Требования к оформлению процедур на Ассемблере, вызываемых из программ на C++

Для правильного разрешения **name decoration** необходимо:

- задавать имена процедур в соответствии с правилами упрощенной **name decoration**;
- не указывать параметры в директивах **PROC** для процедур, непосредственно вызываемых из C++;
- записывать пролог и эпилог для таких процедур, если только в них нет директивы **LOCAL**.

Примеры связи C++ с процедурами на Ассемблере

Задача 1: Вычислить остаток от деления двух целых. Используется самый быстрый способ передачи параметров - регистровый, соглашение вызова - *fastcall*

Часть 1: головная программа на C++

```
#include <iostream>
extern "C"    int __fastcall Remainder(int,int) ;

void main()
{
    std::cout << "remainder=" <<
        Remainder (-12, 5) << std::endl;
```

Пример связи C++ с процедурами на Ассемблере

Часть 2: модуль на Ассемблере

```
.486                                (!)
PUBLIC @Remainder@8
.model flat
.code
@Remainder@8    proc
    mov eax,ecx ;первый параметр
    mov ecx,edx ;второй параметр
    cdq
    idiv ecx
    mov eax,edx
    ret
@Remainder@8    endp
end
```

Примеры связи C++ с процедурами на Ассемблере

Задача 2: Процедура уменьшает в 2 раза свой аргумент. Используется *cdecl* по умолчанию.

Часть 1: головная программа на C++

```
#include <iostream>
//extern "C"    int __cdecl DivideByTwo(int) ;
extern "C"    int DivideByTwo(int) ;
void main()
{
    std::cout << "DivideByTwo=" <<
        DivideByTwo(-12) << std::endl;
```


Пример связи C++ с процедурами на Ассемблере

Часть 2: модуль на Ассемблере

.386

PUBLIC _DivideByTwo

.model flat

.code

_DivideByTwo proc

push ebp

mov ebp, esp

mov eax, [ebp+8]

sar eax, 1 ; арифметический сдвиг вправо

mov esp, ebp

pop ebp

ret

_DivideByTwo endp

end

Примеры связи C++ с процедурами на Ассемблере

Задача 3: написать функцию, получающую два целочисленных параметра a и b длиной 4 байта каждый и возвращающую значение $3a+4b$. Никакие регистры, кроме EAX, использовать нельзя. Соглашения о связях – *stdcall*.

Часть 1: головная программа на C++

```
#include <iostream>
extern "C" int __stdcall F3A4B(int, int);
void main()
{
    int i1=10000, i2= 2000;
    std::cout << F3A4B(i1,i2) << std::endl;
}
```

Пример связи C++ с процедурами на Ассемблере

Часть 2: модуль на Ассемблере

```
.386
PUBLIC _F3A4B@8
.model flat
.code
_F3A4B@8    proc
            push    ebp
            mov     ebp, esp
            push    [ebp+12]
            push    [ebp+8]
            call    p1
            mov     esp, ebp
            pop     ebp
            ret     8
_F3A4B@8    endp
```

Пример связи C++ с процедурами на Ассемблере

Часть 2: модуль на Ассемблере (продолжение)

```
p1          proc  stdcall, a: dword, b: dword
            local tmp:dword
            mov    eax, a
            shl    eax, 2
            sub    eax, a
            mov    tmp, eax
            mov    eax, b
            shl    eax, 2
            add    eax, tmp
            ret
p1          endp
            end
```

Примеры связи C++ с процедурами на Ассемблере

Задача 4: Процедура меняет местами значения своих аргументов. Соглашения о связях – *stdcall*.

Часть 1: головная программа на C++

```
#include <iostream>
extern "C" void __stdcall IntSwap(int&, int&);
void main()
{
    int a = 12, b = -7;
    std::cout << "a=" << a << "b=" << b << std::endl;
    IntSwap(a, b);
    std::cout << "a=" << a << "b=" << b << std::endl;
}
```

Пример связи C++ с процедурами на Ассемблере

Часть 2: модуль на Ассемблере

.386

PUBLIC _IntSwap@8

.model flat

.code

```
_IntSwap@8  proc
    push    ebp
    mov     ebp, esp
    mov     esi, [ebp+8]      ; адрес первого числа
    mov     edi, [ebp+12]    ; адрес второго числа
    mov     eax, [esi]       ; первое число
    xchg    eax, [edi]
    mov     [esi], eax
    mov     esp, ebp
    pop     ebp
    ret     8
```

_IntSwap@8 endp

end