

# Использование Ассемблера в ЯВУ

## Соглашение о регистрах

Регистры, используемые в C++:

SS – сегмент стека

DS – сегмент данных

BP – для указания на список параметров, передаваемых через стек

SP – указатель стека (после завершения подпрограммы стек должен быть восстановлен до своего первоначального состояния на момент вызова подпрограммы)

# Использование Ассемблера в ЯВУ

## Формы комбинирования программ на языках высокого уровня с ассемблером:

- Использование ассемблерных вставок
- Использование внешних процедур и функций.

# Использование Ассемблера в ЯВУ

## Использование ассемблерных вставок

`_asm` КодОперации операнды ; // комментарии

```
_asm  
{  
текст программы на ассемблере ; комментарии  
}
```

# Использование Ассемблера в ЯВУ

## Использование ассемблерных вставок (пример)

**Пример** Даны целые числа  $a$  и  $b$ .  
Вычислить выражение  $a+5b$ .

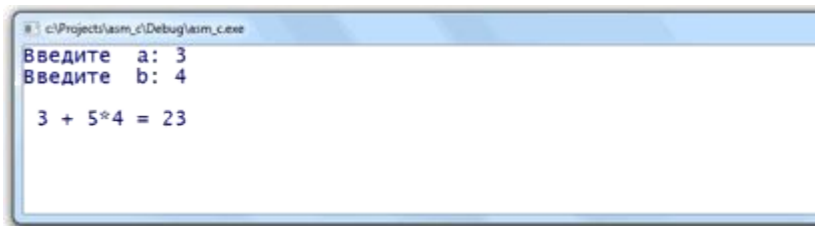
Для вывода

приглашений **Введите**

**a:** и **Введите b:** используем

функцию `CharToOem(_T("Введите ")),s)`,

где  $s$  – указатель на строку,  
которая перекодирует  
русскоязычные сообщения.



```
c:\Projects\asm_c1\Debug\asm_c.exe
Введите a: 3
Введите b: 4
3 + 5*4 = 23
```

```
#include <stdio.h>
#include <windows.h>
#include <tchar.h>
void main()
{
    char s[20];
    int a, b, sum;
    CharToOem(_T("Введите "),s);
    printf("%s a: ", s);
    scanf("%d",&a);
    printf("%s b: ",s);
    scanf("%d",&b);
    _asm
    {
        mov eax, a;
        mov ecx, 5
m:    add eax, b
        loop m
        mov sum, eax
    }
    printf("\n %d + 5*%d = %d", a, b, sum);
    getchar(); getchar();
}
```

# Использование Ассемблера в ЯВУ

## Использование внешних процедур

Для связи посредством внешних процедур создается многофайловая программа. При этом в общем случае возможны два варианта вызова:

- программа на языке высокого уровня вызывает процедуру на языке ассемблера;
- программа на языке ассемблера вызывает процедуру на языке высокого уровня.

# Использование Ассемблера в ЯВУ

## Использование внешних процедур

Основные соглашения по передаче параметров в процедуру

Соглашение	Параметры	Очистка стека	Регистры
Pascal (конвенция языка Паскаль)	Слева направо	Процедура	Нет
C (конвенция C)	Справа налево	Вызывающая программа	Нет
Fastcall (быстрый или регистровый вызов)	Слева направо	Процедура	Задействованы три регистра (EAX, EDX, ECX), далее стек
Stdcall (стандартный вызов)	Справа налево	Процедура	Нет

# Использование Ассемблера в ЯВУ

Конвенция Pascal заключается в том, что параметры из программы на языке высокого уровня передаются в стеке и возвращаются в регистре AX/EAX, — это способ, принятый в языке PASCAL (а также в BASIC, FORTRAN, ADA, OBERON, MODULA2), — просто поместить параметры в стек в естественном порядке.

В этом случае запись

```
some_proc(a,b,c,d);
```

запишется как

```
push a
```

```
push b
```

```
push c
```

```
push d
```

```
call some_proc@16
```

```
some_proc proc
```

```
push ebp
```

```
mov ebp,esp ; пролог
```

```
mov eax, [ebp+20] ; a
```

```
mov ebx, [ebp+16] ; b
```

```
mov ecx, [ebp+12] ; c
```

```
mov edx, [ebp+8] ; d
```

```
...
```

```
pop ebp ; эпилог
```

```
ret 16
```

```
some_proc endp
```

# Использование Ассемблера в ЯВУ

Конвенция C используется, в первую очередь, в языках C и C++, а также в PROLOG и других. Параметры помещаются в стек в обратном порядке, и, в противоположность PASCAL-конвенции, удаление параметров из стека выполняет вызывающая процедура.

Запись `some_proc(a,b,c,d)`

будет выглядеть как

```
push d
push c
push b
push a
call some_proc@16
add esp,16 ; освободить стек
```

```
some_proc proc
push ebp
mov ebp,esp ; пролог
mov eax, [ebp+8] ; a
mov ebx, [ebp+12] ; b
mov ecx, [ebp+16] ; c
mov edx, [ebp+20] ; d
...
pop ebp
ret
some_proc endp
```



# Использование Ассемблера в ЯВУ

## Смешанные конвенции

Существует конвенция передачи параметров **STDCALL**, отличающаяся и от **C**, и от **PASCAL**-конвенций, которая применяется для всех системных функций **Win32 API**. Здесь параметры помещаются в стек в обратном порядке, как в **C**, но процедуры должны очищать стек сами, как в **PASCAL**.

Еще одно отличие от **C**-конвенции – это быстрое или регистровое соглашение **FASTCALL**. В этом случае параметры в функции также передаются по возможности через регистры.

```
some_proc(a,b,c,d,e,f);
```

```
mov a, eax  
mov b, edx  
mov c, ecx  
mov d, [ebp+8]  
mov e, [ebp+12]  
mov f, [ebp+16]
```

# Использование Ассемблера в ЯВУ

## Возврат результата из процедуры

Чтобы вернуть результат в программу на С из процедуры на ассемблере, перед возвратом управления в вызываемой процедуре (на языке ассемблера) необходимо поместить результат в соответствующий регистр:

Тип возвращаемого значения	Регистр
<code>unsigned char</code>	<code>al</code>
<code>char</code>	<code>al</code>
<code>unsigned short</code>	<code>ax</code>
<code>short</code>	<code>ax</code>
<code>unsigned int</code>	<code>eax</code>
<code>int</code>	<code>eax</code>
<code>unsigned long int</code>	<code>edx:eax</code>
<code>long int</code>	<code>edx:eax</code>

# Использование Ассемблера в ЯВУ

## Тонкости вызова методов между C++ и Asm

1. Если надо вызывать из ассемблера библиотечные методы, достаточно в начале секции кода указать, какие именно методы мы собираемся использовать.

```
EXTRN printf : proc ;we'll use printf
```

Далее можно просто использовать call:

```
;printf(ebx,eax)
push eax;
push ebx
call printf
add esp, 8 ;pop x2
```

# Использование Ассемблера в ЯВУ

## Тонкости вызова методов между C++ и Asm

2. Если же надо вызывать пользовательские методы, то кроме п.1 надо еще писать `extern «C»` перед определением метода.

```
extern "C"  
void* readName()  
{  
    char* name = (char*)calloc(1, 255);  
    scanf("%s", name);  
    while (getchar() != '\n');  
    return name;  
}
```

Соответственно, в \*.asm файле:

```
EXTRN readName : proc ;and void* readName()  
и  
call readName ;eax = readName()
```

# Использование Ассемблера в ЯВУ

## Тонкости вызова методов между C++ и Asm

3. случае использования  
Asm-методов в C++  
достаточно лишь указать  
прототип:

```
extern "C"  
{  
    void sayHello();  
}
```

Данный прототип соответствует такому  
объявлению Asm-метода:

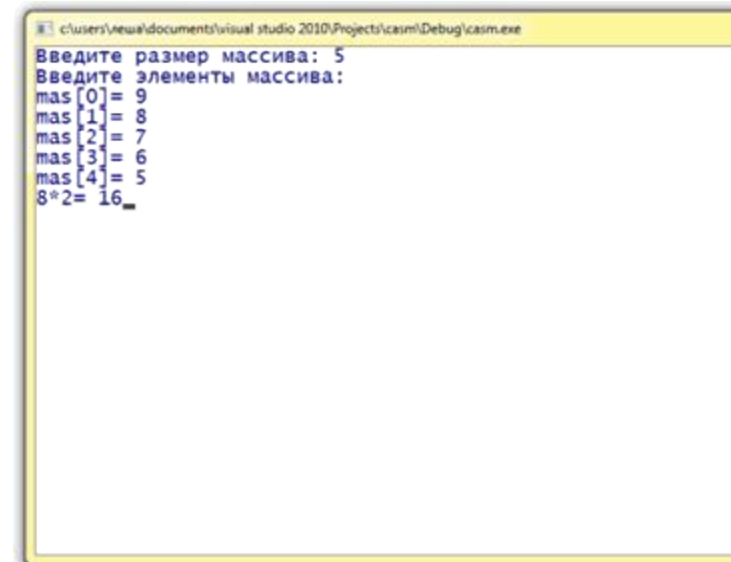
```
sayHello PROC  
  
call readName ;eax = readName()  
lea ebx, helloFormat ;ebx = &helloFormat  
  
;printf(ebx,eax)  
push eax  
push ebx  
call printf  
add esp, 8 ;pop x2  
  
retn  
  
sayHello ENDP
```

# Использование Ассемблера в ЯВУ

**Пример: Умножить на 2 первый элемент массива (нумерация элементов ведется с 0).**

```
//Вызывающая программа file1.cpp
#include <iostream>
using namespace std;
extern "C" int MAS_FUNC (int *, int);
int main() {
    int *mas, n, k;
    system("chcp 1251");
    system("cls");
    cout << "Введите размер массива: ";
    cin >> n;
    mas = new int[n];
    cout << "Введите элементы массива: " << endl;
    for(int i=0; i<n; i++) {
        cout << "mas[" << i <<"]= ";
        cin >> mas[i];
    }
    k = MAS_FUNC(mas, n);
    cout << mas[1] << "*2= " << k;
    cin.get(); cin.get();
    return 0;
}
```

```
;Вызываемая функция file2.asm
.586
.MODEL FLAT, C
.CODE
MAS_FUNC PROC C mas:dword, n:dword
mov esi,mas
mov eax, [esi+4]
shl eax, 1
ret
MAS_FUNC ENDP
END
```



```
c:\users\viewa\documents\visual studio 2010\Projects\casim\Debug\casim.exe
Введите размер массива: 5
Введите элементы массива:
mas[0]= 9
mas[1]= 8
mas[2]= 7
mas[3]= 6
mas[4]= 5
8*2= 16_
```