Процедуры и функции

Что такое процедура

Процедура (подпрограмма) — это основная функциональная единица декомпозиции (разделения на несколько частей) некоторой задачи. Процедура представляет собой группу команд для решения конкретной подзадачи и обладает средствами получения управления из точки вызова задачи более высокого приоритета и возврата управления в эту точку.

Что такое процедура

В простейшем случае программа может состоять из одной процедуры. Процедуру можно определить и как правильным образом оформленную совокупность команд, которая, будучи однократно описана, при необходимости может быть вызвана в любом месте программы.

Что такое функция

Функция – процедура, способная возвращать некоторое значение.

Как использовать

Для описания последовательности команд в виде процедуры в языке ассемблера используются две директивы: PROC и ENDP.

Как использовать

Синтаксис описания процедуры:

```
ИмяПроцедуры PROC расстояние
; тело процедуры
ИмяПроцедуры ENDP
```

Как использовать

Атрибут расстояние может принимать значения near или far и характеризует возможность обращения к процедуре из другого сегмента кода. По умолчанию атрибут расстояние принимает значение near, и именно это значение используется при выборе плоской модели памяти FLAT.

Где размещать?

Процедура может размещаться в любом месте программы, но так, чтобы на нее случайным образом не попало управление. Если процедуру просто вставить в общий поток команд, то микропроцессор будет воспринимать команды процедуры как часть этого потока.

Где размещать?

- в начале программы (до первой исполняемой команды);
- в конце (после команды, возвращающей управление операционной системе);
- промежуточный вариант — тело процедуры располагается внутри другой процедуры или основной программы;
- в другом модуле.

В начале

```
...
.code
myproc proc near
ret
myproc endp
start proc
call myproc
...
start endp
end start
```

В конце

```
...
.code
start proc
call myproc
...
start endp
myproc proc near
ret
myproc endp
end start
```

Ключевые слова

Поскольку имя процедуры обладает теми же атрибутами, что и метка в команде перехода, то обратиться к процедуре можно с помощью любой команды условного или безусловного перехода.

Ключевые слова

В системе команд микропроцессора есть две команды, осуществляющие работу с контекстом. Это команды **call** и **ret**:

call ИмяПроцедуры@num — вызов процедуры (подпрограммы).

ret **число** — возврат управления вызывающей программе.

Ключевые слова

число — необязательный параметр, обозначающий количество байт, удаляемых из стека при возврате из процедуры.

@num – количество байт, которое занимают в стеке переданные аргументы для процедуры (параметр является особенностью использования транслятора MASM).

Объединение процедур, расположенных в разных модулях

В разных модулях

Так как отдельный модуль — это функционально автономный объект, то он ничего не знает о внутреннем устройстве других модулей, и наоборот, другим модулям также ничего не известно о внутреннем устройстве данного модуля. Но каждый модуль должен иметь такие средства, с помощью которых он извещал бы транслятор о том, что некоторый объект (процедура, переменная) должен быть видимым вне этого модуля.

В разных модулях

И наоборот, нужно объяснить транслятору, что некоторый объект находится вне данного модуля. Это позволит транслятору правильно сформировать машинные команды, оставив некоторые их поля незаполненными. Позднее, на этапе компоновки настраивает модули и разрешает все внешние ссылки в объединяемых модулях.

В разных модулях

Директива **extern** предназначена для объявления некоторого имени внешним по отношению к данному модулю. Это имя в другом модуле должно быть объявлено в директиве **public**.

В разных модулях

Директива public предназначена для объявления некоторого имени, определенного в этом модуле и видимого в других модулях.

В разных модулях

Синтаксис этих директив следующий:

extern имя:тип, ..., имя:тип

public имя, ..., имя

В разных модулях

Тип определяет тип идентификатора. Указание типа необходимо для того, чтобы транслятор правильно сформировал соответствующую машинную команду. Действительные адреса будут вычислены на этапе компоновки, когда будут разрешаться внешние ссылки.

В разных модулях

Возможные значения типа определяются допустимыми типами объектов для этих директив:

если **имя** — это имя переменной, то тип может принимать значения byte, word, dword, qword и tbyte;

если **имя** — это имя процедуры, то тип может принимать значения near или far; в компиляторе MASM после имени процедуры необходимо указывать число байтов в стеке, которые занимают аргументы функции:

```
extern p1@0:near
```

если **имя** — это имя константы, то тип должен быть abs.

Пример

```
;Модуль 1
.586
.model flat, stdcall
.data
extern p1@0:near
.code
start proc
call p1@0
ret
start endp
end start
```

```
;Модуль 2
.586
.model flat, stdcall
public p1
.data
.code
p1 proc
ret
p1 endp
end
```

Передача аргументов

Что такое аргументы

Аргумент — это ссылка на некоторые данные, которые требуются для выполнения возложенных на модуль функций и размещенных вне этого модуля.

Что такое константы

Константы — данные, значения которых не могут изменяться.

Что такое сигнатура процедуры

Сигнатура процедуры (функции) — это имя функции, тип возвращаемого значения и список аргументов с указанием порядка их следования и типов.

Что такое семантика процедуры

Семантика процедуры (функции) — это описание того, что данная функция делает. Семантика функции включает в себя описание того, что является результатом вычисления функции, как и от чего этот результат зависит. Обычно результат выполнения зависит только от значений аргументов функции

Передача аргументов

Для передачи аргументов в языке ассемблера существуют следующие способы:

- через регистры;
- через общую область памяти;
- через стек;
- с помощью директив extern и public.

Передача аргументов через регистры

Передача аргументов через регистры – это наиболее простой в реализации способ передачи данных. Данные, переданные подобным способом, становятся доступными немедленно после передачи управления процедуре. Этот способ очень популярен при небольшом объеме передаваемых данных.

Передача аргументов через стек

Наиболее часто используется для передачи аргументов при вызове процедур. Суть этого способа заключается в том, что вызывающая процедура самостоятельно заносит в стек передаваемые данные, после чего передает управление вызываемой процедуре.

Передача аргументов через стек

При передаче управления процедуре микропроцессор автоматически записывает в вершину стека 4 байта. Эти байты являются адресом возврата в вызывающую программу.

Передача аргументов через стек

Стек обслуживается тремя регистрами:

- ESS - указатель дна стека (начала сегмента стека);
- ESP - указатель вершины стека;
- EBP - указатель базы.

Передача аргументов через стек

В начало процедуры рекомендуется включить дополнительный фрагмент кода. Он имеет свое название — **пролог процедуры**. Код пролога состоит всего из двух команд:

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

Передача аргументов через стек

Конец процедуры также должен содержать действия, обеспечивающие корректный возврат из процедуры. Фрагмент кода, выполняющего такие действия, имеет свое название — **эпилог процедуры.**

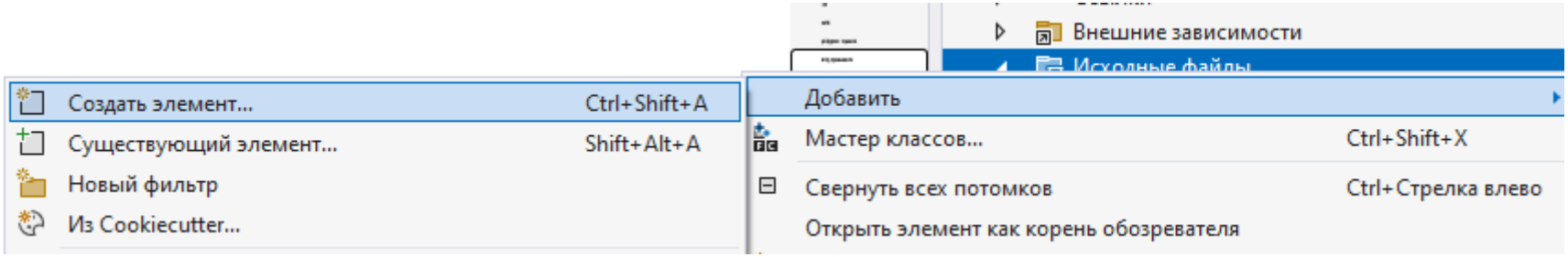
Передача аргументов через стек

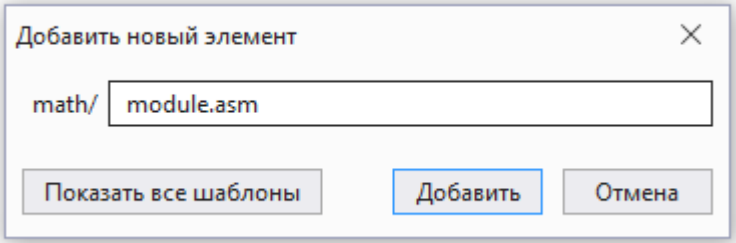
Нужно откорректировать содержимое стека, убрав из него ставшие ненужными аргументы, передававшиеся в процедуру. Способы:

- используя последовательность из n команд pop xx.
- используя машинную команду ret n в качестве последней исполняемой команды в процедуре, где n — количество байт, на которое нужно увеличить содержимое регистра esp

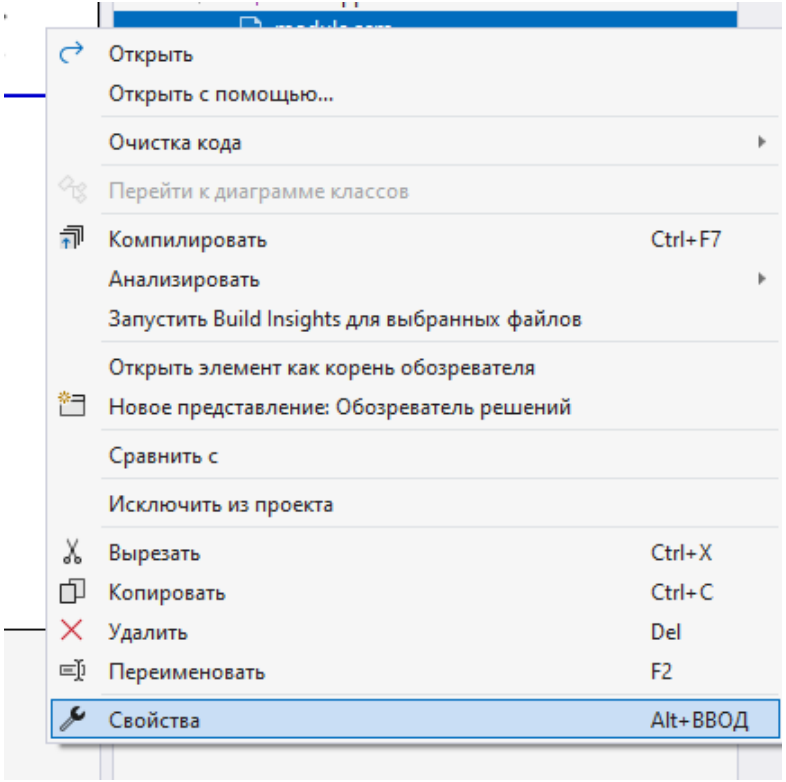
```
.586
.model flat, stdcall
.stack 4096
.data
.code
proc_1 proc         ; начало процедуры
push ebp            ; пролог: сохранение EBP
mov ebp, esp        ; пролог: инициализация EBP
mov eax, [ebp+8]    ; доступ к аргументу 4
mov ebx, [ebp+12] ; доступ к аргументу 3
mov ecx, [ebp+16] ; доступ к аргументу 2
pop ebp             ; эпилог: восстановление EBP
ret 12
proc_1 endp
main proc
push 2
push 3
push 4
call proc_1
ret
main endp
end main
```

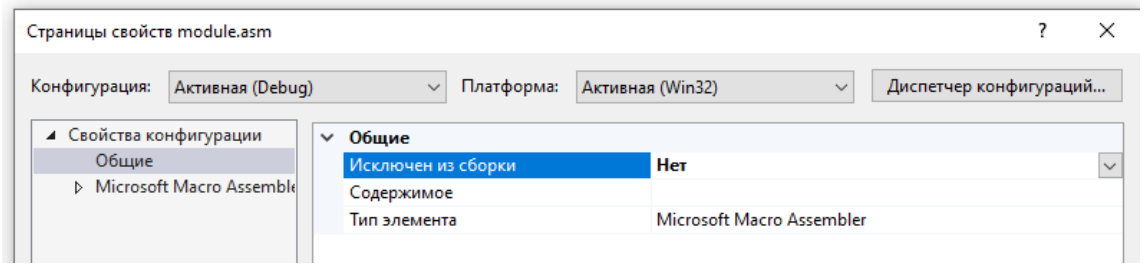
Создание ассемблерного модуля





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

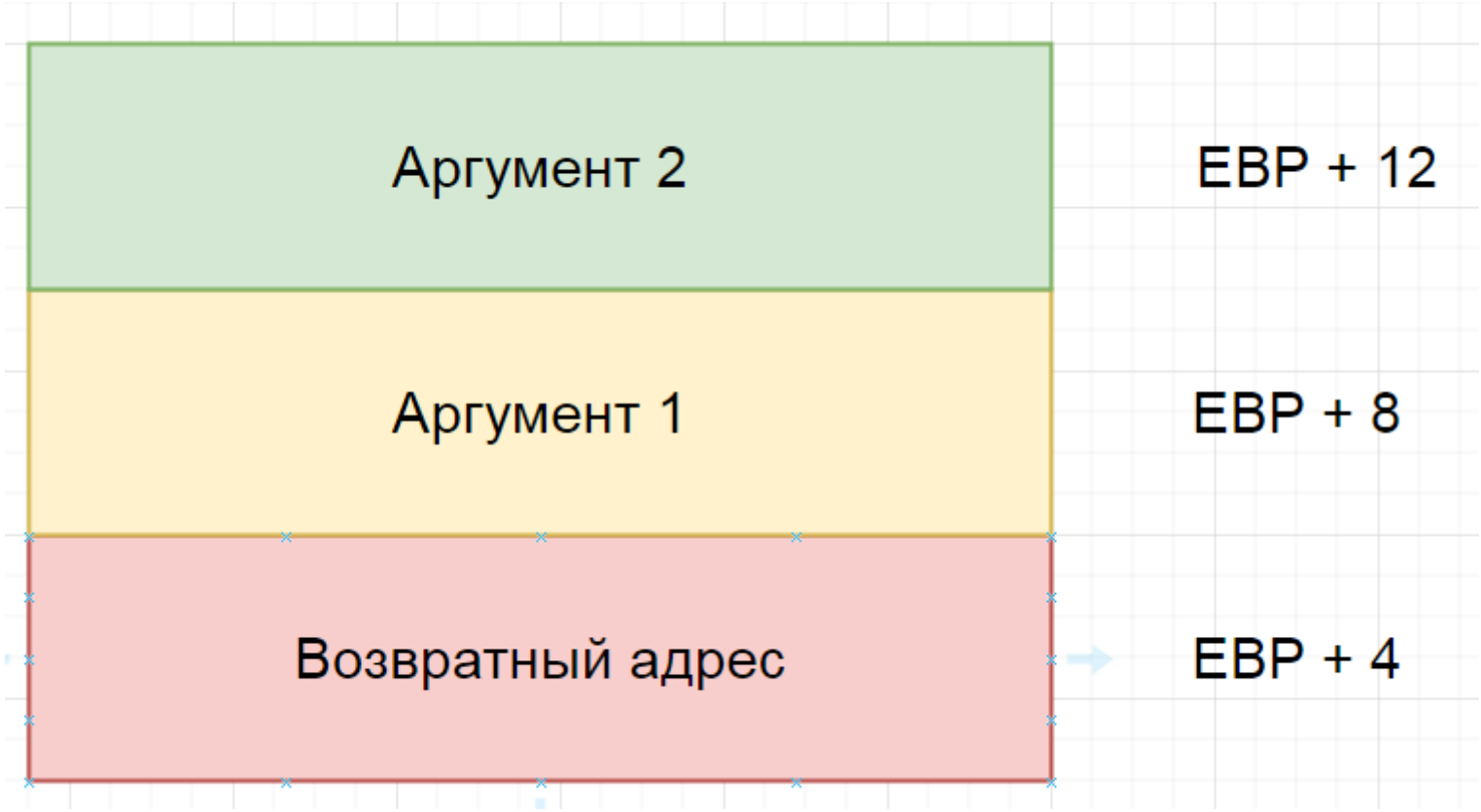
Инструкции push ebp и mov ebp, esp образуют стандартный пролог функции в ассемблере x86.

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.