

2022

Глава 3 *Модульное программирование*

МГТУ им. Н.Э. Баумана

Факультет Информатика и системы управления

Кафедра Компьютерные системы и сети

Лектор: д.т.н., проф.

Иванова Галина Сергеевна

3.1 Организация передачи управления в процедуру и обратно

Процедура в ассемблере – это относительно самостоятельный фрагмент, к которому возможно обращение из разных мест программы.

- На языках высокого уровня такие фрагменты оформляют соответствующим образом и называют подпрограммами: *функциями* или *процедурами* в зависимости от способа возврата результата.
- Поддержка модульного принципа для ассемблера означает, что в языке существуют специальные *машинные* команды вызова подпрограммы и обратной передачи управления.
- Кроме того существует несколько директив, поддерживающих отдельную трансляцию модулей и их последующую совместную компоновку, и также создание библиотек модулей.

Команды вызова процедуры и возврата управления

1. Команда вызова процедуры:

CALL rel32/r32/m32 ; вызов внутрисегментной
; процедуры (*near* - ближний)

CALL sreg:r32/m48 ; вызов межсегментной процедуры
; (*far* - дальний)

2. Команда возврата управления:

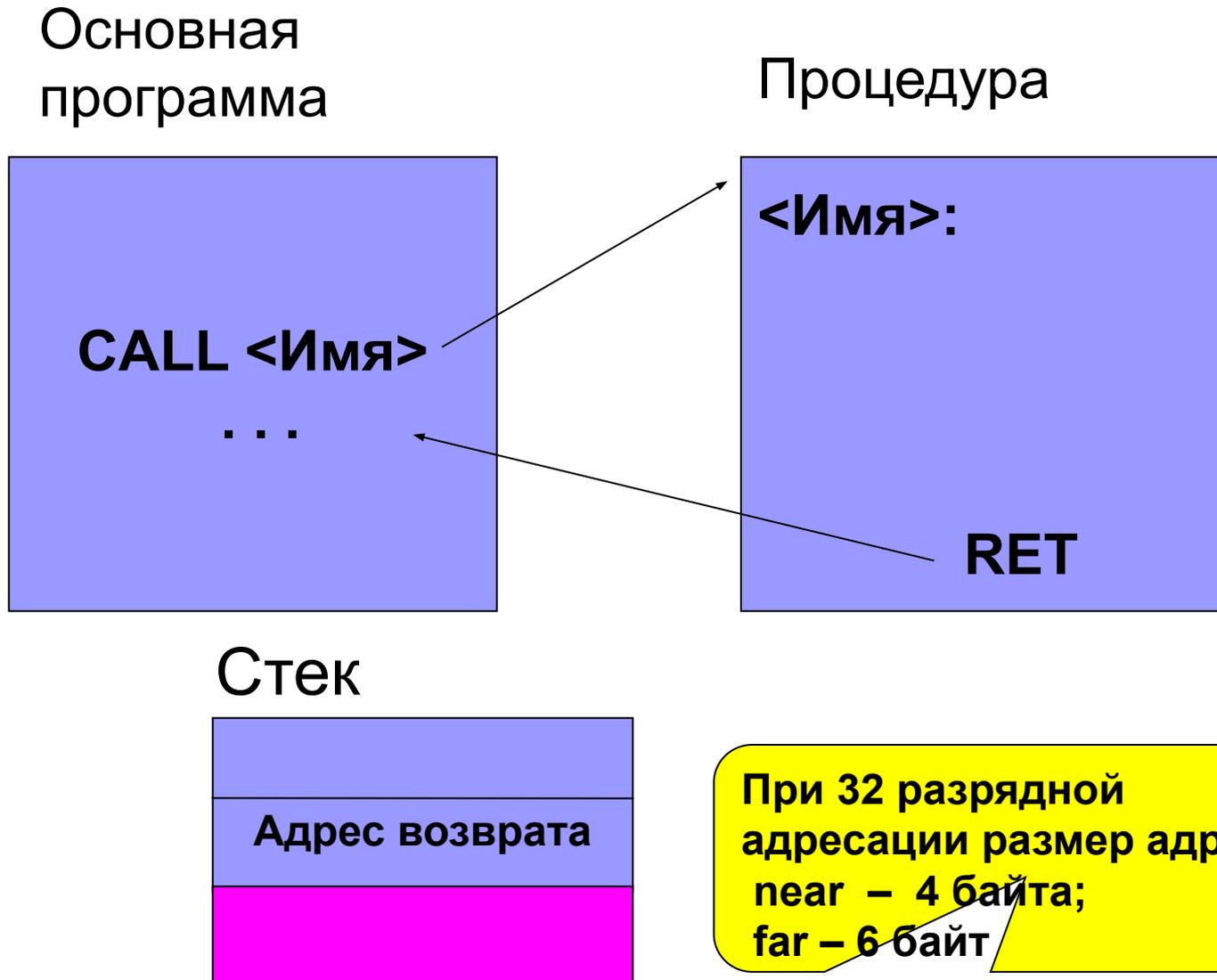
RET [<Целое>]

где <Целое> – количество байт, извлекаемых из стека при возврате управления – используется для удаления из стека параметров процедуры (см. далее).

При выполнении команды вызова процедуры автоматически в стек заносится адрес команды, следующей за командой вызова процедуры, – адрес возврата.

Команда возврата управления выбирает этот адрес из стека и осуществляет переход по нему.

Организация передачи управления в процедуру



Локальные метки

Для того чтобы метки внутри подпрограмм были уникальны, в них используют локальные метки, начинающиеся с точки.

В этом случае ассемблер создает метку, добавляя к ней предыдущую, как правило, имя подпрограммы, например:

```
prog1:
```

```
.cycle    ...  
          ...  
          loop   .cycle  
          ...  
          ret
```

При ассемблировании будет построена метка `prog1.cycle`.

Пример 3.1 Программа с процедурой MaxDword

```

      section .data
A      dd      56
B      dd      34

      section .bss
D      resd   1

      section .text
      call   MaxDword ; вызов процедуры
      ...

```

; вывод сообщения, ввод Enter и завершение программы

Текст
процедуры

Текст процедуры MaxDword

MaxDword:

```
    push    EAX        ; сохранить регистр
    mov     EAX, [A]   ; загрузить 1-е число в регистр
    cmp     EAX, [B]   ; сравнить числа
    jg      .con       ; если 1-е больше, то на запись
    mov     EAX, [B]   ; загрузить 2-е число в регистр
.con:  mov     [D], EAX ; записать результат в память
    pop     EAX        ; восстановить регистр
    ret                ; вернуть управление
```

3.2 Передача данных в подпрограмму

Данные могут быть переданы в подпрограмму:

- **через регистры** – перед вызовом процедуры параметры или их адреса загружаются в регистры, также в регистрах возвращаются результаты;
- **напрямую** – с использованием механизма глобальных переменных:
 - при совместной трансляции,
 - при отдельной трансляции;
- **через таблицу адресов** – в программе создается таблица, содержащая адреса параметров, и адрес этой таблице передается в процедуру через регистр;
- **через стек** – перед вызовом процедуры параметры или их адреса заносятся в стек, после завершения процедуры они из стека удаляются.

3.2.1 Передача параметров в регистрах

Пример 3.2а. Определение суммы двух целых чисел

```
    section .data
A    dd    56
B    dd    34

    section .bss
D    resd 1

    section .text

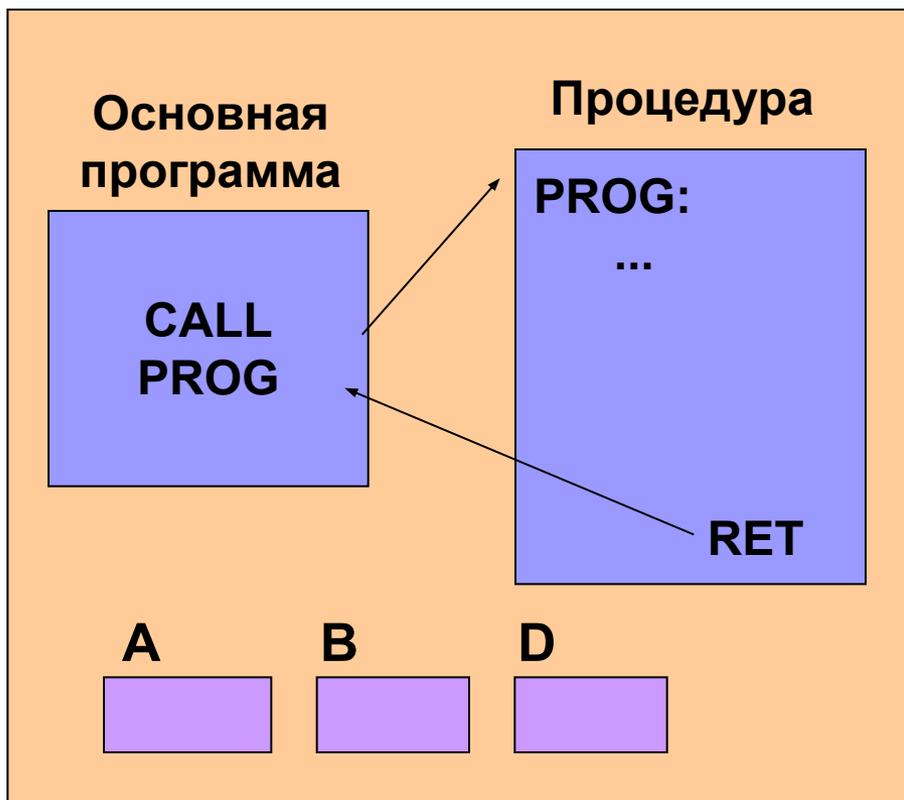
; Занесение параметров в регистры
    lea    EDX, [D]    ; адрес результата
    mov    EAX, [A]    ; первое число
    mov    EBX, [B]    ; второе число

    call   SumDword    ; вызов процедуры
    ...

SumDword:
    add    EAX, EBX
    mov    [EDX], EAX
    ret
```

3.2.2 Процедуры с глобальными переменными (совместная трансляция)

Исходный модуль



При совместной трансляции, когда основная программа и процедура объединены в один исходный модуль, ассемблер строит общую таблицу символьных имен. Следовательно, и основная программа и процедура могут обращаться к символьным именам, объявленным в том же модуле.

Способ не технологичен:

- процедуры не универсальны;
- большое количество ошибок.

Процедура, работающая с глобальными переменными при совместной трансляции

Пример 3.2b. Определение суммы двух целых чисел.

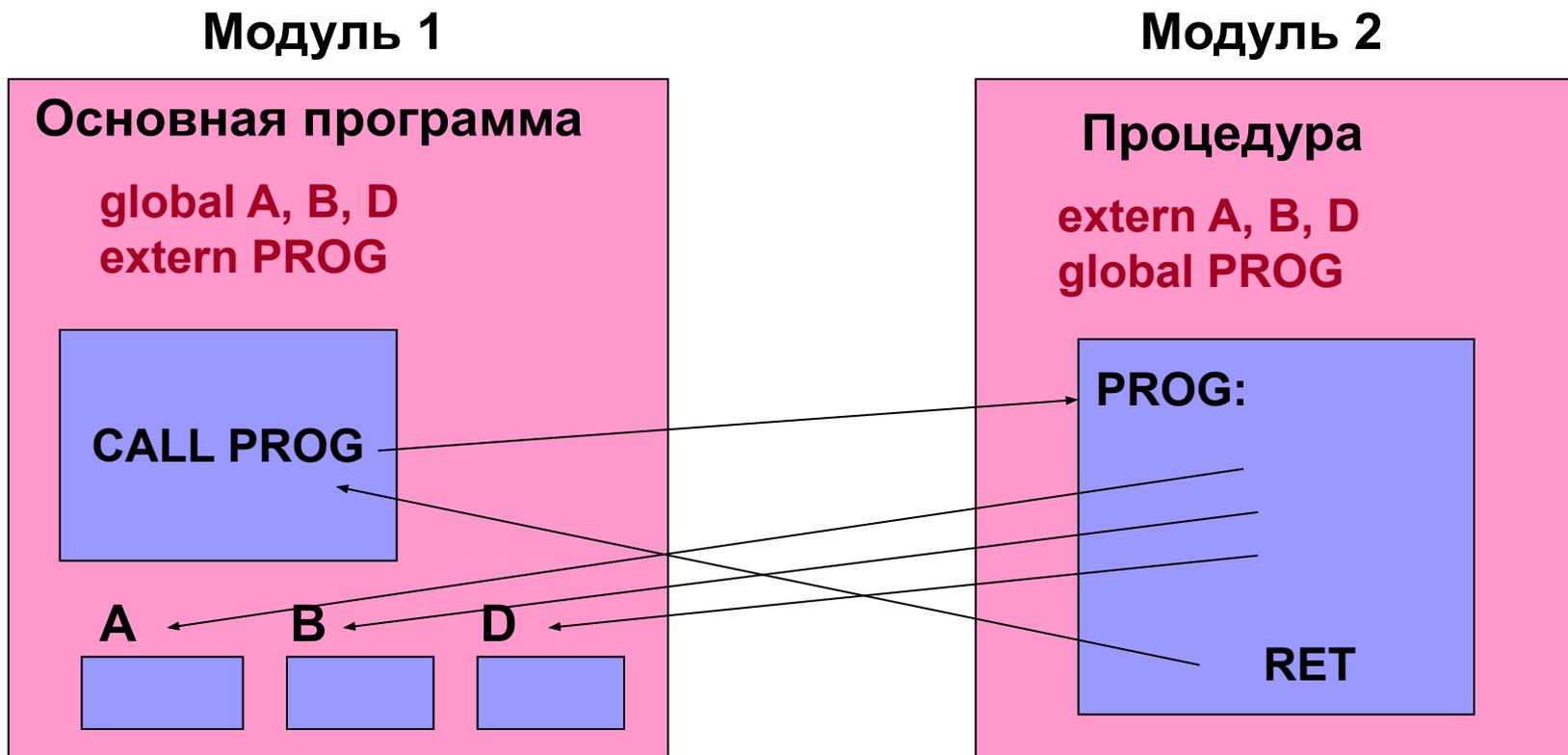
```
section .data
A    dd    56    ; первое число
B    dd    34    ; второе число

section .bss
D    resd  1     ; место для результата

section .text
call SumDword
. . .

SumDword:
mov   EAX, [A]   ; поместили в регистр 1-е число
add   EAX, [B]   ; сложили со вторым
mov   [D], EAX   ; результат отправили на место
ret
```

3.2.3 Многомодульные программы



Объединение модулей осуществляется во время компоновки программ. Программа и процедуры, размещенные в разных исходных модулях, на этапе ассемблирования «не видят» символических имен друг друга. Чтобы сделать имена видимыми за пределами модуля, их объявляют «внешними». Для этого используют директивы `global` и `extern`.

Раздельная трансляция. Основная программа

Пример 3.2с. Определение суммы двух целых чисел.

```

        section .data
A       dd      56
B       dd      34

        section .bss
D       resd    1

global A,B,D      ; объявление внутренних имен
extern SumDword   ; объявление внеш. имен

section .text
call    SumDword  ; вызов подпрограммы

. . .
```

Раздельная трансляция. Процедура в отдельном файле

```
extern A,B,D  
global SumDword  
  
section .text
```

SumDword:

```
push    EAX  
mov     EAX, [A]  
add     EAX, [B]  
mov     [D], EAX  
pop     EAX  
ret
```

3.2.4 Передача параметров через таблицу адресов

Пример 3.2d. Сумма элементов массива целых чисел.

```
section .data
ary      dw      5,6,1,7,3,4 ; массив целых чисел
count    dd      6          ; размер массива

section .bss
sum      resw   1          ; сумма элементов
tabl     resd   3          ; таблица адресов параметров

section .text
; формирование таблицы адресов параметров
mov      dword[tabl], ary
mov      dword[tabl+4], count
mov      dword[tabl+8], sum
mov      EBX, tabl

call     masculc
. . .
```



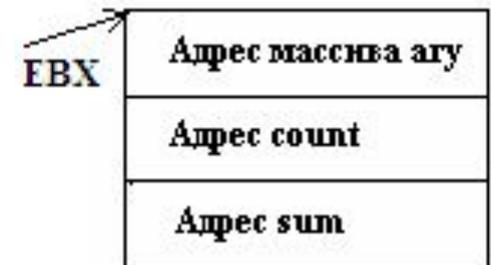
Процедура, получающая параметры через таблицу адресов

masculc:

```
push    AX      ; сохранение регистров
push    ECX
push    EDI
push    ESI
```

; использование таблицы адресов параметров

```
mov     ESI, [EBX] ; адрес массива
mov     EDI, [EBX+4] ; адрес размера
mov     ECX, [EDI] ; размер массива
mov     EDI, [EBX+8] ; адрес результата
```



Процедура, получающая параметры через таблицу адресов (2)

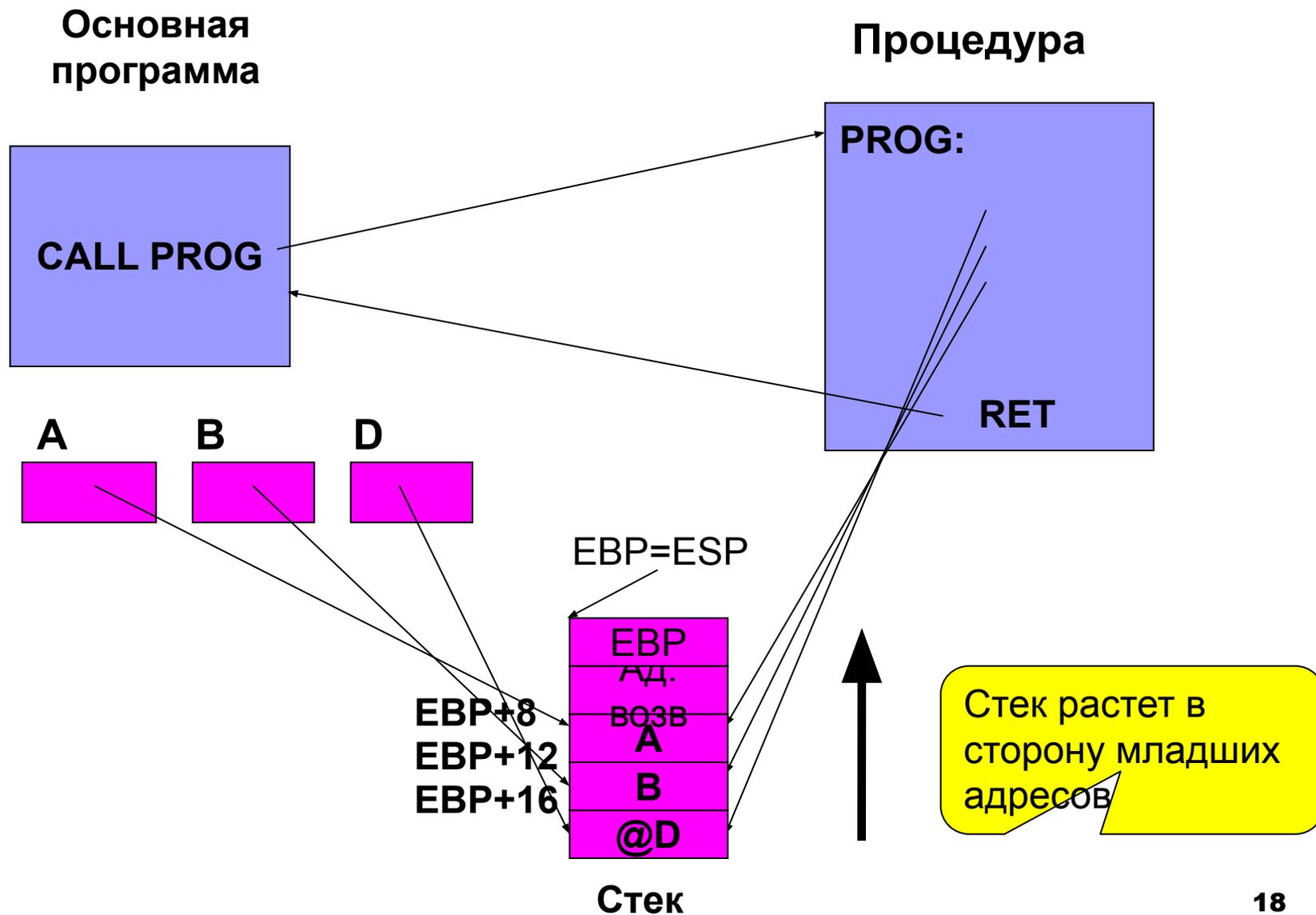
; суммирование элементов массива

```
        xor     AX, AX  
.cycle: add     AX, [ESI]  
        add     ESI, 2  
        loop    .cycle
```

; формирование результатов

```
        mov     [EDI], AX  
        pop     ESI      ; восстановление регистров  
        pop     EDI  
        pop     ECX  
        pop     AX  
        ret
```

3.2.5 Передача параметров через стек



Пример 3.2 е. Максимальное из двух чисел.

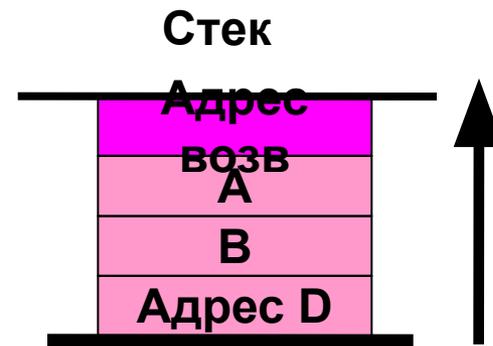
```
section .data
A dd 56
B dd 34
```

```
section .bss
D resd 1
```

```
section .text
lea EBX, [D] ; получение адреса результата
push EBX ; загрузка в стек адреса результата
push dword[B] ; загрузка в стек второго числа
push dword[A] ; загрузка в стек первого числа

call MaxDword
. . .
```

Получение
управления
процедурой



Исходное
состояние
стека

Процедура, получающая параметры через стек

MaxDword:

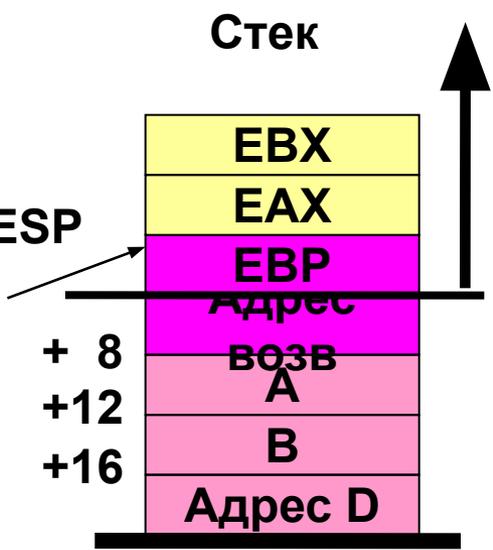
```
push    EBP
mov     EBP, ESP
push    EAX
push    EBX
mov     EBX, [EBP+16] ; адрес D
mov     EAX, [EBP+8]  ; A
cmp     EAX, [EBP+12] ; B
jg     .con
mov     EAX, [EBP+12]
.con:   mov     [EBX], EAX
pop     EBX
pop     EAX
mov     ESP, EBP
pop     EBP
ret     12
```

Пролог

Получение
управления
процедурой

Эпилог

Удаление из стека
области параметров



Исходное
состояние
стека

3.3 Организация ввода-вывода в консольном режиме

Осуществляются посредством вызова системных функций ввода-вывода.

1. 32-х разрядная программа

Вызов функций через прерывание «Диспетчер системных функций»:

```
int      80h
```

или с помощью команды `sysenter`:

```
push    .adret
```

```
push    ECX
```

```
push    EDX
```

```
push    EBP
```

```
mov     EBP,ESP
```

```
sysenter
```

```
.adret: ...
```

2. 64-х разрядная программа

Вызов функций через прерывание «Диспетчер системных функций»:

```
int      80h      или посредством      syscall
```

3.3 Организация ввода-вывода в консольном режиме

1. Параметры системной функции ввода строки:

32-х разрядная программа	64-х разрядная программа	Примечание
eax = 3	rax = 0	Номер функции
ebx = 0	rdi = 0	Номер файла <code>stdin</code> в таблице файлов
ecx = адрес	rsi = адрес	Адрес буфера ввода
edx = целое	rdx = целое	Размер буфера ввода

2. Параметры системной функции вывода:

32-х разрядная программа	64-х разрядная программа	Примечание
eax = 4	rax = 1	Номер функции
ebx = 1	rdi = 1	Номер файла <code>stdout</code> в таблице файлов
ecx = адрес	rsi = адрес	Адрес буфера вывода
edx = целое	rdx = целое	Размер буфера вывода

Пример 3.3 Программа извлечения квадратного корня из целого числа с точностью до целой части

Свойство суммы арифметической прогрессии:

$$1 = 1^2$$

$$1+3 = 4 = 2^2$$

$$1+3+5 = 9 = 3^2$$

$$1+3+5+7 = 16 = 4^2$$

Программа извлечения корня квадратного. Объявление данных

section .data

```
OutMsg  db      'Enter value <65024:',10
lenOut  equ     $-OutMsg
Otw     db      'Root ='
rez     db      '  ',10
lenOtw  equ     $-Otw
```

section .bss

```
InBuf   resb    10
lenIn   equ     $-InBuf
```

Программа извлечения корня квадратного.

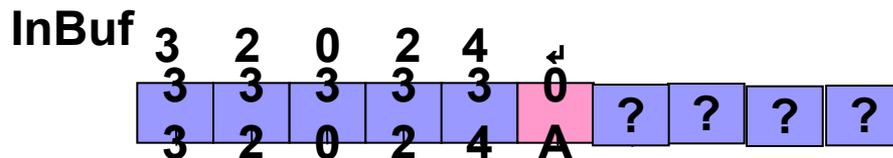
Ввод исходных данных

```
section .text

_start:

; ВЫВОД запроса на ввод
vvod:  mov     eax, 4
       mov     ebx, 1
       mov     ecx, OutMsg
       mov     edx, lenOut
       int    80h

; ВВОД ЧИСЛА
       mov     eax, 3
       mov     ebx, 0
       mov     ecx, InBuf
       mov     edx, lenIn
       int    80h
```



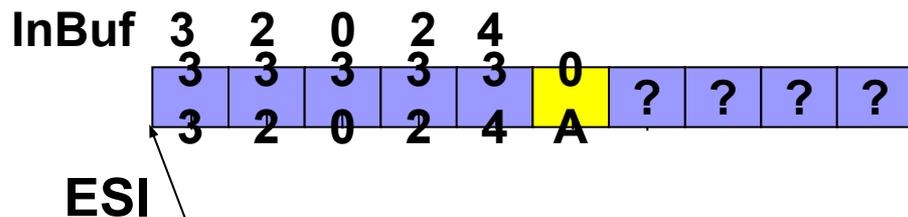
Программа извлечения корня квадратного (2)

; Преобразование

```

mov     BH, '9'
mov     BL, '0'
lea     ESI, [InBuf]
cld
xor     DI, DI
.cycle: lodsb
cmp     AL, 10
je      calc
cmp     AL, BL
jb      vvod
cmp     AL, BH
ja      vvod
sub     AL, 30h
cbw
push   AX
mov     AX, 10
mul    DI
pop     DI
add    AX, DI
mov    DI, AX
jmp    .cycle

```

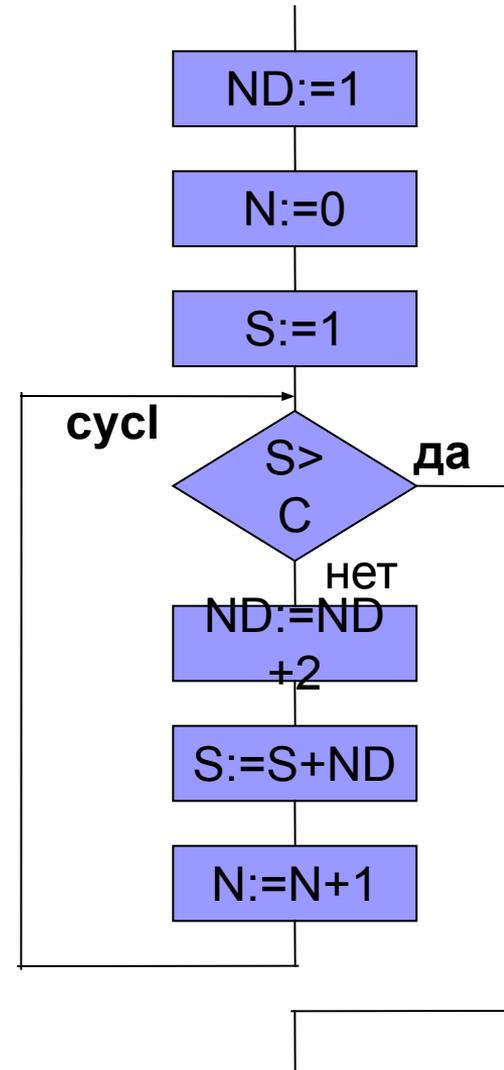


; обнуляем будущее число
; загружаем символ (цифру)
; если 0, то на вычисление
; сравниваем с кодом нуля
; "ниже" - на ввод
; сравниваем с кодом девяти
; "выше" - на ввод
; получаем цифру из символа
; расширяем до слова
; сохраняем в стеке
; заносим 10
; умножаем, результат в DX:AX
; в DI - очередная цифра
; в DI - накопленное число

Программа извлечения корня квадратного (3)

;Вычисление sqrt(dx#ax)

```
calc:    mov     BX,1
         mov     CX,0
         mov     AX,1 ; сумма
.cycle:  cmp     AX,DI
         ja     preobr
         add     BX,2
         add     AX,BX
         jc     vvod
         inc     CX
         jmp    .cycle
```



Программа извлечения корня квадратного (4)

; Преобразование числа в строку

```
preobr:  mov     AX,CX
         mov     EDI,2
         mov     BX,10

.cycle:  cwd             ; расширили слово до двойного
         div     BX             ; делим результат на 10
         add     DL,30h        ; получаем из остатка код
                                     ; цифры
         mov     [EDI+rez],DL ; пишем символ в
                                     ; выводимую строку
         dec     EDI          ; переводим указатель на
                                     ; предыдущую позицию
         cmp     AX,0         ; преобразовали все число?
         jne     .cycle
```

Программа извлечения корня квадратного (5)

```
; вывод результата  
mov     eax, 4  
mov     ebx, 1  
mov     ecx, 0tw  
mov     edx, len0tw  
int     80h
```

3.4 Связь разноязыковых модулей

Основные проблемы связи разноязыковых модулей:

- осуществление совместной компоновки модулей;
- организация передачи и возврата управления;
- передача данных в подпрограмму:
 - с использованием глобальных переменных,
 - с использованием стека (по значению и по ссылке),
- обеспечение возврата результата функции;
- обеспечение корректного использования регистров процессора.

Конвенции о связях WINDOW's

Конвенции о связи определяют правила передачи параметров.

№	Название в MASM32	Delphi Pascal	C++Builder	Visual C++	Порядок записи пар-в	Удаление пар-ров из стека	Использование регистров
1	PASCAL	pascal	<code>__ pascal</code>	-	прямой	процедура	-
2	C	cdecl	<code>__ cdecl</code>	<code>__ cdecl</code>	обратный	осн. прогр.	-
3	STDCALL	stdcall	<code>__ stdcall</code>	<code>__ stdcall</code>	обратный	процедура	-
4	-	register	<code>__ fastcall</code>	<code>__ fastcall</code>	обратный	процедура	до 3-х

Конвенции о связях WINDOW's (2)

- тип вызова: **NEAR**;
- модель памяти: (**FLAT**);
- пролог и эпилог – стандартные, текст зависит от конвенции и наличия локальных переменных:

- пролог:

```
push EBP
```

```
movEBP, ESP
```

```
[subESP, <Размер памяти локальных переменных>]
```

- эпилог:

```
movESP, EBP
```

```
popEBP
```

```
ret[<Размер области параметров>]
```

Конвенция о связях Linux

Операционные системы *Linux*, как и другие *Unix*-системы, использует единственную конвенцию, которая включена в стандартизованный двоичный интерфейс приложения (англ. *System V Application Binary Interface* или сокр. *System V ABI*).

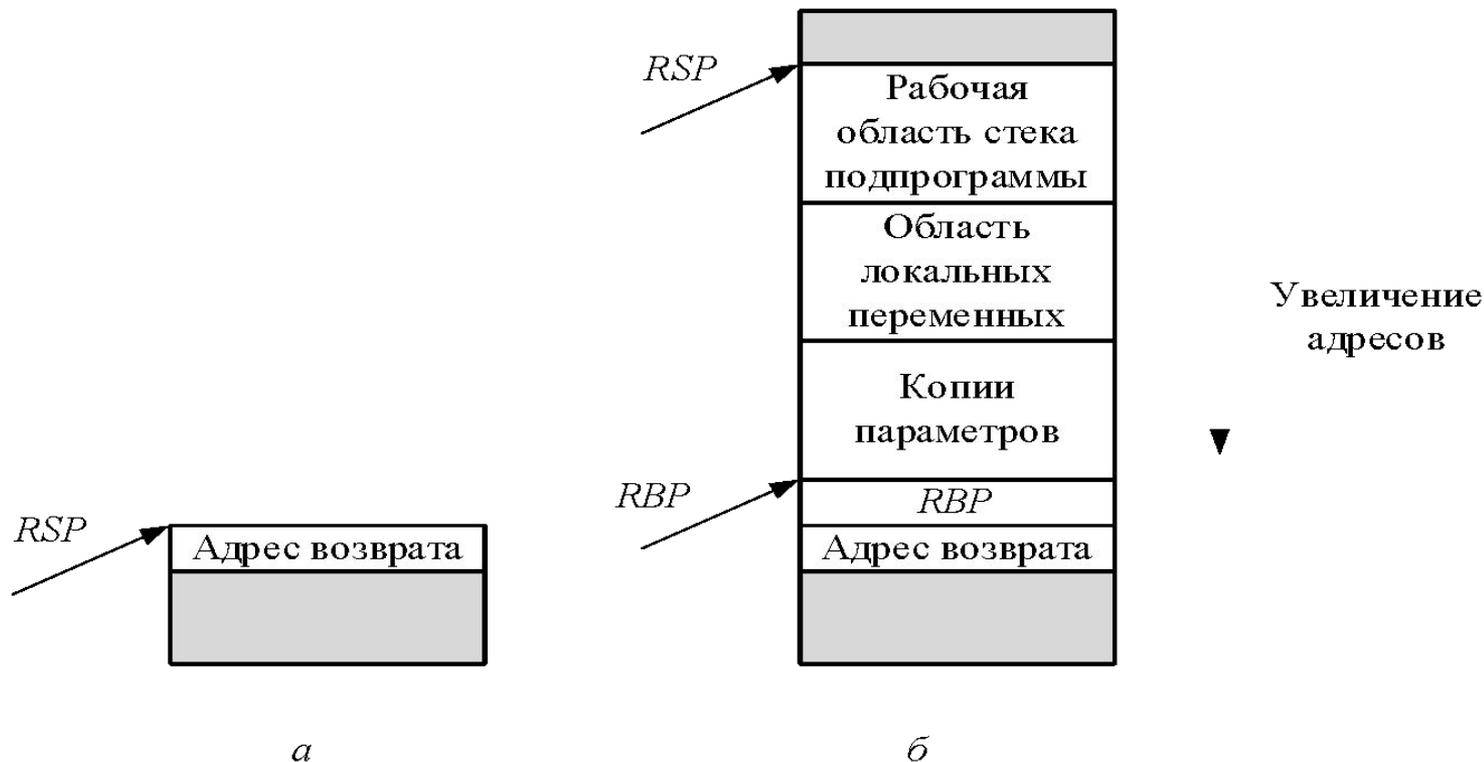
Указанная конвенция предполагает, что до 6-ти параметров передается в регистрах, остальные – в стеке в обратном порядке. Порядок занесения параметров в регистры прямой, т.е. первым записывается в регистр первый параметр и т.д.

В 32-х разрядной программе для передачи параметров используют регистры *EBX, ECX, EDX, ESI, EDI, EBP*; функции, за исключением функций, результатом которых является строка, возвращают результат в регистре *EAX*. Если функция возвращает строку, то адрес результата передается в регистре *EBX*, т.е. в первом регистре, используемом для передачи параметров.

В 64-х разрядной программе для передачи параметров используют регистры *RDI, RSI, RDX, R10, R8, R9*; функции, за исключением функций, результатом которых является строка, возвращают результат в регистре *RAX*. Если функция возвращает строку, то адрес результата передается в регистре *RDI*, т.е. в первом регистре, используемом для передачи параметров.

Структура стека 64-х разрядной программы:

а – в момент вызова подпрограммы;
б – во время работы подпрограммы



Область локальных переменных и область, содержащая копии параметров, адресуются относительно адреса в регистре *RBP*. Рабочая область стека программы адресуется содержимым регистра *RSP*.

3.4.1 *Lazarus (Free PASCAL) – NASM*

При написании программы на Free Pascal, вызывающей подпрограмму на ассемблере следует:

- в модуле на Free Pascal процедуры и функции, реализованные на ассемблере, объявить и описать как внешние **external**, например:
`procedure ADD1 (A,B:integer; Var C:integer); external;`
- если некая подпрограмма на Free Pascal должна будет вызываться из модуля на ассемблере, то ее прототип поместить в интерфейсную секцию, т.е. сделать доступной из других модулей:

interface

```
    procedure Print(n:integer);
```

и уточнить ее внутреннее имя в объектном коде модуля;

- отключить оптимизацию при компиляции модуля на Free Pascal (см. настройки среды в методических указаниях по выполнению лабораторных работ);

Lazarus (Free PASCAL) – NASM (2)

- для работы с модулем на ассемблере добавить в Lazarus инструмент (меню **Сервис/Настроить внешние средства**), назначив в качестве инструмента ассемблер `nasm`:

Заголовок: **NASM**

Имя файла программы: `/usr/bin/nasm`

Параметры: **-f elf64 \$EdFile()**

-I \$Path(EdFile())\$NameOnly(\$EdFile()).lst

-o \$Path(\$EdFile())\$NameOnly(\$EdFile()).o

Рабочий каталог: **\$(ProjPath)**

Также выбираем галочкой пункт «Искать сообщения *FPC* в вводе».

- модуль на ассемблере создать в редакторе среды, транслировать и подключить полученный объектный модуль в секцию реализации модуля Free Pascal:

```
implementation
```

```
{ $1 <Имя объектного модуля> }
```

Free PASCAL – NASM

- СОВМЕСТИМОСТЬ ЧАСТО ИСПОЛЬЗУЕМЫХ ДАННЫХ:

Word – 2 байта,

Byte, Char, Boolean – 1 байт,

Integer, Pointer – 4 байта,

массив – располагается в памяти по строкам,

строка (**shortstring**) – содержит байт длины и далее символы;

- параметры передаются через стек:

- по значению – в стеке копия значения,

- по ссылке – в стеке указатель на параметр;

- результаты функций возвращаются через регистры:

- байт, слово – в **AX**,

- двойное слово, указатель – в **EAX (RAX)**,

- строка – адрес в **EBX (RDI)**.

Пример 3.4 Free PASCAL – NASM.

Программа на Free Pascal

Описание на Free Pascal:

Implementation

```
{$1 m_add.o}
```

```
procedure ADD1 (A,B:integer; Var C:integer);external;
```

Вызов процедуры:

```
Procedure Form1.Button1Click (Sender:TObject) ;
```

```
var A,B,C:integer;
```

```
Begin
```

```
    A:=strtoint(Edit1.text) ;
```

```
    B:=strtoint(Edit2.text) ;
```

```
    ADD1 (A,B,C) ;
```

```
    Edit3.text:=inttostr(C) ;
```

```
End;
```

Модуль на ассемблере

Определение суммы двух целых чисел.

```
procedure ADD1 (A,B:integer; Var C:integer);external;
```

Модуль на ассемблере:

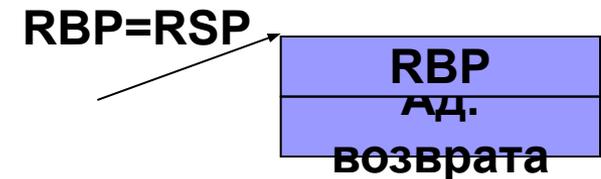
```
section .text  
  
global UNIT1_$$_ADD1$LONGINT$LONGINT$LONGINT  
UNIT1_$$_ADD1$LONGINT$LONGINT$LONGINT:  
  
push    rbp  
mov     rbp, rsp  
add     esi, edi  
mov     [rdx], esi  
mov     rsp, rbp  
pop     rbp  
ret
```

Параметры:

A -> RDI

B -> RSI

адрес C -> RDX



Lazarus IDE v2.0.12 - project1 (идёт отладка ...)

Файл Правка Поиск Вид Код Проект Запуск Пакет Сервис Окно Справка

Ассемблер

```

0000000000460B1C 8945e8      mov     DWORD PTR [rbp-0x18],eax
unit1.pas:67  SUM(a,b,c);
0000000000460B1F 488d55e4    lea    rdx,[rbp-0x1c]
0000000000460B23 8b75e8      mov     esi,DWORD PTR [rbp-0x18]
0000000000460B26 8b7dec      mov     edi,DWORD PTR [rbp-0x14]
0000000000460B29 e8a2010000 call   0x460cd0 <SUM>
unit1.pas:68  Edit5.Text:=inttostr(c);
0000000000460B2E 488dbd70ffff lea    rdi,[rbp-0x90]
0000000000460B35 e8e679fcff call   0x428520 <fpc_ansistr_decr_ref>
0000000000460B3A 8b45e4      mov     eax,DWORD PTR [rbp-0x1c]
0000000000460B3D 89856cffff  mov     DWORD PTR [rbp-0x94],eax
0000000000460B43 48b9ff0000000000 movabs rcx,0xff
0000000000460B4D 488d956cfefff lea    rdx,[rbp-0x194]
0000000000460B54 48c7c6fffffff mov     rsi,0xffffffffffffffff
0000000000460B5B 4863bd6cfffff movsxd rdi,DWORD PTR [rbp-0x94]

```

67: 1 BCT /home/galina/Ex_nasm/p...



Пример 3.5 Процедура без параметров

Увеличение каждого элемента массива AAA на 5.

1. Массив **AAA** должен быть объявлен в секции **Interface**:

```
var AAA:array[1..5] of byte; public; //или cvar;
```

Это сделает его имя неизменяемым.

2. Процедуру следует описать, как внешнюю:

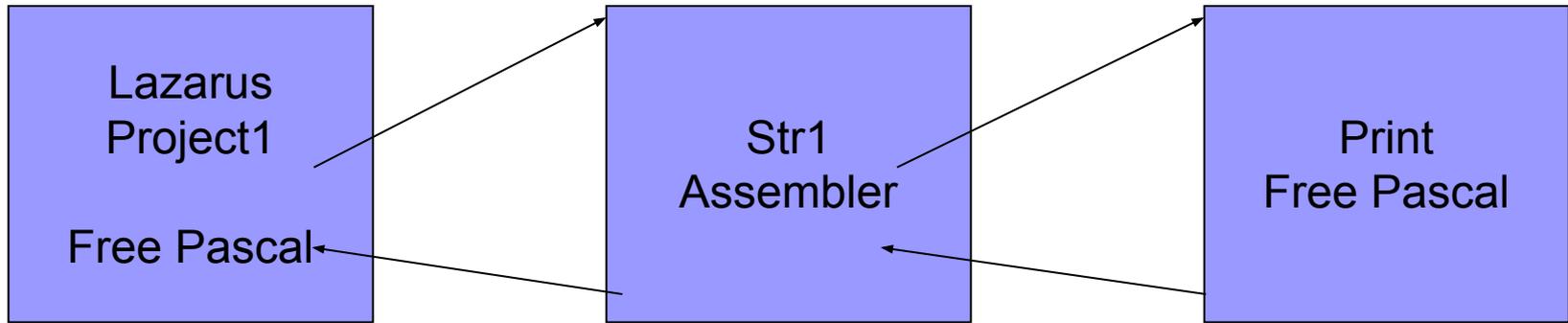
```
{$1 text3.o}
```

```
procedure Array_add; external;
```

3. Текст модуля text3.asm:

```
section .text  
global UNIT1_$$_ARRAY_ADD  
extern AAA ; описание внешнего имени  
UNIT1_$$_ARRAY_ADD:  
mov eax,AAA ; адрес массива AAA  
mov ecx,5  
.cycle: add byte[eax],5  
inc eax  
loop .cycle  
ret
```

Пример 3.6 Free Pascal – Assembler – Free Pascal



implementation

```
{$1 text.o}
```

```
procedure Str1(S:ShortString,St:ShortString);external;
```

```
procedure Form1.Button1Click(Sender:TObject);
```

```
Var S,St:shortstring;
```

```
Begin
```

```
    S:=Edit1.text;
```

```
    Str1(S,St);            // вызов подпрограммы на ассемблере
```

```
                          // S -> EDI, St -> ESI
```

```
    Edit2.text:=St;
```

```
End;
```

Процедура Print для вызова из ассемблера

```
Unit Unit1;  
interface  
    ...  
    procedure Print(n:integer); // внешняя подпрограмма  
  
implementation  
    ...  
    procedure Print(n:integer);  
    begin  
        Form1.Edit3.text:=inttostr(n);  
    end;
```

Внимание! Внутреннее имя процедуры в объектном модуле:

UNIT1_\$\$_PRINT\$LONGINT

Процедура на ассемблере

```
procedure Str1(S:ShortString,St:ShortString;external;
```

```
    global UNIT1_$$_STR1$SHORTSTRING$SHORTSTRING
```

```
    extern UNIT1_$$_PRINT$LONGINT
```

```
    section .text
```

```
UNIT1_$$_STR1$SHORTSTRING$SHORTSTRING:
```

```
    push rbp
```

```
    mov rbp, rsp
```

```
    push rcx
```

```
    push rax
```

```
    mov byte[rsi], 3 ; запись длины строки
```

```
    mov rcx, 3
```

```
    inc rdi ; пропуск байта длины строки
```

Параметры:
адрес S ->RDI
адрес St ->RSI

Процедура на ассемблере (2)

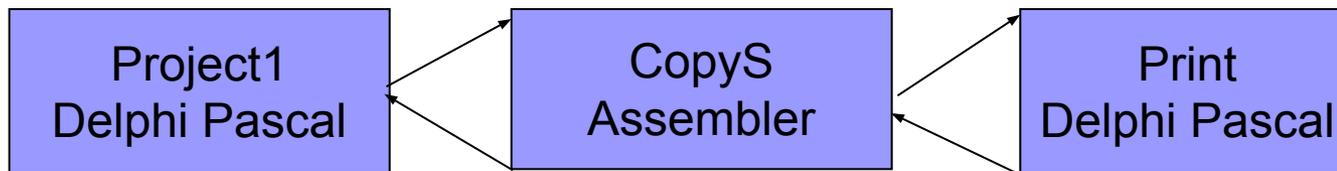
```
        xchg RSI,RDI
.cycle movsb
        inc rsi
        loop .cycle
; вызов подпрограммы на Free Pascal
        mov rdi,3
        call UNIT1_$$_PRINT$LONGINT

        pop rax
        pop rcx
        mov rsp,rbp
        pop rbp
        ret
```

3.5.2 Локальные данные подпрограмм

Паскаль не позволяет создавать в подпрограммах глобальные переменные, поэтому в подпрограммах необходимо работать с локальными данными, размещаемыми в стеке.

Пример 3.7. Организация локальных переменных без использования директив ассемблера



Подпрограмма на ассемблере:

- получает строку,
- копирует в локальную память,
- затем копирует из лок. памяти в результат,
- вызывает Паскаль для вывода длины строки.

Для работы с локальными данными
будем использовать структуры.

Структура

Структура – шаблон с описаниями форматов данных, который можно накладывать на различные участки памяти, чтобы затем обращаться к полям этих участков памяти с помощью имен, определенных в описании структуры.

Формат описания структуры:

```
struc <Имя структуры>  
    <Описание полей>  
endstruc
```

где <Описание полей> – любой набор псевдокоманд определения переменных или вложенных структур.

Пример:

```
struc Student  
    Family      resb 20    ; Фамилия студента  
    Name        resb 15 dup(' ') ; Имя  
    Birthdata   resb 8     ; Дата рождения  
endstruc
```

Последовательность директив описывает, но **не размещает** в памяти структуру данных!!!

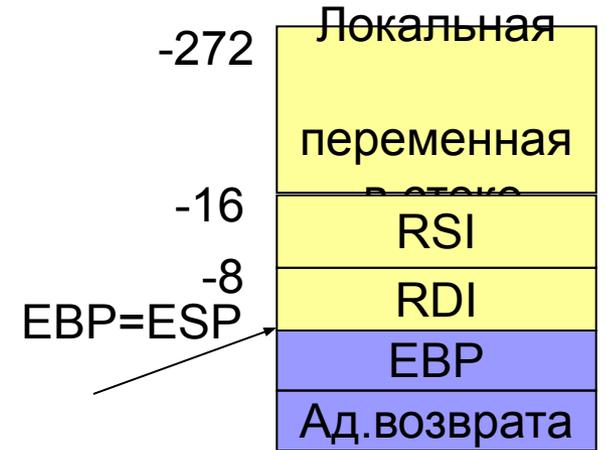
Пример 3.7. Создание локальных переменных

```
interface
procedure Print(n:integer);
implementation
{$1 text.o}
function CopyS(St:ShortString):
    ShortString; external;

procedure Print(n:integer);
begin Form1.Edit3.Text:=inttostr(n);end;

procedure TForm1.Button1Click(Sender: TObject);
Var S,St:ShortString;
begin St:=Edit1.Text;
    S:=CopyS(St);
    Edit2.Text:=S;

end;
```



Адрес S -> RSI
Адрес St -> RDI

Пример 3.7. Создание локальных переменных

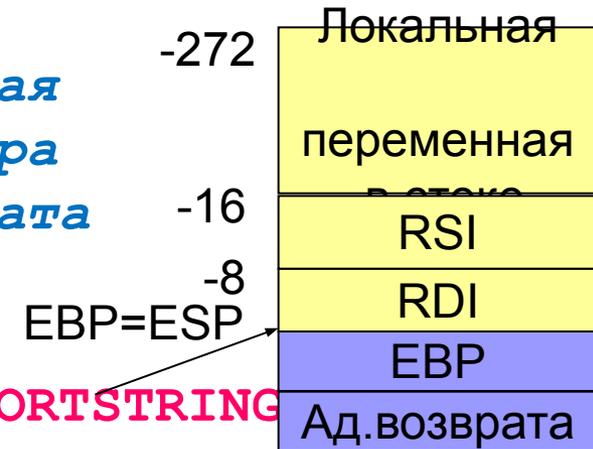
```

struc A           ; описание структуры
    .S    resb    256 ; локальная переменная
    .RSI  resq    1  ; копия адреса параметра
    .RDI  resq    1  ; копия адреса результата
endstruc         ; завершение описания

    section .text
global  UNIT1_$$_COPY$$SHORTSTRING$$SHORTSTRING
extern  UNIT1_$$_PRINT$LONGINT

UNIT1_$$_COPY$$SHORTSTRING$$SHORTSTRING:
    push   RBP           ; сохранение RBP
    mov    RBP,RSP       ; загрузка нового RBP
    sub   RSP,272        ; место под лок. переменные
    mov   [RBP-272+A.RSI],RSI ; адрес строки S
    mov   [RBP-272+A.RDI],RDI ; адрес строки St
    lea  RDI,[RBP-272+A.S] ; обращение к лок.п.
    xor   RAX,RAX
    lodsb                    ; загрузка длины строки
    stosb                    ; сохранение длины строки

```



Адрес S -> RSI
Адрес St -> RDI

Пример 3.7. Создание локальных переменных

```
mov     RCX, RAX    ; загрузка счетчика
cld
rep movsb          ; копирование строки
lea     RSI, [RBP-272+A.S] ; загрузка адреса копии
mov     RDI, [RBP-272+A.RDI]; загрузка адр. рез-та
lods b             ; загрузка длины строки
stos b            ; сохранение длины строки
mov     RCX, RAX    ; загрузка счетчика
rep movsb          ; копирование строки
mov     RDI, RAX    ; длина строки
call    UNIT1_$$_PRINT$LONGINT ; вывод длины
                                           ; строки
mov     RSP, RBP    ; удаление лок. переменных
pop     RBP         ; восстановление старого EBP
ret           ; возврат в программу
```

3.5.2 Qt Creator (CLang) – NASM

При написании программы на C++, вызывающей подпрограмму на ассемблере следует:

- используя пункт меню **Инструменты/Внешние/Настроить...**, добавить внешний инструмент - транслятор ассемблера: сначала добавляем категорию Ассемблер, потом утилиту NASM:

Описание: **NASM**.

Программа: `/usr/bin/nasm`

Параметры: `-f elf64 %{\CurrentDocument:FilePath}`

`-o %{\CurrentRun:Executable:Path}/
%{\CurrentDocument:FileName}.o`

`-I %{\CurrentDocument:Path}/
%{\CurrentDocument:FileName}.lst`

Рабочий каталог: (оставляем пусто)

Стандартный вывод: **Показать в консоли**

Вывод ошибок: **Показать в консоли**

- в модуле на C++ подключаемые процедуры и функции на ассемблере объявить внешними **extern**, например:

```
extern void add1(int a,int b,int *c);
```

Лишние пробелы
внутри имен
файлов убрать

Qt Creator (CLang) – NASM (2)

- подключить к проекту файл – результат ассемблирования с расширением `.o` посредством добавления к файлу проекта `.pro` строки `OBJECTS`, например:

```
OBJECTS += text.o
```

- обеспечить включение исходного модуля на ассемблере в дерево навигатора Qt Creator посредством добавления к файлу проекта строки `DISTFILES`, например:

```
DISTFILES += text.asm
```

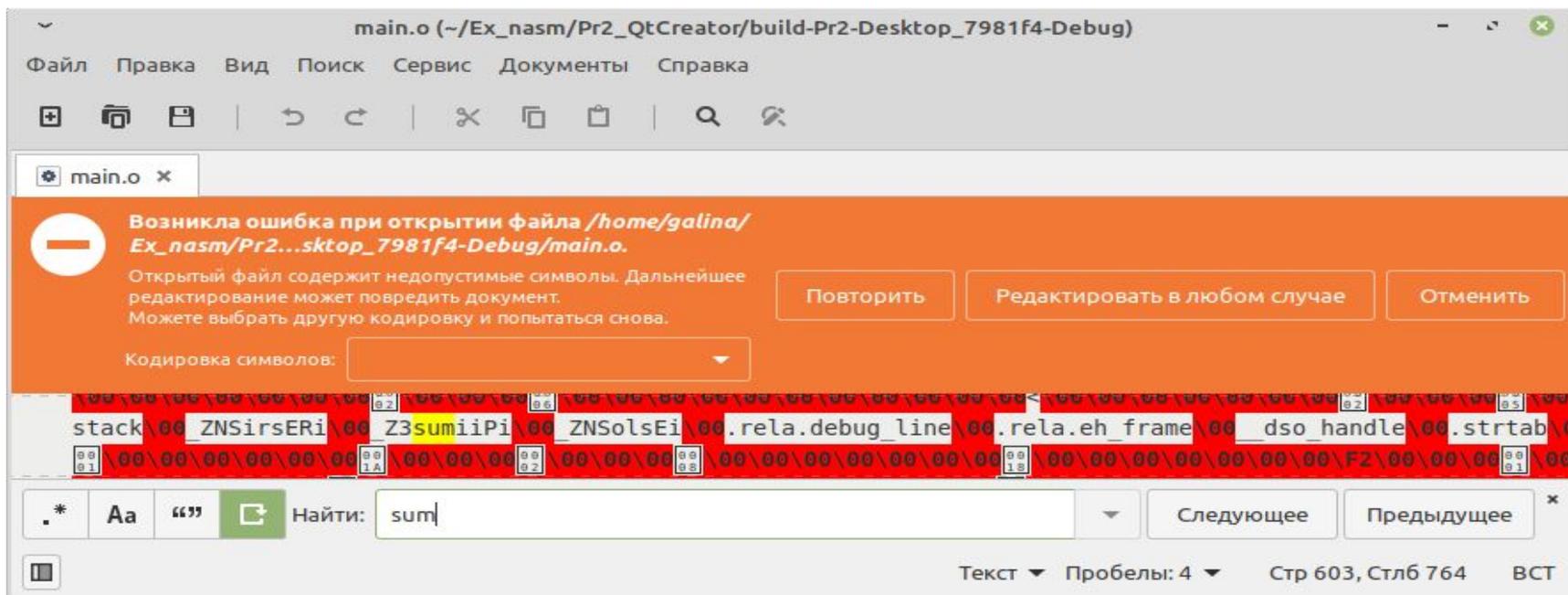
- отключить оптимизацию в процессе компиляции программы посредством добавления к файлу проекта строк `CONFIG`:

```
CONFIG ~= s/-O[0123s]//g
```

```
CONFIG += -O0
```

- задать подпрограмме на ассемблере требуемое компилятором имя, имя следует посмотреть открыв в любом текстовом редакторе содержимое файла объектного модуля вызывающей программы `.o`
- скопировать объектный модуль ассемблера в папку, в которой находится объектный модуль C++, или исправить настройки внешнего инструмента на заданные на слайде 53!!!

Поиск имени подпрограммы в объектном модуле вызывающей программы (редактор хед)



$_Z$ <Целое число><Имя подпрограммы><Описание параметров>,

где <Целое число> – порядковый номер подпрограммы;

<Описание параметров> — строка кодов:

i — параметр целого типа;

c — параметр символьного типа;

Pi — указатель на параметр целого типа;

Pc — указатель на символ;

PKc — указатель на константный символ.

Пример 3.8 Связь Qt Creator - NASM

Программа определения суммы двух целых чисел.

Программа на C++ осуществляет ввод-вывод и вызов подпрограммы:

```
#include <iostream>
```

```
// прототип подпрограммы на ассемблере:
```

```
extern void sum(int x,int y,int *p);
```

```
int main()
```

```
{
```

```
    int d,b,c;
```

```
    std::cin >> d >> b;
```

```
    sum(d,b,&c);
```

```
    std::cout << c << std::endl;
```

```
    return 0;
```

```
}
```

Пример 3.8 Qt Creator – NASM (2)

```
extern void sum(int x,int y,int *p);
```

Текст подпрограммы на ассемблере:

Параметры:

x -> RDI

y -> RSI

*p -> RDX

```
global _Z3sumiiPi
```

```
section .text
```

```
_Z3sumiiPi:
```

```
push    rbp
```

```
mov     rbp, rsp
```

```
add     esi, edi
```

```
mov     [rdx], esi
```

```
mov     rsp, rbp
```

```
pop     rbp
```

```
ret
```

Пример 3.9 Создание внешних переменных в процедуре на ассемблере

Программа на C++:

```
extern void sum1(int a, int b);  
extern int fff;  
  
int main()  
{  
    int a, b;  
    std::cout << "Enter a and b:";  
    std::cin >> a >> b;  
    sum1(a, b);  
    std::cout << "fff=" << fff;  
    return 0;  
}
```

Пример 3.9 Создание внешних переменных в процедуре на ассемблере (2)

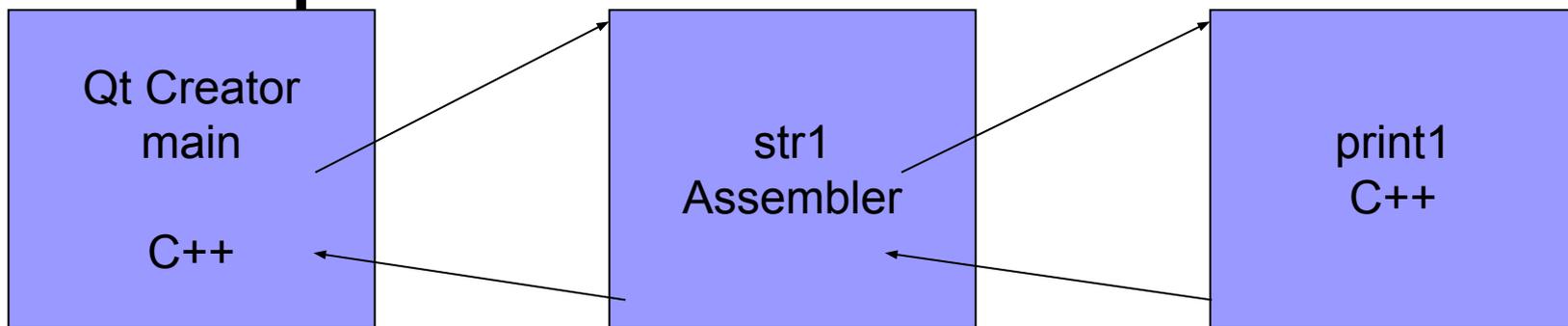
Текст процедуры на ассемблере:

```
    global _Z4sum1ii
    global fff
    section .bss
fff    resd    1
    section .text
_Z4sum1ii:
    push rbp
    mov rbp, rsp

    add esi, edi
    mov [fff], esi

    mov rsp, rbp
    pop rbp
    ret
```

Пример 3.10 Вызов подпрограммы на C++ из ассемблера



Программа на C++ (вводит строку и выводит результат):

```
#include <string.h>
#include <iostream>
extern void str1(const char *s, char *sr);
int main()
{
    char s[20], sr[4];
    std::cout << "Enter s:";
    std::cin >> s;
    str1(s, sr);
    std::cout << sr << std::endl;    return 0;}
```

Пример 3.10 Вызов подпрограммы на C++ из ассемблера (2)

Подпрограмма на C++, вызываемая из процедуры на ассемблере, выводит длину строки на экран:

```
void print1(int n)
{
    std::cout<<n<<' \n' ;
}
```

Пример 3.10 Вызов подпрограммы на C++ из ассемблера (3)

Процедура на ассемблере (копирует 1-й, 3-й и 5-й символы из заданной строки в строку результата):

```
global    _Z4str1PKcPc
extern    _Z6printli
section   .text
_Z4str1PKcPc:
    ; пролог
    push rbp
    mov  rbp, rsp
    ; сохранение содержимого регистров
    push rcx
    push rax
```

Содержимое регистра RAX на самом деле можно не сохранять, поскольку он может быть использован для возврата результата при вызове функции.

Пример 3.10 Вызов подпрограммы на C++ из ассемблера (4)

```
    mov ecx,3
    xchg rdi,rsi
.cycle:
    movsb
    inc rsi
    loop .cycle
    mov byte[rsi],0
; вызов процедуры на C++
    mov rdi,3
    call _Z6printli
; восстановление регистров
    pop rax
    pop rcx
; эпилог
    mov rsp,rbp
    pop rbp
    ret
```