

Лабораторная работа №9. Программирование цикла. Обработка

Архитектура ЭВМ

Осокин Георгий Иванович НММбд-02-22

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Вывод на экран аргументов командной строки	10
2.2	Программа выводющая сумму аргументов	11
2.2.1	Умножение аргументов командной строки	13
3	Задание для самостоятельной работы.	16
4	Выводы	19

Список иллюстраций

2.1	Вывод программы с аргументом 10	8
2.2	Исполнение измененный программы с число 10	9
2.3	Вывод измененной программы аргументом 10	10
2.4	Запуск lab9-2	11
2.5	Исполнение lab9-3	13
2.6	Вывод программы lab9-3.mult	15
3.1	Содержимое файла my-funciton.asm	18
3.2	Вывод программы lab9-4	18

Список таблиц

1 Цель работы

Приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки.

2 Выполнение лабораторной работы

Создадим каталог для выполнения лабораторной, перейдем в него и введем листинг 9.1 в файл

```
%include 'in_out.asm'

SECTION .data

msg1 db 'Введите N: ',0h

SECTION .bss

N: resb 10

SECTION .text

global _start

_start:
; ----- Вывод сообщения 'Введите N: '

mov eax,msg1
call sprint
```

```

; ----- Ввод 'N'
mov ecx, N
mov edx, 10
call sread

; ----- Преобразование 'N' из символа в число

mov eax,N
call atoi
mov [N],eax
; ----- Организация цикла

mov ecx,[N] ; Счетчик цикла, `ecx=N`
label:

mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`

loop label ; `ecx=ecx-1` и если `ecx` не '0'
            ; переход на `label`
call quit

```

Запустим код и введем аргумент

```
> sh run.sh lab9-1
Введите N: 10
10
9
8
7
6
5
4
3
2
1

work/arch-pc/lab09
> 
```

Рис. 2.1: Вывод программы с аргументом 10

Добавим строчку с уменьшением `ecx` на 1

```
sub ecx, 1 ; уменьшаем ECX на 1
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`
```

Исполним программу.


```
work/arch-pc/lab09 took 36s
> sh run.sh lab9-1
Введите N: 10
9
7
5
3
1
```

Рис. 2.2: Исполнение измененный программы с число 10

Обернем, данную часть кода в push, pop. И теперь посмотрим на результат.

```
push ecx
sub ecx, 1
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`
pop ecx
```

```
> sh run.sh lab9-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
work/arch-pc/lab09
```

Рис. 2.3: Вывод измененной программы аргументом 10

2.1 Вывод на экран аргументов командной строки

Скопируем в файл `lab9-2.asm` листинг 9.2

```
%include 'in_out.asm'
```

```
SECTION .text
```

```
global _start
```

```
_start:
```

```
    pop ecx ; Извлекаем из стека в `ecx` количество
             ; аргументов (первое значение в стеке)
```

```
    pop edx ; Извлекаем из стека в `edx` имя программы
             ; (второе значение в стеке)
```

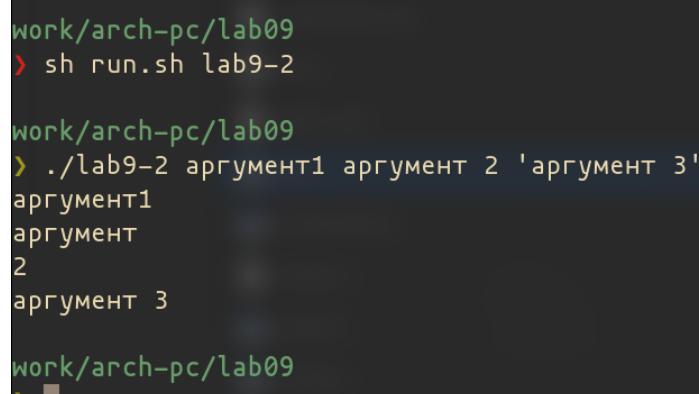
```
    sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
                 ; аргументов без названия программы)
```

```

next:
    cmp ecx, 0 ; проверяем, есть ли еще аргументы
    jz _end ; если аргументов нет выходим из цикла
                ; (переход на метку `_end`)
    pop eax ; иначе извлекаем аргумент из стека
    call sprintf ; вызываем функцию печати
    loop next ; переход к обработке следующего
                ; аргумента (переход на метку `next`)
_end:
    call quit

```

Скомпилируем и запустим файл.



```

work/arch-pc/lab09
> sh run.sh lab9-2

work/arch-pc/lab09
> ./lab9-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3

work/arch-pc/lab09

```

Рис. 2.4: Запуск lab9-2

Программа вывела 4 аргумента и вывела соответствующие строки.

2.2 Программа выводящая сумму аргументов

Скопируем программу из листинга 9.3 в lab9-3.asm

```

#include 'in_out.asm'

SECTION .data
msg db "Результат: ",0

SECTION .text
global _start
_start:

    pop ecx ; Извлекаем из стека в `ecx` количество
              ; аргументов (первое значение в стеке)
    pop edx ; Извлекаем из стека в `edx` имя программы
              ; (второе значение в стеке)
    sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
              ; аргументов без названия программы)
    mov esi, 0 ; Используем `esi` для хранения
              ; промежуточных сумм

next:

    cmp ecx,0h ; проверяем, есть ли еще аргументы
    jz _end ; если аргументов нет выходим из цикла
              ; (переход на метку `_end`)
    pop eax ; иначе извлекаем следующий аргумент из стека
    call atoi ; преобразуем символ в число
    add esi,eax ; добавляем к промежуточной сумме
              ; след. аргумент `esi=esi+eax`
    loop next ; переход к обработке следующего аргумента

_end:

    mov eax, msg ; вывод сообщения "Результат: "
    call sprint
    mov eax, esi ; записываем сумму в регистр `eax`

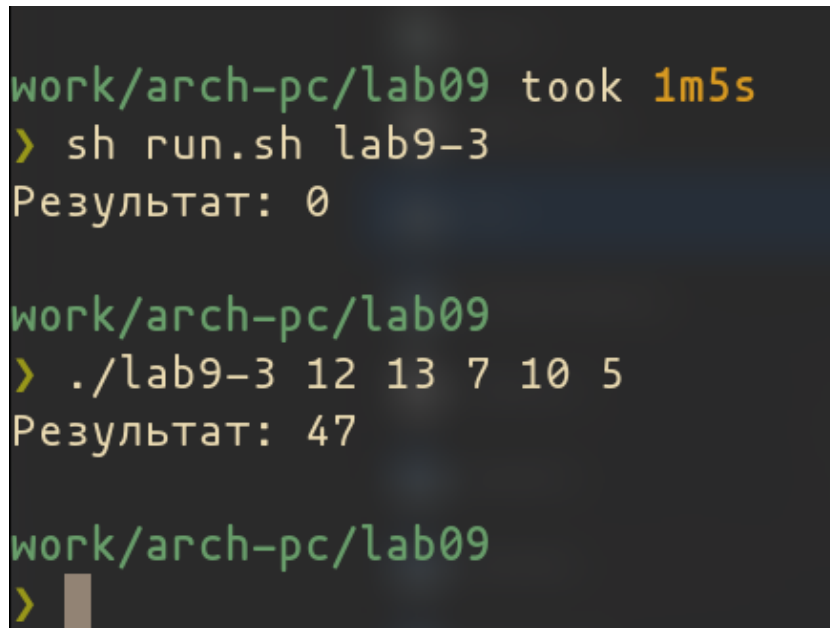
```

`call iprintLF` ; печать результата

`call quit` ; завершение программы

Запустим программу с аргументами 12 13 7 10 5

($12 + 13 + 7 + 10 + 5 = 47$)



```
work/arch-pc/lab09 took 1m5s
> sh run.sh lab9-3
Результат: 0

work/arch-pc/lab09
> ./lab9-3 12 13 7 10 5
Результат: 47

work/arch-pc/lab09
>
```

Рис. 2.5: Исполнение lab9-3

2.2.1 Умножение аргументов командной строки

Скопируем содержимое lab9-3.asm в lab9-3.mult.asm.

Изменим программу так, что бы она не складывала, а изменяла аргументы командной строки.

```
%include 'in_out.asm'

SECTION .data
msg db "Результат: ",0

SECTION .text
```

global _start

_start:

```
pop ecx ; Извлекаем из стека в `ecx` количество  
          ; аргументов (первое значение в стеке)  
pop edx ; Извлекаем из стека в `edx` имя программы  
          ; (второе значение в стеке)  
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество  
          ; аргументов без названия программы)  
mov esi, 1 ; Используем `esi` для хранения  
          ; промежуточных сумм
```

next:

```
cmp ecx,0h ; проверяем, есть ли еще аргументы  
jz _end ; если аргументов нет выходим из цикла  
          ; (переход на метку `_end`)  
pop eax ; иначе извлекаем следующий аргумент из стека  
call atoi ; преобразуем символ в число
```

```
push ecx
```

```
push eax
```

```
mov ecx, eax
```

```
mov eax, esi ; eax = res
```

```
mul ecx ; eax = eax * ecx
```

```
mov esi, eax ; res = eax
```

```
pop eax
```

```
pop ecx
```

loop next ; переход к обработке следующего аргумента

_end:

mov **eax**, msg ; вывод сообщения "Результат: "

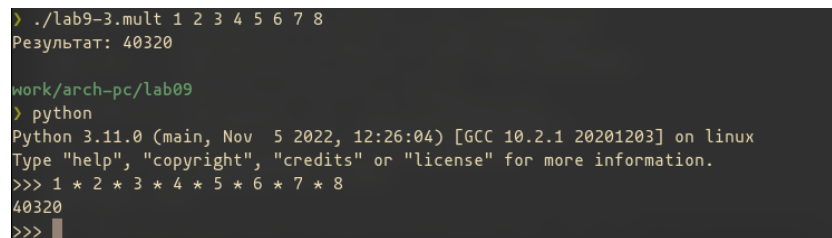
call sprint

mov **eax**, **esi** ; записываем сумму в регистр `eax`

call iprintLF ; печать результата

call quit ; завершение программы

Запустим программу и введем аргументы 1 2 3 4 5 5 6 7 8



```
> ./lab9-3.mult 1 2 3 4 5 5 6 7 8
Результат: 40320

work/arch-pc/lab09
> python
Python 3.11.0 (main, Nov 5 2022, 12:26:04) [GCC 10.2.1 20201203] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 1 * 2 * 3 * 4 * 5 * 6 * 7 * 8
40320
>>>
```

Рис. 2.6: Вывод программы lab9-3.mult

С помощью интерпретатора python убедимся, что результат вычислений верный.

3 Задание для самостоятельной работы.

Так как наш вариант **18**, напомним программу которая будет вычислять сумму функций $f(x) = 5x + 17$ от значений аргументов, введенных в командной строке.

```
%include 'in_out.asm'
```

```
%include 'my-function.asm'
```

```
SECTION .data
```

```
func db "Функция: f(x) = 5x + 17: ",0
```

```
msg db "Результат: ",0
```

```
SECTION .text
```

```
global _start
```

```
_start:
```

```
    pop ecx    ; Извлекаем из стека в `ecx` количество  
                ; аргументов (первое значение в стеке)
```

```
    pop edx    ; Извлекаем из стека в `edx` имя программы  
                ; (второе значение в стеке)
```

```
    sub ecx,1  ; Уменьшаем `ecx` на 1 (количество  
                ; аргументов без названия программы)
```

```
    mov esi, 0 ; Используем `esi` для хранения  
                ; промежуточных сумм
```



```
mov eax, func
```

```
call sprintf
```

```
next:
```

```
cmp ecx, 0h ; проверяем, есть ли еще аргументы
```

```
jz _end ; если аргументов нет выходим из цикла  
; (переход на метку `_end`)
```

```
pop eax ; иначе извлекаем следующий аргумент из стека
```

```
call atoi ; преобразуем символ в число
```

```
call magic_function
```

```
add esi, eax ; добавляем к промежуточной сумме
```

```
loop next ; переход к обработке следующего аргумента
```

```
_end:
```

```
mov eax, msg ; вывод сообщения "Результат: "
```

```
call sprintf
```

```
mov eax, esi ; записываем сумму в регистр `eax`
```

```
call iprintLF ; печать результата
```

```
call quit ; завершение программы
```

Мы подключили файл `my-function.asm` его содержимое - это наша функция.

```

File: my-function.asm
1      ; f(x) = 17 + 5x
2      ; eax = x
3      ; eax = res
4      magic_function:
5
6      push ebx
7      push ecx
8      push edx
9
10     mov ecx,5
11     mul ecx
12     add eax, 17
13
14     pop edx
15     pop ecx
16     pop ebx
17
18
19     ret
20
21

```

Рис. 3.1: Содержимое файла my-funciton.asm

Скомпилируем и запустим программу с аргументами 1 2 3 4.

Убедимся в правильности вычислений с помощью интерпретатора python.

```

work/arch-pc/lab09 took 6s
4% > ./lab9-4 1 2 3 4
Функция: f(x) = 5x + 17:
Результат: 118

work/arch-pc/lab09
4% > python
Python 3.11.0 (main, Nov 5 2022, 12:26:04) [GCC 10.2.1 20201203] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 1 * 5 + 17 + 2*5 + 17 + 3 * 5 + 17 + 4 * 5 + 17
118
>>>
0 > gdb 1 > python

```

Рис. 3.2: Вывод программы lab9-4

4 Выводы

Мы приобрели навыки написания программ с использованием циклов и обработкой аргументов командной строки. Мы написали программы, которые умножают аргументы друг на друга, складывают и складывают значения функции от этих аргументов.