

Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки.

Славинский Владислав Вадимович

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выполнение задания для самостоятельной работы	17
4	Вывод	19

Список иллюстраций

2.1	Создание lab8-1	6
2.2	Код программы	7
2.3	Запуск	7
2.4	Замена	8
2.5	Запуск с нечетным числом	9
2.6	Запуск с четным числом	10
2.7	Программа	11
2.8	Запуск	12
2.9	Код программы	13
2.10	Запуск	13
2.11	Код программы	14
2.12	Запуск	14
2.13	Код программы	15
2.14	Запуск	16
3.1	Создание lab8-4.asm	17
3.2	Код программы	18
3.3	Запуск программы	18

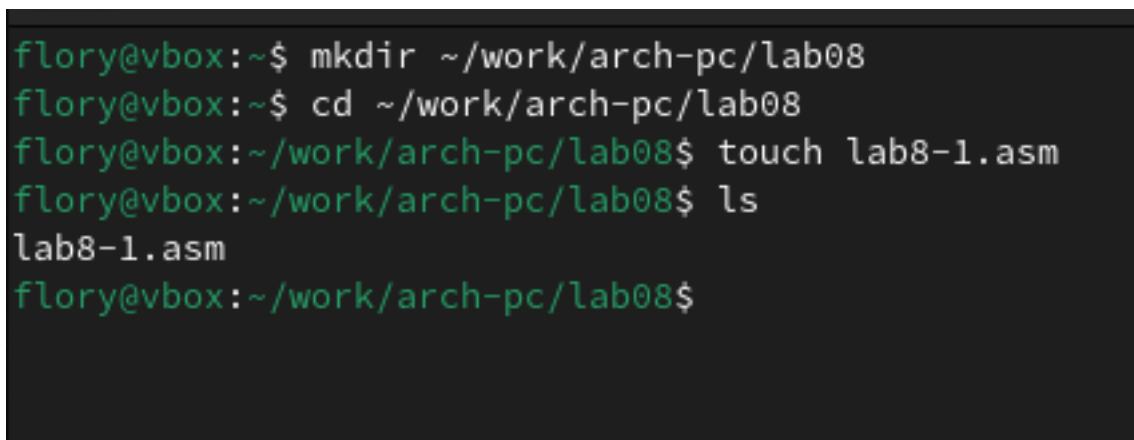
Список таблиц

1 Цель работы

Приобрести навыки написания программ с использованием циклов и обработкой аргументов командной строки.

2 Выполнение лабораторной работы

1) Создал каталог для лабораторной работы №8 и создал файл lab8-1.asm: (Рис. 2.1)



```
flory@vbox:~$ mkdir ~/work/arch-pc/lab08
flory@vbox:~$ cd ~/work/arch-pc/lab08
flory@vbox:~/work/arch-pc/lab08$ touch lab8-1.asm
flory@vbox:~/work/arch-pc/lab08$ ls
lab8-1.asm
flory@vbox:~/work/arch-pc/lab08$
```

Рис. 2.1: Создание lab8-1

2) Ввел код программы вывода значений регистра есх из листинга 8.1: (Рис. 2.2)

```

lab8-1.asm      [-M--] 24 L:[ 1+ 5 6/ 31] *(275 / 844b) 0010 0x00A
;
; Программа вывода значений регистра 'ecx'
;
-----
%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

```

Рис. 2.2: Код программы

3) Создал исполняемый файл и запустил его: (Рис. 2.3)

```

flory@vbox:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
flory@vbox:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
flory@vbox:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 4
4
3
2
1
flory@vbox:~/work/arch-pc/lab08$

```

Рис. 2.3: Запуск

4) Теперь изменим программу, чтобы в цикле отнималась единица у регистра ecx: (Рис. 2.4)

```

lab8-1.asm      [-M--] 24 L: [ 1+25 26/ 29] *(551 / 659b) 1072 0x430
%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:
sub ecx,1 ; `ecx=ecx-1`
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

```

Рис. 2.4: Замена

5) Запустил измененный файл: (Рис. 2.5)


```
flory@vbox:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
flory@vbox:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
flory@vbox:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 5
4
2
0
4294967294
4294967292
4294967290
4294967288
4294967286
4294967284
4294967282
4294967280
4294967278
4294967276
4294967274
4294967272
4294967270
4294967268
4294967266
4294967264
4294967262
4294967260
4294967258
4294967256
4294967254
4294967252
4294967250
4294967248
4294967246
4294967244
4294967242
4294967240
4294967238
```

Рис. 2.5: Запуск с нечетным числом

Цикл выполняется бесконечное кол-во раз. Цикл останавливается в тот момент, когда `ecx` равен 0. Каждое выполнение цикла уменьшается на 2, и из-за нечетного числа, оно не достигнет нуля.

6)Теперь давайте запустим программу с четным числом: (Рис. 2.6)

```
1
flory@vbox:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 6
5
3
1
flory@vbox:~/work/arch-pc/lab08$
```

Рис. 2.6: Запуск с четным числом

Значит, кол-во итераций цикла не соответствует N как при вводе четного числа, так и при вводе нечетного

7)Давайте теперь сохраним корректность работы программы, добавив команды push и pop:(Рис. 2.7)

```

#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:
push ecx
sub ecx,1 ; `ecx=ecx-1`
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`
pop ecx
loop label ; `ecx=ecx-1` и если `ecx` не `0`
; переход на `label`
call quit

```

Рис. 2.7: Программа

8)Запустил его и убедился в работоспособности:(Рис. 2.8)

```
flory@vbox:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
flory@vbox:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
flory@vbox:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 3
2
1
0
flory@vbox:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 4
3
2
1
0
flory@vbox:~/work/arch-pc/lab08$
```

Рис. 2.8: Запуск

Как видим, теперь в обоих случаях программа выводит все числа до нуля.

9)Теперь создам файл lab8-2.asm и введу код программы, выводящая на экран аргументы командной строки:(Рис. 2.9)

```

lab8-2.asm      [----]  9 L:[  1+19  20/ 20] *(943 / 943b) <EOF>
#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

```

Рис. 2.9: Код программы

10)Создам исполняемый файл и запущу его, указав аргументы:(Рис. 2.10)

```

flory@vbox:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
flory@vbox:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
flory@vbox:~/work/arch-pc/lab08$ ./lab8-2
flory@vbox:~/work/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
flory@vbox:~/work/arch-pc/lab08$

```

Рис. 2.10: Запуск

Программа обработала 4 аргумента.

11)Создам файл lab8-3.asm и введу текст программы из листинга 8.3:(Рис. 2.11)

```

lab8-3.asm      [----]  7 L:[  1+ 5   6/ 29] *(103 /1428b) 0010 0x00A
#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

```

Рис. 2.11: Код программы

12)Запустил его.:(Рис. 2.12)

```

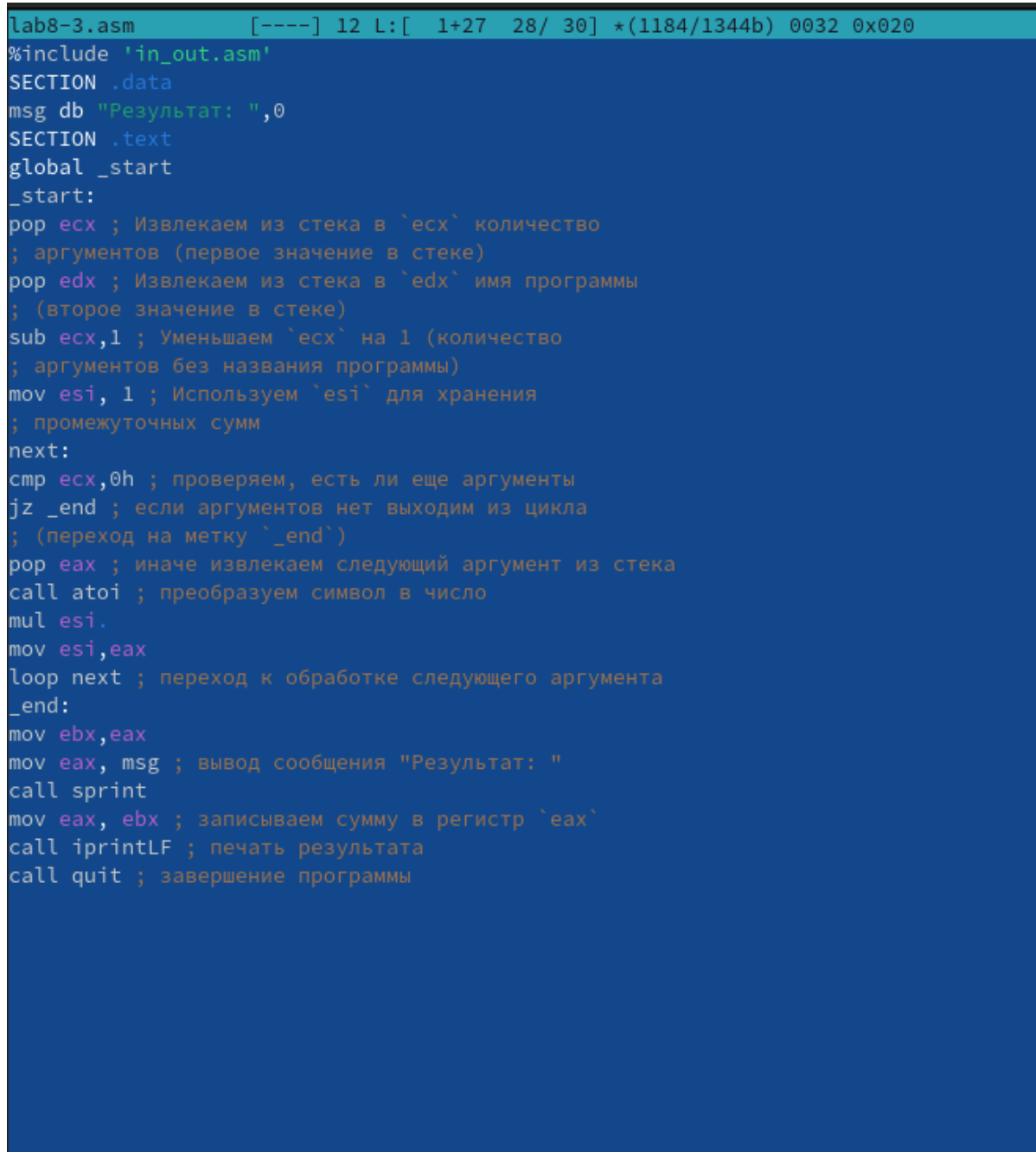
flory@vbox:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
flory@vbox:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
flory@vbox:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
flory@vbox:~/work/arch-pc/lab08$

```

Рис. 2.12: Запуск

Программа выводит сумму всех аргументов. Изменим её теперь, чтобы она находила произведение всех аргументов.

13)Изменяю код программы, чтобы выполнялось произведение:(Рис. 2.13)

The image shows a screenshot of an assembly code editor. The title bar at the top reads 'lab8-3.asm' followed by some system information. The code is written in x86 assembly and is designed to calculate the product of command-line arguments. It starts with an include directive for 'in_out.asm', followed by a data section defining a message string 'Результат: '. The main logic is in the text section, starting with a global _start label. It uses the stack to retrieve arguments, decrementing the count (ecx) as it goes. A loop labeled 'next' checks if there are more arguments; if not, it jumps to '_end'. Inside the loop, it retrieves an argument, converts it from a string to an integer using 'atoi', and multiplies it into the running product stored in 'esi'. After the loop, it prints the message and the final product using 'sprintf' and 'iprintLF', then calls 'quit' to end the program.

```
lab8-3.asm [-----] 12 L:[ 1+27 28/ 30] *(1184/1344b) 0032 0x020
#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
mul esi.
mov esi,eax
loop next ; переход к обработке следующего аргумента
_end:
mov ebx,eax
mov eax, msg ; вывод сообщения "Результат: "
call sprintf
mov eax, ebx ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
```

Рис. 2.13: Код программы

14)Создал исполняемый файл и запустил его. Проверил его на работоспособ-

ность:(Рис. 2.14)

```
flory@vbox:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
flory@vbox:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
flory@vbox:~/work/arch-pc/lab08$ ./lab8-3 2 4 4 2
Результат: 64
flory@vbox:~/work/arch-pc/lab08$
```

Рис. 2.14: Запуск

3 Выполнение задания для самостоятельной работы

1)Создам файл laba8-4.asm для самостоятельной работы. Вариант 10. :(Рис. 3.1)

```
flory@vbox:~/work/arch-pc/lab08$ touch lab8-4.asm
flory@vbox:~/work/arch-pc/lab08$ ls
in_out.asm  lab8-1.asm  lab8-2      lab8-2.o  lab8-3.asm  lab8-4.asm
lab8-1      lab8-1.o    lab8-2.asm  lab8-3    lab8-3.o
```

Рис. 3.1: Создание lab8-4.asm

2)Написал код программы, которая находит сумму значений $f(x)$ для $x=x_1, x_2, \dots, x_n$: (Рис. 3.2)

```

lab8-4.asm      [----] 11 L: [ 1+21 22/ 41] *(270 / 475b) 0111 0x06F
%include 'in_out.asm'

SECTION .data
msg db "Результат: ", 0
msg2 db "Функция: f(x) = 5(2 + x)", 0

SECTION .text
global _start

_start:
    pop ecx
    pop edx
    sub ecx, 1
    ....
    mov esi, 0

next:
    cmp ecx, 0
    jz _end

    pop eax
    call atoi

    add eax, 2
    mov ebx, 5
    mul ebx
    add esi, eax

    loop next

_end:
    mov eax, msg2
    call sprintf

    mov eax, msg
    call sprintf

    mov eax, esi
    call iprintLF

    call quit

```

Рис. 3.2: Код программы

3) Создал исполняемый файл и запустил его. Убедился в работоспособности, посчитав значения еще и в ручную: (Рис. 3.3)

```

flory@vbox: ~/work/arch-pc/lab08$ nasm -f elf lab8-4.asm
flory@vbox: ~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-4 lab8-4.o
flory@vbox: ~/work/arch-pc/lab08$ ./lab8-4 1 2 3 4
Функция: f(x) = 5(2 + x)
Результат: 90
flory@vbox: ~/work/arch-pc/lab08$

```

Рис. 3.3: Запуск программы

4 Вывод

В ходе выполнения лабораторной работы я приобрел навыки написания программ с использованием циклов и обработки аргументов командной строки.