

# **Отчёт по лабораторной работе №8**

**Дисциплина: архитектура компьютеров и операционные системы**

Черная София Витальевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
4.1	Реализация циклов в NASM . . . . .	9
4.2	Обработка аргументов командной строки . . . . .	12
4.3	Задание для самостоятельной работы . . . . .	15
<b>5</b>	<b>Выводы</b>	<b>17</b>

## Список иллюстраций

4.1	Создание файлов для лабораторной работы . . . . .	9
4.2	Ввод текста из листинга 8.1 . . . . .	9
4.3	Запуск исполняемого файла . . . . .	10
4.4	Изменение текста программы . . . . .	10
4.5	Запуск обновленной программы . . . . .	11
4.6	Изменение текста программы . . . . .	11
4.7	Запуск исполняемого файла . . . . .	12
4.8	Ввод текста программы из листинга 8.2 . . . . .	12
4.9	Запуск исполняемого файла . . . . .	13
4.10	Ввод текста программы из листинга 8.3 . . . . .	13
4.11	Запуск исполняемого файла . . . . .	14
4.12	Изменение текста программы . . . . .	14
4.13	Запуск исполняемого файла . . . . .	14
4.14	Текст программы . . . . .	15
4.15	Запуск исполняемого файла и проверка его работы . . . . .	15

## Список таблиц

# 1 Цель работы

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

## 2 Задание

1. Реализация циклов в NASM.
2. Обработка аргументов командной строки.
3. Задание для самостоятельной работы.

### 3 Теоретическое введение

Стек — это структура данных, организованная по принципу LIFO («Last In — First Out» или «последним пришёл — первым ушёл»). Стек является частью архитектуры процессора и реализован на аппаратном уровне. Для работы со стеком в процессоре есть специальные регистры (ss, bp, sp) и команды. Основной функцией стека является функция сохранения адресов возврата и передачи аргументов при вызове процедур. Кроме того, в нём выделяется память для локальных переменных и могут временно храниться значения регистров. Стек имеет вершину, адрес последнего добавленного элемента, который хранится в регистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек указатель стека уменьшается, а при извлечении — увеличивается.

Команда push размещает значение в стеке, т.е. помещает значение в ячейку памяти, на которую указывает регистр esp, после этого значение регистра esp увеличивается на 4. Данная команда имеет один операнд — значение, которое необходимо поместить в стек.

Команда pop извлекает значение из стека, т.е. извлекает значение из ячейки памяти, на которую указывает регистр esp, после этого уменьшает значение регистра esp на 4. У этой команды также один операнд, который может быть регистром или переменной в памяти. Нужно помнить, что извлечённый из стека элемент не стирается из памяти и остаётся как “мусор”, который будет перезаписан при записи нового значения в стек.

Для организации циклов существуют специальные инструкции. Для всех ин-

струкций максимальное количество проходов задаётся в регистре esx. Наиболее простой является инструкция loor. Она позволяет организовать безусловный цикл.



## 4 Выполнение лабораторной работы

### 4.1 Реализация циклов в NASM

Создаю каталог для программ лабораторной работы № 8, перехожу в него и создаю файл lab8-1.asm. (рис. 4.15).

```
(svchernaya@svchernaya)-[~]  
$ mkdir ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab08  
  
(svchernaya@svchernaya)-[~]  
$ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab08  
  
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab08]  
$ touch lab8-1.asm
```

Рис. 4.1: Создание файлов для лабораторной работы

Ввожу в файл lab8-1.asm текст программы из листинга 8.1. (рис. 4.15).

```
Open [v] [i] *lab8-1.asm  
~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08  
1 %include 'in_out.asm'  
2 SECTION .data  
3 msg1 db 'Введите N: ',0h  
4 SECTION .bss  
5 N: resb 10  
6 SECTION .text  
7 global _start  
8 _start:  
9 ; ----- Вывод сообщения 'Введите N: '  
10 mov eax,msg1  
11 call sprint  
12 ; ----- Ввод 'N'  
13 mov ecx, N  
14 mov edx, 10  
15 call sread  
16 ; ----- Преобразование 'N' из символа в число  
17 mov eax,N  
18 call atoi  
19 mov [N],eax  
20 ; ----- Организация цикла  
21 mov ecx,[N] ; Счетчик цикла, 'ecx=N'  
22 label:  
23 mov [N],ecx  
24 mov eax,[N]  
25 call iprintf ; Вывод значения 'N'  
26 loop label ; 'ecx=ecx-1' и если 'ecx' не '0'  
27 ; переход на 'label'  
28 call quit
```

Рис. 4.2: Ввод текста из листинга 8.1

Создаю исполняемый файл и проверяю его работу. (рис. 4.15).

```
(svchernaya@svchernaya) - [~/2023-2024/Архитектура компьютера/arch-pc/lab08]
$ nasm -f elf lab8-1.asm
$ ld -m elf_i386 -o lab8-1 lab8-1.o
$ ./lab8-1
Введите N: 7
7
6
5
4
3
2
1
```

Рис. 4.3: Запуск исполняемого файла

Данная программа выводит числа от N до 1 включительно.

Изменяю текст программы, добавив изменение значения регистра ecx в цикле.  
(рис. 4.15).

```
~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08/lab8-1.asm - Mousepad
File Edit Search View Document Help
1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
8 _start:
9 ; ----- Вывод сообщения 'Введите N: '
10 mov eax,msg1
11 call sprint
12 ; ----- Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 ; ----- Преобразование 'N' из символа в число
17 mov eax,N
18 call atoi
19 mov [N],eax
20 ; ----- Организация цикла
21 mov ecx,[N] ; Счетчик цикла, 'ecx=N'
22 label:
23 sub ecx,1 ; 'ecx=ecx-1'
24 mov [N],ecx
25 mov eax,[N]
26 call iprintLF ; Вывод значения 'N'
27 loop label ; 'ecx-ecx-1' и если 'ecx' не '0'
28 ; переход на 'label'
29 call quit
30
```

Рис. 4.4: Изменение текста программы

Создаю исполняемый файл и проверяю его работу. (рис. 4.15).

```

File Actions Edit View Help
55503913  push ecx
55503911  call printf
55503909  pop ecx
55503907  call printf
55503905  pop ecx
55503903  call printf
55503901  pop ecx
55503899  call printf
55503897  pop ecx
55503895  call printf
55503893  pop ecx
55503891  call printf
55503889  pop ecx
55503887  call printf
55503885  pop ecx
55503883  call printf
55503881  pop ecx
55503879  call printf
55503877  pop ecx
55503875  call printf
55503873  pop ecx
55503871  call printf
55503869  pop ecx

```

Рис. 4.5: Запуск обновленной программы

В данном случае число проходов цикла не соответствует введенному с клавиатуры значению.

Вношу изменения в текст программы, добавив команды push и pop для сохранения значения счетчика цикла loop. (рис. 4.15).

```

File Edit Search View Document Help
1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
8 _start:
9 ; ----- Вывод сообщения 'Введите N: '
10 mov eax,msg1
11 call sprint
12 ; ----- Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 ; ----- Преобразование 'N' из символа в число
17 mov eax,N
18 call atoi
19 mov [N],eax
20 ; ----- Организация цикла
21 mov ecx,[N] ; Счетчик цикла, 'ecx=N'
22 label:
23 push ecx ; добавление значения ecx в стек
24 sub ecx,1 ; 'ecx=ecx-1'
25 mov [N],ecx
26 mov eax,[N]
27 call iprintf ; Вывод значения 'N'
28 pop ecx ; извлечение значения ecx из стека
29 loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
30 ; переход на 'label'
31 call quit
32

```

Рис. 4.6: Изменение текста программы

Создаю исполняемый файл и проверяю его работу.(рис. 4.15).

```
(svchernaya@svchernaya) [~/./2023-2024/Архитектура компьютера/arch-pc/lab08]
$ nasm -f elf lab8-1.asm

(svchernaya@svchernaya) [~/./2023-2024/Архитектура компьютера/arch-pc/lab08]
$ ld -m elf_i386 -o lab8-1 lab8-1.o

(svchernaya@svchernaya) [~/./2023-2024/Архитектура компьютера/arch-pc/lab08]
$ ./lab8-1
Введите N: 5
4
3
2
1
0
```

Рис. 4.7: Запуск исполняемого файла

В данном случае число проходов цикла соответствует введенному с клавиатуры значению и выводит числа от N-1 до 0 включительно.

## 4.2 Обработка аргументов командной строки

Создаю файл lab8-2.asm в каталоге ~/work/study/2023-2024/‘Архитектура компьютера’/arch-pc/lab08 и ввожу в него текст программы из листинга 8.2. (рис. 4.15).

```
1 %include 'in_out.asm'
2 SECTION .text
3 global _start
4 _start:
5 pop ecx ; Извлекаем из стека в `ecx` количество
6 ; аргументов (первое значение в стеке)
7 pop edx ; Извлекаем из стека в `edx` имя программы
8 ; (второе значение в стеке)
9 sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
10 ; аргументов без названия программы)
11 next:
12 cmp ecx, 0 ; проверяем, есть ли еще аргументы
13 jz _end ; если аргументов нет выходим из цикла
14 ; (переход на метку `_end`)
15 pop eax ; иначе извлекаем аргумент из стека
16 call sprintf ; вызываем функцию печати
17 loop next ; переход к обработке следующего
18 ; аргумента (переход на метку `next`)
19 _end:
20 call quit
```

Рис. 4.8: Ввод текста программы из листинга 8.2

Создаю исполняемый файл и запускаю его, указав нужные аргументы. (рис. 4.15).

```

(svchernaya@svchernaya)-[~/./2023-2024/Архитектура компьютера/arch-pc/Lab08]
$ nasm -f elf lab8-2.asm

(svchernaya@svchernaya)-[~/./2023-2024/Архитектура компьютера/arch-pc/Lab08]
$ ld -m elf_i386 -o lab8-2 lab8-2.o

(svchernaya@svchernaya)-[~/./2023-2024/Архитектура компьютера/arch-pc/Lab08]
$ ./lab8-2 аргумент1 аргумент 2 "аргумент 3"
аргумент1
аргумент
2
аргумент 3

```

Рис. 4.9: Запуск исполняемого файла

Программа вывела 4 аргумента, так как аргумент 2 не взят в кавычки, в отличие от аргумента 3, поэтому из-за пробела программа считывает “2” как отдельный аргумент.

Рассмотрим пример программы, которая выводит сумму чисел, которые передаются в программу как аргументы. Создаю файл lab8-3.asm в каталоге ~/work/archpc/lab08 и ввожу в него текст программы из листинга 8.3. (рис. 4.15).

```

1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx ; Извлекаем из стека в `ecx` количество
8 ; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в `edx` имя программы
10 ; (второе значение в стеке)
11 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
12 ; аргументов без названия программы)
13 mov esi, 0 ; Используем `esi` для хранения
14 ; промежуточных сумм
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку `_end`)
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 add esi,eax ; добавляем к промежуточной сумме
22 ; след. аргумент `esi=esi+eax`
23 loop next ; переход к обработке следующего аргумента
24 _end:
25 mov eax,msg ; вывод сообщения "Результат: "
26 call sprint
27 mov eax,esi ; записываем сумму в регистр `eax`
28 call iprintf ; печать результата
29 call quit ; завершение программы

```

Рис. 4.10: Ввод текста программы из листинга 8.3

Создаю исполняемый файл и запускаю его, указав аргументы. (рис. 4.15).

```

(svchernaya@svchernaya)~[/2023-2024/Архитектура компьютера/arch-pc/lab08]
$ nasm -f elf lab8-3.asm

(svchernaya@svchernaya)~[/2023-2024/Архитектура компьютера/arch-pc/lab08]
$ ld -m elf_i386 -o lab8-3 lab8-3.o

(svchernaya@svchernaya)~[/2023-2024/Архитектура компьютера/arch-pc/lab08]
$ ./lab8-3 15 19 20
Результат: 54

```

Рис. 4.11: Запуск исполняемого файла

Изменяю текст программы из листинга 8.3 для вычисления произведения аргументов командной строки. (рис. 4.15).

```

File Edit Search View Document Help
5 global _start
6 _start:
7 por ecx ; Извлекаем из стека в `ecx` количество
8 ; аргументов (первое значение в стеке)
9 por edx ; Извлекаем из стека в `edx` имя программы
10 ; (второе значение в стеке)
11 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
12 ; аргументов без названия программы)
13 mov esi,1 ; Используем `esi` для хранения
14 ; промежуточных сумм
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку `_end`)
19 por eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 mul esi
22 mov esi,eax ; добавляем к промежуточной сумме
23 ; след. аргумент `esi=esi+eax`
24 loop next ; переход к обработке следующего аргумента
25 _end:
26 mov eax, msg ; вывод сообщения "Результат: "
27 call sprint

```

Рис. 4.12: Изменение текста программы

Создаю исполняемый файл и запускаю его, указав аргументы. (рис. 4.15).

```

(svchernaya@svchernaya)~[/2023-2024/Архитектура компьютера/arch-pc/lab08]
$ nasm -f elf lab8-3.asm

(svchernaya@svchernaya)~[/2023-2024/Архитектура компьютера/arch-pc/lab08]
$ ld -m elf_i386 -o lab8-3 lab8-3.o

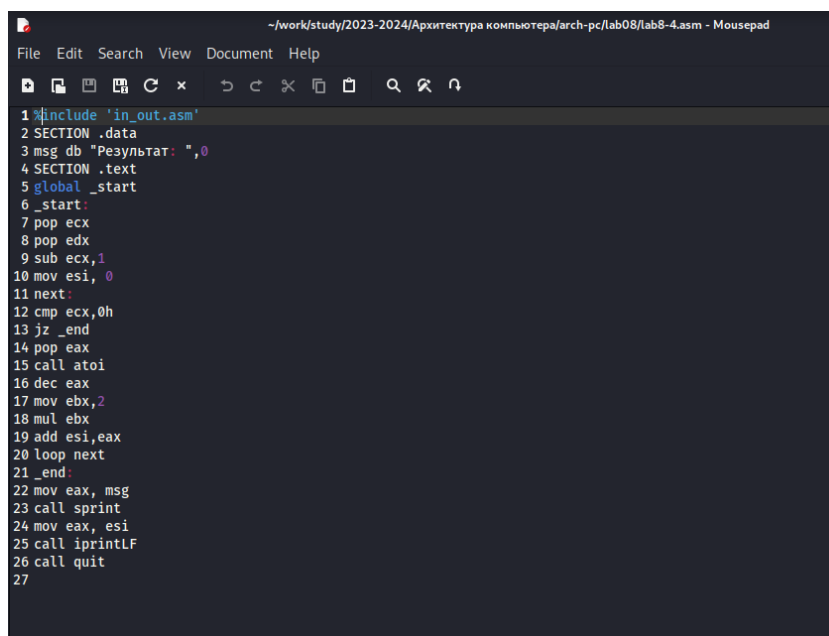
(svchernaya@svchernaya)~[/2023-2024/Архитектура компьютера/arch-pc/lab08]
$ ./lab8-3 15 19 20
Результат: 5700

```

Рис. 4.13: Запуск исполняемого файла

## 4.3 Задание для самостоятельной работы

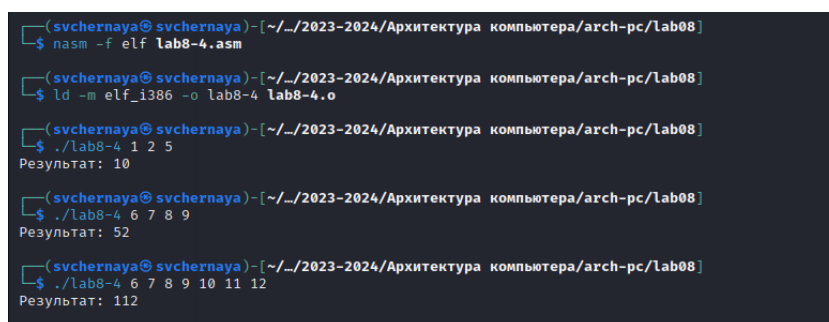
Пишу текст программы, которая находит сумму значений функции  $f(x) = 2 \cdot (x - 1)$  в соответствии с моим номером варианта (4) для  $x = x_1, x_2, \dots, x_n$ . Значения  $x_i$  передаются как аргументы. (рис. 4.15).



```
~\work\study\2023-2024\Архитектура компьютера\arch-pc\lab08\lab8-4.asm - Mousepad
File Edit Search View Document Help
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx
8 pop edx
9 sub ecx,1
10 mov esi, 0
11 next:
12 cmp ecx,0h
13 jz _end
14 pop eax
15 call atoi
16 dec eax
17 mov ebx,2
18 mul ebx
19 add esi,eax
20 loop next
21 _end:
22 mov eax, msg
23 call sprint
24 mov eax, esi
25 call iprintLF
26 call quit
27
```

Рис. 4.14: Текст программы

Создаю исполняемый файл и проверьте его работу на нескольких наборах  $x = x_1, x_2, \dots, x_n$ . (рис. 4.15).



```
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab08]
$ nasm -f elf lab8-4.asm
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab08]
$ ld -m elf_i386 -o lab8-4 lab8-4.o
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab08]
$ ./lab8-4 1 2 5
Результат: 10
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab08]
$ ./lab8-4 6 7 8 9
Результат: 52
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab08]
$ ./lab8-4 6 7 8 9 10 11 12
Результат: 112
```

Рис. 4.15: Запуск исполняемого файла и проверка его работы

Программа работает корректно.

Текст программы:

```
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi, 0
next:
cmp ecx,0h
jz _end
pop eax
call atoi
dec eax
mov ebx,2
mul ebx
add esi,eax
loop next
_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
```



## 5 Выводы

Благодаря данной лабораторной работе я приобрела навыки написания программ использованием циклов и обработкой аргументов командной строки, что поможет мне при выполнении последующих лабораторных работ. # Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005 — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.