

# **Отчет по лабораторной работе №8**

**Дисциплина: архитектура компьютера**

Гомазкова Алина

# 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 . . . . .	13
4.3	Задание для самостоятельной работы . . . . .	16
<b>5</b>	<b>Выводы</b>	<b>18</b>
	<b>Список литературы</b>	<b>19</b>

## Список иллюстраций

4.1	Рис. 1 Создание файлов для лабораторной работы . . . . .	9
4.2	Рис. 2 Ввод текста из листинга 8.1 . . . . .	10
4.3	Рис. 3 Запуск исполняемого файла . . . . .	10
4.4	Рис. 4 Изменение текста программы . . . . .	11
4.5	Рис. 5 Запуск обновленной программы . . . . .	11
4.6	Рис. 6 Изменение текста программы . . . . .	12
4.7	Рис. 7 Запуск исполняемого файла . . . . .	12
4.8	Рис. 8 Ввод текста программы из листинга 8.2 . . . . .	13
4.9	Рис. 9 Запуск исполняемого файла . . . . .	13
4.10	Рис. 10 Ввод текста программы из листинга 8.3 . . . . .	14
4.11	Рис. 11 Запуск исполняемого файла . . . . .	14
4.12	Рис. 12 Изменение текста программы . . . . .	15
4.13	Рис. 13 Запуск исполняемого файла . . . . .	15
4.14	Рис. 14 Текст программы . . . . .	16
4.15	Рис. 15 Запуск исполняемого файла и проверка его работы . . . .	17

## **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.1)

```
[alinagomazkova@10 ~]$ mkdir -p ~/work/arch-pc/lab08  
[alinagomazkova@10 ~]$ cd ~/work/arch-pc/lab08  
[alinagomazkova@10 lab08]$ touch lab8-1.asm
```

Рис. 4.1: Рис. 1 Создание файлов для лабораторной работы

Ввожу в файл lab8-1.asm текст программы из листинга 8.1 (рис. 2)

```

mc [alinagomazkova@10.0.2.15]:~/work/arch-pc/lab08
lab8-1.asm  [-M--]  9 L: [ 1+30 31/ 31] *(844 / 844b) <EOF>
;
; 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 iprintf ; Вывод значения 'N'
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'
call quit

```

Рис. 4.2: Рис. 2 Ввод текста из листинга 8.1

Создаю исполняемый файл и проверяю его работу (рис. 3)

```

[alinagomazkova@10 lab08]$ nasm -f elf lab8-1.asm
[alinagomazkova@10 lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[alinagomazkova@10 lab08]$ ./lab8-1
Введите N: 6
6
5
4
3
2
1

```

Рис. 4.3: Рис. 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)

```
mc [alinagomazkova@10.0.2.15]:~/work/arch-pc/la
/home/alinagomazkova/work/arch-pc/lab08/lab8-1.asm
;-----
; 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:
sub ecx,1 ; `ecx=ecx-1`
mov [N],ecx
mov eax,[N]
call iprintLF
loop label ; `ecx=ecx-1` и если `ecx` не '0'
; переход на `label`
call quit
```

Рис. 4.4: Рис. 4 Изменение текста программы

Создаю исполняемый файл и проверяю его работу (рис. 5)

```
4288048486
4288048484
4288048482
4288048480
4288048478
4288048476
4288048474
4288048472
4288048470
4288048468
4288048466
4288048464
4288048462
```

Рис. 4.5: Рис. 5 Запуск обновленной программы

В данном случае число проходов цикла не соответствует введенному с клавиатуры

туры значению.

Вношу изменения в текст программы, добавив команды push и pop для сохранения значения счетчика цикла loop (рис. 6)

```
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 ; добавление значения ecx в стек
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF
pop ecx ; извлечение значения ecx из стека
loop label
```

Рис. 4.6: Рис. 6 Изменение текста программы

Создаю исполняемый файл и проверяю его работу (рис. 7)

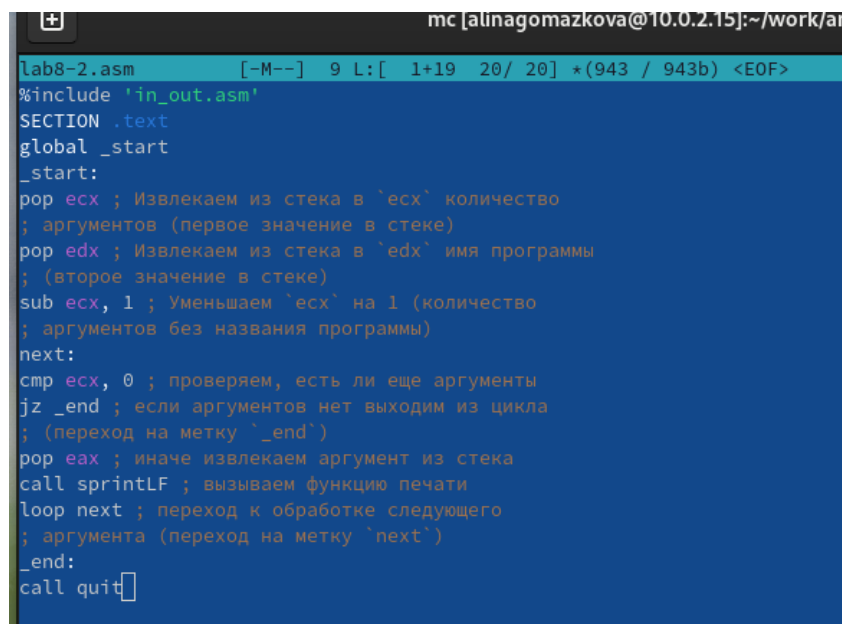
```
[alinagomazkova@10 lab08]$ nasm -f elf lab8-1.asm
[alinagomazkova@10 lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[alinagomazkova@10 lab08]$ ./lab8-1
Введите N: 6
5
4
3
2
1
0
```

Рис. 4.7: Рис. 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

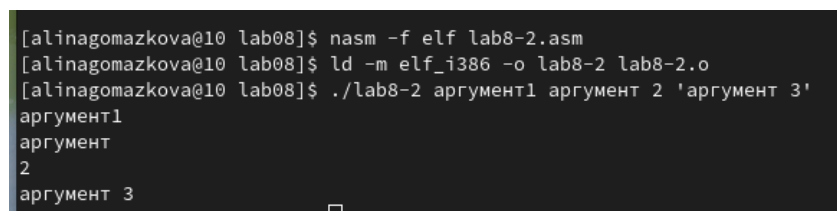
Создаю файл lab8-2.asm и ввожу в него текст программы из листинга 8.2 (рис. 8)



```
lab8-2.asm      [-M--]  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. 4.8: Рис. 8 Ввод текста программы из листинга 8.2

Создаю исполняемый файл и запускаю его, указав нужные аргументы (рис. 9)



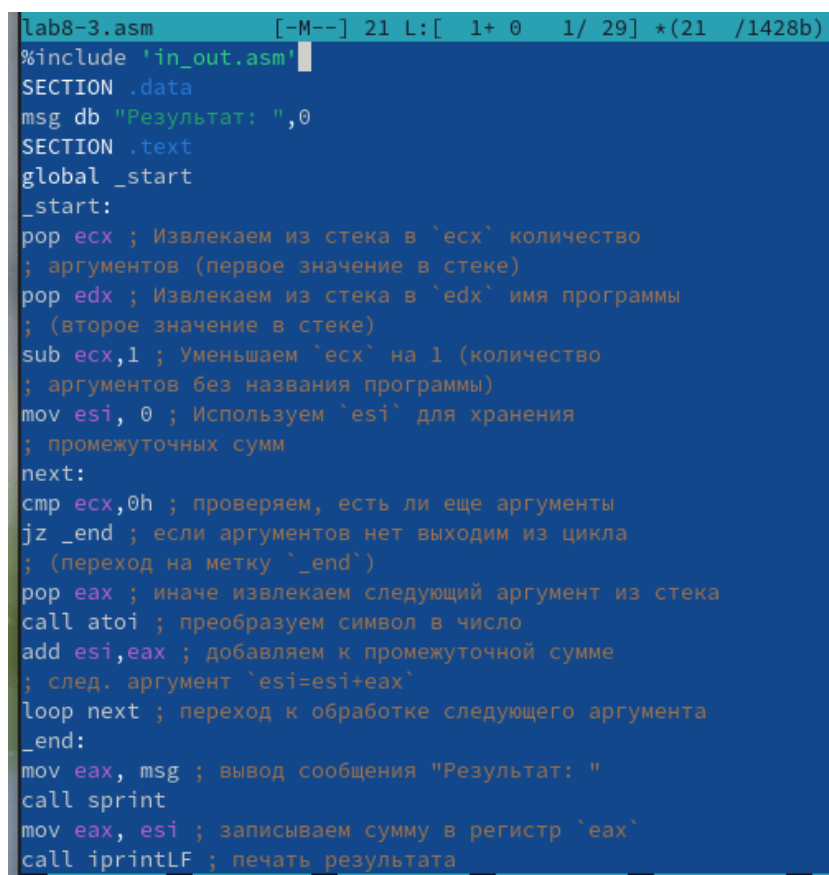
```
[alinagomazkova@10 lab08]$ nasm -f elf lab8-2.asm
[alinagomazkova@10 lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
[alinagomazkova@10 lab08]$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
```

Рис. 4.9: Рис. 9 Запуск исполняемого файла

Программа вывела 4 аргумента, так как аргумент 2 не взят в кавычки, в отличии

от аргумента 3, поэтому из-за пробела программа считывает “2” как отдельный аргумент.

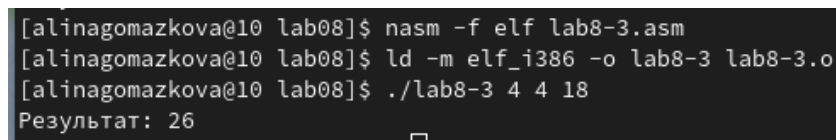
Рассмотрим пример программы, которая выводит сумму чисел, которые передаются в программу как аргументы. Создаю файл lab8-3.asm и ввожу в него текст программы из листинга 8.3 (рис. 10)



```
lab8-3.asm [-M--] 21 L:[ 1+ 0 1/ 29] *(21 /1428b)
#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 sprintf
mov eax, esi ; записываем сумму в регистр `eax`
call iprintf ; печать результата
```

Рис. 4.10: Рис. 10 Ввод текста программы из листинга 8.3

Создаю исполняемый файл и запускаю его, указав аргументы (рис. 11)



```
[alinagomazkova@10 lab08]$ nasm -f elf lab8-3.asm
[alinagomazkova@10 lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[alinagomazkova@10 lab08]$ ./lab8-3 4 4 18
Результат: 26
```

Рис. 4.11: Рис. 11 Запуск исполняемого файла

Изменяю текст программы из листинга 8.3 для вычисления произведения аргументов командной строки (рис. 12)

```
/home/alinagomazkova/work/arch-pc/lab08/lab8-3.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, 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 ; 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`
```

Рис. 4.12: Рис. 12 Изменение текста программы

Создаю исполняемый файл и запускаю его, указав аргументы (рис. 13)

```
[alinagomazkova@10 lab08]$ nasm -f elf lab8-3.asm
[alinagomazkova@10 lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[alinagomazkova@10 lab08]$ ./lab8-3 5 5 10
Результат: 250
[alinagomazkova@10 lab08]$
```

Рис. 4.13: Рис. 13 Запуск исполняемого файла

## 4.3 Задание для самостоятельной работы

Пишу текст программы, которая находит сумму значений функции  $f(x) = 10x - 4$  в соответствии с моим номером варианта (9) для  $x = x_1, x_2, \dots, x_n$ . Значения  $x_i$  передаются как аргументы (рис. 14)

```
/home/alina gomazkova/work/arch-pc/lab08/task.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
mov edi, 10
mul edi
add eax, -4
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: "
```

Рис. 4.14: Рис. 14 Текст программы

Создаю исполняемый файл и проверьте его работу на нескольких наборах  $x = x_1, x_2, \dots, x_n$  (рис. 15)



```
[alinagomazkova@10 lab08]$ nasm -f elf task.asm
[alinagomazkova@10 lab08]$ ld -m elf_i386 -o task task.o
[alinagomazkova@10 lab08]$ ./task 1 2 3 4
Результат: 84
[alinagomazkova@10 lab08]$ ./task 4 6 8 9
Результат: 254
[alinagomazkova@10 lab08]$ ./task 10 17 23 64
Результат: 1124
[alinagomazkova@10 lab08]$
```

Рис. 4.15: Рис. 15 Запуск исполняемого файла и проверка его работы

## 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.Архитектура ЭВМ