

# **Отчёт по лабораторной работе №8**

**Дисциплина: Архитектура компьютеров**

Карпова Анастасия Александровна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>5</b>	<b>Выводы</b>	<b>16</b>
	<b>Список литературы</b>	<b>17</b>

## Список иллюстраций

4.1	Создание файла и каталога . . . . .	8
4.2	Ввод программы . . . . .	9
4.3	Создание исполняемого файла и проверка работы . . . . .	9
4.4	Изменение программы . . . . .	10
4.5	Создание исполняемого файла и проверка работы . . . . .	10
4.6	Изменение программы . . . . .	11
4.7	Создание исполняемого файла и проверка работы . . . . .	11
4.8	Создание файла . . . . .	12
4.9	Ввод программы . . . . .	12
4.10	Создание исполняемого файла и проверка работы . . . . .	12
4.11	Создание файла . . . . .	13
4.12	Ввод программы . . . . .	13
4.13	Создание исполняемого файла и проверка работы . . . . .	13
4.14	Изменение программы . . . . .	14
4.15	Создание исполняемого файла и проверка работы . . . . .	14
4.16	Ввод программы . . . . .	15
4.17	Создание исполняемого файла и проверка работы . . . . .	15

# 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. Выполнение заданий 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. Для стека существует две основные операции: • добавление элемента в вершину стека (push); • извлечение элемента из вершины стека (pop). 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. Существует ещё две команды для добавления значений в стек. Это команда pusha, которая помещает в стек содержимое всех регистров общего назначения в следующем порядке: ax, cx, dx, bx, sp, bp, si, di. А также команда pushf, которая служит для перемещения в стек содержимого регистра флагов. Обе эти команды не имеют операндов. 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. Аналогично команде записи в стек существует команда `rora`, которая восстанавливает из стека все регистры общего назначения, и команда `rorf` для перемещения значений из вершины стека в регистр флагов. Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре `ecx`. Наиболее простой является инструкция `loop`. Она позволяет организовать безусловный цикл.

## 4 Выполнение лабораторной работы

### 8.3.1. Реализация циклов в NASM

Создаю каталог для программ лабораторной работы № 8, перехожу в него и создаю файл lab8-1.asm (рис. 4.1).

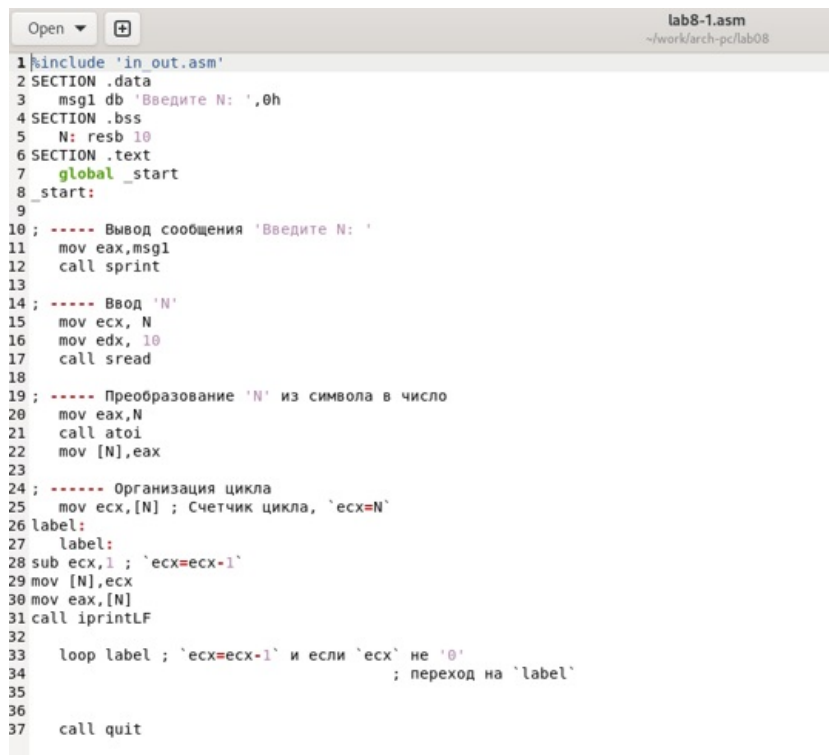


```
akarpova@Justclown:~/work/arch-pc$ mkdir lab08|
akarpova@Justclown:~/work/arch-pc/lab08$ touch lab8-1.asm|
```

Рис. 4.1: Создание файла и каталога

Ввожу в файл lab8-1.asm текст программы из листинга 8.1. (рис. 4.2)





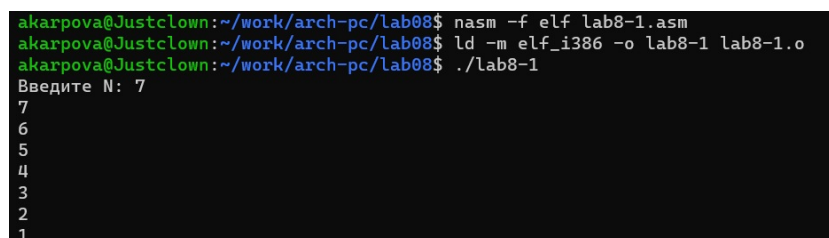
```

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
10 ; ----- Вывод сообщения 'Введите N: '
11     mov eax,msg1
12     call sprint
13
14 ; ----- Ввод 'N'
15     mov ecx, N
16     mov edx, 10
17     call sread
18
19 ; ----- Преобразование 'N' из символа в число
20     mov eax,N
21     call atoi
22     mov [N],eax
23
24 ; ----- Организация цикла
25     mov ecx,[N] ; Счетчик цикла, 'ecx=N'
26 label:
27     label:
28     sub ecx,1 ; 'ecx=ecx-1'
29     mov [N],ecx
30     mov eax,[N]
31     call iprintLF
32
33     loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
34                  ; переход на 'label'
35
36
37     call quit

```

Рис. 4.2: Ввод программы

Создаю исполняемый файл и проверяю его работу. (рис. 4.3)



```

akarpova@JustcLown:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
akarpova@JustcLown:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
akarpova@JustcLown:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 7
7
6
5
4
3
2
1

```

Рис. 4.3: Создание исполняемого файла и проверка работы

Создайте файл lab8-2.asm в каталоге ~/work/arch-pc/lab08

Программа выводит числа от N до 1 включительно Изменяю текст программы добавив изменение значение регистра ecx в цикле. (рис. 4.4)

```

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
27 loop label
28 call quit
29

```

Рис. 4.4: Изменение программы

Создаю исполняемый файл и проверяю его работу (рис. 4.5)

```

akarpova@Justclown:~/work/arch-pc/lab08$ gedit lab8-1.asm
akarpova@Justclown:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
akarpova@Justclown:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
akarpova@Justclown:~/work/arch-pc/lab08$ ./lab8-1
4294895154
4294895152
4294895150
4294895148
4294895146
4294895144
4294895142
4294895140
4294895137

```

Рис. 4.5: Создание исполняемого файла и проверка работы

В данном случае число проходов не соответствует введенному с клавиатуры значению. Снова изменяю текст программы, добавив команды push и pop для сохранения значения счётчика цикла loop (рис. 4.6)

```

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
25 mov [N],ecx
26 mov eax,[N]
27 call iprintLF
28 pop ecx ; извлечение значения ecx из стека
29 loop label
30
31 call quit
32

```

Рис. 4.6: Изменение программы

Создаю исполняемый файл и проверяю его работу (рис. 4.7)

```

akarpova@JustcLown:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
akarpova@JustcLown:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
akarpova@JustcLown:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 7
6
5
4
3
2
1
0

```

Рис. 4.7: Создание исполняемого файла и проверка работы

Теперь число проходов цикла соответствует введенному с клавиатуры значению и выводит числа от N-1 до 0 включительно

### 8.3.2. Обработка аргументов командной строки

Создаю файл lab8-2.asm в каталоге lab08 (рис. 4.8)

```
akarpova@Justclown:~/work/arch-pc/lab08$ touch lab8-2.asm
```

Рис. 4.8: Создание файла

Ввожу в него текст программы из листинга 8.2. (рис. 4.9)

```
1 %include 'in_out.asm'
2 SECTION .text
3 global _start
4
5 _start:
6     pop ecx ; Извлекаем из стека в `ecx` количество
7             ; аргументов (первое значение в стеке)
8
9     pop edx ; Извлекаем из стека в `edx` имя программы
10            ; (второе значение в стеке)
11
12     sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
13              ; аргументов без названия программы)
14
15 next:
16     cmp ecx, 0 ; проверяем, есть ли еще аргументы
17     jz _end ; если аргументов нет выходим из цикла
18             ; (переход на метку `_end`)
19
20     pop eax ; иначе извлекаем аргумент из стека
21     call sprintf ; вызываем функцию печати
22     loop next ; переход к обработке следующего
23              ; аргумента (переход на метку `next`)
24
25 _end:
26     call quit
```

Рис. 4.9: Ввод программы

Создаю исполняемый файл и запускаю его. (рис. 4.10)

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

Рис. 4.10: Создание исполняемого файла и проверка работы

Программа выводит 4 аргумента, так как аргумент 2 не взят в кавычки, в отличие от аргумента 3, поэтому из-за пробела программа не считывает 2 как отдельные аргумент. Рассмотрим пример программы, которая выводит сумму

чисел, которые передаются в программу как аргументы. Создаю файл lab8-3.asm в каталоге lab08. (рис. 4.11)

```
akarpova@Justclown:~/work/arch-pc/lab08$ touch lab8-3.asm
```

Рис. 4.11: Создание файла

Ввожу в него текст программы из листинга 8.3. (рис. 4.12)

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

Рис. 4.12: Ввод программы

Создаю исполняемый файл и запускаю его, указав аргументы (рис. 4.13)

```
akarpova@Justclown:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
akarpova@Justclown:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
akarpova@Justclown:~/work/arch-pc/lab08$ ./lab8-3 13 3 9
Результат: 25
akarpova@Justclown:~/work/arch-pc/lab08$ ./lab8-3 13 3 9 6 7
Результат: 38
akarpova@Justclown:~/work/arch-pc/lab08$ |
```

Рис. 4.13: Создание исполняемого файла и проверка работы

Изменяю текст программы для вычисления произведения аргументов командной строки (рис. 4.14)

```

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 ; Извлекаем из стека в `edx` имя программы
9 ; (второе значение в стеке)
10 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
11 ; аргументов без названия программы)
12 mov esi, 1 ; Используем `esi` для хранения
13 ; промежуточных сумм
14 next:
15 cmp ecx,0h ; проверяем, есть ли еще аргументы
16 jz _end ; если аргументов нет выходим из цикла
17 ; (переход на метку `_end`)
18 pop eax ; иначе извлекаем следующий аргумент из стека
19 call atoi ; преобразуем символ в число
20 mul esi
21 mov 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

```

Рис. 4.14: Изменение программы

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

```

akarpova@Justclown:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
akarpova@Justclown:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
akarpova@Justclown:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 4 5
Результат: 120
akarpova@Justclown:~/work/arch-pc/lab08$ ./lab8-3 13 3 9 6 7
Результат: 14742

```

Рис. 4.15: Создание исполняемого файла и проверка работы

#### 8.4. Задание для самостоятельной работы

Пишу текст программы, которая находит сумму значений функции  $f(x) = 6x+13$  в соответствии с моим вариантом - 15, для  $x_1, x_2, \dots, x_n$ . Значения  $x_i$  передаются как аргументы. (рис. 4.16)

```

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 mov edi, 6
12 next:
13 cmp ecx, 0h
14 jz _end
15 pop eax
16 call atoi
17 mul edi
18 add eax, 13
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

```

Рис. 4.16: Ввод программы

Создаю исполняемый файл и запускаю его, указав аргументы (рис. 4.17)

```

akarpova@Justclown:~/work/arch-pc/lab08$ gedit task8-1.asm
akarpova@Justclown:~/work/arch-pc/lab08$ nasm -f elf task8-1.asm
akarpova@Justclown:~/work/arch-pc/lab08$ ld -m elf_i386 -o task8-1 task8-1.o
akarpova@Justclown:~/work/arch-pc/lab08$ ./task8-1 1 2 3
Результат: 75
akarpova@Justclown:~/work/arch-pc/lab08$ ./task8-1 13 3 7
Результат: 177

```

Рис. 4.17: Создание исполняемого файла и проверка работы

## **5 Выводы**

В ходе лабораторной работы я приобрела навыки написания программ с использованием циклов и обработкой аргументов командной строки.



# Список литературы

Архитектура ЭВМ