

Лабораторная работа №8

Архитектура компьютера

Башиянц Александра Кареновна

Содержание

1	Цель работы	3
2	Задание	4
3	Выполнение лабораторной работы	5
3.1	Реализация циклов в NASM	5
3.2	Обработка аргументов командной строки	7
3.3	Задание для самостоятельной работы	9
4	Выводы	11

1 Цель работы

Цель работы — приобрести навыки написания программ с использованием циклов и обработкой аргументов командной строки.

2 Задание

В этой лабораторной работе необходимо изучить работу циклов и обратку аргументов командной строки.

Необходимо научиться:

- Изучить команды циклов;
- Приобрести навыки циклов;
- Узнать назначение циклов;
- Изучить команды использования аргументов командной строки.

Выполняя это задание, мы получим практический опыт работы циклов и получения аргументов из командной строки.

3 Выполнение лабораторной работы

3.1 Реализация циклов в NASM

Создадим директорию для 8 лабораторной работы и создадим файл lab8-1.asm (рис. 3.1).

```
akbashiyanc@fedoral:~$ mkdir ~/work/arch-pc/lab08
akbashiyanc@fedoral:~$ cd ~/work/arch-pc/lab08
akbashiyanc@fedoral:~/work/arch-pc/lab08$ touch lab8-1.asm
```

Рис. 3.1: Создание директории

Скопируем файл in_out.asm из lab06 с помощью mc (рис. 3.2).

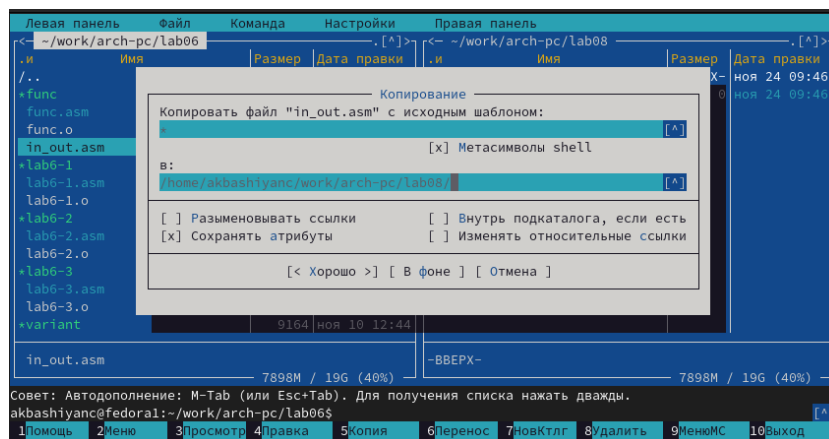


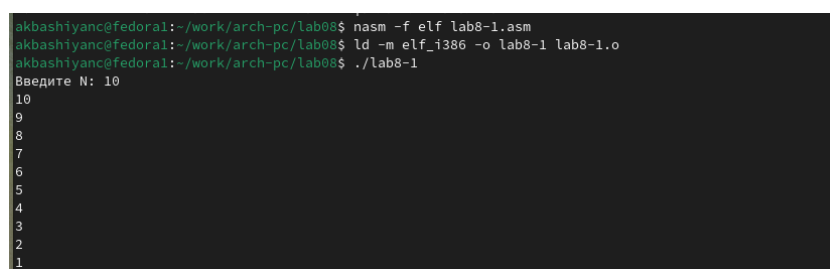
Рис. 3.2: Копирование in_out.asm

Введем код в lab7-1.asm и создадим исполняемый файл и запустим его (рис. 3.3 и 3.4).



```
1 %include 'in_out.asm'
2 SECTION .data
3     msg1 db 'Введите N: ',0h
4
5 SECTION .bss
6     N: resb 10
7 SECTION .text
8     global _start
9 _start:
10 ; ----- Вывод сообщения 'Введите N: '
11     mov eax,msg1
12     call sprint
13 ; ----- Ввод 'N'
14     mov ecx, N
15     mov edx, 10
16     call sread
17 ; ----- Преобразование 'N' из символа в число
18     mov eax,N
19     call atoi
20     mov [N],eax
21 ; ----- Организация цикла
22     mov ecx,[N] ; Счетчик цикла. `ecx=N`
```

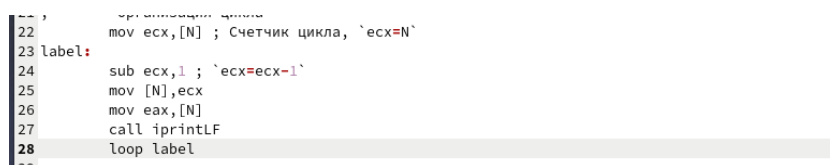
Рис. 3.3: Ввод кода



```
akbashiyan@fedora1:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
akbashiyan@fedora1:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
akbashiyan@fedora1:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 10
10
9
8
7
6
5
4
3
2
1
```

Рис. 3.4: Запуск файла

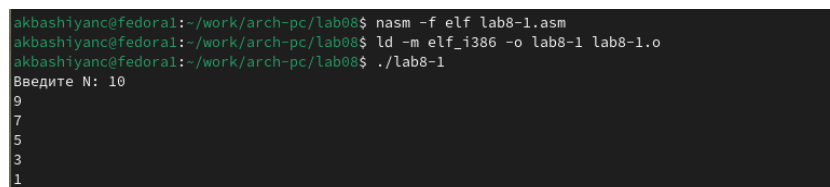
Изменим код так, что использование регистра ecx в теле цикла loop может привести к некорректной работе программы (рис. 3.5).



```
22     mov ecx,[N] ; Счетчик цикла, `ecx=N`
23 label:
24     sub ecx,1 ; `ecx=ecx-1`
25     mov [N],ecx
26     mov eax,[N]
27     call iprintLF
28     loop label
```

Рис. 3.5: Изменение файла

Создадим исполняемый файл и запустим его (рис. 3.6).



```
akbashiyan@fedora1:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
akbashiyan@fedora1:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
akbashiyan@fedora1:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 10
9
7
5
3
1
```

Рис. 3.6: Запуск файла

Заметим, что теперь выводятся только нечетные числа, так как мы, добавив `ecx` в тело цикла `loop`, стали вычитать итератор 2 раза.

Изменим код, добавив команды `push` и `pop` (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла `loop` (рис. 3.7).

```
22      mov ecx,[N] ; счетчик цикла,  ecx=N
23 label:
24      push ecx ; добавление значения ecx в стек
25      sub ecx,1
26      mov [N],ecx
27      mov eax,[N]
28      call iprintLF
29      pop ecx ; извлечение значения ecx из стека
30      loop label
31
```

Рис. 3.7: Изменение файла

Создадим исполняемый файл и запустим его (рис. 3.8).

```
1
akbashiyan@fedora1:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
akbashiyan@fedora1:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
akbashiyan@fedora1:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
```

Рис. 3.8: Запуск файла

Число проходов цикла соответствует значению `N`, но в отличие от изначального кода, выводится числа от `N-1` до 0.

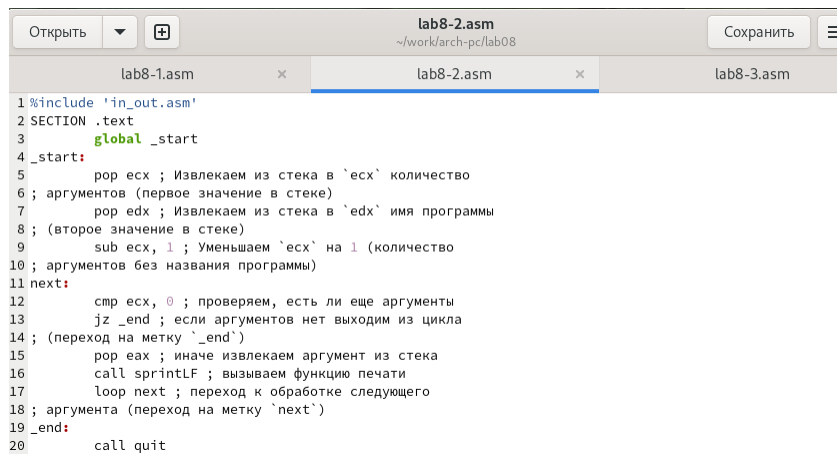
3.2 Обработка аргументов командной строки

Создадим файл `lab8-2.asm` (рис. 3.9).

```
akbashiyan@fedora1:~/work/arch-pc/lab08$ touch lab8-2.asm
```

Рис. 3.9: Создание файла

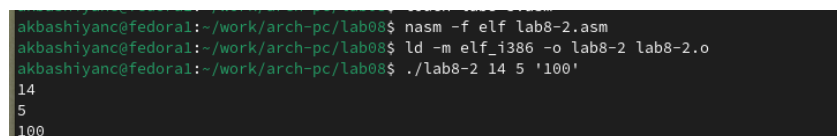
Введем код в `lab8-2.asm` (рис. 3.10).



```
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 printf ; вызываем функцию печати
17     loop next ; переход к обработке следующего
18 ; аргумента (переход на метку `next`)
19 _end:
20     call quit
```

Рис. 3.10: Ввод кода

Создадим исполняемый файл и запустим его (рис. 3.11). Было обработано все 3 аргумента.



```
akbashiyanc@fedoral:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
akbashiyanc@fedoral:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
akbashiyanc@fedoral:~/work/arch-pc/lab08$ ./lab8-2 14 5 '100'
14
5
100
```

Рис. 3.11: Запуск файла

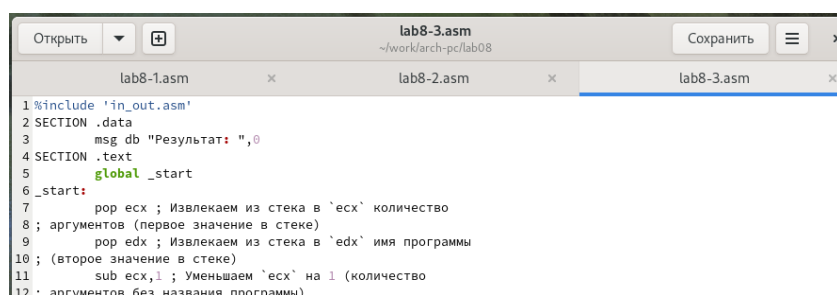
Создадим файл lab8-3.asm (рис. 3.12).



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

Рис. 3.12: Создание файла

Введем код в lab8-2.asm (рис. 3.13).



```
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)
```

Рис. 3.13: Ввод кода

Создадим исполняемый файл и запустим его (рис. 3.14). Было обработано все 3 аргумента.

```
akbashiyan@fedoral:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
akbashiyan@fedoral:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
akbashiyan@fedoral:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 10
Результат: 16
akbashiyan@fedoral:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
```

Рис. 3.14: Запуск файла

Изменим код так, чтобы значения не складывались, а перемножались (рис. 3.15).

```
21
22
23     mul esi
24     mov esi, eax
25 ; след. аргумент `esi=esi*eax`
26
--
```

Рис. 3.15: Изменение файла

Создадим исполняемый файл и запустим его (рис. 3.16).

```
touch: создаем сегментированный образ памяти (собирает на диск)
akbashiyan@fedoral:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
akbashiyan@fedoral:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
akbashiyan@fedoral:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 4
Результат: 24
```

Рис. 3.16: Запуск файла

3.3 Задание для самостоятельной работы

Создадим файл ex1.asm (рис. 3.17).

```
akbashiyan@fedoral:~/work/arch-pc/lab08$ touch ex1.asm
```

Рис. 3.17: Создание файл

Введем код для нахождения суммы функции $f(x)=5*x+17$ (вариант 18) в ex1.asm (рис. 3.18).

```

9
10     pop ecx
11     pop edx
12     sub ecx,1
13     mov esi,0
14     mov eax,msg1
15     call sprintf
16 next:
17     cmp ecx,0
18     jz _end
19     mov ebx,5
20     pop eax
21     call atoi
22     mul ebx
23     add eax,17
24     add esi,eax
25
26     loop next
27
28 _end:

```

Рис. 3.18: Ввод кода

Создадим исполняемый файл и запустим его (рис. 3.19).

```

akbashiyanc@fedora1:~/work/arch-pc/lab08$ nasm -f elf ex1.asm
akbashiyanc@fedora1:~/work/arch-pc/lab08$ ld -m elf_i386 -o ex1 ex1.o
akbashiyanc@fedora1:~/work/arch-pc/lab08$ ./ex1 1 2 3
f(x)=5x+17
Результат: 81
akbashiyanc@fedora1:~/work/arch-pc/lab08$ ./ex1 10 20 100
f(x)=5x+17
Результат: 701

```

Рис. 3.19: Запуск файла

4 Выводы

В ходе выполнения работы были получены навыки написания программ с использованием циклов и обработкой аргументов командной строки.