

Отчет по лабораторной работе №8

Дисциплина: архитектура компьютера

Лаптев Тимофей Сергеевич

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Реализация циклов в NASM	8
4.2	Обработка аргументов командной строки	10
4.3	Задание для самостоятельной работы	11
5	Выводы	13
6	Список литературы	14

Список иллюстраций

4.1	Создание каталога	8
4.2	Запуск программы	8
4.3	Изменение программы	9
4.4	Запуск измененной программы	9
4.5	Добавление push и pop в цикл программы	10
4.6	Запуск второй программы	10
4.7	Запуск третьей программы	11
4.8	Запуск программы для самостоятельной работы	12

Список таблиц

1 Цель работы

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

2 Задание

1. Реализация циклом в NASM
2. Обработка аргументов командной строки
3. Самостоятельное написание программы по материалам лабораторной 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.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Создаю каталог для программ лабораторной работы №8 (рис. -fig. 4.1).

```
[xtade@fedora] [/dev/pts/0] [127]  
[~/work/arch-pc]> cd lab08  
[xtade@fedora] [/dev/pts/0]  
[~/work/arch-pc/lab08]> touch lab8-1.asm
```

Рис. 4.1: Создание каталога

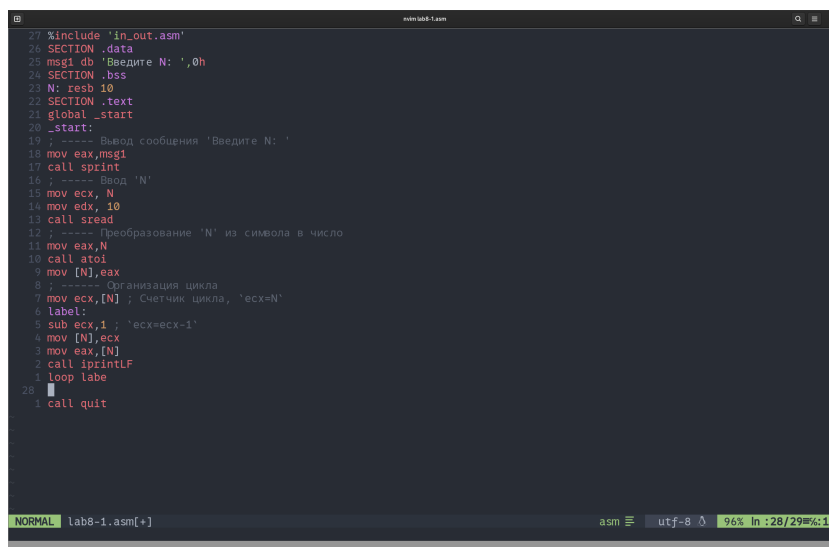
Копирую в созданный файл программу из листинга.

Запускаю программу, она показывает работу циклов в NASM (рис. -fig. 4.2).

```
[-/work/arch-pc/lab08]> nasm -f elf lab8-1.asm  
[xtade@fedora] [/dev/pts/0]  
[-/work/arch-pc/lab08]> ld -m elf_i386 lab8-1.o -o lab8-1  
[xtade@fedora] [/dev/pts/0]  
[-/work/arch-pc/lab08]> ./lab8-1  
Введите N: 2100  
2100  
2099  
2098  
2097  
2096  
2095  
2094  
2093  
2092  
2091  
2090  
2089  
2088  
2087  
2086  
2085  
2084  
2083  
2082  
2081  
2080  
2079  
2078  
2077  
2076  
2075  
2074  
2073  
2072  
2071  
2070  
2069  
2068  
2067
```

Рис. 4.2: Запуск программы

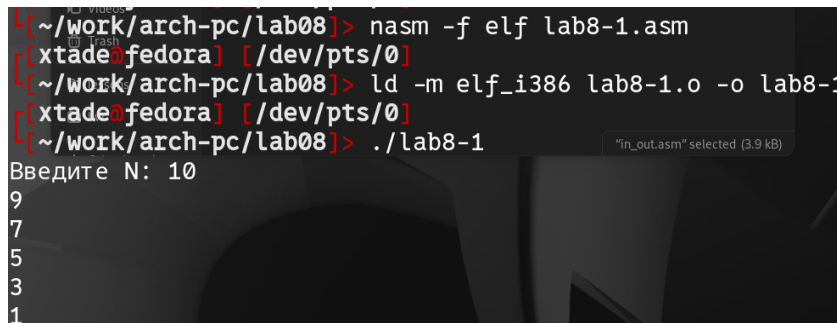
Заменяю программу изначальную так, что в теле цикла я изменяю значение регистра `ecx` (рис. -fig. 4.3).



```
27 %include "in_out.asm"
26 SECTION .data
25 msg1 db 'Введите N: ',0h
24 SECTION .bss
23 N resb 10
22 SECTION .text
21 global _start
20 _start:
19 ; ---- Вывод сообщения 'Введите N: '
18 mov eax,msg1
17 call printf
16 ; ---- Ввод 'N'
15 mov ecx, N
14 mov edx, 10
13 call read
12 ; ---- Преобразование 'N' из символа в число
11 mov eax,N
10 call atoi
9 mov [N],eax
8 ; ---- Организация цикла
7 mov ecx,[N] ; Счетчик цикла, 'ecx=N'
6 label:
5 sub ecx,1 ; 'ecx=ecx-1'
4 mov [N],ecx
3 mov eax,[N]
2 call iprintLF
1 loop label
28
1 call quit
```

Рис. 4.3: Изменение программы

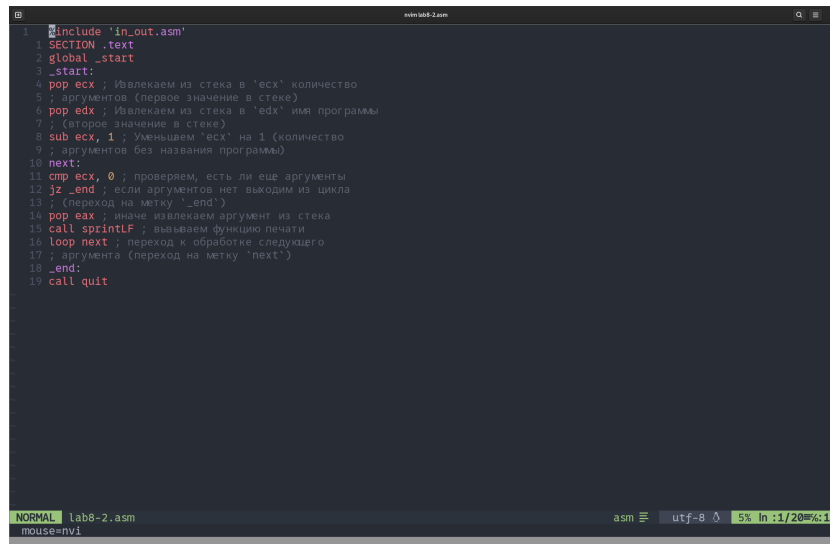
Из-за того, что теперь регистр `ecx` на каждой итерации уменьшается на 2 значения, количество итераций уменьшается вдвое (рис. -fig. 4.4).



```
[~/work/arch-pc/lab08]> nasm -f elf lab8-1.asm
[xtade@fedora] [/dev/pts/0]
[~/work/arch-pc/lab08]> ld -m elf_i386 lab8-1.o -o lab8-1
[xtade@fedora] [/dev/pts/0]
[~/work/arch-pc/lab08]> ./lab8-1
Введите N: 10
9
7
5
3
1
```

Рис. 4.4: Запуск измененной программы

Добавляю команды `push` и `pop` в программу (рис. -fig. 4.5).

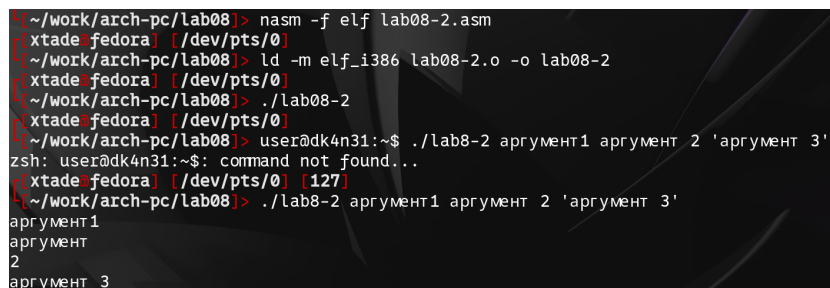


```
1 include 'in_out.asm'
2 SECTION .text
3 global _start
4 _start:
5     pop ecx ; Извлекаем из стека в 'ecx' количество
6     ; аргументов (первое значение в стеке)
7     pop edx ; Извлекаем из стека в 'edx' имя программы
8     ; (второе значение в стеке)
9     sub ecx, 1 ; Уменьшаем 'ecx' на 1 (количество
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. 4.5: Добавление push и pop в цикл программы

4.2 Обработка аргументов командной строки

Создаю новый файл для программы и копирую в него код из следующего листинга. Компилирую программу и запускаю, указав аргументы. Программой было обработано то же количество аргументов, что и было введено (рис. -fig. 4.6).



```
[~/work/arch-pc/lab08] nasm -f elf lab08-2.asm
[xtade@fedora ~]$ ./lab08-2 аргумент1 аргумент 2 'аргумент 3'
[xtade@fedora ~]$ ./lab08-2 аргумент1 аргумент 2 'аргумент 3'
[xtade@fedora ~]$ ./lab08-2 аргумент1 аргумент 2 'аргумент 3'
```

Рис. 4.6: Запуск второй программы

Создаю новый файл для программы и копирую в него код из третьего листинга. Компилирую программу и запускаю, указав в качестве аргументов некоторые числа, программа их складывает (рис. -fig. 4.7).

```
~/work/arch-pc/lab08]> ./lab08-3 12 13 7 10 5
Результат: 47
```

Рис. 4.7: Запуск третьей программы

Изменяю поведение программы так, чтобы указанные аргументы она умножала, а не складывала. Программа действительно теперь умножает данные на вход числа.

4.3 Задание для самостоятельной работы

Пишу программу, которая будет находить сумму значений для функции $f(x) = 3x - 1$, которая совпадает с моим девятым вариантом

Код программы:

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
msg_func db "Функция:  $f(x) = 3x - 1$ ", 0
```

```
msg_result db "Результат: ", 0
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov eax, msg_func
```

```
call sprintf
```

```
pop ecx
```

```
pop edx
```

```
sub ecx, 1
```

```
mov esi, 0
```

```

next:
cmp ecx, 0h
jz _end
pop eax
call atoi

mov ebx, 3
mul ebx
sub eax, 1

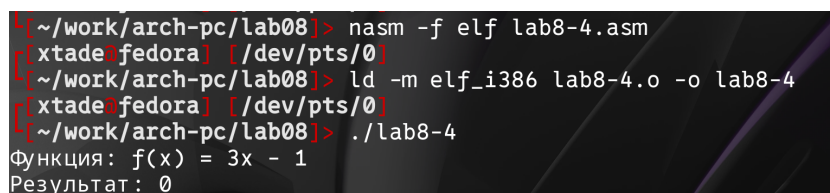
add esi, eax

loop next

_end:
mov eax, msg_result
call sprint
mov eax, esi
call iprintLF
call quit

```

Проверяю работу программы, указав в качестве аргумента несколько чисел (рис. -fig. 4.8).



```

[~/work/arch-pc/lab08]> nasm -f elf lab8-4.asm
[xtade@fedora] [/dev/pts/0]
[~/work/arch-pc/lab08]> ld -m elf_i386 lab8-4.o -o lab8-4
[xtade@fedora] [/dev/pts/0]
[~/work/arch-pc/lab08]> ./lab8-4
Функция: f(x) = 3x - 1
Результат: 0

```

Рис. 4.8: Запуск программы для самостоятельной работы

5 Выводы

В результате выполнения данной лабораторной работы я приобрел навыки написания программ с использованием циклов а также научился обрабатывать аргументы командной строки.

6 Список литературы

1. Курс на ТУИС
2. Лабораторная работа №8
3. Программирование на языке ассемблера NASM Столяров А. В.