

Отчет по лабораторной работе №8

Дисциплина: Архитектура компьютера

Гашимова Эсма Эльшан кызы

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	13
5	Выводы	17
	Список литературы	18

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 измененной программы	10
4.7	Копирование программы из листинга	11
4.8	Запуск второй программы	11
4.9	Копирование программы из третьего листинга	12
4.10	Запуск третьей программы	12
4.11	Изменение третьей программы	13
4.12	Запуск измененной третьей программы	13
4.13	Написание программы для самостоятельной работы	14
4.14	Запуск программы для самостоятельной работы	16

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. 4.1).

```
esmagashimova@fedora:~$ mkdir ~/work/arch-pc/lab08
esmagashimova@fedora:~$ cd ~/work/arch-pc/lab08
bash: /home/esmagashimova/work/arch-pc/lab08: Это каталог
esmagashimova@fedora:~$ cd ~/work/
arch-pc/ study/
esmagashimova@fedora:~$ cd ~/work/arch-pc/lab08
bash: /home/esmagashimova/work/arch-pc/lab08: Это каталог
esmagashimova@fedora:~$ cd ~/
.cache/      .mozilla/    work/        Загрузки/    Общедоступные/
.config/     .ssh/        Видео/       Изображения/ Рабочий стол/
.local/     .texlive2023/ Документы/   Музыка/      Шаблоны/
esmagashimova@fedora:~$ cd ~/work/arch-pc/lab08
esmagashimova@fedora:~/work/arch-pc/lab08$ touch lab8-1.asm
```

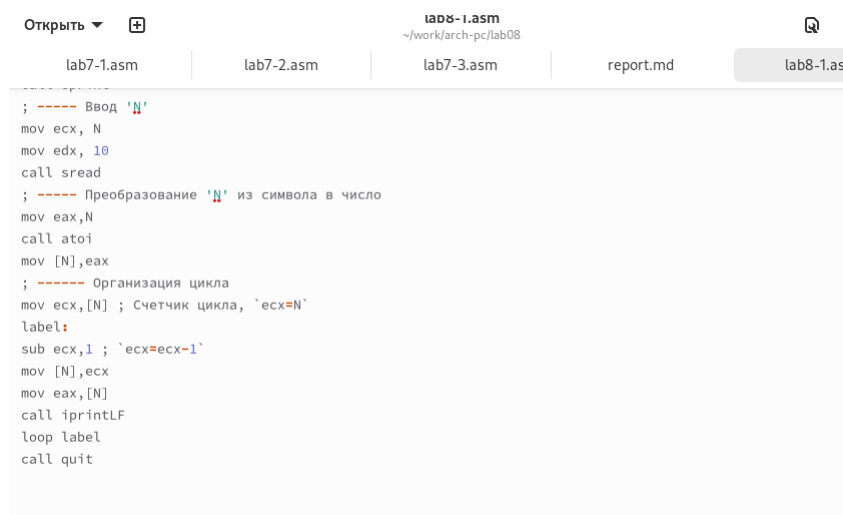
Рис. 4.1: Создание каталога

Копирую в созданный файл программу из листинга и запускаю программу, она показывает работу циклов в NASM (рис. 4.2).

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

Рис. 4.2: Запуск программы

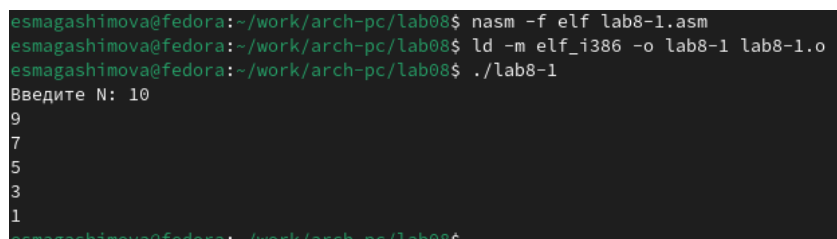
Заменяю программу изначальную так, что в теле цикла изменяю значение регистра ехх (рис. 4.3).



```
Открыть + lab8-1.asm ~/work/arch-pc/lab08
lab7-1.asm lab7-2.asm lab7-3.asm report.md lab8-1.as
; ----- 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 iprintlnLF
loop label
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. 4.4).



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

Рис. 4.4: Запуск измененной программы

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

```

*~/work/arch-pc/lab08/lab8-1.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь

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
call quit

```

Рис. 4.5: Добавление push и pop в цикл программы

Теперь количество итераций совпадает введенному N, но произошло смещение выводимых чисел на -1 (рис. 4.6).

```

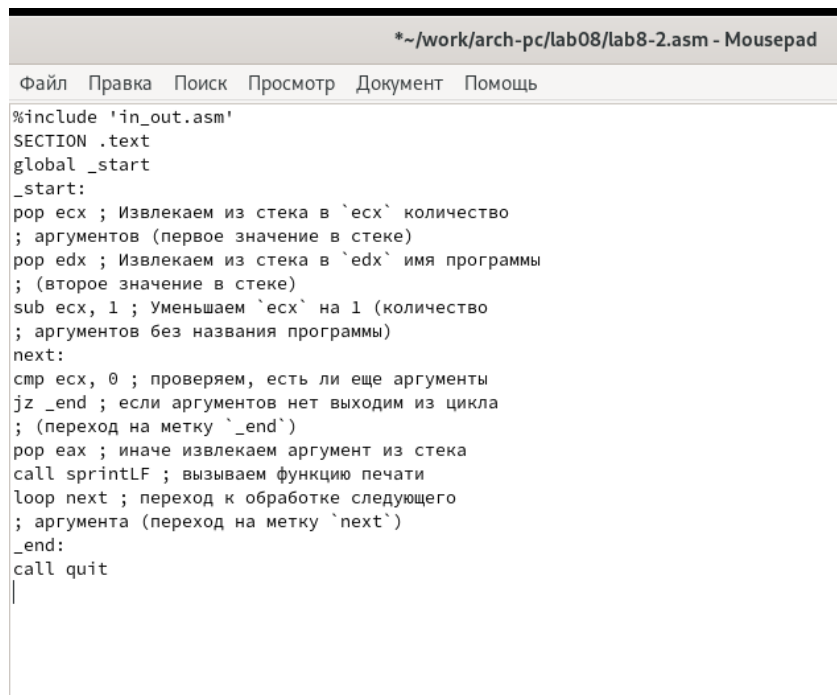
esmagashimova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
esmagashimova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
esmagashimova@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
esmagashimova@fedora:~/work/arch-pc/lab08$

```

Рис. 4.6: Запуск измененной программы

4.2 Обработка аргументов командной строки

Создаю новый файл для программы и копирую в него код из следующего листинга (рис. 4.7).

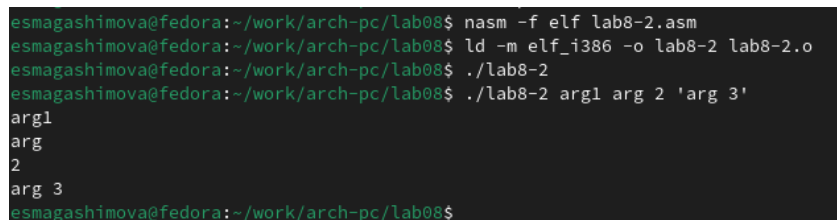


```
*~/work/arch-pc/lab08/lab8-2.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь

%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 sprintLF ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit
|
```

Рис. 4.7: Копирование программы из листинга

Компилирую программу и запускаю, указав аргументы. Программой было обработано то же количество аргументов, что и было введено (рис. 4.8).



```
esmagashimova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
esmagashimova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
esmagashimova@fedora:~/work/arch-pc/lab08$ ./lab8-2
esmagashimova@fedora:~/work/arch-pc/lab08$ ./lab8-2 arg1 arg 2 'arg 3'
arg1
arg
2
arg 3
esmagashimova@fedora:~/work/arch-pc/lab08$
```

Рис. 4.8: Запуск второй программы

Создаю новый файл для программы и копирую в него код из третьего листинга (рис. 4.9).

```
*~/work/arch-pc/lab08/lab8-3.asm - Mousepad
Файл Правка Поиск Просмотр Документ Помощь
%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: "
```

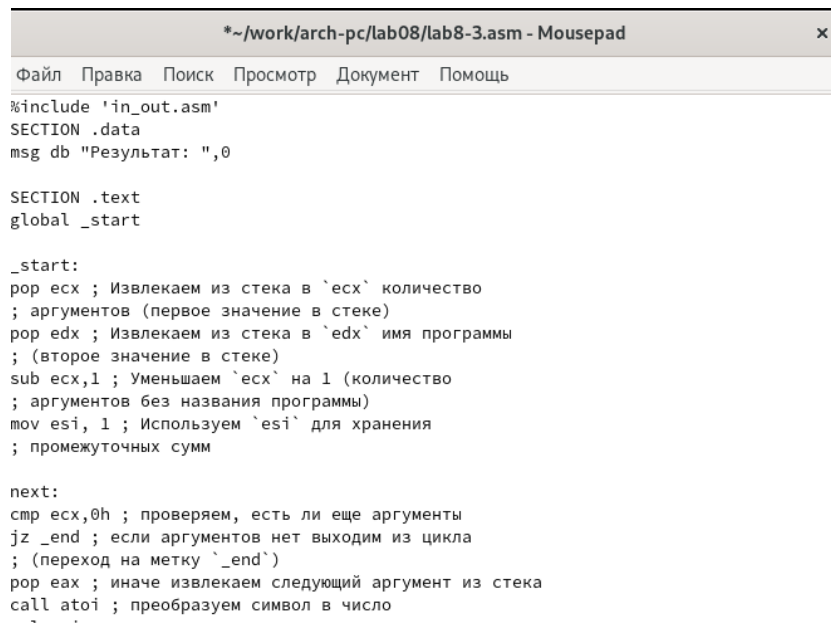
Рис. 4.9: Копирование программы из третьего листинга

Компилирую программу и запускаю, указав в качестве аргументов некоторые числа, программа их складывает (рис. 4.10).

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

Рис. 4.10: Запуск третьей программы

Изменяю поведение программы так, чтобы указанные аргументы она умножала, а не складывала (рис. 4.11).



```
*~/work/arch-pc/lab08/lab8-3.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь

%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0

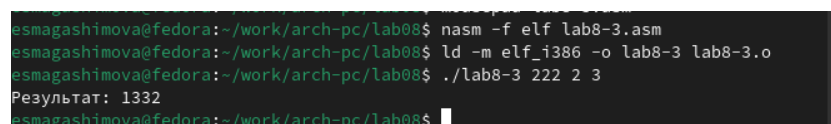
SECTION .text
global _start

_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
```

Рис. 4.11: Изменение третьей программы

Программа действительно теперь умножает данные на вход числа (рис. 4.12).



```
esmagashimova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
esmagashimova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
esmagashimova@fedora:~/work/arch-pc/lab08$ ./lab8-3 222 2 3
Результат: 1332
esmagashimova@fedora:~/work/arch-pc/lab08$
```

Рис. 4.12: Запуск измененной третьей программы

4.3 Задание для самостоятельной работы

Пишу программу, которая будет находить сумма значений для функции $f(x) = 2x+15$, которая совпадает с моим первым вариантом (рис. 4.13).

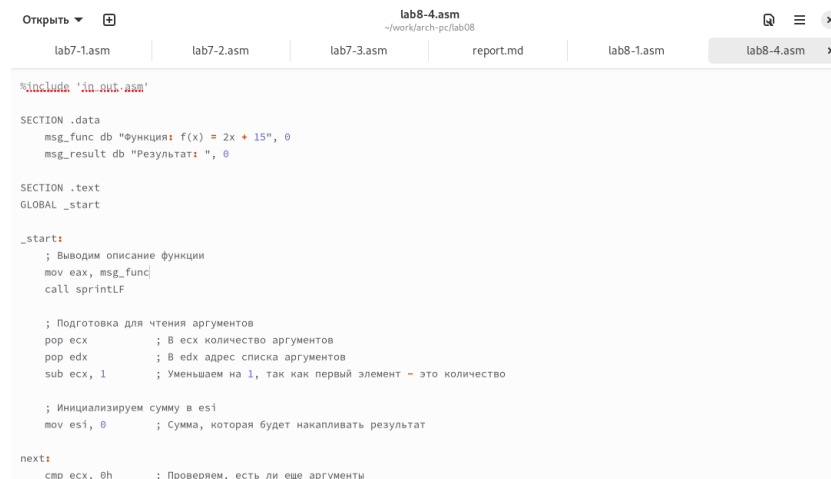


Рис. 4.13: Написание программы для самостоятельной работы

Код программы:

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

```
SECTION .data
```

```
msg_func db "Функция: f(x) = 2x + 15", 0
```

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

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
; Выводим описание функции
```

```
mov eax, msg_func
```

```
call sprintf
```

```
; Подготовка для чтения аргументов
```

```
pop ecx          ; В ecx количество аргументов
```

```
pop edx          ; В edx адрес списка аргументов
```

```
sub ecx, 1       ; Уменьшаем на 1, так как первый элемент - это количество
```

```

; Инициализируем сумму в esi
mov esi, 0          ; Сумма, которая будет накапливать результат

next:
    cmp ecx, 0h      ; Проверяем, есть ли еще аргументы
    jz _end          ; Если нет, переходим к выводу результата

    pop eax           ; Получаем следующий аргумент (x_i)
    call atoi         ; Преобразуем строку в число

    ; Вычисляем  $f(x) = 2x + 15$ 
    mov ebx, 2        ; Загружаем 2 в ebx
    mul ebx           ; eax = eax * ebx (2 * x)
    add eax, 15       ; eax = eax + 15 (2x + 15)

    add esi, eax      ; Добавляем результат в сумму

    loop next         ; Переходим к следующему аргументу

_end:
    ; Выводим результат
    mov eax, msg_result
    call sprint

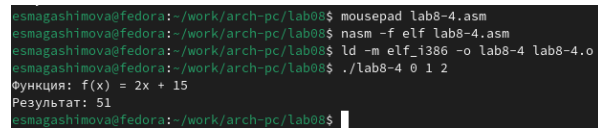
    mov eax, esi      ; Результат суммы
    call iprintLF     ; Выводим результат

    ; Завершаем программу

```

call quit

Проверяю работу программы, указав в качестве аргумента несколько чисел (рис. 4.14).



```
esmagashimova@fedora:~/work/arch-pc/lab8$ mousepad lab8-4.asm
esmagashimova@fedora:~/work/arch-pc/lab8$ nasm -f elf lab8-4.asm
esmagashimova@fedora:~/work/arch-pc/lab8$ ld -m elf_i386 -o lab8-4 lab8-4.o
esmagashimova@fedora:~/work/arch-pc/lab8$ ./lab8-4 0 1 2
Функция: f(x) = 2x + 15
Результат: 51
esmagashimova@fedora:~/work/arch-pc/lab8$
```

Рис. 4.14: Запуск программы для самостоятельной работы

5 Выводы

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

Список литературы

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