

Программирование цикла. Обработка аргументов командной строки.

Лабораторная работа №8.

Дагделен Зейнап Реджеповна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
3.1	Организация стека	7
3.1.1	Добавление элемента в стек.	7
3.1.2	Извлечение элемента из стека.	8
3.2	Инструкции организации циклов	8
4	Выполнение лабораторной работы	9
4.1	Реализация циклов в NASM	9
4.2	Обработка аргументов командной строки	13
4.3	Задание для самостоятельной работы	18
5	Выводы	22
6	Список литературы	23

Список иллюстраций

4.1	Создание каталога и файла в нем	9
4.2	Копирование файла из одной папки в другую и проверка работы команд	9
4.3	Текст программы	10
4.4	Создание исполняемого файла и проверка его работы	10
4.5	Измененный текст программы	11
4.6	Создание исполняемого файла	11
4.7	Работа программы	12
4.8	Создание исполняемого файла и проверка его работы	12
4.9	Создание файла	13
4.10	Текст программы	14
4.11	Создание исполняемого файла и проверка его работы	14
4.12	Создание файла	15
4.13	Текст программы	15
4.14	Создание исполняемого файла и проверка его работы	15
4.15	Текст программы	16
4.16	Создание исполняемого файла и проверка его работы	17
4.17	Создание файла	18
4.18	Текст программы	19
4.19	Создание исполняемого файла и проверка его работы	21

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

3.1 Организация стека

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).

3.1.1 Добавление элемента в стек.

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.

3.1.2 Извлечение элемента из стека.

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.

3.2 Инструкции организации циклов

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, типичная структура которого имеет следующий вид:

```
mov ecx, 100 ; Количество проходов
```

```
NextStep:
```

```
...
```

```
... ; тело цикла
```

```
...
```

```
loop NextStep ; Повторить ecx раз от метки NextStep
```

Инструкция loop выполняется в два этапа. Сначала из регистра ecx вычитается единица и его значение сравнивается с нулём. Если регистр не равен нулю, то выполняется переход к указанной метке. Иначе переход не выполняется и управление передаётся команде, которая следует сразу после команды loop.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Создаю каталог для программ лабораторной работы № 8 с помощью `mkdir`, перехожу в него (команда `cd`) и создаю файл `lab8-1.asm` с помощью `touch` (рис. [4.1]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ mkdir ~/work/arch-pc/lab08
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ cd ~/work/arch-pc/lab08
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ touch lab8-1.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ ls
lab8-1.asm
```

Рис. 4.1: Создание каталога и файла в нем

Так как для дальнейшей работы программ я буду использовать внешний файл `in_iut.asm`, то его необходимо скопировать в папку, в которой я работаю на данный момент (рис. [4.2]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ cp ~/work/arch-pc/lab07/in_out.asm
~/work/arch-pc/lab08
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ ls
in_out.asm lab8-1.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$
```

Рис. 4.2: Копирование файла из одной папки в другую и проверка работы команд

Инструкция `loop` использует регистр `ecx` в качестве счетчика и на каждом шаге уменьшает его значение на единицу. Ввожу в файл `lab8-1.asm` текст программы из листинга 8.1 (рис. [4.3]).

```

zrdagdelen@zrdagdelen: ~/work/arch-pc/lab08
GNU nano 6.2 /home/zrdagdelen/work/arch-pc/lab08/lab
; 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

```

Рис. 4.3: Текст программы

Создаю исполняемый файл и проверяю его работу(рис. [4.4]).

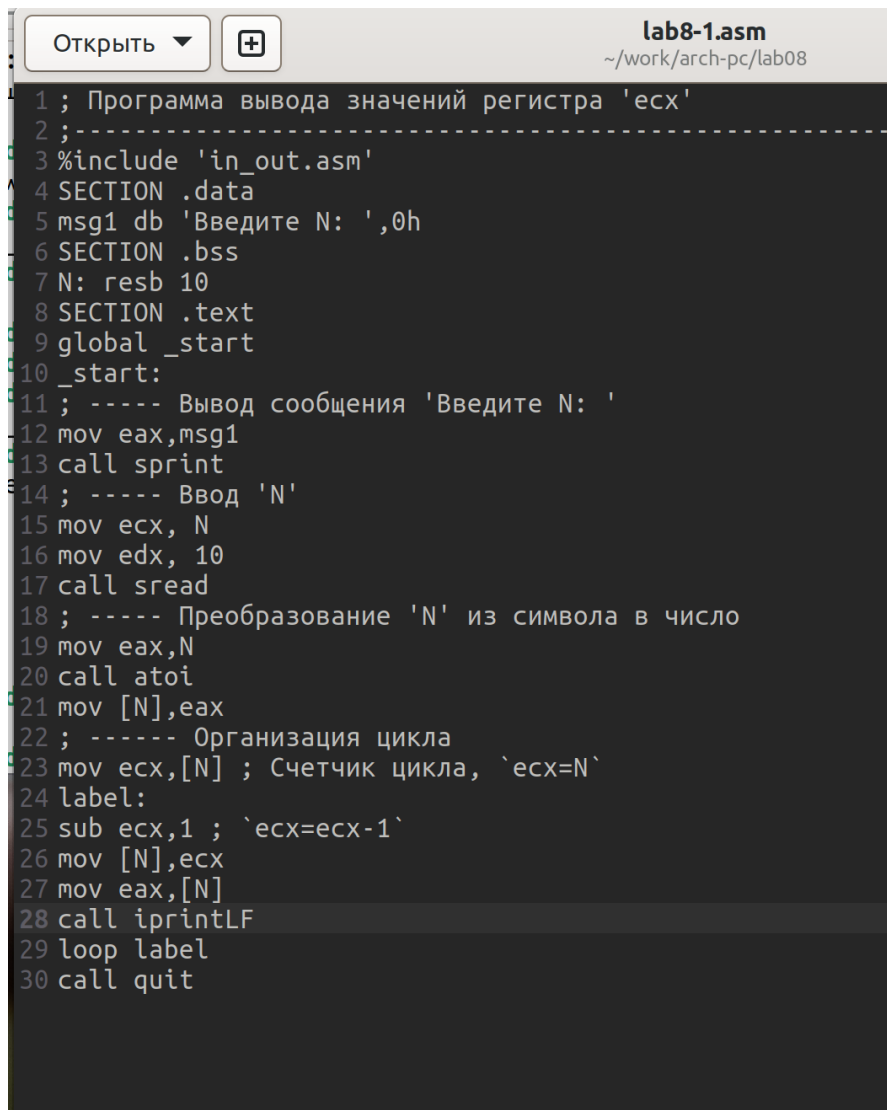
```

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

```

Рис. 4.4: Создание исполняемого файла и проверка его работы

Данный пример показывает, что использование регистра ecx в теле цикла loop может привести к некорректной работе программы. Изменяю текст программы добавив изменение значение регистра ecx в цикле(рис. [4.5]).



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

Рис. 4.5: Измененный текст программы

Создаю исполняемый файл и проверяю его работу(рис. [4.6] - [4.7]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 3
```

Рис. 4.6: Создание исполняемого файла

```

-----
4294793080
4294793078
4294793076
4294793074
4294793072
4294793070
4294793068
4294793066
4294793064
4294793062
4294793060
4294793058
4294793056
4294793054
4294793052
4294793050
4294793048
4294793046
4294793044
4294793042
4294793040
4294793038

```

Рис. 4.7: Работа программы

В итоге я создала бесконечный цикл. Число проходов цикла не соответствует значению N введенному с клавиатуры.

Для использования регистра есх в цикле и сохранения корректности работы программы использую стек. Вношу изменения в текст программы добавив команды push и pop для сохранения значения счетчика цикла loop. Создаю исполняемый файл и проверяю его работу(рис. [4.8]).

```

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

```

Рис. 4.8: Создание исполняемого файла и проверка его работы

В этот раз число проходов цикла соответствует значению N, введенному с

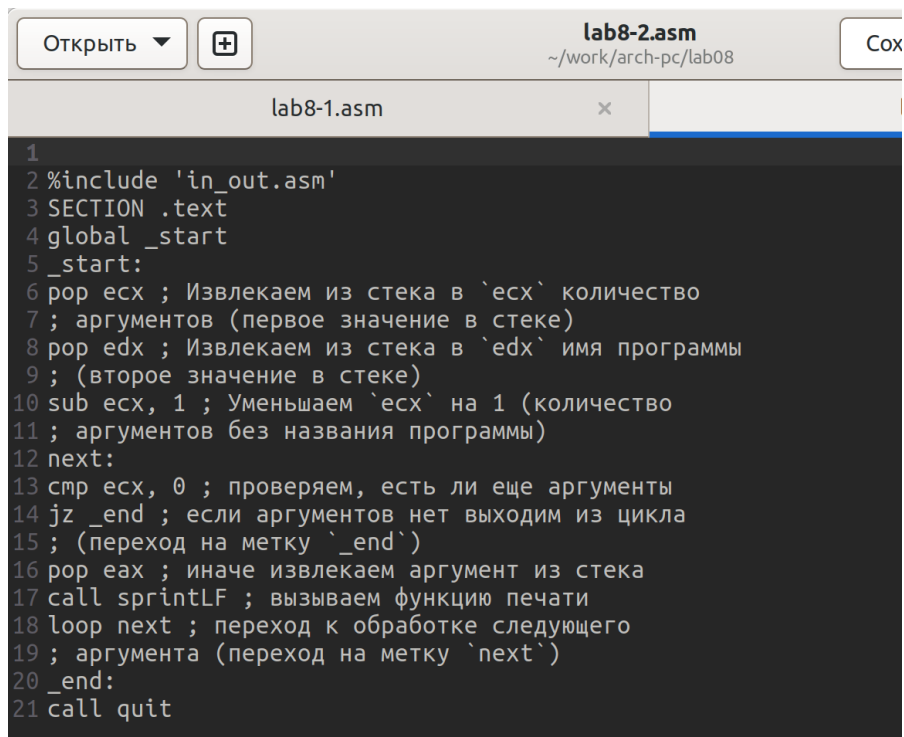
клавиатуры.

4.2 Обработка аргументов командной строки

При разработке программ иногда необходимо указывать аргументы, которые будут использоваться в программе, непосредственно из командной строки при запуске программы. При запуске программы в NASM аргументы командной строки загружаются в стек в обратном порядке, кроме того в стек записывается имя программы и общее количество аргументов. Последние два элемента стека для программы, скомпилированной NASM, – это всегда имя программы и количество переданных аргументов. Таким образом, для того чтобы использовать аргументы в программе, их просто нужно извлечь из стека. Обработку аргументов нужно проводить в цикле. Напишем программу, которая выводит на экран аргументы командной строки. Создаю файл `lab8-2.asm` в каталоге `~/work/arch-pc/lab08` и ввожу в него текст программы из листинга 8.2 (рис. [4.9]–[4.10]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ touch lab8-2.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ ls
in_out.asm lab8-1 lab8-1.asm lab8-1.o lab8-2.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$
```

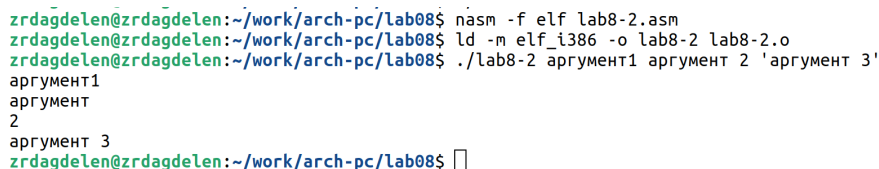
Рис. 4.9: Создание файла



```
1
2 %include 'in_out.asm'
3 SECTION .text
4 global _start
5 _start:
6 por ecx ; Извлекаем из стека в `ecx` количество
7 ; аргументов (первое значение в стеке)
8 por 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 next:
13 cmp ecx, 0 ; проверяем, есть ли еще аргументы
14 jz _end ; если аргументов нет выходим из цикла
15 ; (переход на метку `_end`)
16 por eax ; иначе извлекаем аргумент из стека
17 call sprintf ; вызываем функцию печати
18 loop next ; переход к обработке следующего
19 ; аргумента (переход на метку `next`)
20 _end:
21 call quit
```

Рис. 4.10: Текст программы

Создаю исполняемый файл и запускаю его, указав аргументы: аргумент1 аргумент 2 'аргумент 3' (рис. [4.11]).



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

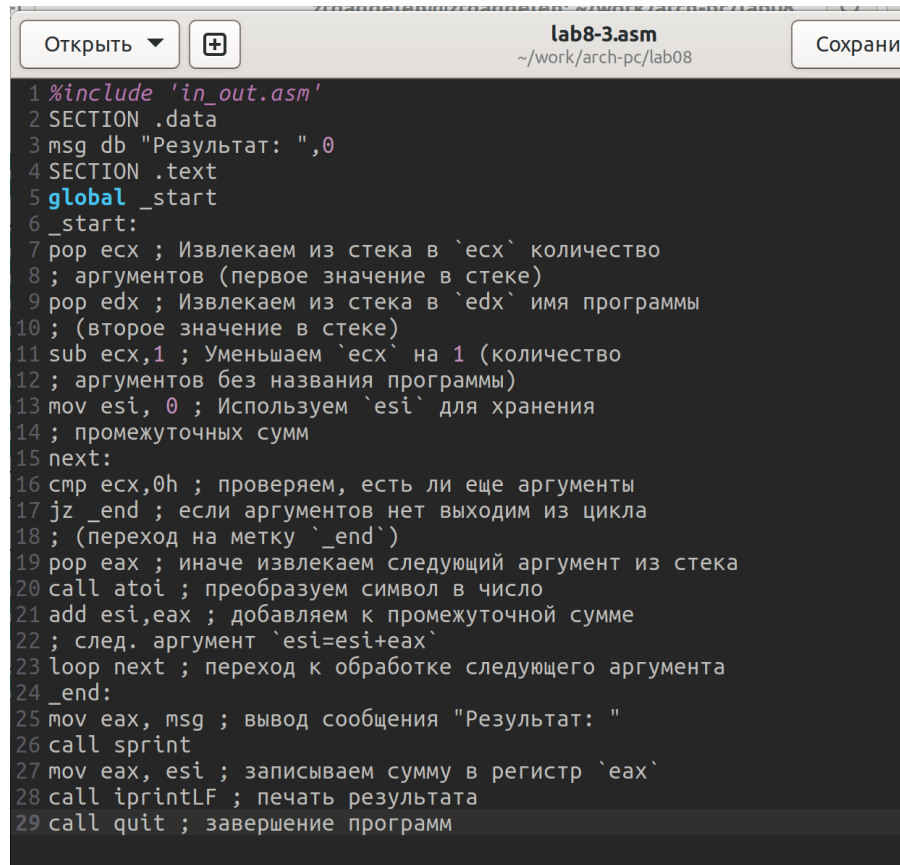
Рис. 4.11: Создание исполняемого файла и проверка его работы

Сколько аргументов было обработано программой? 4 аргумента: каждое аргумент принимается через пробел, а последний аргумент написан в кавычках, из-за чего все, что внутри кавычек воспринимается программой как одна единая строка-аргумент. Напишем программу, которая выводит сумму чисел, которые передаются в программу как аргументы. Создаю файл lab8-3.asm в каталоге ~/work/arch-pc/lab08 и ввожу в него текст программы из листинга 8.3 (рис. [4.12])-

[4.13]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ touch lab8-3.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ ls
in_out.asm lab8-1 lab8-1.asm lab8-1.o lab8-2 lab8-2.asm lab8-2.o lab8-3.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$
```

Рис. 4.12: Создание файла



```
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)
13 mov esi, 0 ; Используем `esi` для хранения
14 ; промежуточных сумм
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку `_end`)
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 add 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 iprintLF ; печать результата
29 call quit ; завершение программ
```

Рис. 4.13: Текст программы

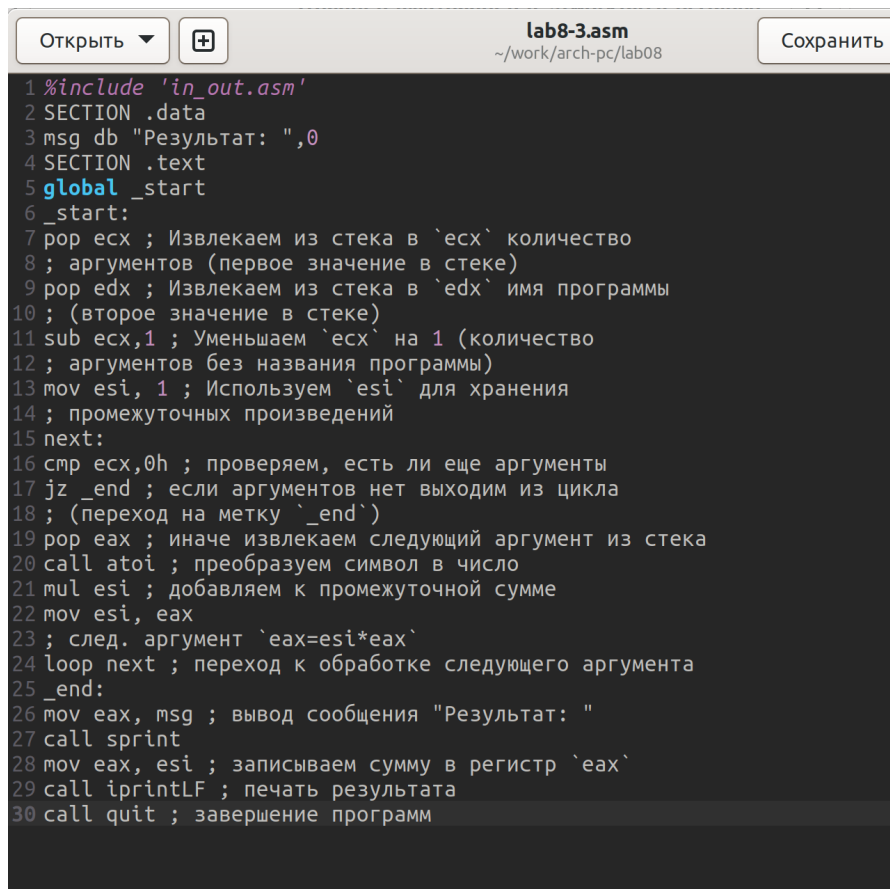
Создаю исполняемый файл и запускаю его, указав аргументы (2, 13, 7, 10, 5) (рис. [4.14]).

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

Рис. 4.14: Создание исполняемого файла и проверка его работы

Программа работает верно.

Изменяю текст программы из листинга 8.3 для вычисления произведения аргументов командной строки(рис. [4.15]).



```
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)
13 mov esi, 1 ; Используем `esi` для хранения
14 ; промежуточных произведений
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку `_end`)
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 mul esi ; добавляем к промежуточной сумме
22 mov esi, eax
23 ; след. аргумент `eax=esi*eax`
24 loop next ; переход к обработке следующего аргумента
25 _end:
26 mov eax, msg ; вывод сообщения "Результат: "
27 call sprint
28 mov eax, esi ; записываем сумму в регистр `eax`
29 call iprintLF ; печать результата
30 call quit ; завершение программ
```

Рис. 4.15: Текст программы

Текст программы:

```
%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 произведений
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 ; добавляем к промежуточному произведению
mov esi, eax
; след. аргумент `eax=esi*eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем произведение в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программ

```

Создаю исполняемый файл и запускаю его, указав аргументы (рис. [4.16]).

```

zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ ./lab8-3 12 2
Результат: 24
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 4 2
Результат: 48
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ 

```

Рис. 4.16: Создание исполняемого файла и проверка его работы

Все работает верно.

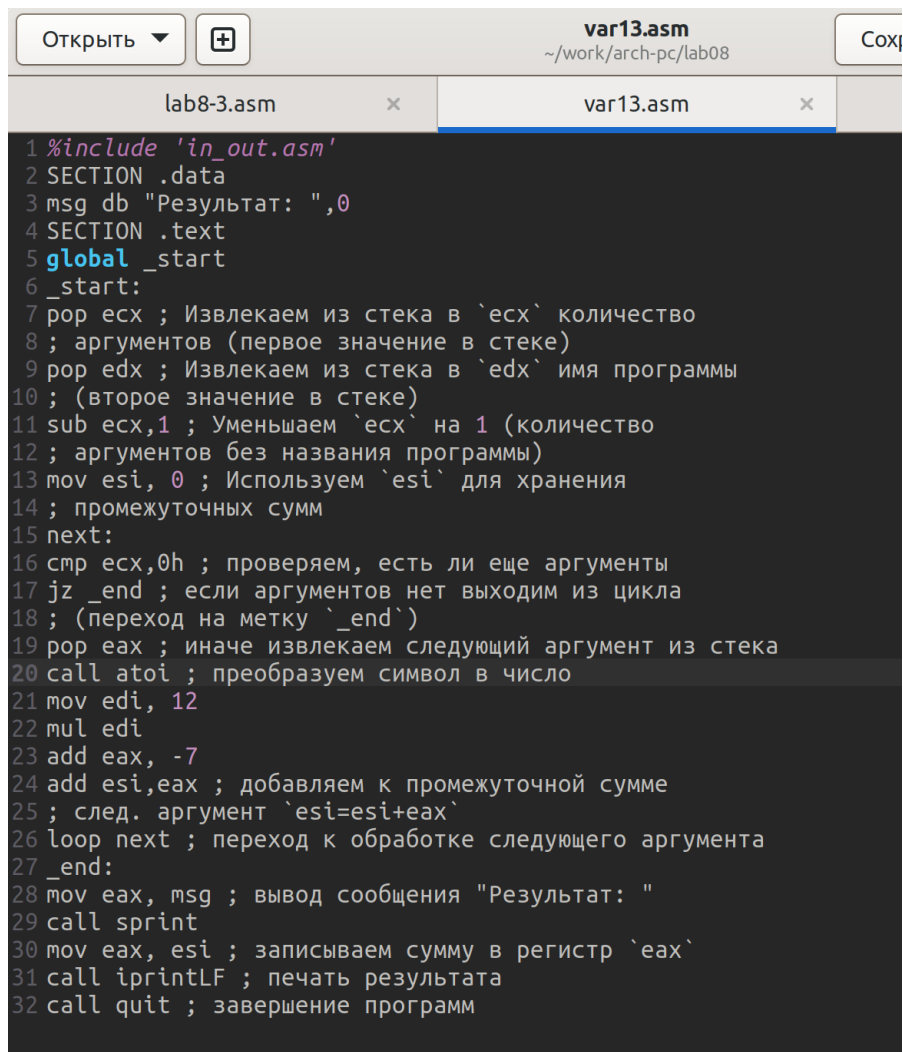
4.3 Задание для самостоятельной работы

1. Напишу программу, которая находит сумму значений функции $f(x)$ для $x=x_1, x_2, \dots, x_n$, т.е. программа должна выводить значение $f(x_1) + f(x_2) + \dots + f(x_n)$.

Так как у меня 13 вариант был при выполнении 6 лабораторной, то пишу программу для функции: $f(x)=12x - 7$. Создаю файл `var13.asm` с помощью `touch` и ввожу текст программы (рис. [4.17]-[4.18]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ touch var13.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ ls
in_out.asm  lab8-1.asm  lab8-2      lab8-2.o  lab8-3.asm  var13.asm
lab8-1      lab8-1.o    lab8-2.asm  lab8-3    lab8-3.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$
```

Рис. 4.17: Создание файла



```
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)
13 mov esi, 0 ; Используем `esi` для хранения
14 ; промежуточных сумм
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку `_end`)
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 mov edi, 12
22 mul edi
23 add eax, -7
24 add esi,eax ; добавляем к промежуточной сумме
25 ; след. аргумент `esi=esi+eax`
26 loop next ; переход к обработке следующего аргумента
27 _end:
28 mov eax, msg ; вывод сообщения "Результат: "
29 call sprint
30 mov eax, esi ; записываем сумму в регистр `eax`
31 call iprintLF ; печать результата
32 call quit ; завершение программ
```

Рис. 4.18: Текст программы

Текст программы:

```
%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, 12
mul edi
add eax, -7
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 sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы

```

Создаю исполняемый файл и проверяю его работу на нескольких наборах $x=x_1, x_2, \dots, x_n$ (рис. [4.19]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ nasm -f elf var13.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ ld -m elf_i386 -o var13 var13.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ ./var13 1 2
Результат: 22
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$ ./var13 3 2 4 12 4
Результат: 265
zrdagdelen@zrdagdelen:~/work/arch-pc/lab08$
```

Рис. 4.19: Создание исполняемого файла и проверка его работы

Программа работает верно.

5 Выводы

Я приобрела навыки написания программ с использованием циклов и обработкой аргументов командной строки.

6 Список литературы

Архитектура ЭВМ