

Отчёт по лабораторной работе №8

Дисциплина: архитектура компьютера

Репкина Елизавета Андреевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Выводы	17
	Список литературы	18

Список иллюстраций

4.1	Выполнение команд	9
4.2	Редактирование файла	9
4.3	Запуск файла	10
4.4	Редактирование файла	10
4.5	Запуск файла	10
4.6	Редактирование файла	11
4.7	Запуск файла	11
4.8	Создание файла	11
4.9	Редактирование файла	12
4.10	Запуск исполняемого файла	12
4.11	Создание файла	12
4.12	Редактирование файла	13
4.13	Запуск исполняемого файла	13
4.14	Редактирование файла	14
4.15	Запуск исполняемого файла	14
4.16	Создание файла	14
4.17	Редактирование файла	15
4.18	Запуск исполняемого файла	15

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. Задания для самостоятельной работы

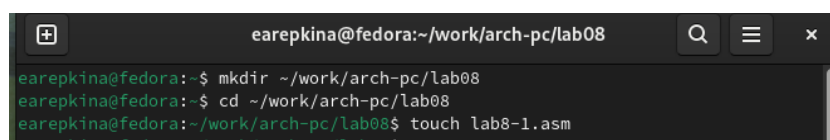
3 Теоретическое введение

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. На рис. 8.1 показана схема организации стека в процессоре. 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). 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. Извлечение элемента из стека. Команда pop извлекает зна-

чение 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 `ropf` для перемещения значений из вершины стека в регистр флагов.

4 Выполнение лабораторной работы

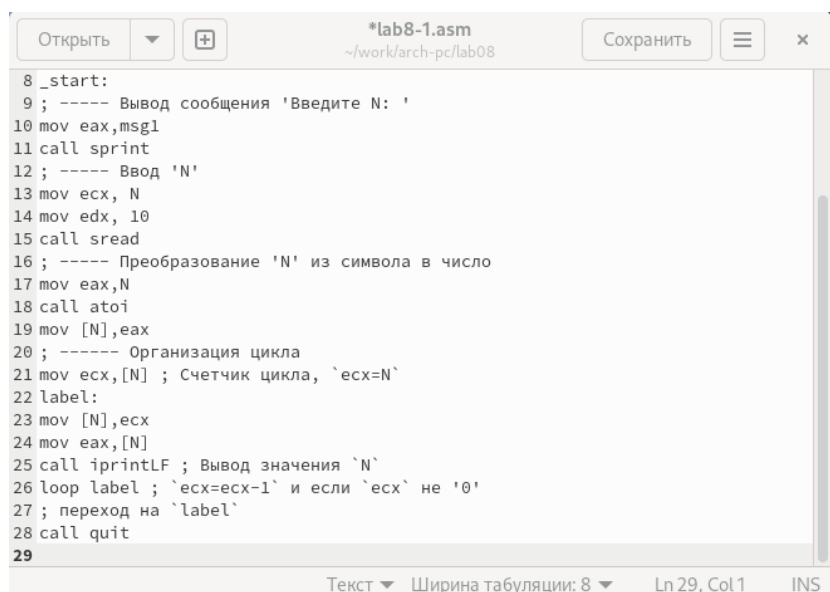
1. Реализация циклов в NASM Создаю каталог для программ лабораторной работы № 8, перехожу в него и создаю файл lab8-1.asm (рис. 4.1)



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

Рис. 4.1: Выполнение команд

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



```
*lab8-1.asm
~/work/arch-pc/lab08
Сохранить

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 mov [N],ecx
24 mov eax,[N]
25 call iprintLF ; Вывод значения `N`
26 loop label ; `ecx=ecx-1` и если `ecx` не '0'
27 ; переход на `label`
28 call quit
29
```

Рис. 4.2: Редактирование файла

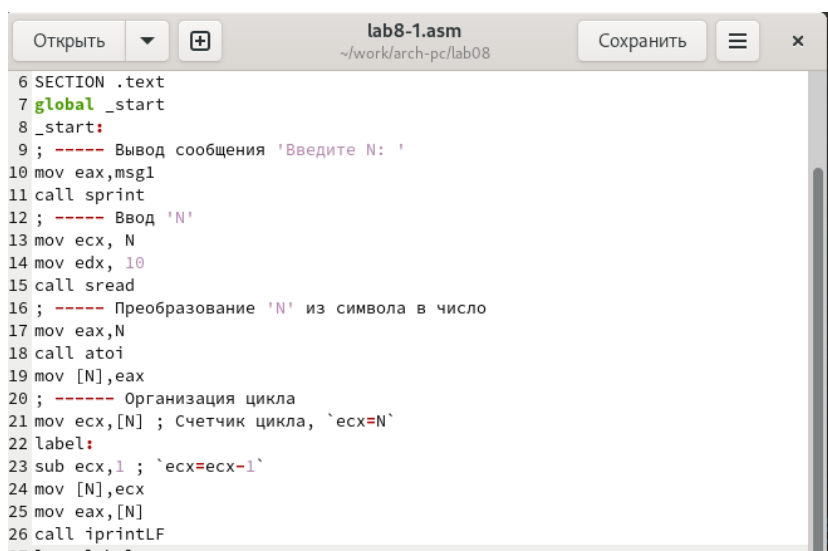
Создаю исполняемый файл и запускаю его.(рис. 4.3)

```
earepkina@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
earepkina@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
earepkina@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 4
4
3
2
1
earepkina@fedora:~/work/arch-pc/lab08$
```

Рис. 4.3: Запуск файла

Данный пример показывает, что использование регистра `ecx` в теле цикла `loop` может привести к некорректной работе программы.

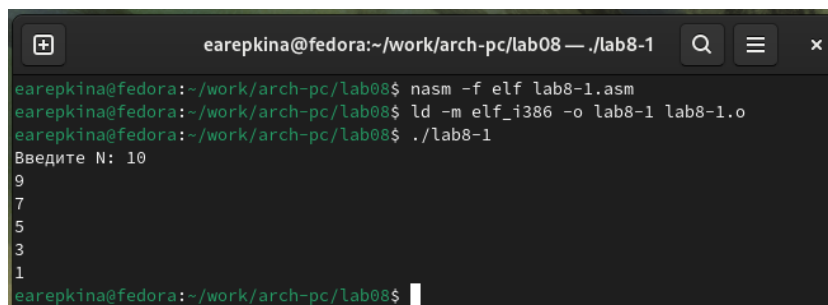
Меняю текст программы добавив изменение значение регистра `ecx` в цикле (рис. 4.4)



```
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
```

Рис. 4.4: Редактирование файла

Создаю исполняемый файл и запускаю его.(рис. 4.5)

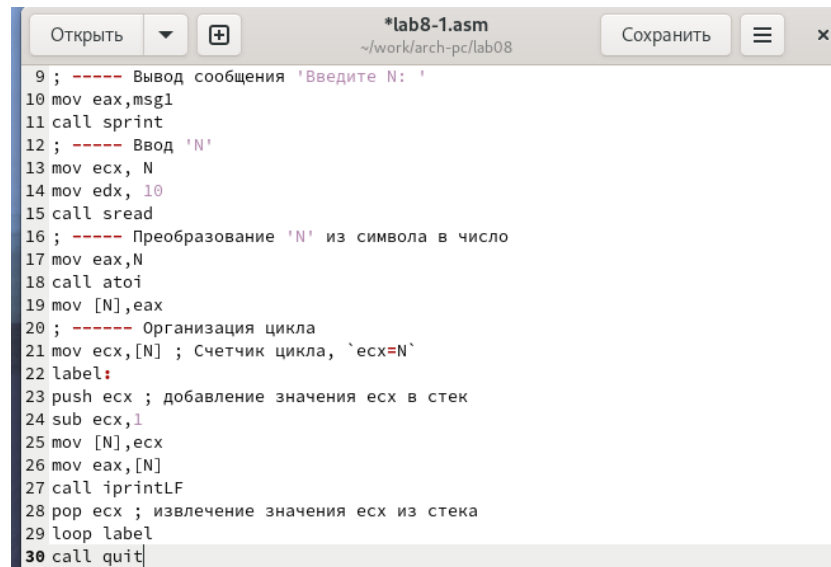


```
earepkina@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
earepkina@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
earepkina@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 10
9
7
5
3
1
earepkina@fedora:~/work/arch-pc/lab08$
```

Рис. 4.5: Запуск файла

В данном случае число проходов цикла не соответствует значению N.

Вношу изменения в текст программы, добавив команды push и pop(рис. 4.6)

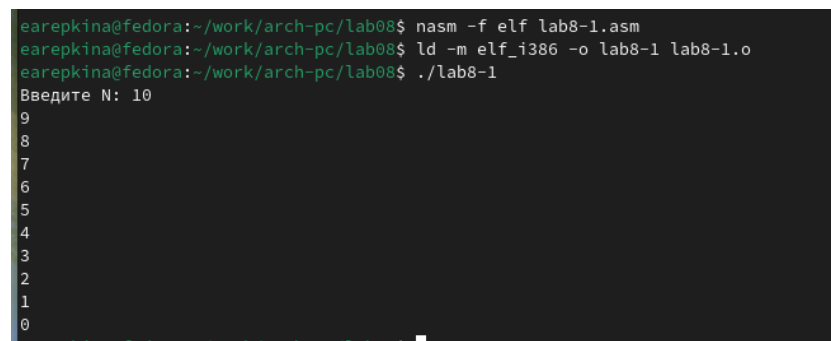


```
*lab8-1.asm
~/work/arch-pc/lab08

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 call quit
```

Рис. 4.6: Редактирование файла

Создаю исполняемый файл и запускаю его.(рис. 4.7)



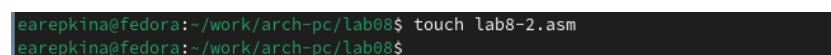
```
earepkina@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
earepkina@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
earepkina@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
earepkina@fedora:~/work/arch-pc/lab08$
```

Рис. 4.7: Запуск файла

В данном случае число проходов цикла соответствует значению N.

2. Обработка аргументов командной строки

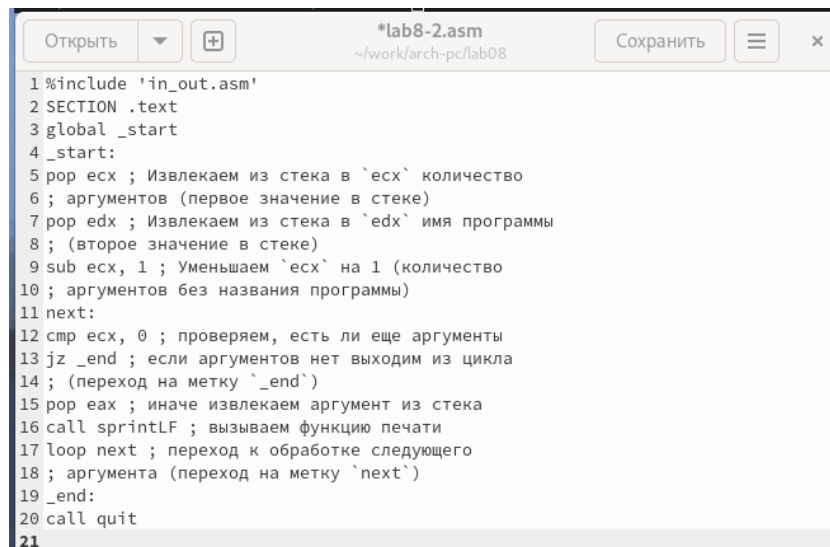
Создаю файл с названием lab8-2.asm (рис. 4.8)



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

Рис. 4.8: Создание файла

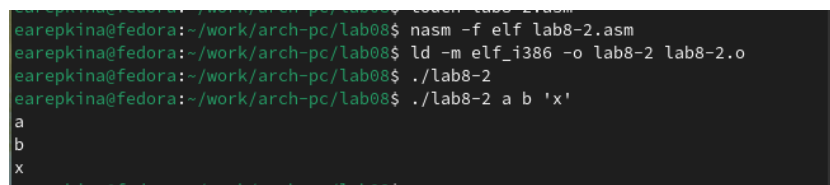
ввожу в него текст программы из листинга 8.2 (рис. 4.9)



```
1 %include 'in_out.asm'
2 SECTION .text
3 global _start
4 _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
21
```

Рис. 4.9: Редактирование файла

Создаю исполняемый файл и запускаю его, указав аргументы (рис. 4.10)




```
earepkina@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
earepkina@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
earepkina@fedora:~/work/arch-pc/lab08$ ./lab8-2
a
b
x
earepkina@fedora:~/work/arch-pc/lab08$
```

Рис. 4.10: Запуск исполняемого файла

Программа обработала 3 аргумента.

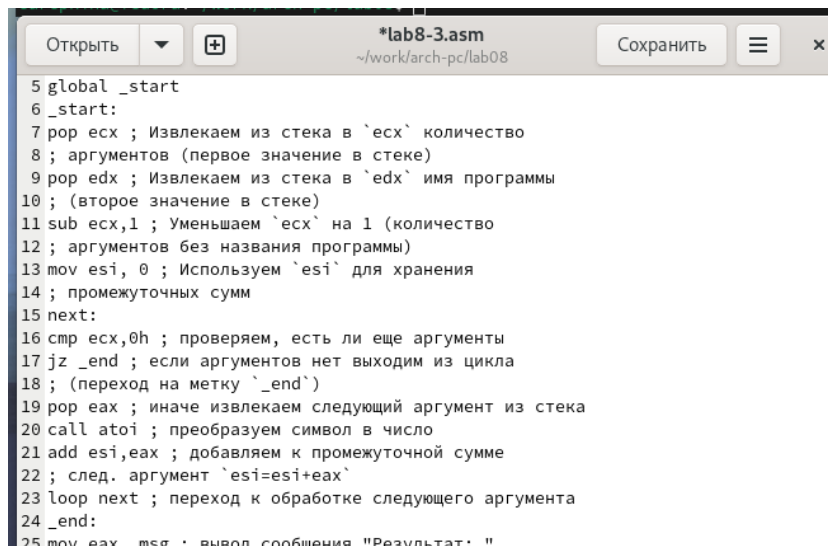
Создаю файл lab8-3.asm (рис. 4.11)



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

Рис. 4.11: Создание файла

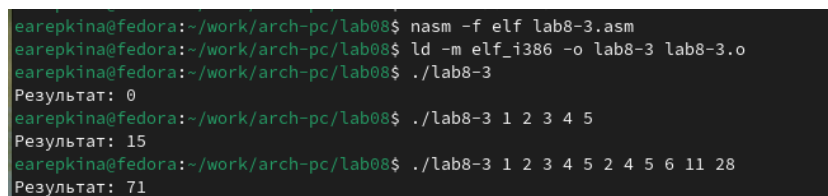
ввожу в него программу из листинга 8.3 (рис. 4.12)



```
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: "
```

Рис. 4.12: Редактирование файла

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




```
earepkina@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
earepkina@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
earepkina@fedora:~/work/arch-pc/lab08$ ./lab8-3
Результат: 0
earepkina@fedora:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 4 5
Результат: 15
earepkina@fedora:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 4 5 2 4 5 6 11 28
Результат: 71
```

Рис. 4.13: Запуск исполняемого файла

Программа работает.

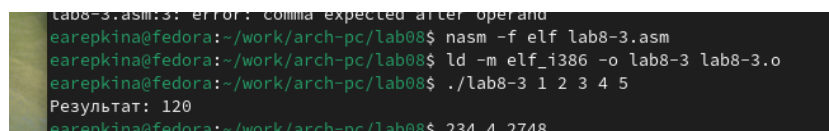
Теперь изменяю текст программы из листинга 8.3 так, чтобы она вычисляла произведение аргументов каждой строки (рис. 4.14)



```
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, 1
11 next:
12 cmp ecx,0h
13 jz _end
14 pop eax
15 call atoi
16 mul esi
17 mov esi, eax
18 loop next ;
19 _end:
20 mov eax, msg
21 call sprint
22 mov eax, esi
23 call iprintLF
24 call quit
```

Рис. 4.14: Редактирование файла

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

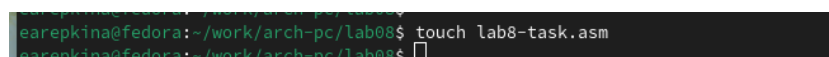


```
lab8-3.asm:3: error: comma expected after operand
earepkina@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
earepkina@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
earepkina@fedora:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 4 5
Результат: 120
earepkina@fedora:~/work/arch-pc/lab08$ 234 4 2748
```

Рис. 4.15: Запуск исполняемого файла

3. Задания для самостоятельной работы

Создаю файл lab8-task.asm (рис. 4.16)



```
earepkina@fedora:~/work/arch-pc/lab08$ touch lab8-task.asm
earepkina@fedora:~/work/arch-pc/lab08$
```

Рис. 4.16: Создание файла

Начинаю написание программы, которая будет вычислять сумму значений $f(x)=7(x+1)$. (вариант 14) (рис. 4.17)

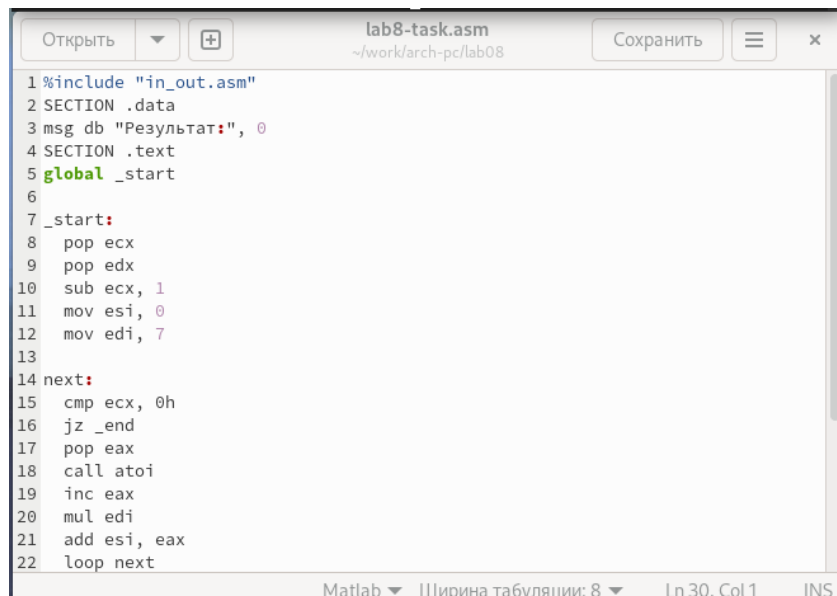


Рис. 4.17: Редактирование файла

Создаю исполняемый файл и запускаю его. (рис. 4.18)

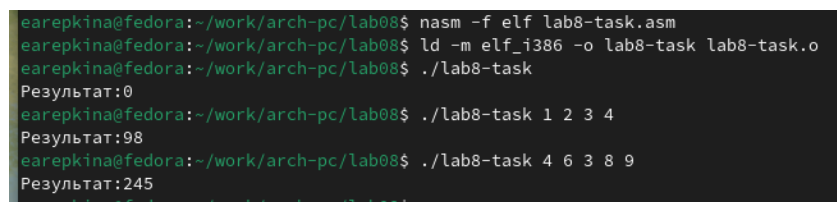


Рис. 4.18: Запуск исполняемого файла

Произведя несложные математические вычисления, делаю вывод, что программа работает верно

Текст программы:

%include "in_out.asm"

SECTION .data

msg db "Результат:", 0

SECTION .text

global _start

_start:

pop ecx

```
pop edx
sub ecx, 1
mov esi, 0
mov edi, 7
next:
cmp ecx, 0h
jz _end
pop eax
call atoi
inc eax
mul edi
add esi, eax
loop next
_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
```


5 Выводы

После выполнения данной лабораторной работы я приобрела навыки написания программ с использованием циклов и обработкой аргументов командной строки

Список литературы