

Отчёт по лабораторной работе №8

Дисциплина: Архитектура компьютера

Луангсуваннавонг Сайпхачан

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
4.1	Реализация переходов в NASM	8
4.2	Обработка аргументов командной строки	13
4.3	Выполнение заданий для самостоятельной работы	19
5	Выводы	23
6	Ответы на вопросы для самопроверки	24
7	Список литературы	26

Список иллюстраций

4.1	Создание файла и каталога	8
4.2	Копирование файла	8
4.3	Редактирование файла	9
4.4	Запуск исполняемого файла	10
4.5	Редактирование файла	10
4.6	Запуск исполняемого файла	10
4.7	Результат работы программы	11
4.8	Результат работы программы	12
4.9	Редактирование файла	12
4.10	Запуск исполняемого файла	13
4.11	Создание файла	13
4.12	Редактирование файла	14
4.13	Запуск исполняемого файла	14
4.14	Создание файла	15
4.15	Редактирование файла	15
4.16	Запуск исполняемого файла	16
4.17	Редактирование файла	17
4.18	Запуск исполняемого файла	18
4.19	Создание файла	19
4.20	Редактирование файла	20
4.21	Запуск исполняемого файла	21

1 Цель работы

Целью данной лабораторной работы является 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. Выполнение заданий для самостоятельной работы

3 Теоретическое введение

Стек — это структура данных, организованная по принципу LIFO (**“Last In — First Out”** или **“последним пришёл — первым ушёл”**), то есть последний элемент, добавленный в стек, будет первым, который из него извлечётся. Стек является важной частью архитектуры процессора и реализуется на аппаратном уровне. Для работы со стеком используются специальные регистры, такие как `ss`, `bp` и `sp`.

Основная функция стека — это хранение адресов возврата и передача аргументов при вызове функций. Также в стеке выделяется память для локальных переменных и могут храниться временные значения регистров. Стек имеет вершину, которая хранит адрес последнего добавленного элемента и находится в регистре `esp` (указатель стека). Противоположная вершине часть называется дном стека. При добавлении элемента в стек значение указателя `esp` уменьшается, а при извлечении — увеличивается.

Существует две основные операции с элементами стека:

`Push` — добавляет элемент в вершину стека. После этого регистр `esp` уменьшается на 4.

`Pop` — извлекает элемент из вершины стека. Регистр `esp` увеличивается на 4 после извлечения.

Команда `push` имеет один операнд — значение, которое нужно поместить в стек. Также есть дополнительные команды для добавления нескольких значений в стек:

`pusha` — помещает в стек содержимое всех регистров общего назначения в

следующем порядке: `ax`, `cx`, `dx`, `bx`, `sp`, `bp`, `si`, `di`. Эта команда не требует операндов.

`pushf` — помещает в стек содержимое регистра флагов. Эта команда также не требует операндов.

Команда `pop` извлекает значение из вершины стека и сохраняет его в указанный операнд (регистр или память). Значение не стирается из памяти и остаётся как “мусор” до тех пор, пока не будет перезаписано новым значением.

Аналогично команде `push` существует команда `popa`, которая восстанавливает все регистры общего назначения из стека, а также команда `popf`, которая восстанавливает значение регистра флагов из стека.

Для организации циклов в ассемблере существуют специальные инструкции. Одной из самых простых является команда `loop`, которая позволяет создать безусловный цикл. Типичная структура цикла выглядит следующим образом:

```
mov ecx, 100    ; Устанавливаем количество итераций
NextStep:
...
... ; тело цикла
...
loop NextStep   ; Повторить цикл `ecx` раз, переходя к метке NextStep
```

Команда `loop` выполняется в два этапа: сначала из регистра `ecx` вычитается единица и его значение сравнивается с нулём. Если значение не равно нулю, то выполнение переходит к указанной метке. Если значение регистра равно нулю, переход не выполняется, и управление передается следующей инструкции, после команды `loop`.

4 Выполнение лабораторной работы

4.1 Реализация переходов в NASM

Я создаю новую директорию, в которой буду создавать файлы с программами для лабораторной работы № 8, используя команду `mkdir`. Затем я перехожу в созданный каталог и создаю файл `lab8-1.asm`, используя команду `touch`. (Рис.4.1)

```
sayprachanh@desktop:~$ mkdir work/study/2024-2025/Архитектура\ компьютера/arch-pc/lab08
sayprachanh@desktop:~$ cd work/study/2024-2025/Архитектура\ компьютера/arch-pc/lab08
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ touch lab8-1.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ls
lab8-1.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

Рис. 4.1: Создание файла и каталога

Я копирую файл `in_out.asm` из последней лабораторной работы, потому что он будет использоваться в других программах (Рис.4.2)

```
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ cp ~/work/study/2024-2025/Архитектура\ компьютера/arch-pc/lab07/in_out.asm .
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ls
in_out.asm  lab8-1.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

Рис. 4.2: Копирование файла

Я открываю созданный файл `lab8-1.asm`, затем вставляю программу, которая реализует цикл (Рис.4.3)


```

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

```

Рис. 4.3: Редактирование файла

Я создаю новый исполняемый файл и запускаю его (Рис.4.4). Я ввожу значение в программу, и программа выводит введенное мной число, а затем следующее число, которое со временем будет уменьшаться.

```

sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf lab8-1.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-1
Введите N: 3
3
2
1
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$

```

Рис. 4.4: Запуск исполняемого файла

Я снова захожу к файлу программы и изменяю программу, добавляя инструкцию: `sub ecx, 1` (Рис.4.5)

```

30  label:
31  sub ecx, 1      ; `ecx=ecx-1`
32  mov [N],ecx
33  mov eax,[N]
34  call iprintLF ; Вывод значения `N`
35  loop label      ; `ecx=ecx-1` и если `ecx` не `0`

```

Рис. 4.5: Редактирование файла

Я создаю исполняемый файл и запускаю его. Затем я ввожу это значение в программу (Рис.4.6)

```

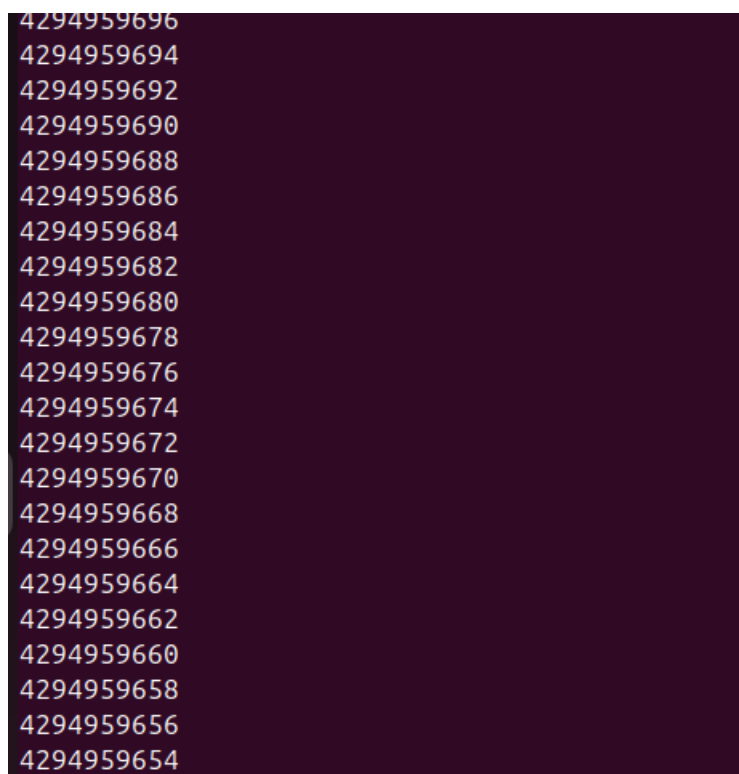
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf lab8-1.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-1
Введите N: 3

```

Рис. 4.6: Запуск исполняемого файла

В результате программа выводит значение не в соответствии с тем, которое было введено, а выводит большое количество чисел, которые продолжают за циклироваться и вычитаться, это создает бесконечный цикл (Рис.4.7)

В программе, `ecx` принимает значение `N`, равное 3, и уменьшается на 2 (один раз с помощью `sub ecx, 1` и один раз с помощью цикла(`loop`)). В инструкции `loop`, если `ecx` равен нулю, цикл остановится, поскольку `ecx` уменьшается на 2, он проходит проверку нулевой точки, что означает, что цикл будет бесконечным



```
4294959696
4294959694
4294959692
4294959690
4294959688
4294959686
4294959684
4294959682
4294959680
4294959678
4294959676
4294959674
4294959672
4294959670
4294959668
4294959666
4294959664
4294959662
4294959660
4294959658
4294959656
4294959654
```

Рис. 4.7: Результат работы программы

Но если я введу значение 2 (или любые четные числа) в программу, она выполнит цикл и выведет четные числа, поскольку `esx` уменьшается на 2 при каждом цикле. (Рис.4.8)

```

sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-1
Введите N: 2
1
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-1
Введите N: 4
3
1
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-1
Введите N: 20
19
17
15
13
11
9
7
5
3
1
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$

```

Рис. 4.8: Результат работы программы

Я снова захожу к файлу программы и изменяю программу. На этот раз я добавляю инструкции `push ecx` и `pop ecx` (Рис.4.9)

```

30 label:
31 push ecx      ; добавление значения ecx в стек
32 sub ecx, 1    ; `ecx=ecx-1`
33 mov [N],ecx
34 mov eax,[N]
35 call iprintLF ; Вывод значения `N`
36 pop ecx      ; извлечение значения ecx из стека
37
38 loop label    ; `ecx=ecx-1` и если `ecx` не '0'

```

Рис. 4.9: Редактирование файла

Я создаю исполняемый файл и запускаю его. (Рис.4.10) Затем я ввожу значение в программу, она выполняет цикл и выводит число в соответствии со значением, по которому я хочу выполнить цикл.

```

sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf lab8-1.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-1
Введите N: 3
2
1
0
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$

```

Рис. 4.10: Запуск исполняемого файла

4.2 Обработка аргументов командной строки

Я создаю новый файл lab8-2.asm с помощью touch (Рис.4.11)

```

sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ touch lab8-2.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ls
in_out.asm lab8-1 lab8-1.asm lab8-1.o lab8-2.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$

```

Рис. 4.11: Создание файла

Я открываю созданный файл lab8-2.asm и вставляю программу, которая отображает аргументы командной строки (Рис.4.12)

```

1  %include 'in_out.asm'
2
3  SECTION .text
4  global _start
5  _start:
6
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
12  sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
13            ; аргументов без названия программы)
14
15  next:
16  cmp ecx, 0 ; проверяем, есть ли еще аргументы
17  jz _end    ; если аргументов нет выходим из цикла
18            ; (переход на метку `_end`)
19  pop eax    ; иначе извлекаем аргумент из стека
20  call printf ; вызываем функцию печати
21  loop next  ; переход к обработке следующего
22            ; аргумента (переход на метку `next`)
23  _end:
24  call quit

```

Рис. 4.12: Редактирование файла

Я создаю исполняемый файл и запускаю его. (Рис.4.13) В результате программа выводит 4 аргумента, так как в аргументе командной строки, строки 'аргумент' и '2' не связаны друг с другом (между ними есть пробел). Поэтому программа рассматривает это как два аргумента вместо одного и выводит "аргумент" и "2" отдельно.

```

sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf lab8-2.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$

```

Рис. 4.13: Запуск исполняемого файла

Я создаю новый файл lab8-3.asm с помощью touch (Рис.4.14)

```
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ touch lab8-3.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

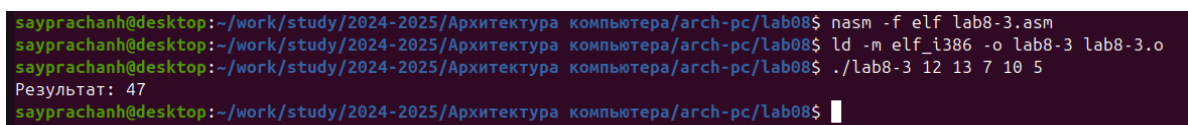
Рис. 4.14: Создание файла

Я открываю созданный файл lab8-3.asm и вставляю программу (Рис.4.15)

```
1  %include 'in_out.asm'
2
3  SECTION .data
4  msg db "Результат: ",0
5
6  SECTION .text
7  global _start
8  _start:
9
10 pop ecx ; Извлекаем из стека в `ecx` количество
11         ; аргументов (первое значение в стеке)
12 pop edx ; Извлекаем из стека в `edx` имя программы
13         ; (второе значение в стеке)
14 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
15         ; аргументов без названия программы)
16 mov esi, 0 ; Используем `esi` для хранения
17         ; промежуточных сумм
18
19 next:
20 cmp ecx,0h ; проверяем, есть ли еще аргументы
21 jz _end    ; если аргументов нет выходим из цикла
22           ; (переход на метку `_end`)
23 pop eax
24 call atoi ; иначе извлекаем следующий аргумент из стека
25           ; преобразуем символ в число
26 add esi,eax ; добавляем к промежуточной сумме
27           ; след. аргумент `esi=esi+eax`
28 loop next  ; переход к обработке следующего аргумента
29
30 _end:
31 mov eax, msg ; вывод сообщения "Результат: "
32 call sprint
33 mov eax, esi ; записываем сумму в регистр `eax`
34 call iprintLF ; печать результата
35 call quit ; завершение программы
```

Рис. 4.15: Редактирование файла

Я создаю исполняемый файл и запускаю его (Рис.4.16), ввожу в программу аргументы в виде чисел, и в результате она выводит сумму чисел, которые были переданы в программу.



```
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf lab8-3.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

Рис. 4.16: Запуск исполняемого файла

Я снова захожу к файлу программы и изменяю программу для вычисления произведения аргументов командной строки. (Рис.4.17)


```

1  %include 'in_out.asm'
2
3  SECTION .data
4  msg db "Результат: ",0
5
6  SECTION .text
7  global _start
8  _start:
9
10 pop ecx ; Извлекаем из стека в `ecx` количество
11         ; аргументов (первое значение в стеке)
12 pop edx ; Извлекаем из стека в `edx` имя программы
13         ; (второе значение в стеке)
14 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
15         ; аргументов без названия программы)
16 mov esi, 1 ; Используем `esi` для хранения
17         ; промежуточных сумм
18
19 next:
20 cmp ecx,0h ; проверяем, есть ли еще аргументы
21 jz _end    ; если аргументов нет выходим из цикла
22           ; (переход на метку `_end`)
23 pop eax
24 call atoi ; иначе извлекаем следующий аргумент из стека
25           ; преобразуем символ в число
26 mul esi   ; умножение на esi
27 mov esi, eax ; след. аргумент `esi=eax`
28
29 loop next ; переход к обработке следующего аргумента
30
31 _end:
32 mov eax, msg ; вывод сообщения "Результат: "
33 call sprint
34 mov eax, esi ; записываем сумму в регистр `eax`
35 call iprintLF ; печать результата
36 call quit ; завершение программы

```

Рис. 4.17: Редактирование файла

Я создаю исполняемый файл и запускаю его. (Рис.4.18) Я вставляю аргументы командной строки, и на этот раз программа выводит результат умножения аргументов.

```

sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf lab8-3.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 54600
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$

```

Рис. 4.18: Запуск исполняемого файла

Код редактируемой 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 сумм
```

```
next:
```

```
cmp ecx,0h ; проверяем, есть ли еще аргументы
```

```
jz _end ; если аргументов нет выходим из цикла
        ; (переход на метку `_end`)
```

```
pop eax
```

```

call atoi ; иначе извлекаем следующий аргумент из стека
          ; преобразуем символ в число
mul esi   ; умножение на esi
mov esi, eax ; след. аргумент `esi=eax`

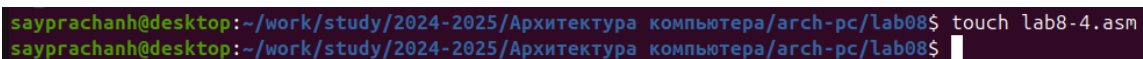
loop next ; переход к обработке следующего аргумента

_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

```

4.3 Выполнение заданий для самостоятельной работы

Сначала я создаю новый файл lab8-4.asm, используя команду touch (Рис.4.19)



```

sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ touch lab8-4.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$

```

Рис. 4.19: Создание файла

Я открываю созданный файл lab8-4.asm и начинаю вставлять программу (Рис.4.20). Поскольку мой вариант равен 13 (из лабораторной работы № 6), моя функция будет $f(x) = 12x - 7$.

```

1  %include 'in_out.asm' ; подключение внешнего файла
2
3  section .data
4  msg1 db "Функция: f(x)=12x-7 ",0
5  result db "Результат: ",0
6
7  section .text
8  global _start
9  _start:
10 pop ecx
11 pop edx
12 sub ecx, 1
13 mov esi, 0 ; esi = 0
14
15 next:
16 cmp ecx, 0
17 jz _end
18
19 pop eax
20 call atoi
21 mov ebx, 12
22 mul ebx ; eax = eax * ebx(12)
23
24 mov ebx, 7
25 sub eax, ebx ; eax = eax - ebx(7)
26
27 add esi, eax ; esi = esi + eax
28
29 loop next
30
31 _end:
32 mov eax, msg1
33 call sprintf ;Вывод сообщения 'Функция: f(x)=12x-7 '
34 mov eax, result
35 call sprintf ;Вывод сообщения 'Результат: '
36 mov eax, esi
37 call sprintf ;Результат
38
39 call quit
~

```

Рис. 4.20: Редактирование файла

После этого я создаю исполняемый файл и запускаю его (Рис.4.21). Я ввожу аргументы командной строки в программу, и она выдает результат. Чтобы прове-

ритель правильность работы программы, я вычисляю результат самостоятельно. Программа работает правильно.

```
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf lab8-4.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 -o lab8-4 lab8-4.o
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-4 1 2 3 4
Функция: f(x)=12x-7
Результат: 92
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-4 0 1 2
Функция: f(x)=12x-7
Результат: 15
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-4 -1 0 1 2
Функция: f(x)=12x-7
Результат: 8
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

Рис. 4.21: Запуск исполняемого файла

Программа для выполнения задачи 1

```
%include 'in_out.asm' ; подключение внешнего файла
```

```
section .data
```

```
msg1 db "Функция: f(x)=12x-7 ",0
```

```
result db "Результат: ",0
```

```
section .text
```

```
global _start
```

```
_start:
```

```
pop ecx
```

```
pop edx
```

```
sub ecx, 1
```

```
mov esi, 0 ; esi = 0
```

```
next:
```

```
cmp ecx, 0
```

```
jz _end
```

```

pop eax
call atoi
mov ebx, 12
mul ebx      ; eax = eax * ebx(12)

mov ebx, 7
sub eax, ebx ; eax = eax - ebx(7)

add esi, eax ; esi = esi + eax

loop next

_end:
mov eax, msg1
call sprintf      ; Вывод сообщения 'Функция: f(x)=12x-7 '
mov eax, result
call sprintf      ; Вывод сообщения 'Результат: '
mov eax, esi
call sprintf      ; Результат

call quit

```

5 Выводы

При выполнении данной лабораторной работы, Я приобрел навыки написания программ с использованием циклов и обработки аргументов командной строки.

6 Ответы на вопросы для самопроверки

1. Опишите работу команды loop.

Команда loop используется для создания цикла в ассемблере.

Команда loop уменьшает значение регистра есх на 1 и проверяет, не равно ли оно нулю. Если есх не ноль, происходит переход к метке (начало следующей итерации). Если есх равен нулю, выполнение продолжается с инструкции после loop.

Пример:

```
mov есх, 5
NextStep:
    ....
    .... ; тело цикла
    ....
    loop NextStep
```

2. Как организовать цикл с помощью команд условных переходов, не прибегая к специальным командам управления циклами?

Цикл можно создать, используя dec есх для уменьшения значения есх и команду jnz для проверки, не равен ли есх нулю. Пример:

```
mov есх, 5
NextStep:
```



```
....  
.... ; тело цикла  
....  
dec ecx  
jnz NextStep
```

3. Дайте определение понятия «стек».

Стек — это структура данных по принципу LIFO (последним пришёл — первым ушёл), где последний добавленный элемент извлекается первым. Он используется для хранения адресов возврата, аргументов и временных данных.

4. Как осуществляется порядок выборки содержащихся в стеке данных?

Данные извлекаются по принципу LIFO: последний добавленный элемент извлекается первым. При выполнении `pop` значение извлекается, а указатель стека `esp` увеличивается на 4.

7 Список литературы

Архитектура ЭВМ