

Лабораторная работа №8

Архитектура компьютера

Алехин Давид Андреевич

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Выводы	18
	Список литературы	19

Список иллюстраций

4.1	lab08/lab8-1.asm	9
4.2	8.1	9
4.3	./lab8-1	10
4.4	8.1v2	10
4.5	./8.1v2	11
4.6	8.1v3	12
4.7	./8.1v3	13
4.8	8.2	14
4.9	./8.2	14
4.10	8.3	15
4.11	./8.3	15
4.12	8.3v2	16
4.13	./8.3v2	16
4.14	lab8-1s.asm	17
4.15	./lab8-1s.asm	17

Список таблиц

1 Цель работы

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

2 Задание

3 Теоретическое введение

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. 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).

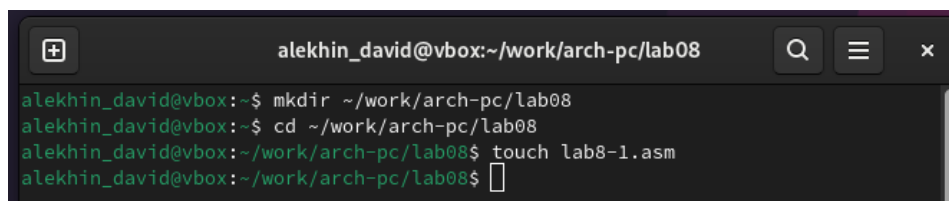
Добавление элемента в стек Команда push размещает значение в стеке, т.е. помещает значение в ячейку памяти, на которую указывает регистр esp, после этого значение регистра esp увеличивается на 4. Данная команда имеет один операнд — значение, которое необходимо поместить в стек. Существует ещё две команды 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 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. Примеры: `pop eax`; Поместить значение из стека в регистр `eax` `pop [buf]`; Поместить значение из стека в `buf` `pop word[si]`; Поместить значение из стека в слово по адресу в `si` Аналогично команде записи в стек существует команда `popa`, которая восстанавливает из стека все регистры общего назначения, и команда `popf` для перемещения значений из вершины стека в регистр флагов.

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

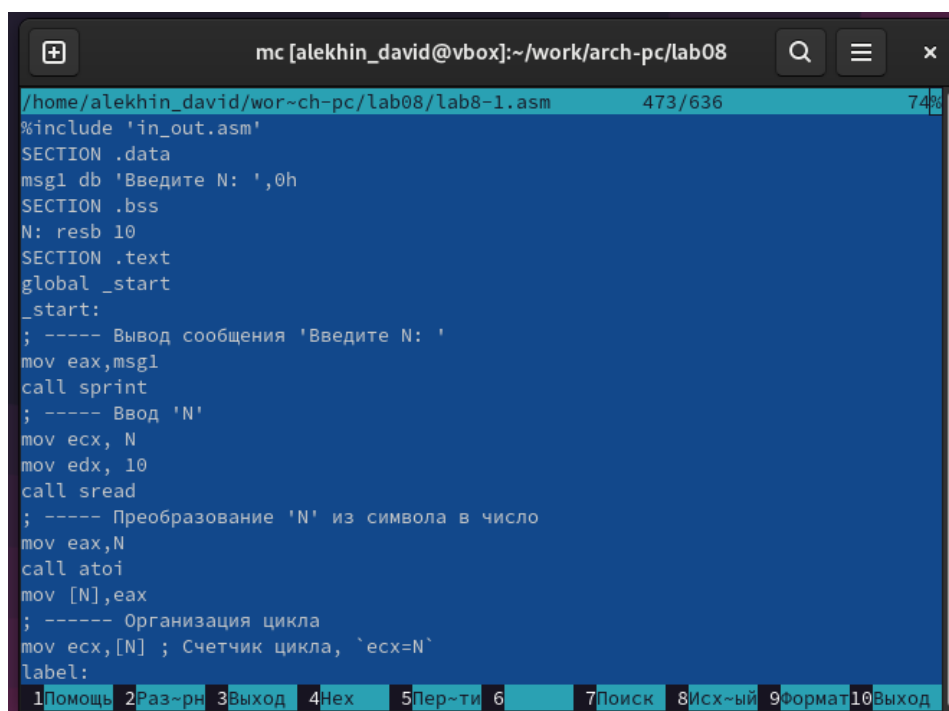
Создаю каталог lab08 и файл lab8-1.asm .(рис. 4.1).



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

Рис. 4.1: lab08/lab8-1.asm

Ввожу туда текст команды 8.1. Программа вывода значений регистра есх. (рис. 4.2).



```
/home/alekhin_david/work/arch-pc/lab08/lab8-1.asm 473/636 74%
%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
mov [N],eax
; ----- Организация цикла
mov ecx,[N] ; Счетчик цикла, `ecx=N`
label:
1Помощь 2Разрыв 3Выход 4Чех 5Пер-ти 6 7Поиск 8Исх-ый 9Формат10Выход
```

Рис. 4.2: 8.1

Компеллирую и запускаю команду 8.1. (рис. 4.3).

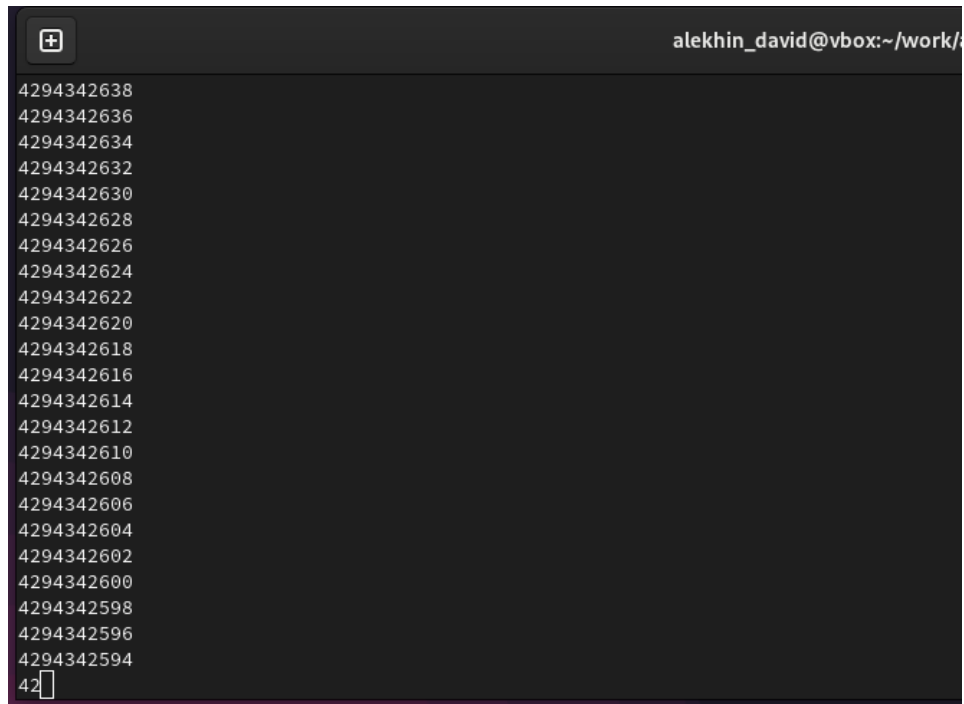
```
alekhin_david@vbox:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
alekhin_david@vbox:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
alekhin_david@vbox:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 12
12
11
10
9
8
7
6
5
4
3
2
1
alekhin_david@vbox:~/work/arch-pc/lab08$
```

Рис. 4.3: ./lab8-1

Меняю текст программы, в теле label добавляю строку `sub eax,1`. Циклы закольцевался и стал бесконечным. (рис. 4.4), (рис. 4.5).

```
mc [alekhin_david@vbox]~/work/arch-pc/lab08
lab8-1.asm [-M--] 10 L: [ 7+20 27/ 27] *(545 / 545b) <EOF> [*] [X]
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:
sub ecx,1 ; `ecx=ecx-1`
mov [N],ecx
mov eax,[N]
call iprintLF
loop label
```

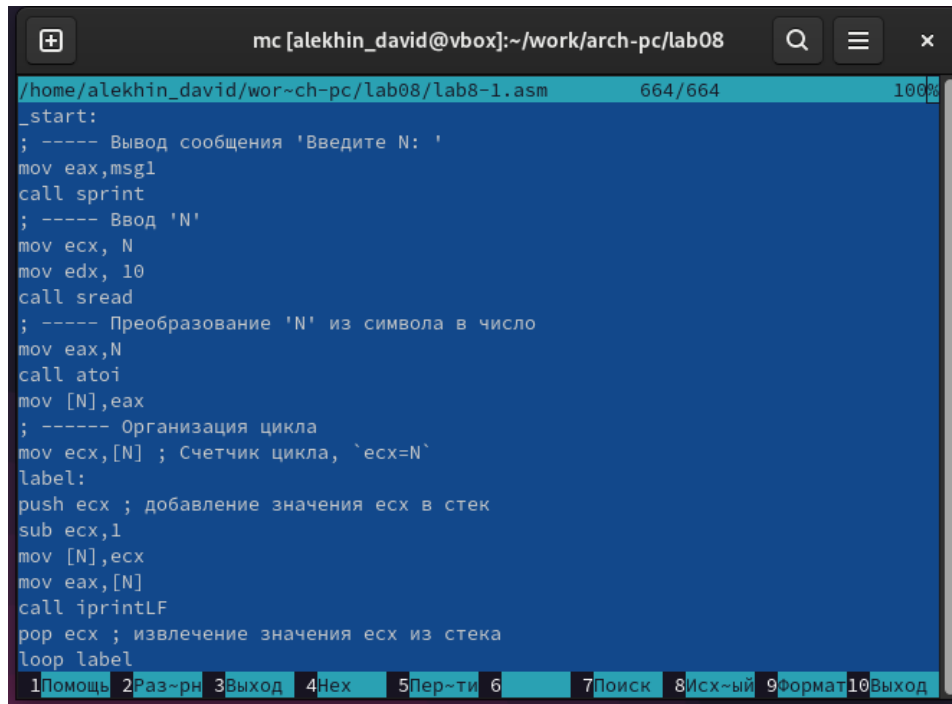
Рис. 4.4: 8.1v2



```
alekhin_david@vbox:~/work/
4294342638
4294342636
4294342634
4294342632
4294342630
4294342628
4294342626
4294342624
4294342622
4294342620
4294342618
4294342616
4294342614
4294342612
4294342610
4294342608
4294342606
4294342604
4294342602
4294342600
4294342598
4294342596
4294342594
42
```

Рис. 4.5: ./8.1v2

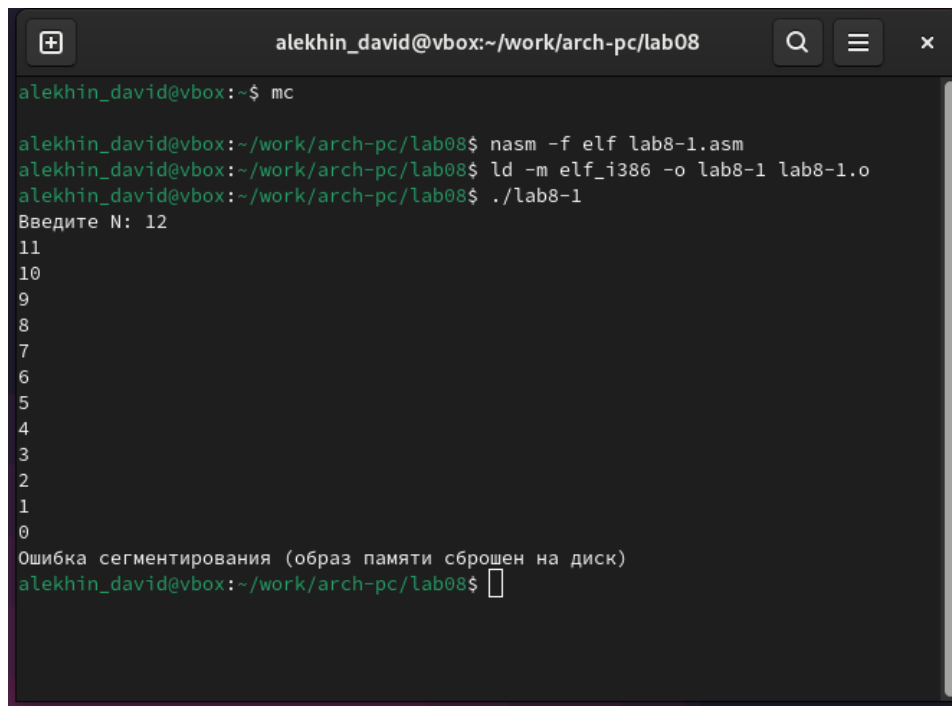
Меняю текст программы так, чтобы цикл и счетчик работали корректно. (рис. 4.6).



```
mc [alekhin_david@vbox]:~/work/arch-pc/lab08
/home/alekhin_david/work/arch-pc/lab08/lab8-1.asm 664/664 100%
_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
1Помощь 2Раз-рн 3Выход 4Нех 5Пер-ти 6 7Поиск 8Исх-ый 9Формат10Выход
```

Рис. 4.6: 8.1v3

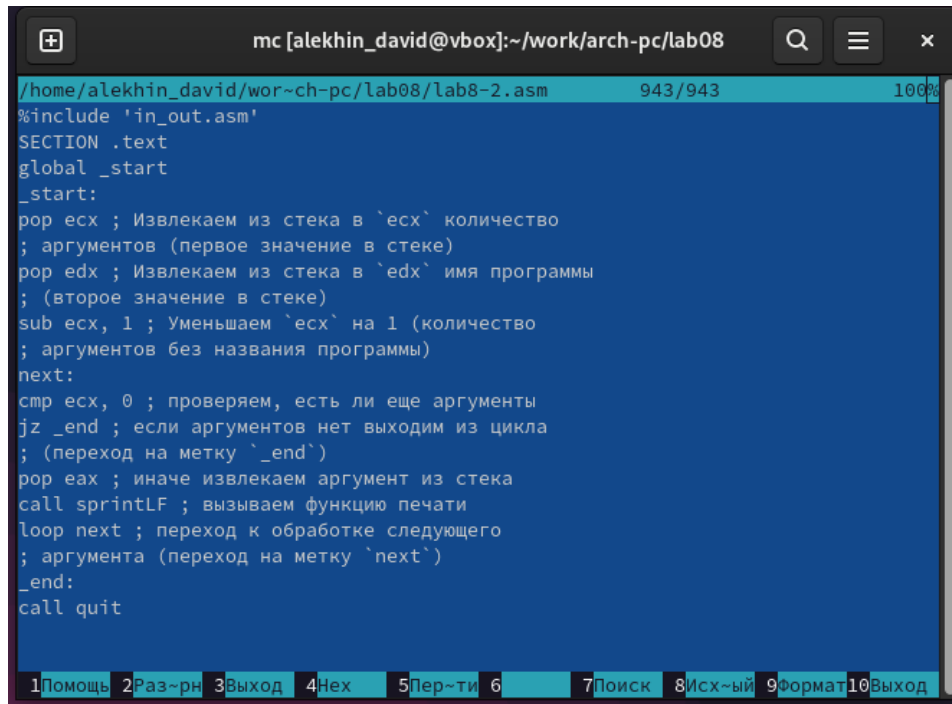
После изменения программы, число проходки циклов стал соответствовать числу введенному с клавиатуры. (рис. 4.7).



```
alekhin_david@vbox:~/work/arch-pc/lab08
alekhin_david@vbox:~$ mc
alekhin_david@vbox:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
alekhin_david@vbox:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
alekhin_david@vbox:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 12
11
10
9
8
7
6
5
4
3
2
1
0
Ошибка сегментирования (образ памяти сброшен на диск)
alekhin_david@vbox:~/work/arch-pc/lab08$
```

Рис. 4.7: ./8.1v3

Создаю lab8-2.asm и ввожу туда текст команды 8.2. Программа выводющая на экран аргументы командной строки. (рис. 4.8).

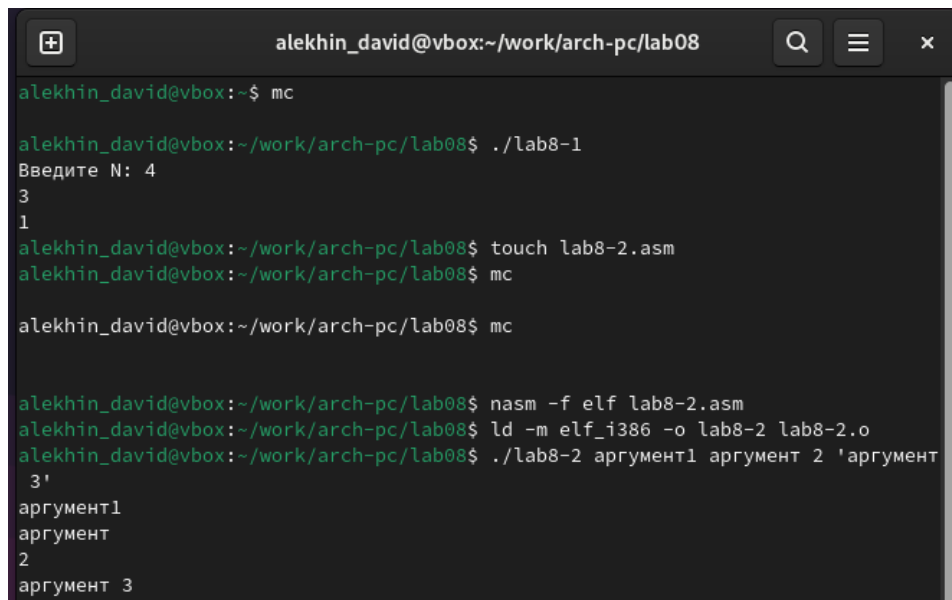


The screenshot shows a text editor window titled 'mc [alekhin_david@vbox]:~/work/arch-pc/lab08'. The file being edited is '/home/alekhin_david/work/arch-pc/lab08/lab8-2.asm' with 943/943 lines and 100% zoom. The code is written in assembly and includes comments in Russian. At the bottom, there is a menu bar with options: 1Помощь, 2Раз-рн, 3Выход, 4Нех, 5Пер-ти, 6, 7Поиск, 8Исх-ый, 9Формат, 10Выход.

```
%include 'in_out.asm'
SECTION .text
global _start
_start:
por ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
por 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`)
por eax ; иначе извлекаем аргумент из стека
call sprintf ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit
```

Рис. 4.8: 8.2

При запуске программа выводит все 3 аргумента которые ввели, но в разном виде. (рис. 4.9).

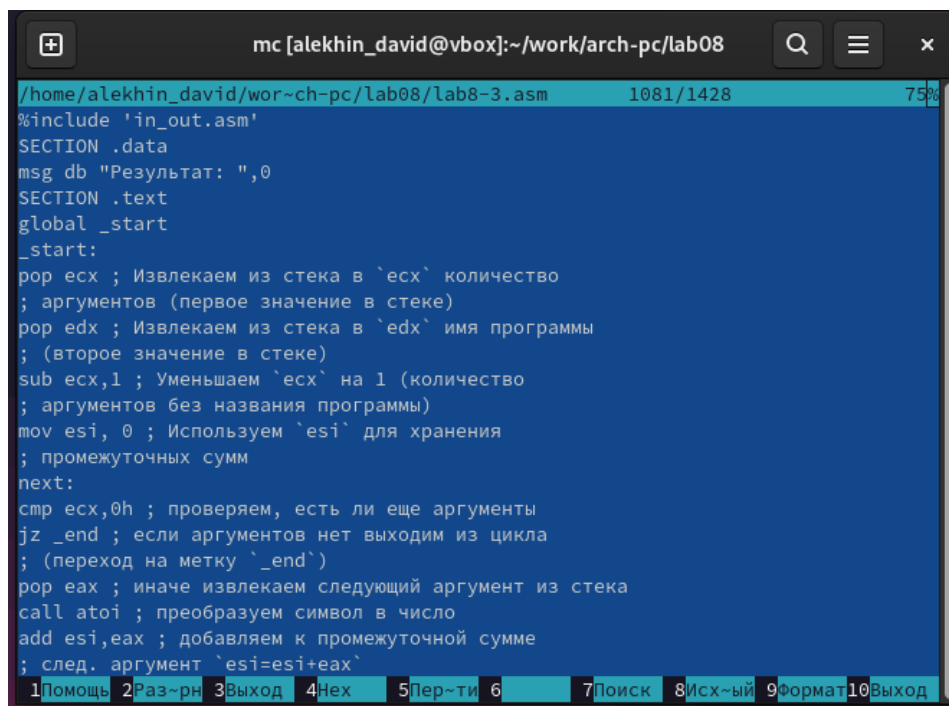


The screenshot shows a terminal window titled 'alekhin_david@vbox:~/work/arch-pc/lab08'. It displays the commands used to compile and run the program, and the output of the program.

```
alekhin_david@vbox:~$ mc
alekhin_david@vbox:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 4
3
1
alekhin_david@vbox:~/work/arch-pc/lab08$ touch lab8-2.asm
alekhin_david@vbox:~/work/arch-pc/lab08$ mc
alekhin_david@vbox:~/work/arch-pc/lab08$ mc
alekhin_david@vbox:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
alekhin_david@vbox:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
alekhin_david@vbox:~/work/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент 2 'аргумент
3'
аргумент1
аргумент
2
аргумент 3
```

Рис. 4.9: ./8.2

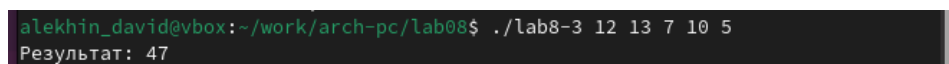
Создаю lab8-3.asm и ввожу туда текст команды 8.3. программа вычисления суммы аргументов командной строки. (рис. 4.10).



```
mc [alekhin_david@vbox]:~/work/arch-pc/lab08
/home/alekhin_david/work/arch-pc/lab08/lab8-3.asm 1081/1428 75%
%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`
1Помощь 2Раз~рн 3Выход 4Нех 5Пер~ти 6 7Поиск 8Исх~ый 9Формат10Выход
```

Рис. 4.10: 8.3

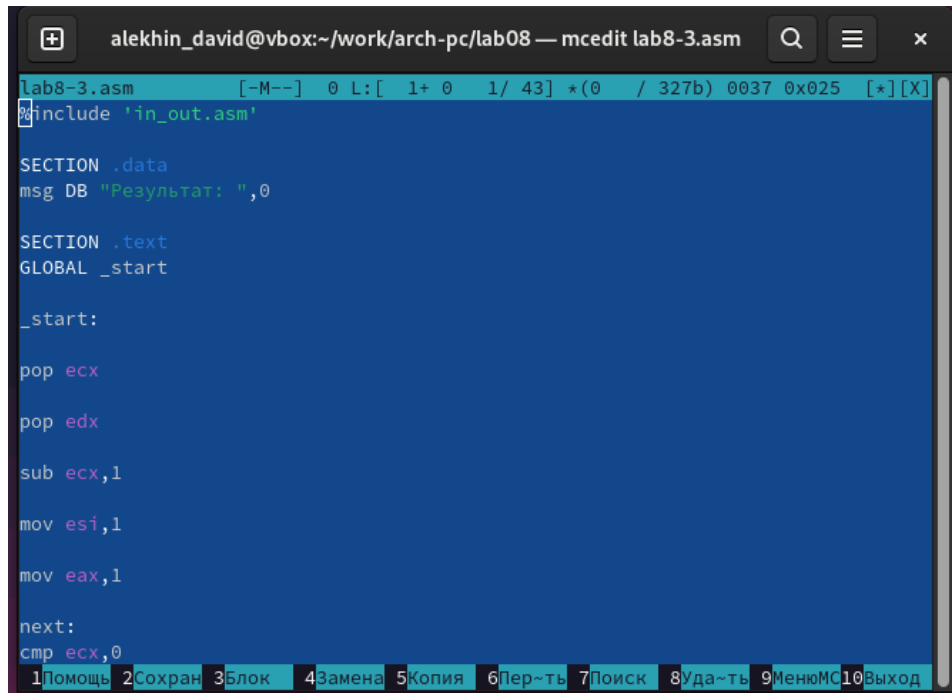
Запускаю 8.3 и получаю сумму. (рис. 4.11).



```
alekhin_david@vbox:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
```

Рис. 4.11: ./8.3

Меняю текст 8.3 так, чтобы получить произведение аргументов. (рис. 4.12) (рис. 4.13).



```
lab8-3.asm      [-M--]  0 L:[ 1+ 0 1/ 43] *(0 / 327b) 0037 0x025 [*][X]
#include 'in_out.asm'

SECTION .data
msg DB "Результат: ",0

SECTION .text
GLOBAL _start

_start:

pop ecx

pop edx

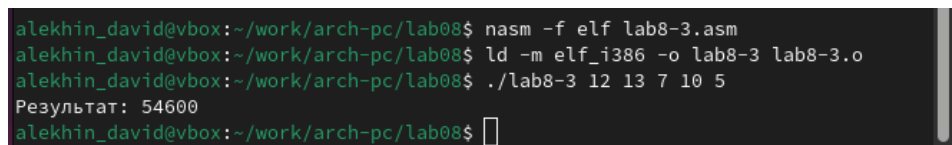
sub ecx,1

mov esi,1

mov eax,1

next:
cmp ecx,0
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Уда~ть 9МенюМС10Выход
```

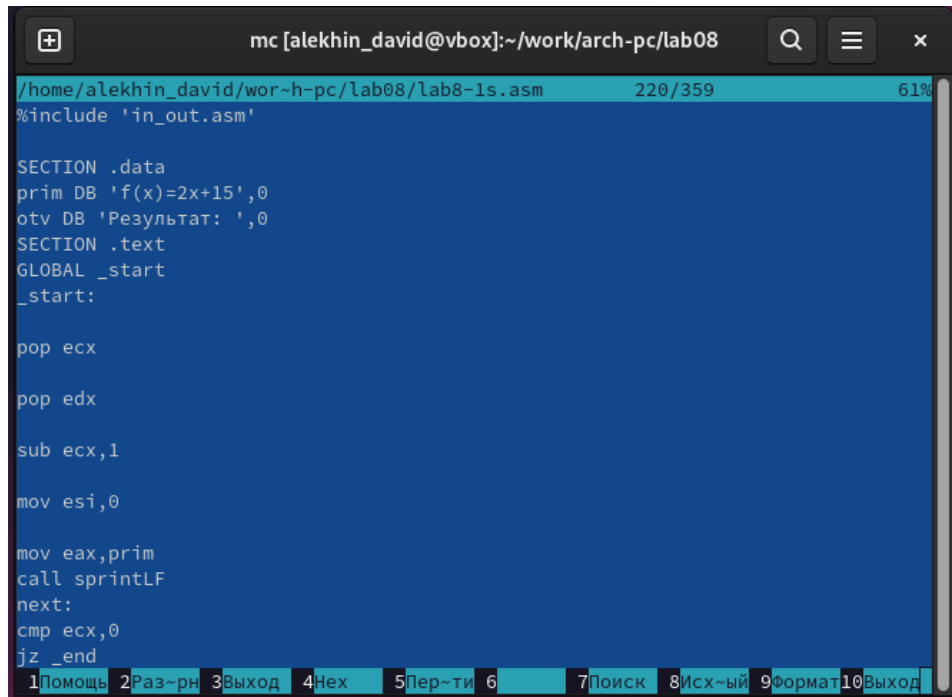
Рис. 4.12: 8.3v2



```
alekhin_david@vbox:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
alekhin_david@vbox:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
alekhin_david@vbox:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 54600
alekhin_david@vbox:~/work/arch-pc/lab08$
```

Рис. 4.13: ./8.3v2

Пишу команду для самостоятельной работы (1 вариант). (рис. 4.14).



```
mc [alekhin_david@vbox]:~/work/arch-pc/lab08
/home/alekhin_david/wor~h-pc/lab08/lab8-1s.asm 220/359 61%
#include 'in_out.asm'

SECTION .data
prim DB 'f(x)=2x+15',0
otv DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:

pop ecx

pop edx

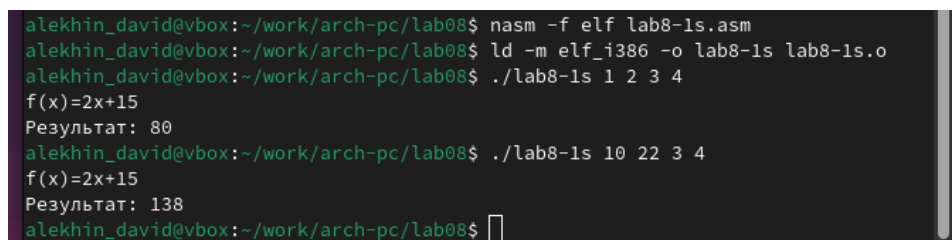
sub ecx,1

mov esi,0

mov eax,prim
call sprintLF
next:
cmp ecx,0
jz _end
```

Рис. 4.14: lab8-1s.asm

Компеллирую и запускаю проверяя различные значения аргумента. (рис. 4.15).



```
alekhin_david@vbox:~/work/arch-pc/lab08$ nasm -f elf lab8-1s.asm
alekhin_david@vbox:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1s lab8-1s.o
alekhin_david@vbox:~/work/arch-pc/lab08$ ./lab8-1s 1 2 3 4
f(x)=2x+15
Результат: 80
alekhin_david@vbox:~/work/arch-pc/lab08$ ./lab8-1s 10 22 3 4
f(x)=2x+15
Результат: 138
alekhin_david@vbox:~/work/arch-pc/lab08$
```

Рис. 4.15: ./lab8-1s.asm

5 Выводы

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

Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learning-bash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс,
- 11.
12. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
13. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
14. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ- Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
15. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-

- е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.
16. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
 17. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер,
 18. — 1120 с. — (Классика Computer Science).