

Отчёт по лабораторной работе №8

дисциплина: Архитектура компьютера

Маслова Анна Павловна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выполнение заданий для самостоятельной работы	15
4	Выводы	19
	Список литературы	20

Список иллюстраций

2.1	Создание каталога lab08 и файла lab8-1.asm	6
2.2	Текст файла lab8-1.asm	7
2.3	Создание исполняемого файла lab8-1	7
2.4	Исправление текста программы в файле lab8-1.asm	8
2.5	Запуск изменённого исполняемого файла lab8-1	9
2.6	Добавление команд push и pop в текст файла lab8-1.asm	10
2.7	Запуск файла lab8-1 с командами push и pop	10
2.8	Текст программы в файле lab8-2.asm	11
2.9	Создание и запуск исполняемого файла lab8-2	12
2.10	Текст программы в файле lab8-3.asm	13
2.11	Создание и запуск исполняемого файла lab8-3	13
2.12	Изменённый текст программы в файле lab8-3.asm	14
2.13	Создание и запуск изменённого исполняемого файла lab8-3	14
3.1	Текст программы в файле func.asm	17
3.2	Создание и запуск исполняемого файла func.asm	17

Список таблиц

1 Цель работы

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

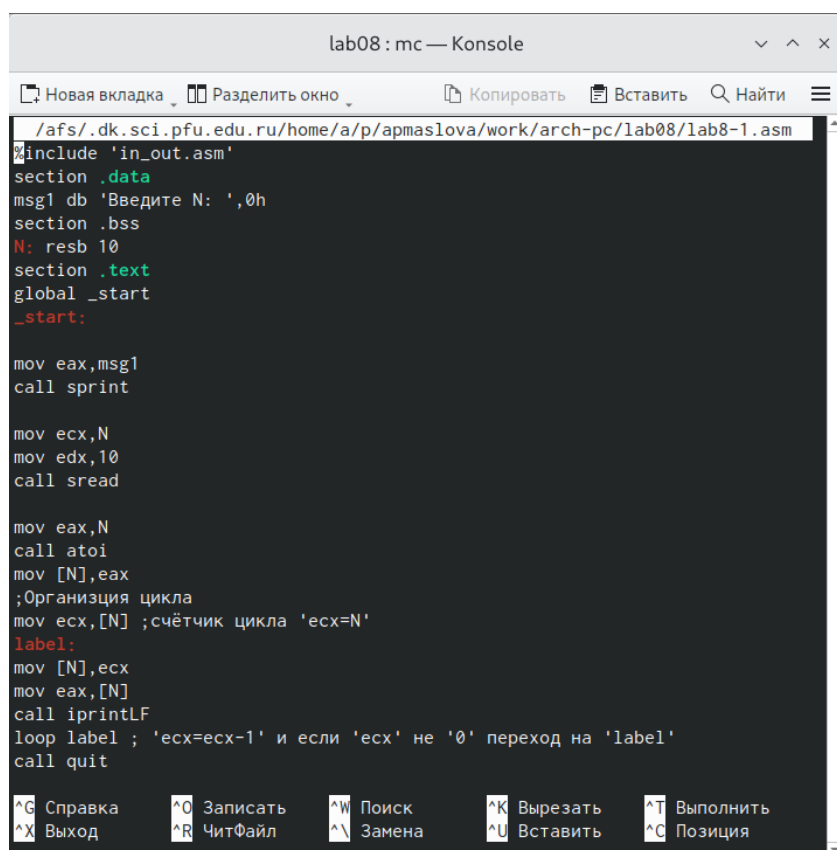
2 Выполнение лабораторной работы

Для начала создаём каталог для программ лабораторной работы №8, перейдём в него и создадим файл *lab8-1.asm* (рис. 2.1).

```
apmaslova@dk3n31 ~ $ mkdir work/arch-pc/lab08
apmaslova@dk3n31 ~ $ cd work/arch-pc/lab08
apmaslova@dk3n31 ~/work/arch-pc/lab08 $ touch lab8-1.asm
apmaslova@dk3n31 ~/work/arch-pc/lab08 $ ls
lab8-1.asm
apmaslova@dk3n31 ~/work/arch-pc/lab08 $
```

Рис. 2.1: Создание каталога lab08 и файла lab8-1.asm

Введём в файл *lab8-1.asm* текст программы вывода значений регистра *ecx* (рис. 2.2).



```
lab08: mc — Konsole
/afs/.dk.sci.pfu.edu.ru/home/a/p/apmaslova/work/arch-pc/lab08/lab8-1.asm
%include 'in_out.asm'
section .data
msg1 db 'Введите N: ',0h
section .bss
N: resb 10
section .text
global _start
_start:

mov eax,msg1
call sprint

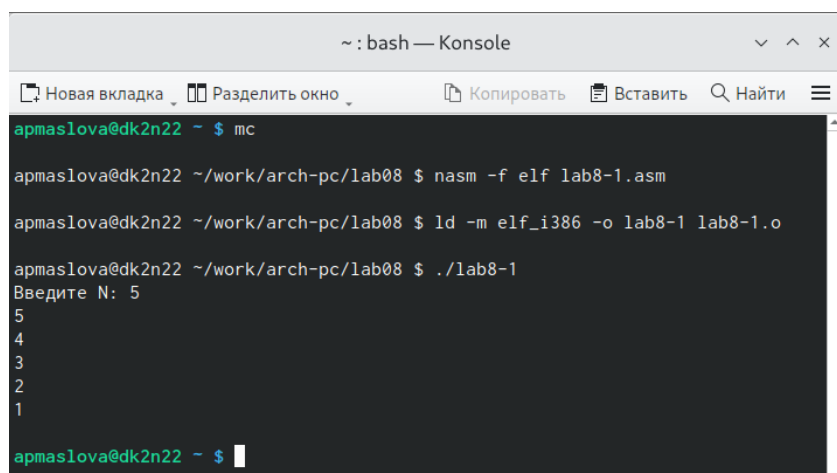
mov ecx,N
mov edx,10
call sread

mov eax,N
call atoi
mov [N],eax
;Организция цикла
mov ecx,[N] ;счётчик цикла 'ecx=N'
label:
mov [N],ecx
mov eax,[N]
call iprintLF
loop label ; 'ecx=ecx-1' и если 'ecx' не '0' переход на 'label'
call quit

^G Справка      ^O Записать     ^W Поиск        ^K Вырезать     ^T Выполнить
^X Выход        ^R ЧитФайл     ^\ Замена       ^U Вставить     ^C Позиция
```

Рис. 2.2: Текст файла lab8-1.asm

Создадим исполняемый файл и проверим его работу (рис. 2.3).

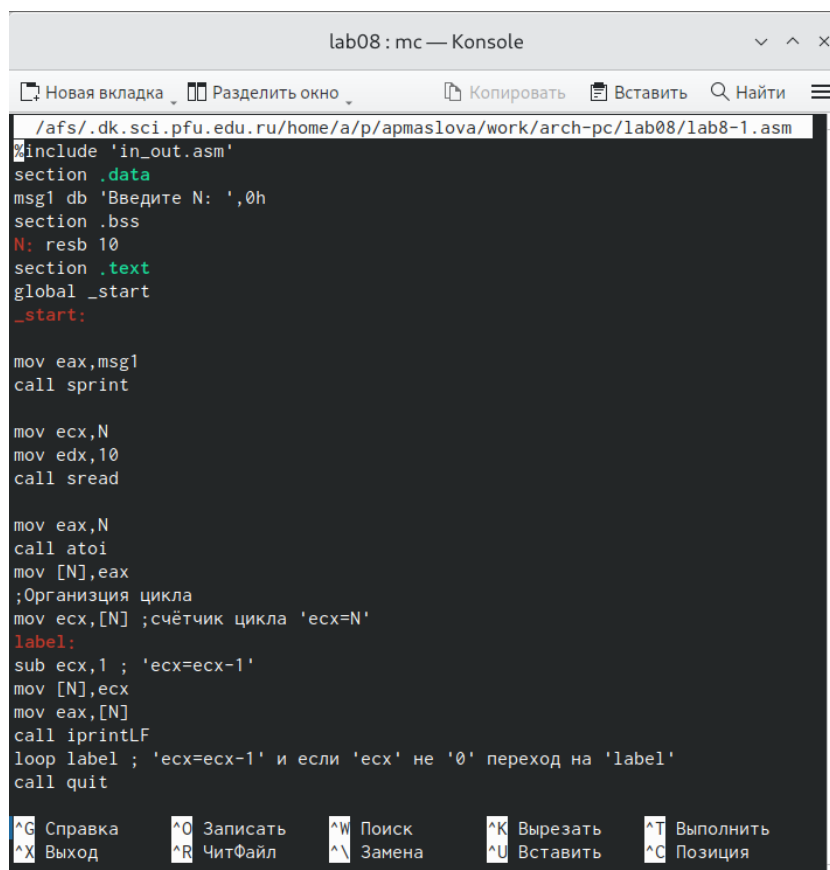


```
~ : bash — Konsole
apmaslova@dk2n22 ~ $ mc
apmaslova@dk2n22 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
apmaslova@dk2n22 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
apmaslova@dk2n22 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 5
5
4
3
2
1
apmaslova@dk2n22 ~ $
```

Рис. 2.3: Создание исполняемого файла lab8-1

На экран вывелись значения от N до 1.

Данный пример показывает, что использование регистра *ecx* в теле цикла *loop* может привести к некорректной работе программы. Подкорректируем текст программы добавив изменение значения регистра *ecx* в цикле (рис. 2.4).



```
lab08 : mc — Konsole
/afs/.dk.sci.pfu.edu.ru/home/a/p/apmaslova/work/arch-pc/lab08/lab8-1.asm
%include 'in_out.asm'
section .data
msg1 db 'Введите N: ',0h
section .bss
N: resb 10
section .text
global _start
_start:

mov eax,msg1
call sprint

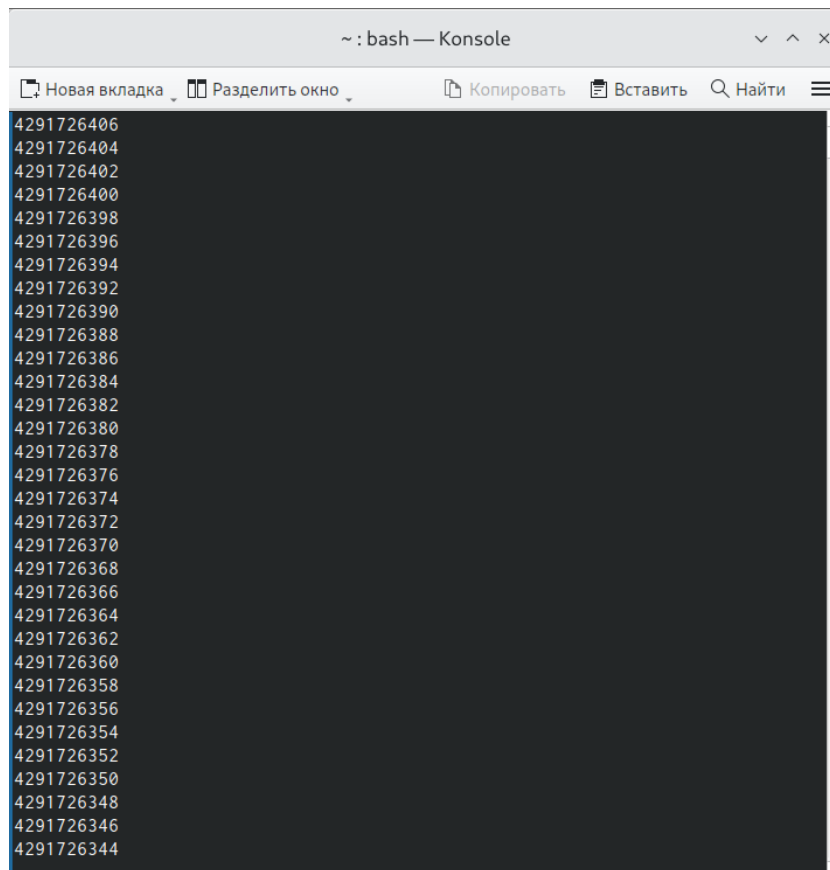
mov ecx,N
mov edx,10
call sread

mov eax,N
call atoi
mov [N],eax
;Организция цикла
mov ecx,[N] ;счётчик цикла 'ecx=N'
label:
sub ecx,1 ; 'ecx=ecx-1'
mov [N],ecx
mov eax,[N]
call iprintLF
loop label ; 'ecx=ecx-1' и если 'ecx' не '0' переход на 'label'
call quit

^G Справка      ^O Записать     ^W Поиск        ^K Вырезать     ^T Выполнить
^X Выход        ^R ЧитФайл     ^\ Замена       ^U Вставить     ^C Позиция
```

Рис. 2.4: Исправление текста программы в файле lab8-1.asm

Создадим исполняемый файл и проверим его работу (рис. 2.5).

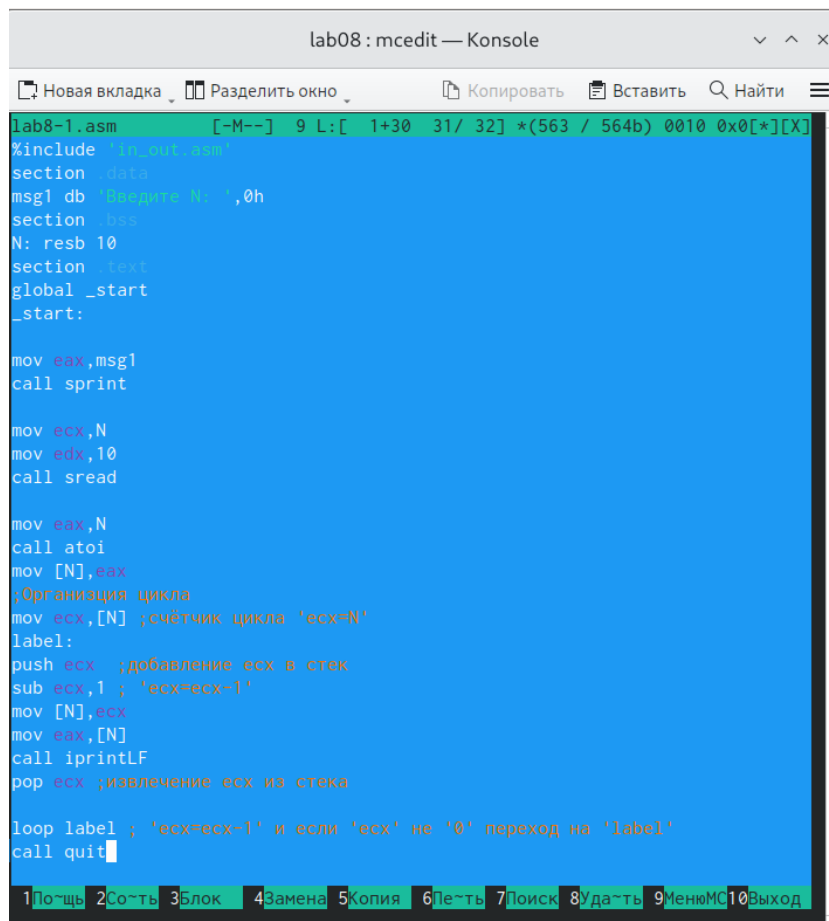


```
~ : bash — Konsole
Новая вкладка Разделить окно Копировать Вставить Найти
4291726406
4291726404
4291726402
4291726400
4291726398
4291726396
4291726394
4291726392
4291726390
4291726388
4291726386
4291726384
4291726382
4291726380
4291726378
4291726376
4291726374
4291726372
4291726370
4291726368
4291726366
4291726364
4291726362
4291726360
4291726358
4291726356
4291726354
4291726352
4291726350
4291726348
4291726346
4291726344
```

Рис. 2.5: Запуск изменённого исполняемого файла lab8-1

Как мы видим, в этом случае программа работает долго, число проходов гораздо больше заявленного N . Регистр *ecx* принимает большие чётные значения, меньшие, но близкие к 5000000000.

Для использования регистра *ecx* в цикле и сохранения корректности работы программы можно использовать стек. Внесём изменения в текст программы добавив команды *push* и *pop* (добавления в стек *push* и извлечения из стека *pop*) для сохранения значения счетчика цикла *loop* (рис. 2.6).



```
lab8-1.asm [-M--] 9 L: [ 1+30 31/ 32] *(563 / 564b) 0010 0x0[*][X]
#include "in_out.asm"
section .data
msg1 db "Введите N: ",0h
section .bss
N: resb 10
section .text
global _start
_start:

mov eax,msg1
call sprint

mov ecx,N
mov edx,10
call sread

mov eax,N
call atoi
mov [N],eax
;Организация цикла
mov ecx,[N] ;счётчик цикла 'ecx=N'
label:
push ecx ;добавление ecx в стек
sub ecx,1 ; 'ecx=ecx-1'
mov [N],ecx
mov eax,[N]
call iprintLF
pop ecx ;извлечение ecx из стека

loop label ; 'ecx=ecx-1' и если 'ecx' не '0' переход на 'label'
call quit
```

Рис. 2.6: Добавление команд push и pop в текст файла lab8-1.asm

Создадим исполняемый файл и проверим его работу (рис. 2.7).



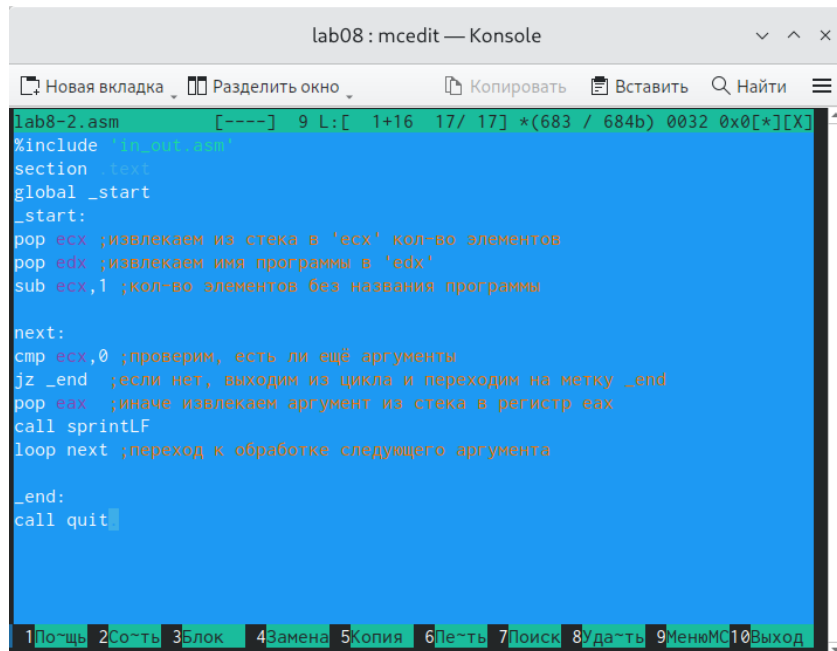
```
apmaslova@dk1n22 ~/work/arch-pc/lab08 $ mcedit lab8-1.asm
apmaslova@dk1n22 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
apmaslova@dk1n22 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
apmaslova@dk1n22 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 5
4
3
2
1
0
apmaslova@dk1n22 ~/work/arch-pc/lab08 $
```

Рис. 2.7: Запуск файла lab8-1 с командами push и pop

В данном случае число проходов цикла соответствует введённому числу N . На

экран вывелись числа от $N-1$ до 0.

Далее познакомимся с другой программой. Создадим файл *lab8-2.asm* в каталоге `~/work/arch-рс/lab08` и введём в него текст программы, которая выводит на экран аргументы командной строки (рис. 2.8).



```
lab8-2.asm  [----]  9  L:[ 1+16 17/ 17] *(683 / 684b) 0032 0x0[*][X]
%include "mcs51.asm"
section .text
global _start
_start:
pop ecx ;извлекаем из стека в 'ecx' кол-во элементов
pop edx ;извлекаем имя программы в 'edx'
sub ecx,1 ;кол-во элементов без названия программы

next:
cmp ecx,0 ;проверим, есть ли ещё аргументы
jz _end ;если нет, выходим из цикла и переходим на метку _end
pop eax ;иначе извлекаем аргумент из стека в регистр eax
call printf
loop next ;переход к обработке следующего аргумента

_end:
call quit
```

Рис. 2.8: Текст программы в файле *lab8-2.asm*

Создаём исполняемый файл и запустим его (рис. 2.9), указав следующие аргументы:

```
армаслова@dk1n22:~$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
```

```

армаслова@dk1n22 ~/work/arch-pc/lab08 $ touch lab8-2.asm
армаслова@dk1n22 ~/work/arch-pc/lab08 $ mcedit lab8-2.asm

армаслова@dk1n22 ~/work/arch-pc/lab08 $ mcedit lab8-2.asm

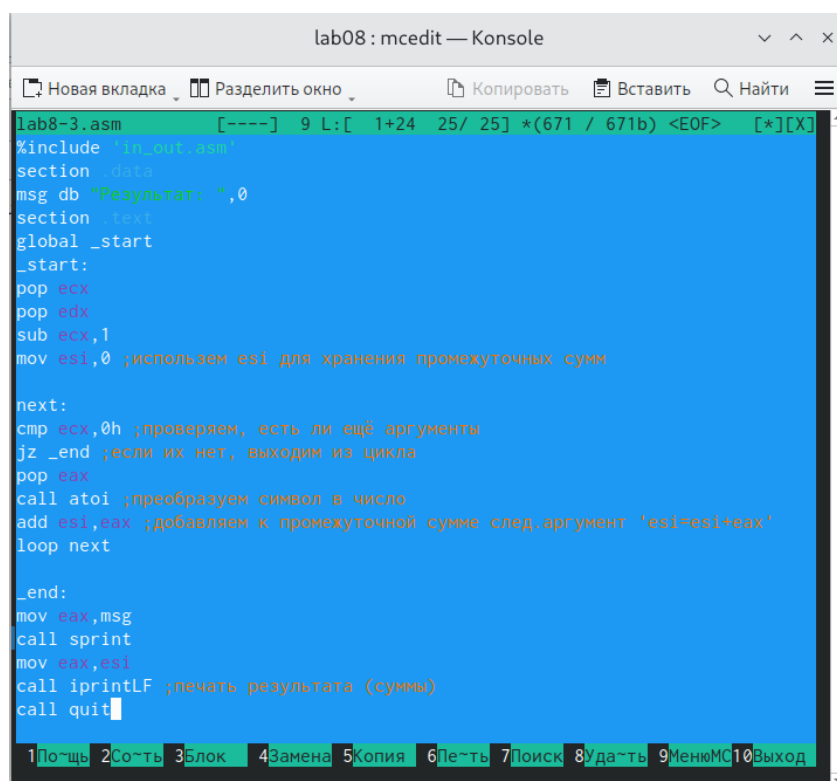
армаслова@dk1n22 ~/work/arch-pc/lab08 $ nasm -f elf lab8-2.asm
армаслова@dk1n22 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
армаслова@dk1n22 ~/work/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент2 'аргуме
нт3'
аргумент1
аргумент2
аргумент3
армаслова@dk1n22 ~/work/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент 2 'аргум
ент3'
аргумент1
аргумент
2
аргумент3
армаслова@dk1n22 ~/work/arch-pc/lab08 $

```

Рис. 2.9: Создание и запуск исполняемого файла lab8-2

Программа обработала 4 аргумента, т.к. аргумент 2 был прочитан как аргумент и 2, т.е. разделён на два разных аргумента, которые поочередно были выведены на экран.

Рассмотрим следующую программу. Создадим файл *lab8-3.asm* в каталоге `~/work/arch-pc/lab08` и введём в него текст программы, которая выводит сумму чисел, которые передаются в программу как аргументы (рис. 2.10).



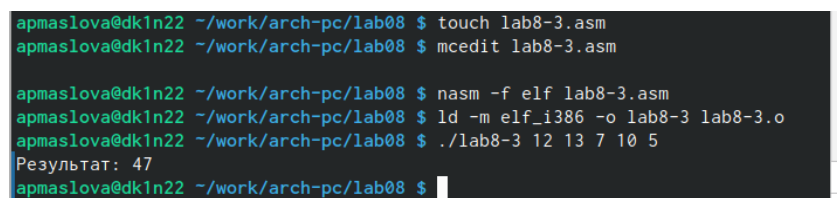
```
lab8-3.asm [----] 9 L:[ 1+24 25/ 25] *(671 / 671b) <EOF> [*][X]
#include "mcout.asm"
section .data
msg db "Результат: ",0
section .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi,0 ;используем esi для хранения промежуточных сумм

next:
cmp ecx,0h ;проверяем, есть ли ещё аргументы
jz _end ;если их нет, выходим из цикла
pop eax
call atoi ;преобразуем символ в число
add esi,eax ;добавляем к промежуточной сумме след. аргумент 'esi=esi+eax'
loop next

_end:
mov eax,msg
call sprint
mov eax,esi
call iprintLF ;печать результата (суммы)
call quit
```

Рис. 2.10: Текст программы в файле lab8-3.asm

Создадим исполняемый файл и запустим его, указав аргументы (рис. 2.11).



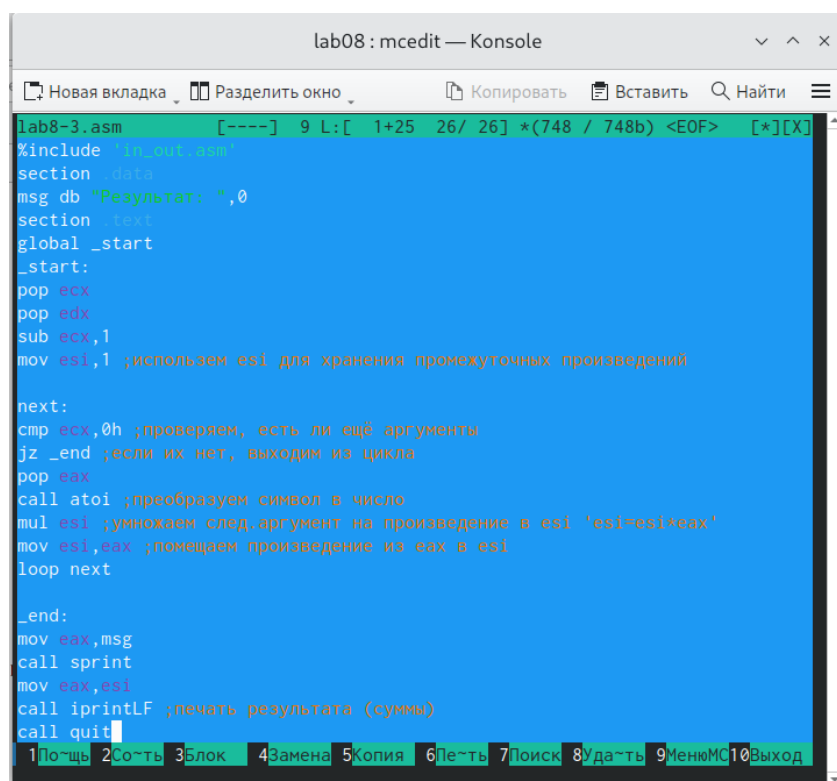
```
apmaslova@dk1n22 ~/work/arch-pc/lab08 $ touch lab8-3.asm
apmaslova@dk1n22 ~/work/arch-pc/lab08 $ mcedit lab8-3.asm

apmaslova@dk1n22 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
apmaslova@dk1n22 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
apmaslova@dk1n22 ~/work/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 47
apmaslova@dk1n22 ~/work/arch-pc/lab08 $
```

Рис. 2.11: Создание и запуск исполняемого файла lab8-3

Программа работает корректно: выводит на экран сумму аргументов командной строки.

А теперь изменим текст этой программы для вычисления произведения аргументов командной строки. Введём в файл *lab8-3.asm* текст из рис. 2.12 :



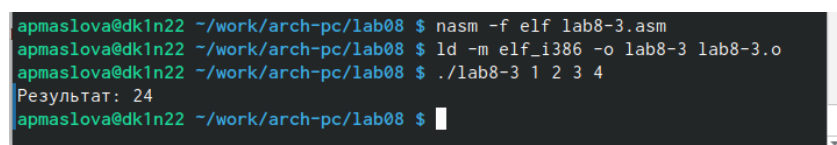
```
lab8-3.asm [----] 9 L:[ 1+25 26/ 26] *(748 / 748b) <EOF> [*][X]
#include "lab08.asm"
section .data
msg db "Результат: ",0
section .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi,1 ;используем esi для хранения промежуточных произведений

next:
cmp ecx,0h ;проверяем, есть ли ещё аргументы
jz _end ;если их нет, выходим из цикла
pop eax
call atoi ;преобразуем символ в число
mul esi ;умножаем след. аргумент на произведение в esi 'esi=esi*eax'
mov esi,eax ;помещаем произведение из eax в esi
loop next

_end:
mov eax,msg
call sprint
mov eax,esi
call iprintlnLF ;печать результата (суммы)
call quit
```

Рис. 2.12: Изменённый текст программы в файле lab8-3.asm

Создадим исполняемый файл и запустим его, указав аргументы (рис. 2.13).



```
apmaslova@dk1n22 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
apmaslova@dk1n22 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
apmaslova@dk1n22 ~/work/arch-pc/lab08 $ ./lab8-3 1 2 3 4
Результат: 24
apmaslova@dk1n22 ~/work/arch-pc/lab08 $
```

Рис. 2.13: Создание и запуск изменённого исполняемого файла lab8-3

Как мы видим, программа верно вычисляет произведение аргументов командной строки.

3 Выполнение заданий для самостоятельной работы

Напишем программу, которая находит сумму значений функции $f(x)$ для $x = x_1, x_2, \dots, x_n$. То есть программа должна выводить значения $f(x_1) + f(x_2) + \dots + f(x_n)$.

Варианту №15 соответствует следующая функция:

$$f(x) = 6x + 13$$

В том же каталоге создадим файл *func.asm* и внесём в него текст программы из листинга 8.4 (рис. 3.1).

Листинг 8.4. Программа, вычисления суммы значений функции

```
%include 'in_out.asm'

section .data
msg db "Результат: ",0

section .text
global _start
_start:
```

```

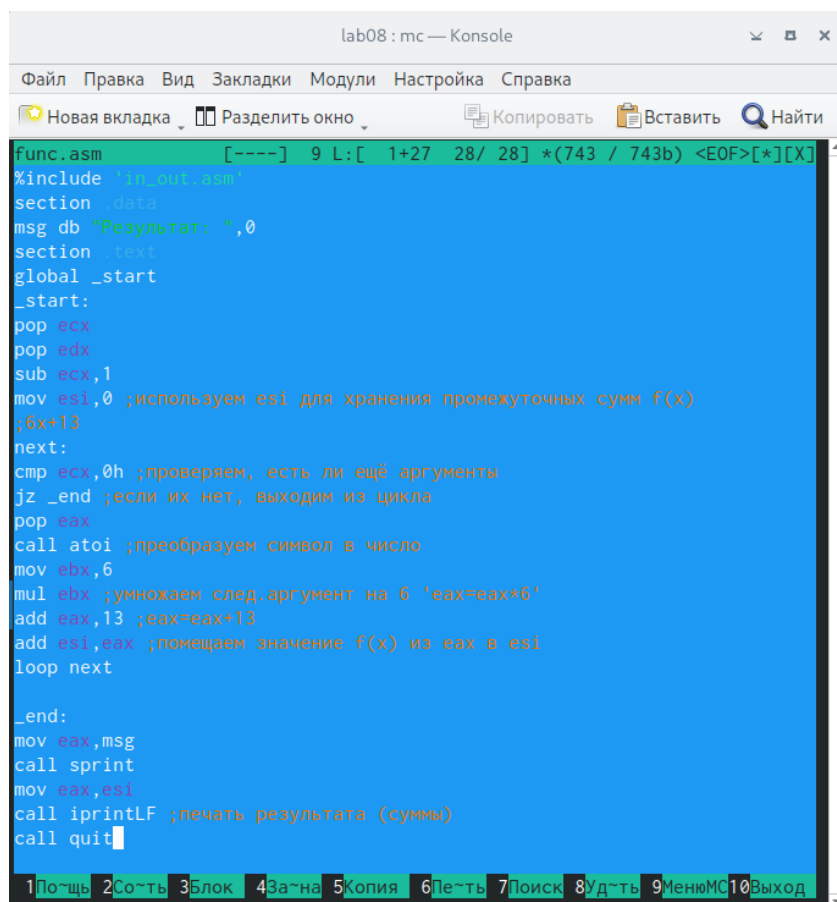
pop ecx
pop edx
sub ecx,1
mov esi,0 ;используем esi для хранения промежуточных сумм f(x)

;6x+13
next:
cmp ecx,0h ;проверяем, есть ли ещё аргументы
jz _end ;если их нет, выходим из цикла
pop eax
call atoi ;преобразуем символ в число
mov ebx,6
mul ebx ;умножаем след.аргумент на 6 'eax=eax*6'
add eax,13 ;eax=eax+13
add esi,eax ;помещаем значение f(x) из eax в esi
loop next

_end:
mov eax,msg
call sprint
mov eax,esi
call iprintLF ;печать результата (суммы)

call quit

```

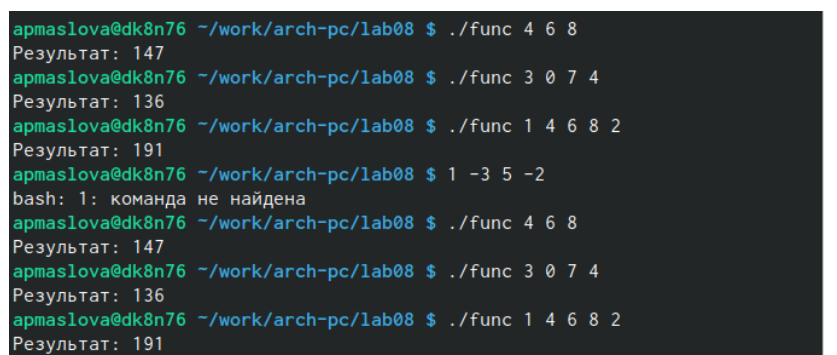



```
func.asm [----] 9 L: [ 1+27 28/ 28] *(743 / 743b) <EOF>[*][X]
#include "ln_out.asm"
section .data
msg db "Результат: ",0
section .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi,0 ;используем esi для хранения промежуточных сумм f(x)
;6x+13
next:
cmp ecx,0h ;проверяем, есть ли ещё аргументы
jz _end ;если их нет, выходим из цикла
pop eax
call atoi ;преобразуем символ в число
mov ebx,6
mul ebx ;умножаем след. аргумент на 6 'eax=eax*6'
add eax,13 ;eax=eax+13
add esi,eax ;помещаем значение f(x) из eax в esi
loop next

_end:
mov eax,msg
call sprint
mov eax,esi
call iprintLF ;печать результата (суммы)
call quit
```

Рис. 3.1: Текст программы в файле func.asm

Создадим исполняемый файл и несколько раз запустим его, указав аргументы (рис. 3.2).



```
армаслова@dk8n76 ~/work/arch-pc/lab08 $ ./func 4 6 8
Результат: 147
армаслова@dk8n76 ~/work/arch-pc/lab08 $ ./func 3 0 7 4
Результат: 136
армаслова@dk8n76 ~/work/arch-pc/lab08 $ ./func 1 4 6 8 2
Результат: 191
армаслова@dk8n76 ~/work/arch-pc/lab08 $ 1 -3 5 -2
bash: 1: команда не найдена
армаслова@dk8n76 ~/work/arch-pc/lab08 $ ./func 4 6 8
Результат: 147
армаслова@dk8n76 ~/work/arch-pc/lab08 $ ./func 3 0 7 4
Результат: 136
армаслова@dk8n76 ~/work/arch-pc/lab08 $ ./func 1 4 6 8 2
Результат: 191
```

Рис. 3.2: Создание и запуск исполняемого файла func.asm

Как мы видим, программа работает корректно и верно считает сумму значений функции.

4 Выводы

Мы научились писать программы 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.