

РУДН. Архитектура компьютеров

Отчёт по лабораторной работе №8

Косинов Никита Андреевич, НПИМбв-02-20

Содержание

1	Цель работы	5
2	Ход работы	6
3	Реализация цикла фиксированной длины	7
4	Реализация цикла, зависящего от аргументов	15
5	Работа с аргументами	17
6	Самостоятельная работа	22
7	Выводы	26

Список иллюстраций

3.1	Создание файла	7
3.2	Код программы lab8-1	8
3.3	Исполнение lab8-1	9
3.4	Код программы lab8-1-1	10
3.5	Исполнение lab8-1-1	12
3.6	Код программы lab8-1-2	12
3.7	Исполнение lab8-1-2	14
4.1	Код программы lab8-2	15
4.2	Исполнение lab8-2	16
5.1	Код программы lab8-3	17
5.2	Исполнение lab8-3	19
5.3	Код программы lab8-3-1	19
5.4	Исполнение lab8-3-1	21
6.1	Код программы sr	22
6.2	Код программы sr	23
6.3	Исполнение sr	25

Список таблиц

1 Цель работы

Многие логики программ требуют повторять одну и ту же операцию несколько раз: фиксированное, либо зависящее от параметров, заданных пользователем. В любом случае, в Ассемблере предусмотрена такая возможность.

Цель данной работы - научиться писать программы, использующие логику циклов, а также познакомиться со структурой стека.

2 Ход работы

Лабораторная работа выполнена с использованием консоли **ОС Linux** и языка программирования ассемблера **NASM**.

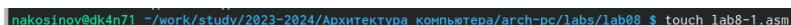
1. Реализация цикла фиксированной длины;
2. Реализация цикла, зависящего от аргументов;
3. Работа с аргументами.

В конце выполнена самостоятельная работа.

3 Реализация цикла фиксированной длины

Чтобы организовать цикл в **NASM**, можно использовать инструкцию **loop**. Она уменьшает значение, хранящееся в регистре **ecx** на 1, и, если получает не ноль, переходит на указанную метку. Таким образом, метка указанная над командой **loop** возвращает нас на несколько шагов назад, но с новым значением **ecx**. Так мы получаем цикл на **ecx** операций.

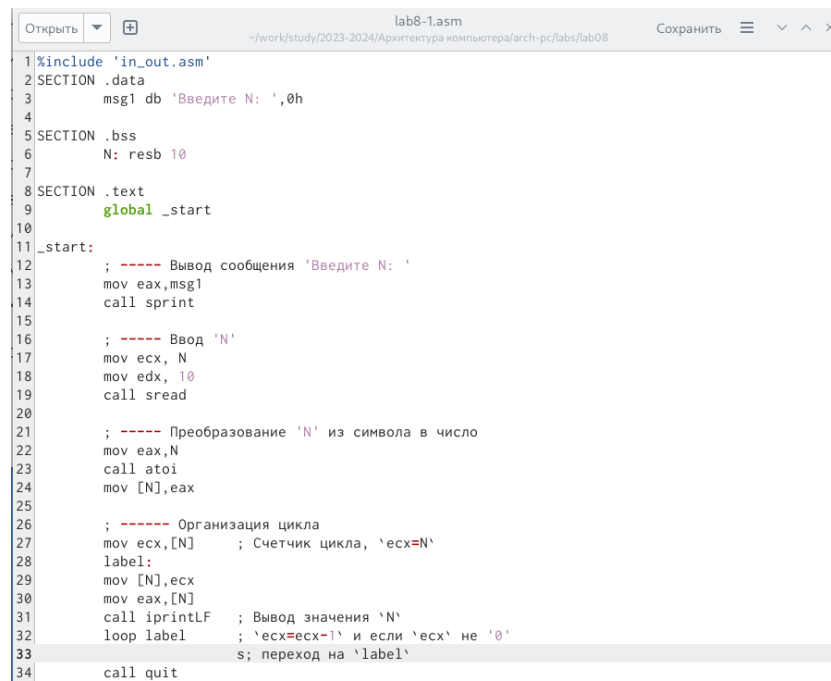
1. Создаём рабочий файл.



```
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ touch lab8-1.asm
```

Рис. 3.1: Создание файла

2. Пишем код с использованием цикла длины **N**, где последний вводится пользователем.



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

Рис. 3.2: Код программы lab8-1

```
%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
```



```

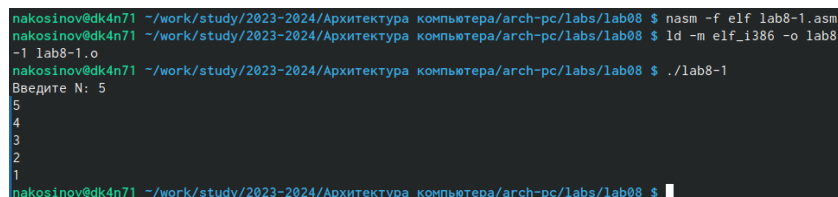
; ----- Ввод 'N'
mov ecx, N
mov edx, 10
call sread

; ----- Преобразование 'N' из символа в число
mov eax, N
call atoi
mov [N], eax

; ----- Организация цикла
mov ecx, [N]      ; Счетчик цикла, `ecx=N`
label:
    mov [N], ecx
    mov eax, [N]
    call iprintLF  ; Вывод значения `N`
    loop label    ; `ecx=ecx-1` и если `ecx` не '0'
                  ; переход на `label`
call quit

```

3. Компилируем и запускаем. Изначально в **ecx** записывает число 5, далее каждую итерацию мы выводим его на экран и уменьшаем на 1. По достижении 0 программа завершает работу



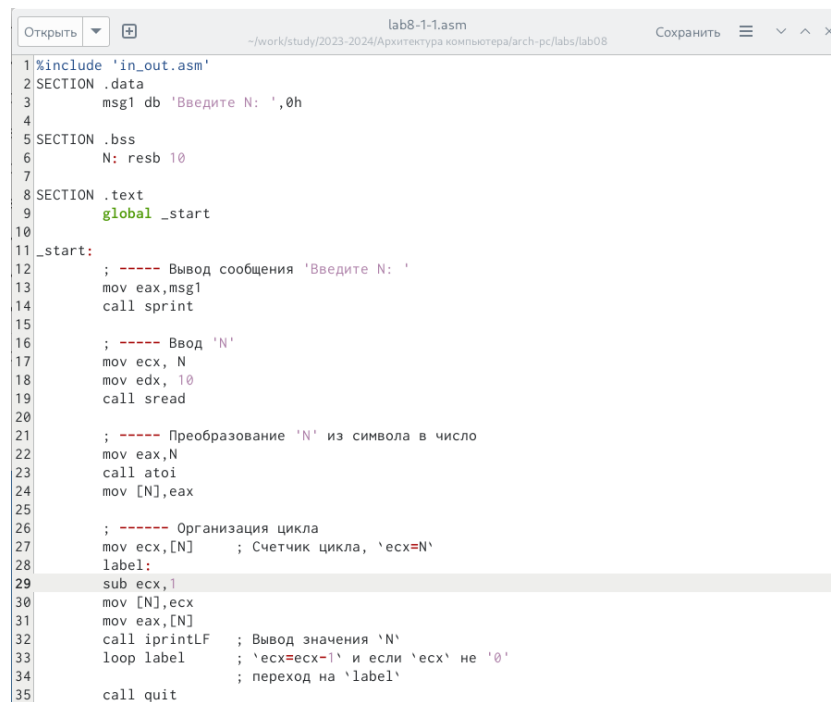
```

nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ nasm -f elf lab8-1.asm
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ld -m elf_i386 -o lab8
-lab8-1.o
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ./lab8-1
Введите N: 5
5
4
3
2
1
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $

```

Рис. 3.3: Исполнение lab8-1

4. Изменим работу программы, добавив уменьшение **ecx** на 1 вручную.



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

Рис. 3.4: Код программы lab8-1-1

```
%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:
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF    ; Вывод значения `N`
loop label      ; `ecx=ecx-1` и если `ecx` не '0'
                ; переход на `label`
call quit

```

5. Запустим, попробовав **N=7** и **N=6**. Во втором случае видно, что выводятся числа с шагом 2. Действительно, **ecx** уменьшается дважды на 1 за одну итерацию: нами и инструкцией **loop**. В первом случае же, раз **N** нечётно, после 1 мы получим -1, т.е. не 0, и программы продолжит работу. При этом, -1 в **ecx** - это самое большое число, поэтому мы видим такой результат.

```

4294673950
4294673948
4294673946
4294673944
4294673942
4294673940
4294673938
4294673936
4294673934
4294673932
4294^C
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ nasm -f elf lab8-1-1.a
sm
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ld -m elf_i386 -o lab8
-1-1 lab8-1-1.o
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ./lab8-1-1
Введите N: 6
5
3
1
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $

```

Рис. 3.5: Исполнение lab8-1-1

- Для исправления такого рода недочёта, можно перед изменением значения **ecx** его сохранить, например, в стек. А перед концом шага цикла достать обратно.

```

lab8-1-2.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08
Сохранить

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

```

Рис. 3.6: Код программы lab8-1-2

```
%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:

push ecx

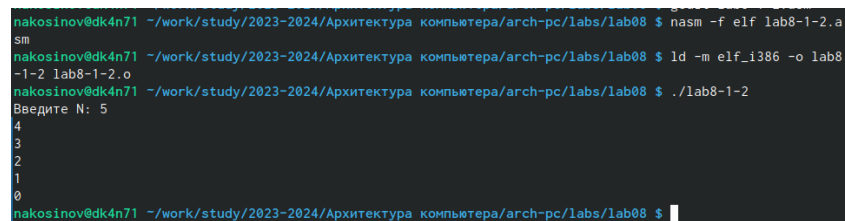
sub ecx,1

```

mov [N],ecx
mov eax,[N]
call iprintLF    ; Вывод значения `N`
pop ecx
loop label    ; `ecx=ecx-1` и если `ecx` не `0`
               ; переход на `label`
call quit

```

7. Компилируем и запустим. Получим значения **ecx**, уменьшенные на 1.



```

nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ nasm -f elf lab8-1-2.a
sm
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ld -m elf_i386 -o lab8
-1-2 lab8-1-2.o
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ./lab8-1-2
Введите N: 5
4
3
2
1
0
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $

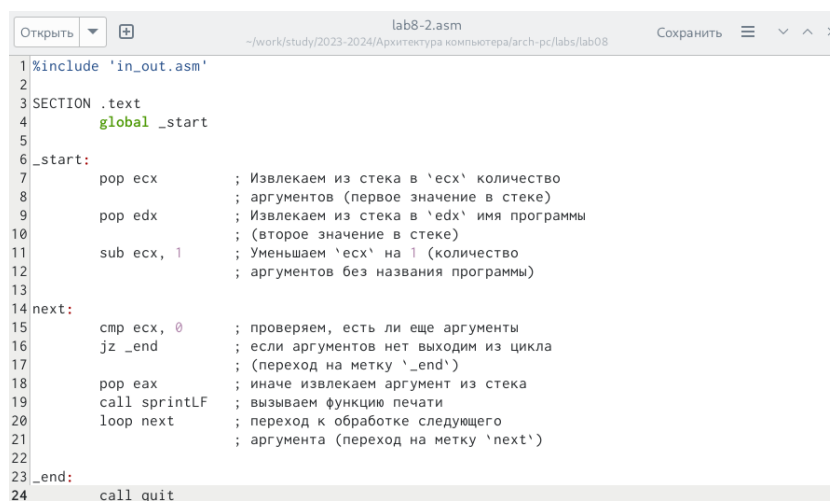
```

Рис. 3.7: Исполнение lab8-1-2

4 Реализация цикла, зависящего от аргументов

При запуске программы есть возможность указать какие-либо аргументы, которые программа может использовать. Попробуем написать код, который работает с введенными с клавиатуры аргументами.

1. Пишем код программы: вывести указанные аргументы, каждый на новой строке. Количество аргументов при этом можно взять из стека аргументов, пропуская при этом название программы, также записанное в стек.



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

Рис. 4.1: Код программы lab8-2

```
%include 'in_out.asm'
```

```

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)

next:

    cmp ecx, 0    ; проверяем, есть ли еще аргументы
    jz _end       ; если аргументов нет выходим из цикла
                  ; (переход на метку `_end`)
    pop 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

```

2. Компилируем и запускаем.

```

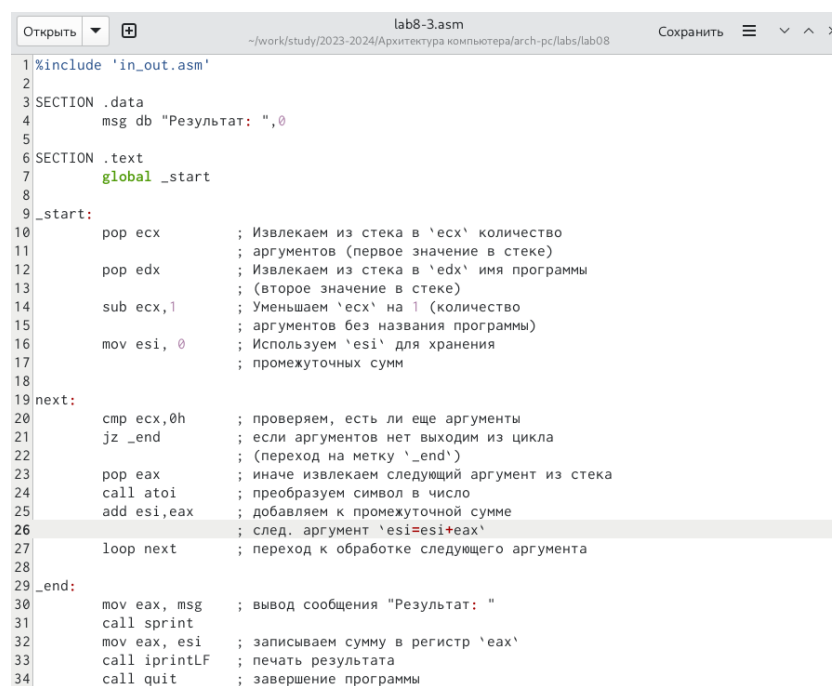
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ nasm -f elf lab8-2.asm
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ./lab8-2
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ./lab8-2 arg1 arg 2 'a
rg3'
arg1
arg
2
arg3
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $

```

Рис. 4.2: Исполнение lab8-2

5 Работа с аргументами

1. Попробуем сделать с аргументами что-то дополнительно, например, посчитать их сумму.



```
1 %include 'in_out.asm'
2
3 SECTION .data
4     msg db "Результат: ",0
5
6 SECTION .text
7     global _start
8
9 _start:
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        ; иначе извлекаем следующий аргумент из стека
24    call atoi      ; преобразуем символ в число
25    add esi,eax    ; добавляем к промежуточной сумме
26    ; след. аргумент 'esi=esi+eax'
27    loop next      ; переход к обработке следующего аргумента
28
29 _end:
30    mov eax, msg   ; вывод сообщения "Результат: "
31    call sprint
32    mov eax, esi    ; записываем сумму в регистр 'eax'
33    call iprintfLF ; печать результата
34    call quit      ; завершение программы
```

Рис. 5.1: Код программы lab8-3

```
%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`
    loop next    ; переход к обработке следующего аргумента
```

```
_end:
```

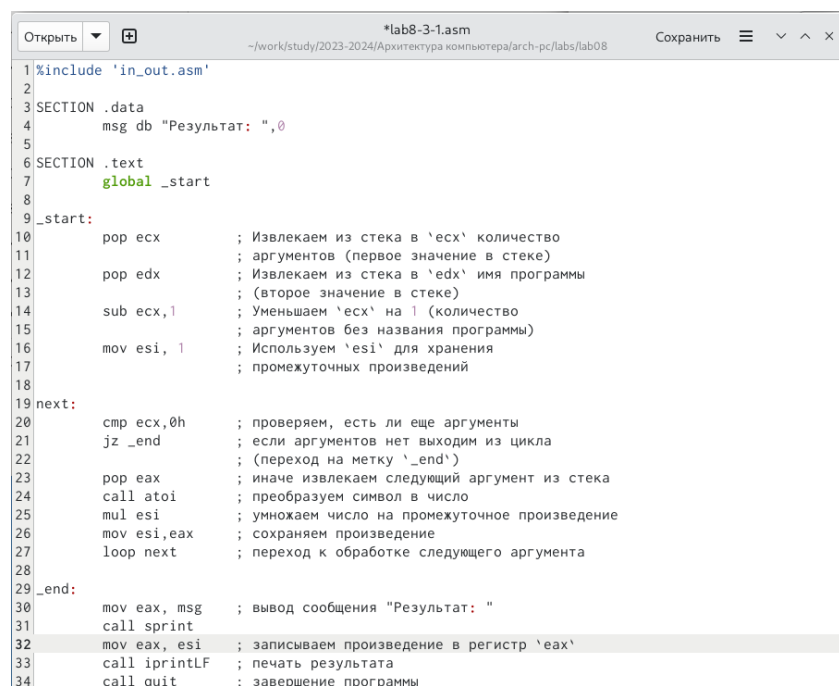
```
    mov eax, msg ; вывод сообщения "Результат: "
    call sprint
    mov eax, esi  ; записываем сумму в регистр `eax`
    call iprintLF ; печать результата
    call quit     ; завершение программы
```

2. Компилируем и запускаем. Программа работает корректно: $1 + 2 + 3 + \dots + 10 = \frac{11 \times 10}{2} = 55$.

```
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ nasm -f elf lab8-3.asm
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ld -m elf_i386 -o lab8
-lab8-3.o
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ./lab8-3 1 2 3 4 5 6 7
8 9 10
Результат: 55
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $
```

Рис. 5.2: Исполнение lab8-3

3. Изменим программу, чтобы она считала произведение чисел. Не забываем “обнулить” регистр `esi` единицей.



```
Открыть [иконка] *lab8-3-1.asm Сохранить [иконка] [иконка] [иконка]
~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08

1 %include 'in_out.asm'
2
3 SECTION .data
4     msg db "Результат: ",0
5
6 SECTION .text
7     global _start
8
9 _start:
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 произведений
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        ; иначе извлекаем следующий аргумент из стека
24     call atoi      ; преобразуем символ в число
25     mul esi        ; умножаем число на промежуточное произведение
26     mov esi,eax    ; сохраняем произведение
27     loop next      ; переход к обработке следующего аргумента
28
29 _end:
30     mov eax, msg    ; вывод сообщения "Результат: "
31     call sprint
32     mov eax, esi    ; записываем произведение в регистр 'eax'
33     call iprintLF  ; печать результата
34     call quit      ; завершение программы
```

Рис. 5.3: Код программы lab8-3-1

```
%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      ; умножаем число на промежуточное произведение
    mov esi,eax  ; сохраняем произведение
    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 программы

4. Запустим и посчитаем, например $6! = 720$.

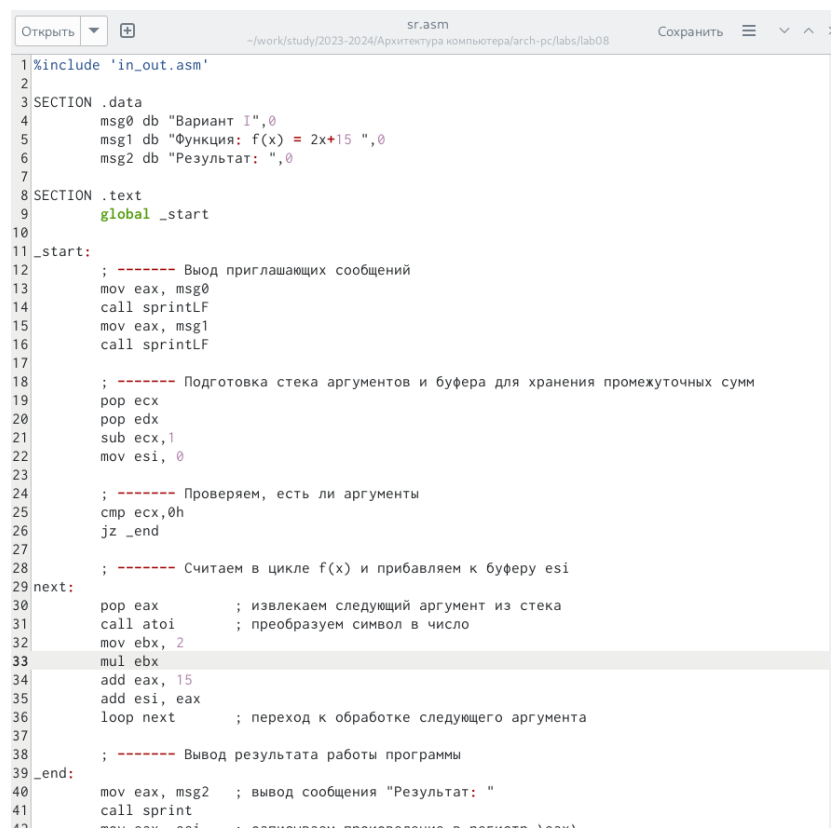
```
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ gedit lab8-3-1.asm
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ nasm -f elf lab8-3-1.asm
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ld -m elf_i386 -o lab8-3-1 lab8-3-1.o
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ./lab8-3-1 1 2 3 4 5 6
Результат: 720
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $
```

Рис. 5.4: Исполнение lab8-3-1

6 Самостоятельная работа

В самостоятельной работе предлагается взять функцию $f(x) = 2x + 15$ и посчитать сумму значений этой функции от введенных аргументов.

1. Напишем код с комментариями.



```
1 %include 'in_out.asm'
2
3 SECTION .data
4     msg0 db "Вариант I",0
5     msg1 db "Функция: f(x) = 2x+15 ",0
6     msg2 db "Результат: ",0
7
8 SECTION .text
9     global _start
10
11 _start:
12     ; ----- Вывод приглашающих сообщений
13     mov eax, msg0
14     call sprintf
15     mov eax, msg1
16     call sprintf
17
18     ; ----- Подготовка стека аргументов и буфера для хранения промежуточных сумм
19     pop ecx
20     pop edx
21     sub ecx,1
22     mov esi, 0
23
24     ; ----- Проверяем, есть ли аргументы
25     cmp ecx,0h
26     jz _end
27
28     ; ----- Считаем в цикле f(x) и прибавляем к буферу esi
29 next:
30     pop eax             ; извлекаем следующий аргумент из стека
31     call atoi           ; преобразуем символ в число
32     mov ebx, 2
33     mul ebx
34     add eax, 15
35     add esi, eax
36     loop next          ; переход к обработке следующего аргумента
37
38     ; ----- Вывод результата работы программы
39 _end:
40     mov eax, msg2      ; вывод сообщения "Результат: "
41     call sprintf
42     mov eax, esi        ; записываем результат в регистр EAX
```

Рис. 6.1: Код программы sr

42	mov eax, esi	; записываем произведение в регистр 'eax'
43	call iprintLF	; печать результата
44	call quit	; завершение программы

Рис. 6.2: Код программы sr

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
msg0 db "Вариант I",0
msg1 db "Функция: f(x) = 2x+15 ",0
msg2 db "Результат: ",0
```

```
SECTION .text
```

```
global _start
```

```
_start:
```

```
; ----- Вывод приглашающих сообщений
```

```
mov eax, msg0
call sprintLF
mov eax, msg1
call sprintLF
```

```
; ----- Подготовка стека аргументов и буфера для хранения промежуточных сум
```

```
pop ecx
pop edx
sub ecx,1
mov esi, 0
```

```
; ----- Проверяем, есть ли аргументы
```

```
cmp ecx,0h
```

```
jz _end
```

```
; ----- Считаем в цикле f(x) и прибавляем к буферу esi
```

```
next:
```

```
pop eax      ; извлекаем следующий аргумент из стека
```

```
call atoi    ; преобразуем символ в число
```

```
mov ebx, 2
```

```
mul ebx
```

```
add eax, 15
```

```
add esi, eax
```

```
loop next    ; переход к обработке следующего аргумента
```

```
; ----- Вывод результата работы программы
```

```
_end:
```

```
mov eax, msg2 ; вывод сообщения "Результат: "
```

```
call sprint
```

```
mov eax, esi   ; записываем произведение в регистр `eax`
```

```
call iprintLF  ; печать результата
```

```
call quit     ; завершение программы
```

2. Скомпилируем и запустим несколько раз, указывая разные наборы аргументов. Убедимся, что во всех случаях программа работает корректно, в том числе, если аргументы не указаны.


```

nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ gedit sr.asm
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ nasm -f elf sr.asm
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ld -m elf_i386 -o sr s
r.o
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ./sr
Вариант I
Функция:  $f(x) = 2x+15$ 
Результат: 0
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ./sr 1 2 3 4
Вариант I
Функция:  $f(x) = 2x+15$ 
Результат: 80
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ./sr 1 1 1 1
Вариант I
Функция:  $f(x) = 2x+15$ 
Результат: 68
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $ ./sr 0 0 0 0 0
Вариант I
Функция:  $f(x) = 2x+15$ 
Результат: 75
nakosinov@dk4n71 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08 $

```

Рис. 6.3: Исполнение sr

7 Выводы

В ходе данной лабораторной работы мы познакомились со структурой стека, научились брать из автоматически созданного стека аргументы и их количество, реализовали работу с аргументами с использованием циклической структуры.