

# **Отчёт по лабораторной работе №8**

**2023**

Просина Ксения Максимовна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
3.1	Реализация циклов в NASM . . . . .	7
3.2	Обработка аргументов командной строки . . . . .	14
3.3	Задание для самостоятельной работы . . . . .	19
<b>4</b>	<b>Выводы</b>	<b>22</b>
	<b>Список литературы</b>	<b>23</b>

## Список иллюстраций

3.1	Создание каталога и файла . . . . .	7
3.2	Создание исполняемого файла и проверка . . . . .	9
3.3	Измененный код . . . . .	10
3.4	Вывод результата после компоновки файла . . . . .	11
3.5	Измененный код . . . . .	13
3.6	Вывод результата после компоновки файла . . . . .	14
3.7	Создание исполняемого файла и проверка . . . . .	16
3.8	Создание исполняемого файла и проверка . . . . .	17
3.9	Компоновка и проверка . . . . .	18
3.10	Код . . . . .	18
3.11	Таблица заданий . . . . .	19
3.12	Создание нового файла . . . . .	20
3.13	Решенный пример . . . . .	21

## Список таблиц

# 1 Цель работы

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

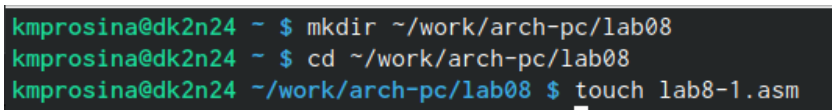
## 2 Теоретическое введение

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).

## 3 Выполнение лабораторной работы

### 3.1 Реализация циклов в NASM

Создайте каталог для программ лабораторной работы № 8, перейдите в него и создайте файл lab8-1.asm: `mkdir ~/work/arch-pc/lab08 cd ~/work/arch-pc/lab08 touch lab8-1.asm`



```
kmprosina@dk2n24 ~ $ mkdir ~/work/arch-pc/lab08
kmprosina@dk2n24 ~ $ cd ~/work/arch-pc/lab08
kmprosina@dk2n24 ~/work/arch-pc/lab08 $ touch lab8-1.asm
```

Рис. 3.1: Создание каталога и файла

При реализации циклов в NASM с использованием инструкции `loop` необходимо помнить о том, что эта инструкция использует регистр `ecx` в качестве счетчика и на каждом шаге уменьшает его значение на единицу. В качестве примера рассмотрим программу, которая выводит значение регистра `ecx`. Внимательно изучите текст программы (Листинг 8.1).

Листинг 8.1. Программа вывода значений регистра `ecx` ;-----  
; Программа вывода значений регистра 'ecx' ;-----  
`%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

```

Введите в файл lab8-1.asm текст программы из листинга 8.1. Создайте исполняемый файл и проверьте его работу.



```
kmprosina@dk2n24 ~/work/arch-pc/lab08 $ gedit lab8-1.asm
kmprosina@dk2n24 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
kmprosina@dk2n24 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab08-1 lab8-1.o
kmprosina@dk2n24 ~/work/arch-pc/lab08 $ ./lab08-1
Введите N: 15
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
```

Рис. 3.2: Создание исполняемого файла и проверка

Данный пример показывает, что использование регистра `ecx` в теле цикла `loop` может привести к некорректной работе программы. Измените текст программы добавив изменение значение регистра `ecx` в цикле: `label:`

```
sub ecx,1 ; ecx=ecx-1
mov [N],ecx
mov eax,[N]
call iprintLF
loop label
```

Создайте исполняемый файл и проверьте его работу. Какие значения принимает регистр `ecx` в цикле? Соответствует ли число проходов цикла значению ☒ введенному с клавиатуры?

```

1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
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 ; Вывод значения 'N'
27 loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
28 ; переход на 'label'
29 call quit

```

Рис. 3.3: Измененный код

```
4294923358
4294923356
4294923354
4294923352
4294923350
4294923348
4294923346
4294923344
4294923342
4294923340
4294923338
4294923336
4294923334
4294923332
4294923330
4294923328
4294923326
4294923324
4294923322
4294923320
4294923318
4294923316
4294923314
4294923312
4294923310
4294923308
4294923306
4294923304
4294923302
4294923300
4294923298
4294923296
4294923294
4294923292
4294923290
4294923288
4294923286
4294923284
4294923282
4294923280
4294923278
4294923276
4294923274
4294923272
42949^C
```

Рис. 3.4: Вывод результата после компоновки файла

После изменения кода числа начали принимать огромные значения, а цикл стал бесконечным, поэтому его пришлось прервать. Число проходов цикла не соответствует значению  $N$ , введенному с клавиатуры

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

`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` Создайте исполняемый файл и проверьте его работу. Соответствует ли в данном случае число проходов цикла значению ☒ введенному с клавиатуры?

```

1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
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 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. 3.5: Измененный код

```
kmprosina@dk2n24 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
kmprosina@dk2n24 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
kmprosina@dk2n24 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
```

Рис. 3.6: Вывод результата после компоновки файла

В данном случае число проходов цикла меньше на единицу введенного N

## 3.2 Обработка аргументов командной строки

При разработке программ иногда встает необходимость указывать аргументы, которые будут использоваться в программе, непосредственно из командной строки при запуске программы. При запуске программы в NASM аргументы командной строки загружаются в стек в обратном порядке, кроме того в стек записывается имя программы и общее количество аргументов. Последние два элемента стека для программы, скомпилированной NASM, – это всегда имя программы и количество переданных аргументов. Таким образом, для того чтобы использовать аргументы в программе, их просто нужно извлечь из стека. Обработку аргументов нужно проводить в цикле. Т.е. сначала нужно извлечь из стека количество аргументов, а затем циклично для каждого аргумента выполнить логику программы. В качестве примера рассмотрим программу, которая выводит на экран аргументы командной строки. Внимательно изучите текст программы (Листинг 8.2). Листинг 8.2. Программа выводящая на экран аргументы командной строки ; ————— ; Обра-

```

ботка аргументов командной строки ;-----
%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 sprintLF ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку next)
_end:
call quit

```

Создайте файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и введите в него текст программы из листинга 8.2. Создайте исполняемый файл и запустите его, указав аргументы: user@dk4n31:~\$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3' Сколько аргументов было обработано программой?

```

kmprosina@dk2n24 ~/work/arch-pc/lab08 $ touch lab8-2.asm
kmprosina@dk2n24 ~/work/arch-pc/lab08 $ gedit lab8-2.asm
kmprosina@dk2n24 ~/work/arch-pc/lab08 $ nasm -f elf lab8-2.asm
kmprosina@dk2n24 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
kmprosina@dk2n24 ~/work/arch-pc/lab08 $ ./lab8-2 1 2 '3'
1
2
3
kmprosina@dk2n24 ~/work/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3

```

Рис. 3.7: Создание исполняемого файла и проверка

Всего было выведено 4 аргумента. Второй аргумент был зачтен как два разных, поскольку между ними стоит пробел. Чтобы сделать их единым аргументом, необходимо поставить их в кавычки, как в примере 3 аргумента

Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. Создайте файл lab8-3.asm в каталоге ~/work/arch-pc/lab08 и введите в него текст программы из листинга 8.3.

Листинг 8.3. Программа вычисления суммы аргументов командной строки

```

%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

```

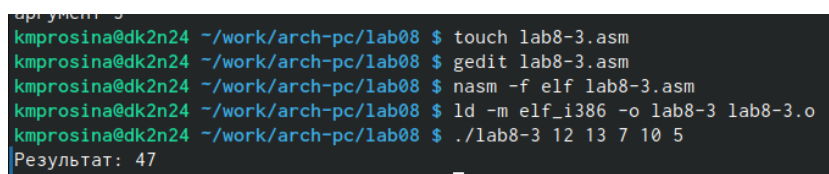


```

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

```

Создайте исполняемый файл и запустите его, указав аргументы. Пример результата работы программы: user@dk4n31:~\$ ./main 12 13 7 10 5 Результат: 47 user@dk4n31:~\$



```

user@dk4n31:~$ cd /work/arch-pc/lab08
kmprosina@dk2n24 /work/arch-pc/lab08 $ touch lab8-3.asm
kmprosina@dk2n24 /work/arch-pc/lab08 $ gedit lab8-3.asm
kmprosina@dk2n24 /work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
kmprosina@dk2n24 /work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
kmprosina@dk2n24 /work/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 47

```

Рис. 3.8: Создание исполняемого файла и проверка

Измените текст программы из листинга 8.3 для вычисления произведения аргументов командной строки.

```

Результат: 80
kmprosina@dk8n74 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
kmprosina@dk8n74 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
kmprosina@dk8n74 ~/work/arch-pc/lab08 $ ./lab8-3 1 2 3 4
Результат: 24
kmprosina@dk8n74 ~/work/arch-pc/lab08 $ ./lab8-3 2 2 2 2
Результат: 16
kmprosina@dk8n74 ~/work/arch-pc/lab08 $ ./lab8-3 9 9
Результат: 81
kmprosina@dk8n74 ~/work/arch-pc/lab08 $ 

```

Рис. 3.9: Компоновка и проверка

```

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

```

Рис. 3.10: Код

В ходе преобразований кода, мне удалось сделать так, чтобы аргументы перемножались.

`mul esi` Здесь мы умножаем промежуточное произведение на след. аргумент  
`mov esi,eax` После этого мы присваиваем еси получившееся произведение, чтобы впоследствии цикл умножил его на следующий элемент.

### 3.3 Задание для самостоятельной работы

1. Напишите программу, которая находит сумму значений функции  $f(x)$  для  $x = x_1, x_2, \dots, x_n$ , т.е. программа должна выводить значение  $f(x_1) + f(x_2) + \dots + f(x_n)$ . Значения  $x_i$  передаются как аргументы. Вид функции  $f(x)$  выбрать из таблицы 8.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах  $x = x_1, x_2, \dots, x_n$ . Пример работы программы для функции  $f(x) = x + 2$  и набора  $x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4$ :  

```
user@dk4n31:~$ ./main 1 2 3 4
Функция: f(x)=x+2
Результат: 18
user@dk4n31:~$
```

Моим вариантом является 19 (8х-3) согласно этой таблице:

**Таблица 8.1.** Выражения для  $f(x)$  для задания №1.

Номер варианта	$f(x)$	Номер варианта	$f(x)$
<b>1</b>	$2x + 15$	<b>11</b>	$15x + 2$
<b>2</b>	$3x - 1$	<b>12</b>	$15x - 9$
<b>3</b>	$10x - 5$	<b>13</b>	$12x - 7$
<b>4</b>	$2(x - 1)$	<b>14</b>	$7(x + 1)$
<b>5</b>	$4x + 3$	<b>15</b>	$6x + 13$
<b>6</b>	$4x - 3$	<b>16</b>	$30x - 11$
<b>7</b>	$3(x + 2)$	<b>17</b>	$10(x - 1)$
<b>8</b>	$7 + 2x$	<b>18</b>	$17 + 5x$
<b>9</b>	$10x - 4$	<b>19</b>	$8x - 3$
<b>10</b>	$5(2 + x)$	<b>20</b>	$3(10 + x)$

Рис. 3.11: Таблица заданий

Для создания файла я использовала команду touch, а для редактирования - gedit.

```
kmprosina@dk2n24 ~/work/arch-pc/lab08 $ touch lab8-4.asm
kmprosina@dk2n24 ~/work/arch-pc/lab08 $ gedit lab8-4.asm
```

Рис. 3.12: Создание нового файла

В итоге у меня получился такой код для решения:

```
%include 'in_out.asm'

SECTION .data
msg1 db "Функция: f(x)=8x-3",0
msg2 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

mov ebx,8
mul ebx
sub eax,3
```

add esi, eax

loop next ; переход к обработке следующего аргумента

\_end:

mov eax, msg1 ; вывод сообщения "Функция: f(x)=8x-3:"

call sprintLF

mov eax, msg2 ; вывод сообщения "Результат:"

call sprint

mov eax, esi ; записываем итог в регистр eax

call iprintLF ; печать результата

call quit ; завершение программы

```
kmprosina@dk3n35 ~/work/arch-pc/lab08 $ nasm -f elf lab8-4.asm
kmprosina@dk3n35 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-4 lab8-4.o
kmprosina@dk3n35 ~/work/arch-pc/lab08 $ ./lab8-4 2
Функция: f(x)=8x-3
Результат: 13
kmprosina@dk3n35 ~/work/arch-pc/lab08 $ ./lab8-4 2 2
Функция: f(x)=8x-3
Результат: 26
kmprosina@dk3n35 ~/work/arch-pc/lab08 $ ./main 1 2 3 4
bash: ./main: Нет такого файла или каталога
kmprosina@dk3n35 ~/work/arch-pc/lab08 $ ./lab8-4 1 2 3 4
Функция: f(x)=8x-3
Результат: 68
```

Рис. 3.13: Решенный пример

Объясню вычисление. `mov ebx, 8` Здесь я присвоила ебкс число восемь, чтобы впоследствии с ним можно было работать `mul ebx` Этой командой я умножила аргумент на 8 `sub eax, 3` `sub` позволяет вычесть из аргумента 3 `add esi, eax` далее мы добавляем к счетчику суммы функций получившийся измененный аргумент

## 4 Выводы

Были приобретены навыки написания программ с использованием циклов и обработкой аргументов командной строки

## **Список литературы**