

# **Лабораторная работа номер 8**

**Архитектура компьютера**

Титков Ярослав Максимович

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
4.1	Реализация циклов в NASM: . . . . .	8
4.2	Обработка аргументов командной строки . . . . .	10
4.3	Задания для самостоятельной работы: . . . . .	11
<b>5</b>	<b>Выводы</b>	<b>13</b>

## Список иллюстраций

4.1	Создал каталог lab08, затем перешёл в него и создал файл lab8-1.asm	8
4.2	Вставил нужную функцию из Листинга 8.1 и добавил все изменения	9
4.3	Проверил работоспособность файла . . . . .	9
4.4	Создал файл lab8-2.asm ввёл программу из листинга 8.2 и запустил	10
4.5	Создал файл lab8-3.asm и переписал программу из листинга 8.3 .	11
4.6	Запустил программу lab8-3 . . . . .	11
4.7	Изменил текст программы для вычисления произведения аргумен- тов командой строки и запустил . . . . .	11
4.8	Создал программу со значением $f(x)=6x+13$ . . . . .	12
4.9	Запустил программу . . . . .	12

# 1 Цель работы

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. Задание для самостоятельной работы

### 3 Теоретическое введение

тек — 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. 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). 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. 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. Аналогично команде записи в стек существует команда `rora`, которая восстанавливает из стека все регистры общего назначения, и команда `rorf` для перемещения значений из вершины стека в регистр `flag`. 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
```

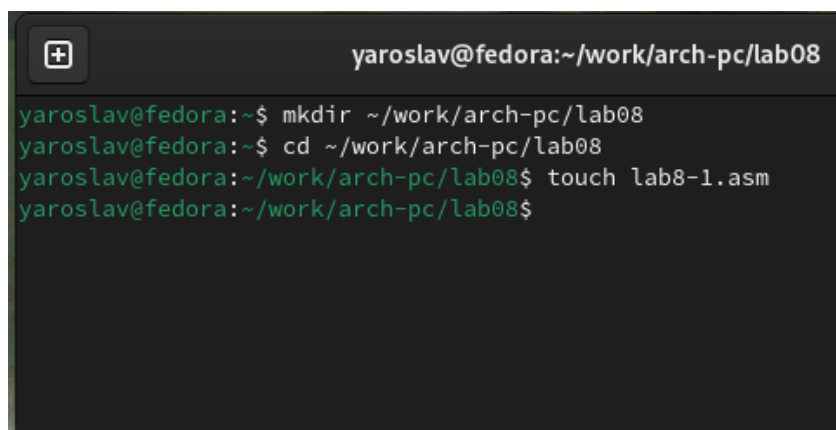
```
NextStep: ... .. ; тело цикла ...
```

```
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

### 4.1 Реализация циклов в NASM:



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

Рис. 4.1: Создал каталог lab08, затем перешёл в него и создал файл lab8-1.asm



```
GNU nano 7.2 lab8-1.asm
#include "in_out.asm"
SECTION .data
sgl db 'Введите N: ',0h

SECTION .bss
: 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 ; добавление значения ecx в стек
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения 'N'
pop ecx ; извлечение значения ecx из стека
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'

call quit
```

Рис. 4.2: Вставил нужную функцию из Листинга 8.1 и добавил все изменения

```
yaroslav@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
yaroslav@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
yaroslav@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 1
0
yaroslav@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 2
1
0
yaroslav@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 3
2
1
0
yaroslav@fedora:~/work/arch-pc/lab08$
```

Рис. 4.3: Проверил работоспособность файла

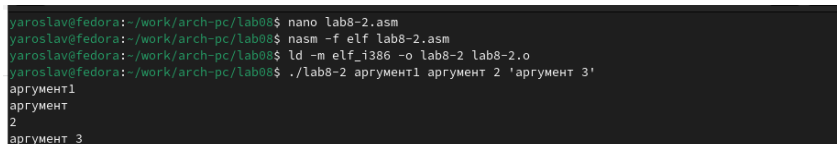
Ответы на вопросы: Вопрос: Какие значения принимает регистр ecx в цикле?  
Соответствует ли число проходов цикла значению N введенному с клавиатуры?  
Ответ: В измененном коде регистр ecx принимает значения от N до 1 (вклю-

чительно). В каждой итерации цикла значение `ecx` уменьшается на 1. Да, число проходов цикла соответствует значению `N`, введенному с клавиатуры. Цикл выполняется `N` раз, так как `ecx` изначально равен `N` и уменьшается на 1 в каждой итерации до тех пор, пока не станет равным 0.

Вопрос: Соответствует ли в данном случае число проходов цикла значению `N` введенному с клавиатуры?

Ответ: Да, в данном случае число проходов цикла соответствует значению `N`, введенному с клавиатуры. Благодаря использованию стека для сохранения и восстановления значения `ecx`, программа корректно выполняет цикл `N` раз.

## 4.2 Обработка аргументов командной строки



```
yaroslav@fedora: ~/work/arch-pc/lab08$ nano lab8-2.asm
yaroslav@fedora: ~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
yaroslav@fedora: ~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
yaroslav@fedora: ~/work/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
2
аргумент 3
```

Рис. 4.4: Создал файл `lab8-2.asm` ввёл программу из листинга 8.2 и запустил

Ответы на вопросы Вопрос: Сколько аргументов было обработано программой?

Ответ: Первый аргумент: `аргумент1`, Второй аргумент: `аргумент2`, Третий аргумент: `аргумент 3`. Программа извлекает аргументы из стека и выводит их на экран. Количество аргументов, которые будут обработаны, равно количеству аргументов, переданных при запуске программы, за вычетом имени самой программы.

```

GNU nano 7.2                                lab8-3.asm                                Изме
%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 iprintf ; печать результата
call quit ; завершение программы

```

Рис. 4.5: Создал файл lab8-3.asm и переписал программу из листинга 8.3

```

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

```

Рис. 4.6: Запустил программу lab8-3

```

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

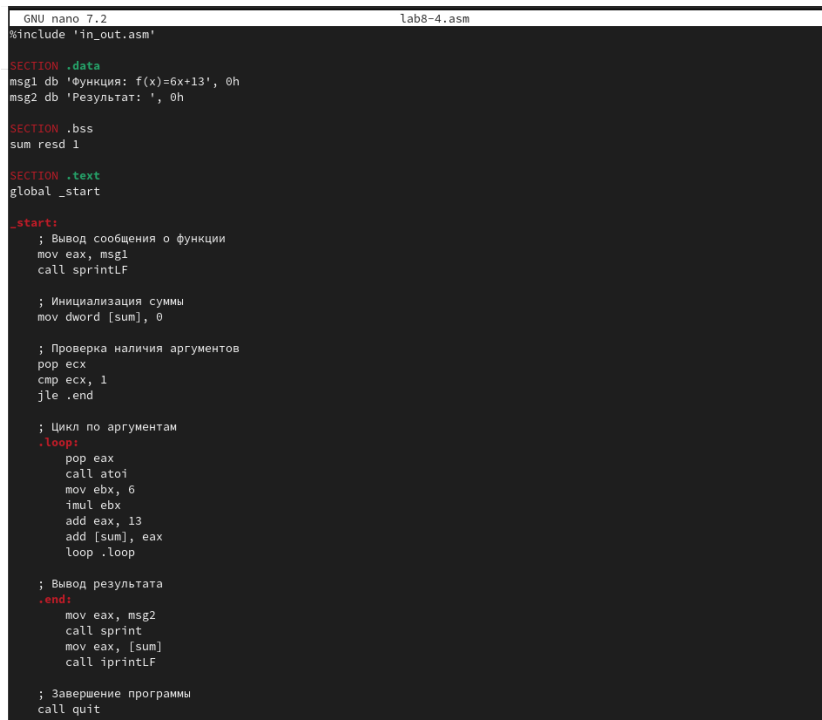
```

Рис. 4.7: Изменил текст программы для вычисления произведения аргументов командой строки и запустил

## 4.3 Задания 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  $f(x_2) + f(x_1) + \dots + f(x_n)$ . Значения  $x_i$  передаются как аргументы. Вид функции  $f(x)$  выбрать из таблицы 8.1 вариантов заданий в соответствии с вариантом (15 вариантов), полученным при выполнении лабораторной

работы № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах  $x = x_1, x_2, \dots, x_n$



```
GNU nano 7.2 lab8-4.asm
#include 'in_out.asm'

SECTION .data
msg1 db 'Функция: f(x)=6x+13', 0h
msg2 db 'Результат: ', 0h

SECTION .bss
sum resd 1

SECTION .text
global _start

_start:
; Вывод сообщения о функции
mov eax, msg1
call sprintf

; Инициализация суммы
mov dword [sum], 0

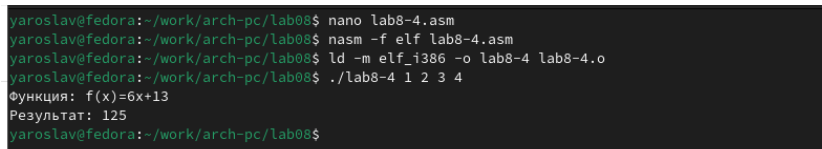
; Проверка наличия аргументов
pop ecx
cmp ecx, 1
jle .end

; Цикл по аргументам
.loop:
pop eax
call atoi
mov ebx, 6
imul ebx
add eax, 13
add [sum], eax
loop .loop

; Вывод результата
.end:
mov eax, msg2
call sprintf
mov eax, [sum]
call sprintf

; Завершение программы
call quit
```

Рис. 4.8: Создал программу со значением  $f(x)=6x+13$



```
yaroslav@fedora:~/work/arch-pc/lab08$ nano lab8-4.asm
yaroslav@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-4.asm
yaroslav@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-4 lab8-4.o
yaroslav@fedora:~/work/arch-pc/lab08$ ./lab8-4 1 2 3 4
Функция: f(x)=6x+13
Результат: 125
yaroslav@fedora:~/work/arch-pc/lab08$
```

Рис. 4.9: Запустил программу

## **5 Выводы**

В ходе выполнения задания были приобретены навыки написания программ с использованием циклов и обработкой аргументов командной строки на языке ассемблера.