РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 8

дисциплина: Архитектура компьютера

Студент: Ибрагимов Гаджимурад Шамильевич

Группа: НКАбд-02-25

МОСКВА

2025 г.

СОДЕРЖАНИЕ

1.ЦЕЛЬ РАБОТЫ

Целью моей работы является приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки

2.ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ

2.1 Сущность стека

Стек (англ. *stack* **— "стопка")** — это структура данных, работающая по принципу LIFO (Last In, First Out) — "последним пришёл, первым ушёл". Реализация стека приведена во многих языках программирования.

2.2 Функционал стека

С данной структурой данных возможен ограниченный список действий:

Основные операции со стеком:

1. `push(x)` — добавить элемент `x` на вершину стека.
2. `pop()` — удалить верхний элемент.
3. `top()` — возвращает верхний элемент без удаления.
4. `empty()` — проверка на пустоту (возвращает true/false)

Реализация стека напоминает чипсы Pringles;)

2.3 Циклы

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
```

Иструкция 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.

3. ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

3.1.1 Реализация циклов в NASM

Создаем каталог для программ лабораторной работы № 8, переходим в него и создаём файл lab8-1.asm

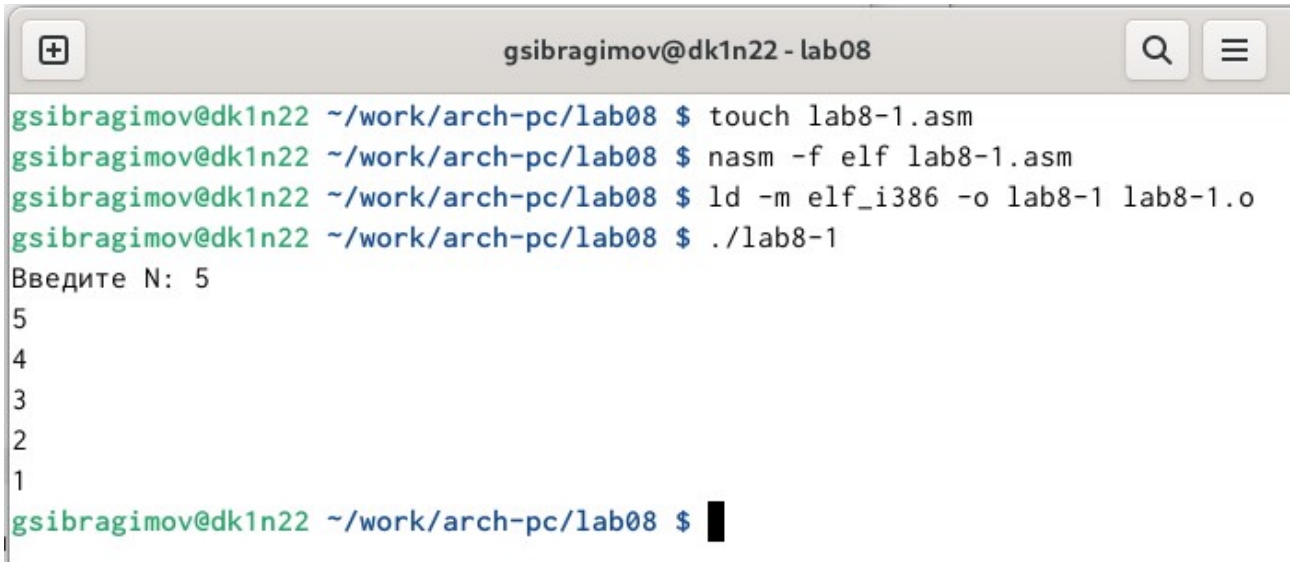
```
gsibragimov@dk1n22 ~/work/arch-pc/lab08 $ touch lab8-1.asm
gsibragimov@dk1n22 ~/work/arch-pc/lab08 $
```

Реализация цикла с N числом итераций с последующим уменьшением числа на единицу:

Открыть | lab8-1.asm ~/work/arch-pc/lab08 | Сохранить

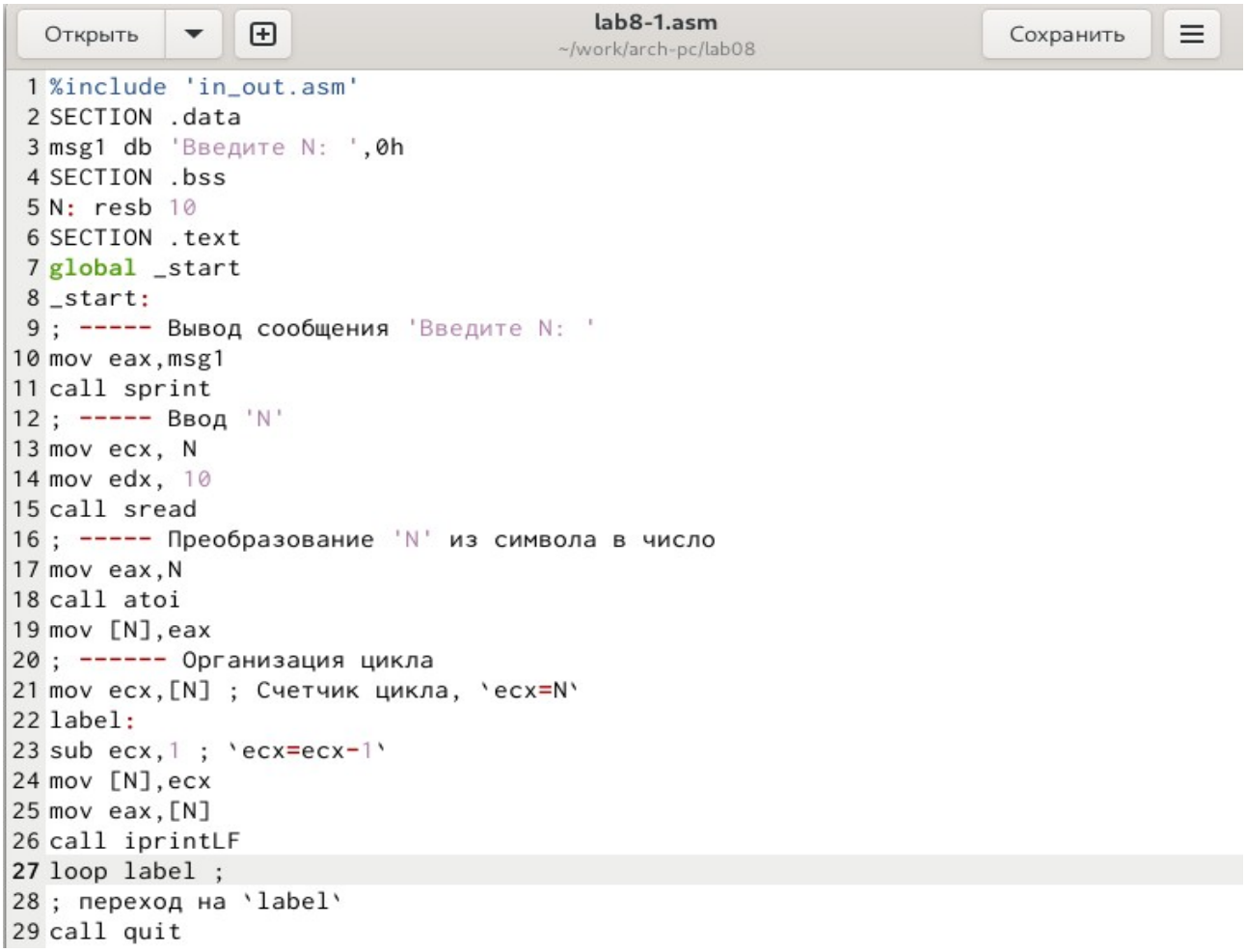
```
%include 'in_out.asm'
SECTION .data
msg1 db 'Введите N: ',0h
SECTION .bss
N: resb 10
SECTION .text
global _start
_start:
; ----- Вывод сообщения 'Введите N: '
mov eax,msg1
call sprint
; ----- Ввод 'N'
mov ecx, N
mov edx, 10
call sread
; ----- Преобразование 'N' из символа в число
mov eax,N
call atoi
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
```

Запустим программу и посмотрим на результат:

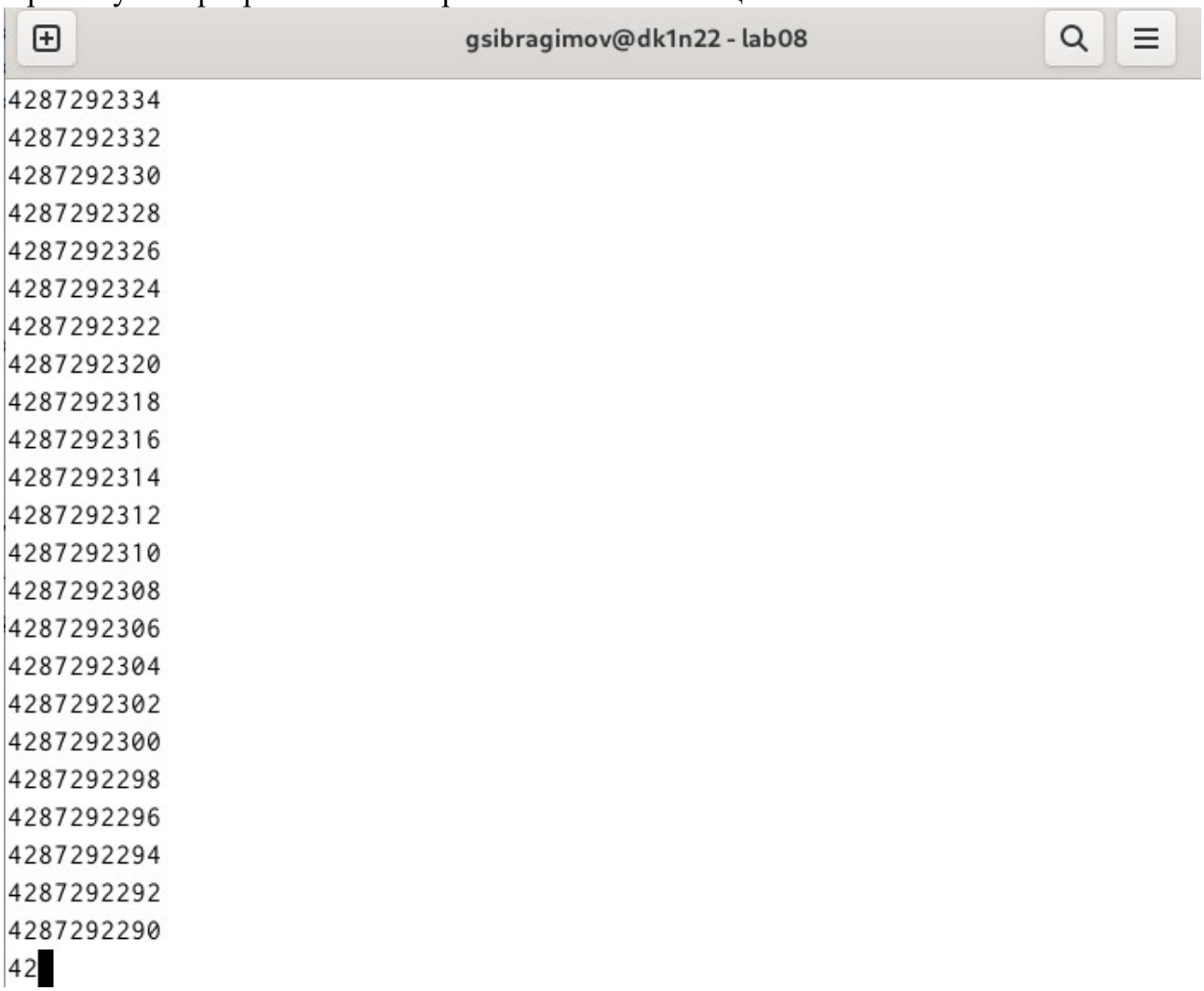


3.1.2 Интересный пример

Данный пример показывает, что использование регистра ecx в теле цилка loop может привести к некорректной работе программы. Изменим текст программы добавив и изменив значение регистра ecx в цикле:



При запуске программы нас встречает бесконечный цикл:



3.1.3 Для восстановления работоспособности программы изменим код:

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

Открыть | lab8-1.asm ~/work/arch-pc/lab08 | Сохранить

```
%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 ; 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 ;
; переход на `label`
call quit
```

Запустим программу:

gsibragimov@dk1n22 - lab08

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

Число проходов соответствует счетчику цикла N введенному с клавиатуры.

3.2.1 Обработка аргументов командной строки

Создаем файл lab8-2.asm в нашем рабочем каталоге и вводим в него следующий текст программы:

```
gsibragimov@dk1n22 ~/work/arch-pc/lab08 $ ls
in_out.asm  lab8-1  lab8-1.asm  lab8-1.o  lab8-2  lab8-2.asm  lab8-2.o
gsibragimov@dk1n22 ~/work/arch-pc/lab08 $
```

Создаем исполняемый файл и запускаем его, указав аргументы следующие: аргумент1 аргумент 2 'аргумент 3'

```
gsibragimov@dk1n22 ~/work/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
gsibragimov@dk1n22 ~/work/arch-pc/lab08 $
```

3.3.1 Программа вычисления суммы аргументов командной строки

Рассмотрим еще один пример программы, которая выводит сумму чисел, которые передаются в программу как аргументы.

Создаем файл lab8-3.asm

```
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ls
in_out.asm  lab8-1.asm  lab8-2      lab8-2.o
lab8-1      lab8-1.o    lab8-2.asm  lab8-3.asm
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $
```

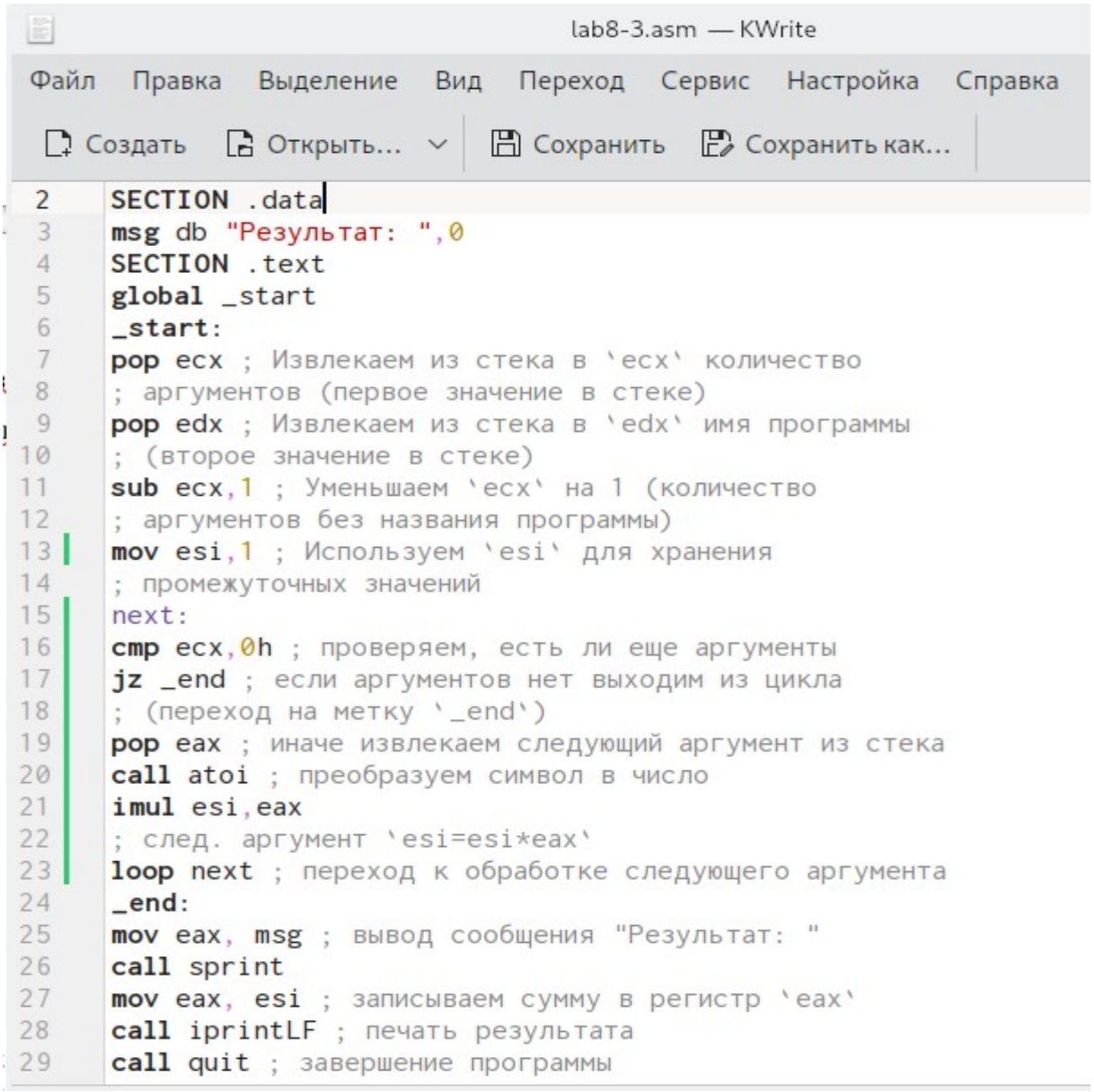
Открыть | lab8-3.asm ~/work/arch-pc/lab08 | Сохранить

```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
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 mov esi, 0 ; Используем `esi` для хранения
14 ; промежуточных сумм
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку `_end`)
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 add esi,eax ; добавляем к промежуточной сумме
22 ; след. аргумент `esi=esi+eax`
23 loop next ; переход к обработке следующего аргумента
24 _end:
25 mov eax, msg ; вывод сообщения "Результат: "
26 call sprint
27 mov eax, esi ; записываем сумму в регистр `eax`
28 call iprintLF ; печать результата
29 call quit ; завершение программы
```

Создаем исполняемый файл и запускаем его:

```
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-3
Результат: 0
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 47
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $
```

3.3.2 Давайте изменим текст программы для вычисления произведения аргументов командной строки.



Запускаем программу, указав аргументы:

```
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-3 2 1 3
Результат: 6
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $
```

Результат есть произведение аргументов

4. ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

4.1. Напишите программу, которая находит сумму значений функции $f(x)$ для $x = x1, x2, ..., xn$, т.е. программа должна выводить значение $f(x1) + f(x2) + ... + f(xn)$. Значения xi передаются как аргументы.

Вид функции $f(x)$: **13** $12x - 7$

Создаем файл 1.asm

```
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ touch 1.asm
```

Текст программы:

Открыть | 1.asm ~/work/arch-pc/lab08 | Сохранить

```
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0h
section .bss
x resb 10
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 значений

next:
cmp ecx, 0h  ; Если аргументов нет, то переход к завершению программы
jz _end

pop eax ; извлекаем аргумент из стека
call atoi ; преобразуем в число
mov [x], eax
mov ebx, 12
imul ebx
sub eax, 7
add esi, eax
loop next

_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
```

Создаем исполняемый файл и проверяем его работу на нескольких значениях:

```
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf 1.asm
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o 1 1.o
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ./1
Результат: 0
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ./1 1 2 3 4 5
Результат: 145
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ./1 2 2 2
Результат: 51
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ./1 3 3
Результат: 58
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $
```

5. ВЫВОД

В результате выполнения данной лабораторной работы мы научились пользоваться циклами для решения задач, а также познакоимились со структурой данных “стек”.

6. ИСТОЧНИКИ

1. 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.

2. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с.