Отчёта по лабораторной работе №8

Команды безусловного и условного переходов в Nasm. Программирование ветвлений.

Мантуров Татархан Бесланович

Содержание

Список иллюстраций

1 Цель работы

Изучить работу циклов и обработкой аргументов командной строки.

2 Задание

Написать программы с использованием циклов и обработкой аргументов командной строки.

3 Выполнение лабораторной работы

Создаем каталог для программ ЛБ8, и в нем создаем файл

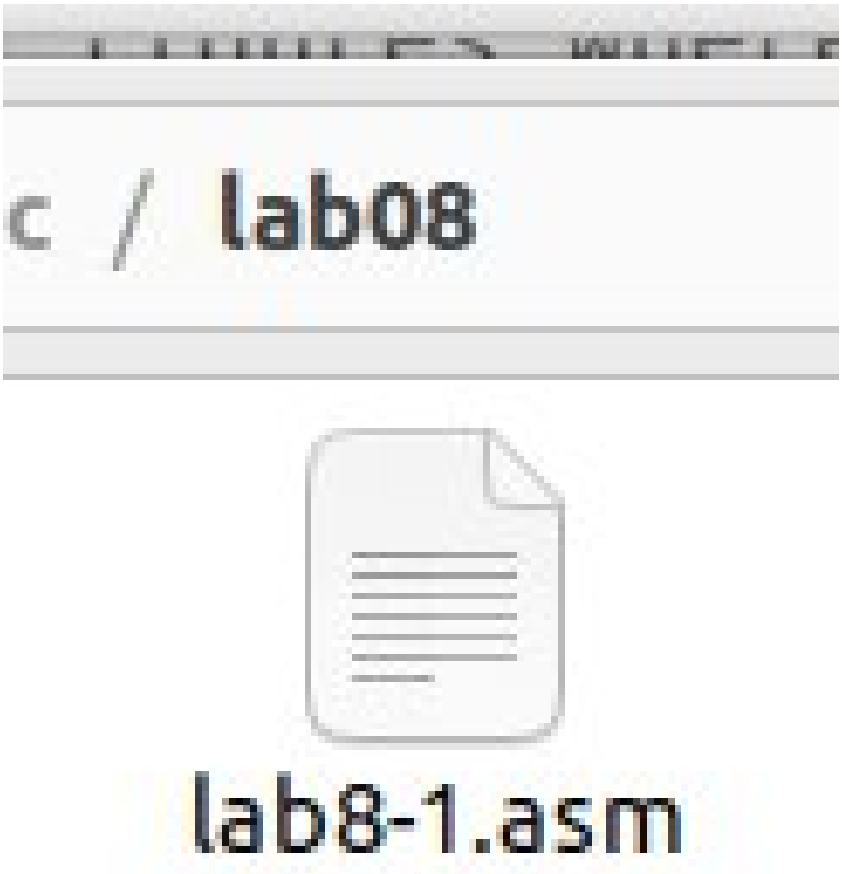

Рис. 3.1: Создание каталога

Заполняем его в соответствии с листингом 8.1

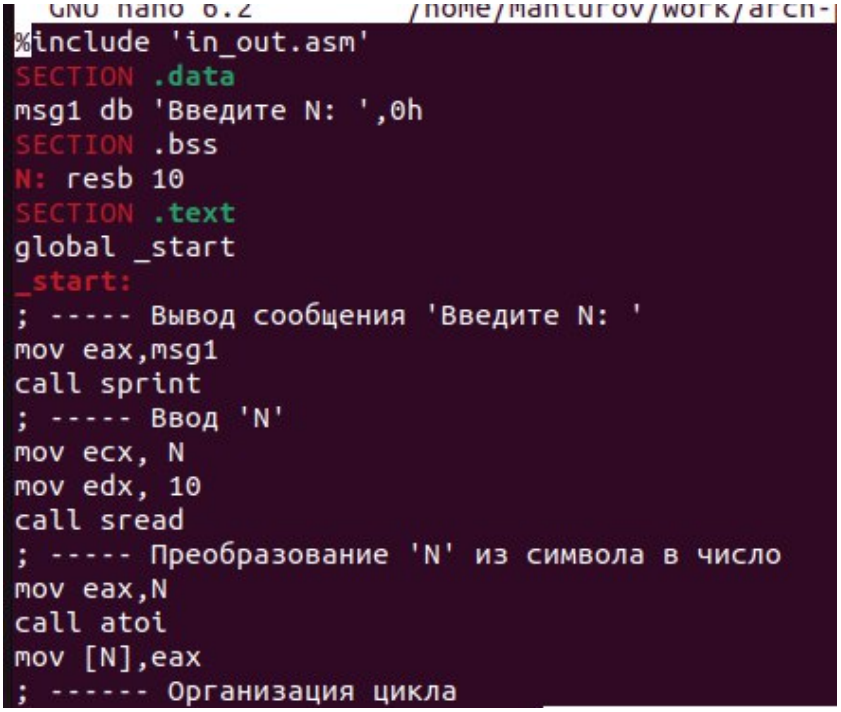

Рис. 3.2: Заплнение 8.1

Создаем исполняемый файл и запускаем его

Рис. 3.3: Проверка

Снова открываем файл для редактирования и изменяем его, добавив изменение значения регистра в цикле

```
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
```

Рис. 3.4: Вносим изменения

Создаем исполняемый файл и запускаем его

```
manturov@ubuntu:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 12
11
```

Рис. 3.5: Проверка

Регистр ecx принимает значения, в цикле label данный регистр уменьшается на 2 командой sub и loop). Число проходов цикла не соответсвует числу N, так как уменьшается на 2. Снова открываем файл для редактирования и изменяем его, чтобы все корректно работало

```
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
```

Рис. 3.6: Редактирование

Создаем исполняемый файл и запускаем его

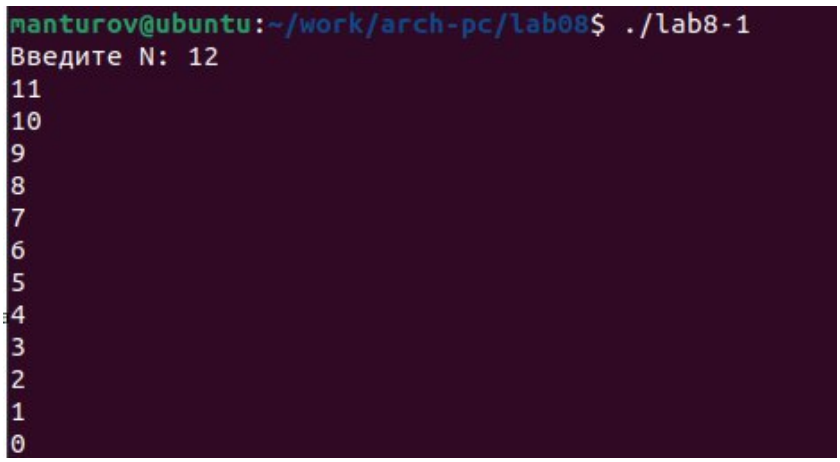

Рис. 3.7: Проверка работы

В данном случае число проходов цикла равна числу N.

Создаем новый файл

Рис. 3.8: Новый файл

Заполняем его в соответствии с листингом 8.2

```
 GNU nano 6.2                  /home/manturov/work/arch-pc/
%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
```

Рис. 3.9: Заполняем 8.2

Создаем исполняемый файл и проверяем его работу, указав аргументы

```
manturov@ubuntu:~/work/arch-pc/lab08$ ./lab8-2 1 2 '3'
1
2
3
```

Рис. 3.10: Проверка

Програмой было обработано 3 аргумента.

Создаем новый файл main.asm

Рис. 3.11: Новый файл

Открываем файл и заполняем его в соответствии с листингом 8.3

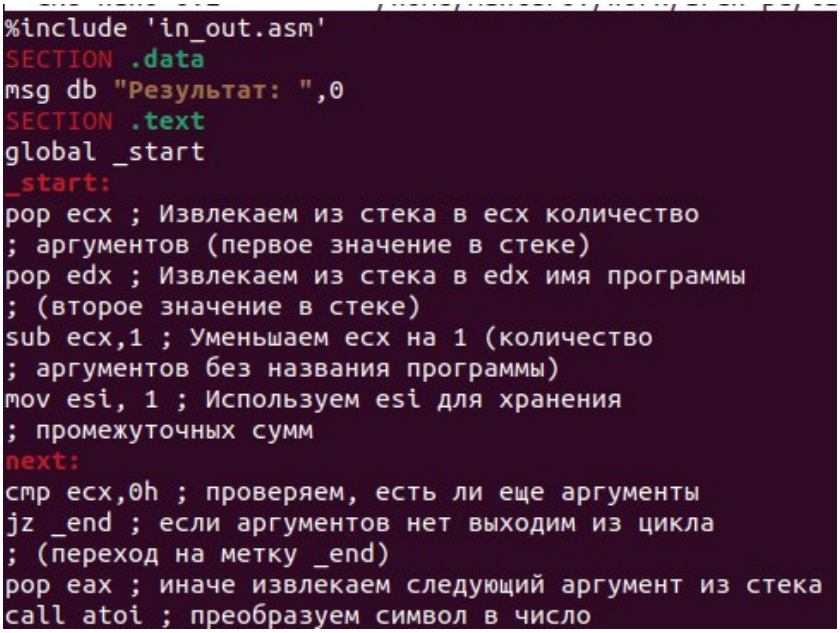

Рис. 3.12: Заполнение 8.3

Создаём исполняемый файл и запускаем его, указав аргументы

Рис. 3.13: Проверка

Снова открываем файл для редактирования и изменяем его, чтобы вычислялось произведение вводимых значений

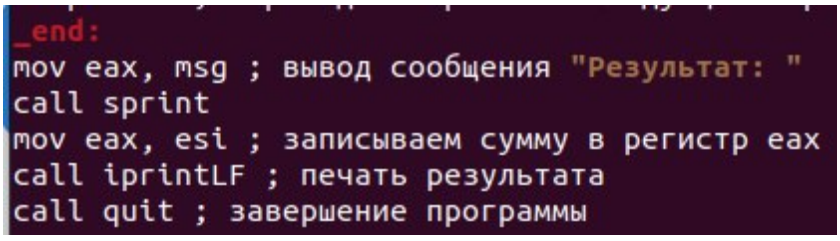

Рис. 3.14: Редактирование

Создаём исполняемый файл и запускаем его, указав аргументы

Рис. 3.15: Проверка работы

3.1 Самостоятельная работа

Вариант 2 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) 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 = x1, x2, …, xn. Создаем новый файл

Рис. 3.16: Новый файл

Открываем его и пишем программу, которая выведет сумму значений, получившихся после решения выражения 3(10+x)

```
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 eax, 10
mov ebx, 3
mul ebx
add esi,eax ; добавляем к промежуточной сумме
```

Рис. 3.17: Пишем программу

Транслируем файл и смотрим на работу программы

```
manturov@ubuntu:~/work/arch-pc/lab08$ ./samos 1 2 3 4
Функция: f(x)=3(10+x)
Результат: 150
```

Рис. 3.18: Проверка

4 Выводы

Мы научились решать программы с использованием циклов и обработкой аргументов командной строки.