Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки

Андрюшин Никита Сергеевич

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Научиться работать с циклами на языке Ассемблера, а также научиться обрабатывать аргументы командной строки

2 Выполнение лабораторной работы

Для начала выполнения лабораторной работы создадим рабочую директорию и файл lab8-1.asm (рис. 2.1):

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

Рис. 2.1: Создание рабочей директории и файла lab8-1.asm

Далее, запустим Midnight commander (рис. 2.2):

```
nsandryushin@nsandryushin:~/work/arch-pc/lab08$ mc
```

Рис. 2.2: Запуск Midnight commander

Теперь, вставим в ранее созданный файл из листинга 8.1. Он должен запускать цикл и выводить каждую итерацию число, на единицу меньше предыдущего (начинается выводить с числа N) (рис. 2.3):

```
%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
```

Рис. 2.3: Вставка кода из файла листинга 8.1

Чтобы собрать код, нужен файл in_out.asm. скопируем его из директории прошлой лабораторной работы (рис. 2.4):

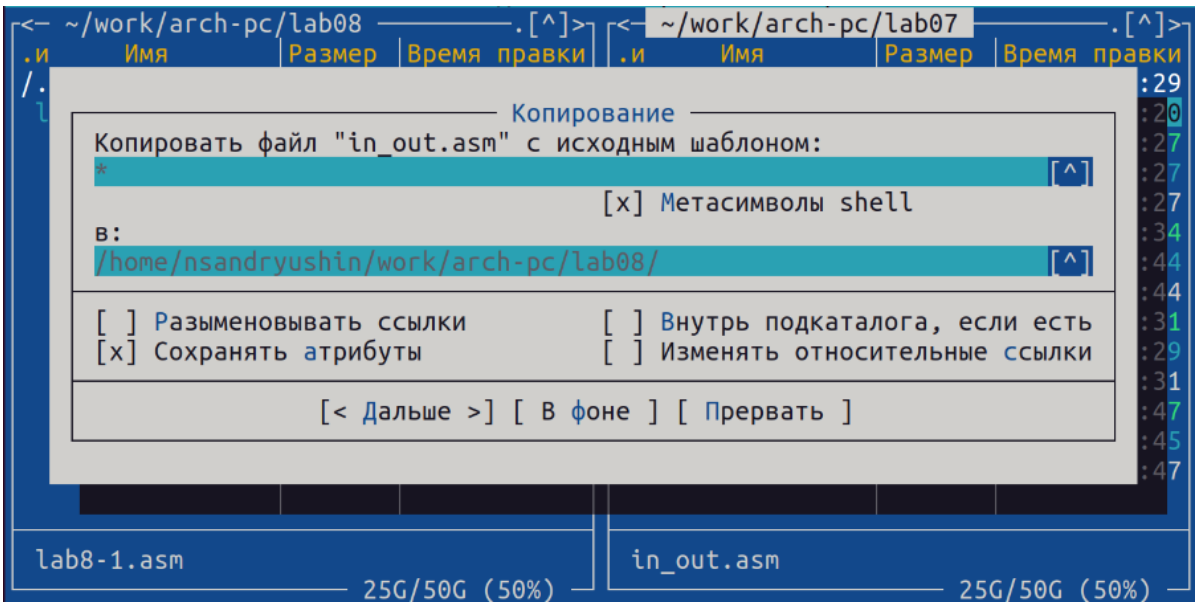

Рис. 2.4: Копирование файла in_out.asm в рабочую директорию

Теперь соберём программу и посмотрим на результат выполнения (рис. 2.5):

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

Рис. 2.5: Сборка программы из файла lab8-1.asm и её запуск

Как видим, она выводит числа он N до единицы включительно. Теперь попробуем изменить код, чтобы в цикле также отнималась единица у регистра ecx (рис. 2.6):

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

Рис. 2.6: Изменение файла lab8-1.asm

Попробуем собрать программу и запустить её (рис. 2.7):

```
nsandryushin@nsandryushin:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
nsandryushin@nsandryushin:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.
o
nsandryushin@nsandryushin:~/work/arch-pc/lab08$ ./lab8-1
```

Рис. 2.7: Повторная сборка программы из файла lab8-1.asm и её запуск

Введём в качестве N число 5 и посмотрим на результат выполнения (рис. 2.8):

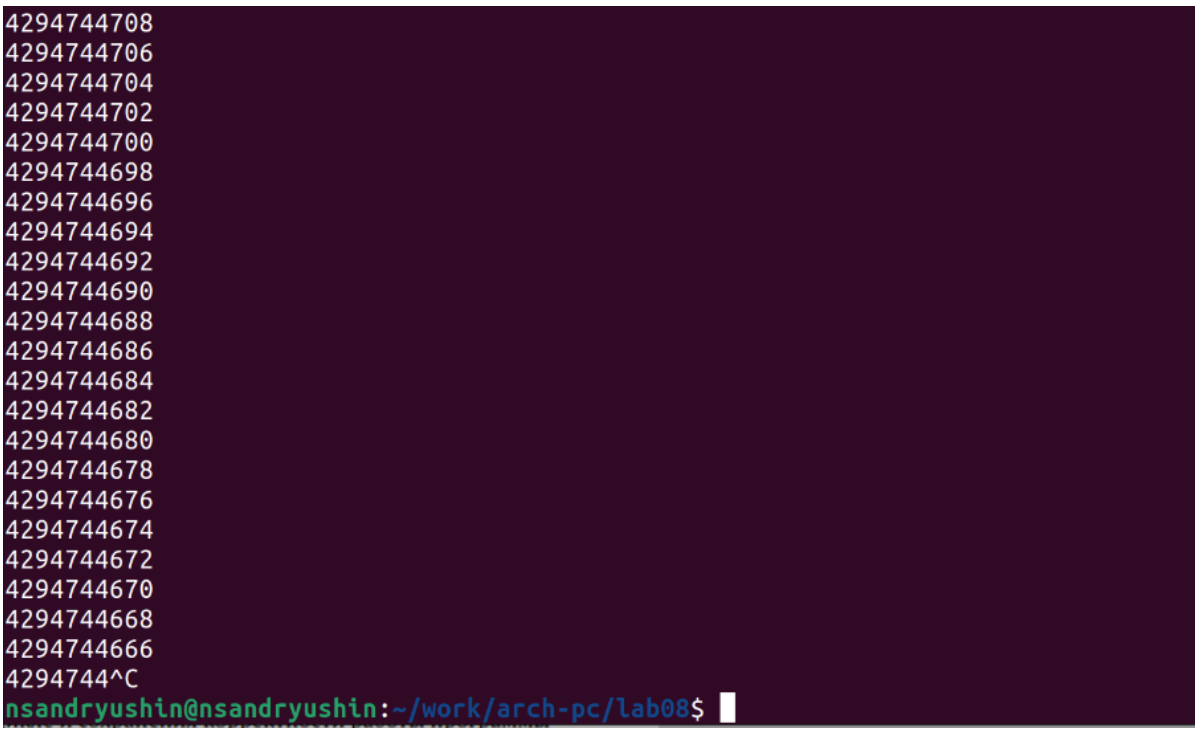

Рис. 2.8: Результат вывода

Как видим, цикл выполняется бесконечное количество раз. Это связано с тем, что цикл останавливается в тот момент, когда при проверке ecx равен 0, но он каждое выполнение цикла уменьшается на 2, из-за чего, в случае нечётного числа, никогда не достигнет нуля. Регистр ecx меняет своё значение дважды: стандартно -1 после каждой итерации и -1 в теле цикла из-за команды sub. Если на вход подать чётное число, цикл прогонится N/2 раз, выводя числа от N-1 до 1 (выводит через одно число) (рис. 2.9):

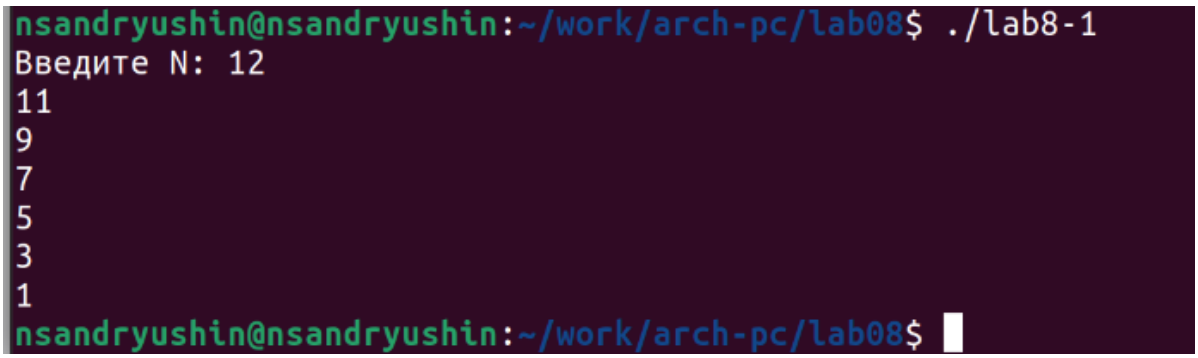

Рис. 2.9: Результат вывода для чётного N

Таким образом, количество итераций цикла не равно N ни при подаче на вход чётного числа, ни при подаче нечётного.

Теперь попробуем изменить программу так, чтобы она сохраняла значение регистра ecx в стек (рис. 2.10):

```
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
loop label ; `ecx=ecx-1` и если `ecx` не '0'
; переход на `label`
call quit
```

Рис. 2.10: Редактирование файла lab8-1.asm

Попробуем собрать и запустить программу (рис. 2.11):

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

Рис. 2.11: Повторная сборка программы из файла lab8-1.asm и её запуск

Теперь, программа выводит все числа от N-1 до нуля. Таким образом, число прогонов цикла равно числу N. Создадим второй файл (рис. 2.12):

```
nsandryushin@nsandryushin:~/work/arch-pc/lab08$ touch lab8-2.asm
nsandryushin@nsandryushin:~/work/arch-pc/lab08$
```

Рис. 2.12: Создание второго файла: lab8-2.asm

И вставим в него код из файла листинга 8.2 (рис. 2.13):

```
%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. 2.13: Запись кода из листинга 8.2 в файл lab8-2.asm

Соберём и запустим его, указав некоторые аргументы. Посмотрим на результат (рис. 2.14):

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

Рис. 2.14: Сборка программы из файла lab8-2.asm и её запуск

Как видим, он обработал 4 аргумента. Аргументы разделяются пробелом, либо,

когда аргумент содержит в себе пробел, обрамляется в кавычки. Создадим третий файл (рис. 2.15):

```
nsandryushin@nsandryushin:~/work/arch-pc/lab08$ touch lab8-3.asm
nsandryushin@nsandryushin:~/work/arch-pc/lab08$
```

Рис. 2.15: Создание третьего файла: lab8-3.asm

И вставим в него код из листинга 8.3. Он будет находить сумму всех аргументов (рис. 2.16):

```
%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
```

Рис. 2.16: Запись кода из листинга 8.3 в файл lab8-3.asm

Теперь соберём программу и запустим её (рис. 2.17):

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

Рис. 2.17: Сборка программы из файла lab8-2.asm и её запуск

Как видим, программа действительно выводит сумму всех аргументов. Изменим её так, чтобы она находила не сумму, а произведение всех аргументов (рис. 2.18):

```
; аргументов без названия программы)
mov esi, 1 ; 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
mul esi
mov esi,eax
loop next ; переход к обработке следующего аргумента
_end:
mov ebx, eax
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, ebx ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
```

Рис. 2.18: Изменение файла lab8-3.asm

Соберём программу и запустим её (рис. 2.19):

```
nsandryushin@nsandryushin:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
nsandryushin@nsandryushin:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.
o
nsandryushin@nsandryushin:~/work/arch-pc/lab08$ ./lab8-3 2 3 4 5
Результат: 120
nsandryushin@nsandryushin:~/work/arch-pc/lab08$
```

Рис. 2.19: Повторная сборка программы из файла lab8-3.asm и её запуск

Как видим, программа выводит правильный ответ

3 Выполнение задания для самостоятельной работы

Для выполнения самостоятельной работы создадим файл в формате .asm (рис. 3.1):

Рис. 3.1: Создание файла самостоятельной работы

В рамках самостоятельной работы необходимо сделать задание под вариантом 9. Так, необходимо сложить результаты выполнения функции f(x)=10x-4 для всех введённых аргументов (рис. 3.2 и рис. 3.3):

```
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg2 db "Функция: f(x)=10x-4"
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
```

Рис. 3.2: Код файла самостоятельной работы

```
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, 10
mul ebx
sub eax, 4
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg2
call sprintLF
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
```

Рис. 3.3: Код файла самостоятельной работы (продолжение)

Соберём и запустим программу, вводя различные аргументы (рис. 3.4):

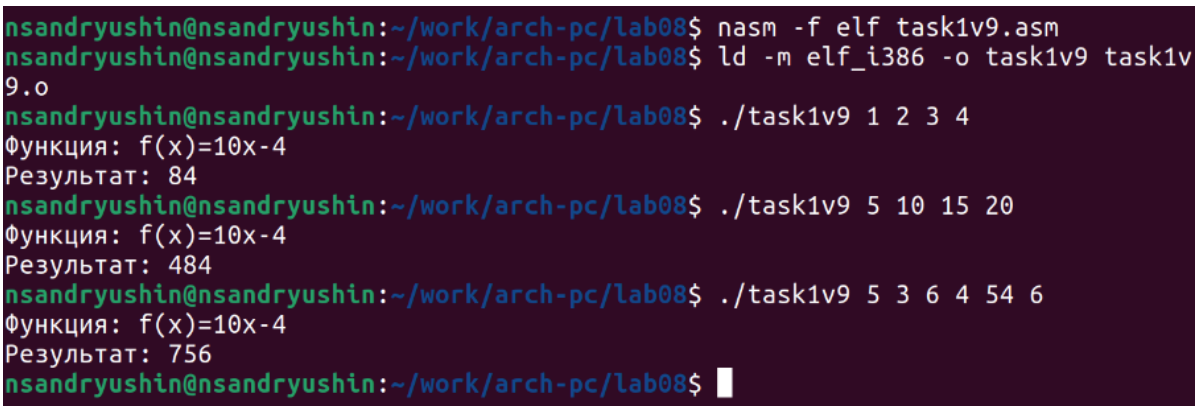

Рис. 3.4: Сборка и запуск программы первого задания самостоятельной работы, а также результат выполнения

Пересчитав результат вручную, убеждаемся, что программа работает верно

4 Выводы

В результате выполнения лабораторной работы были получены навыки работы с циклами и обработкой аргументов из командной строки. Были написаны программы, использующие все вышеописанные аспекты