Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки

Коровкин Никита Михайлович

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 создадим новую папку для 8-й лабораторной работы и первый файл.(рис.1)

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

Рис. 2.1: Создание папки и файла lab8-1.asm

Запустив Midnight commander, вставляем код из первого листинга.(рис.2)

```
%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
```

Рис. 2.2: Вставляем код из листинга

Код должен запускать цикл и выводить каждую итерацию число, которое будет на единицу меньше предыдущего. Чтобы он работал перенесем файл in_out.asm из папки предыдущей лабораторной работы(рис.3)

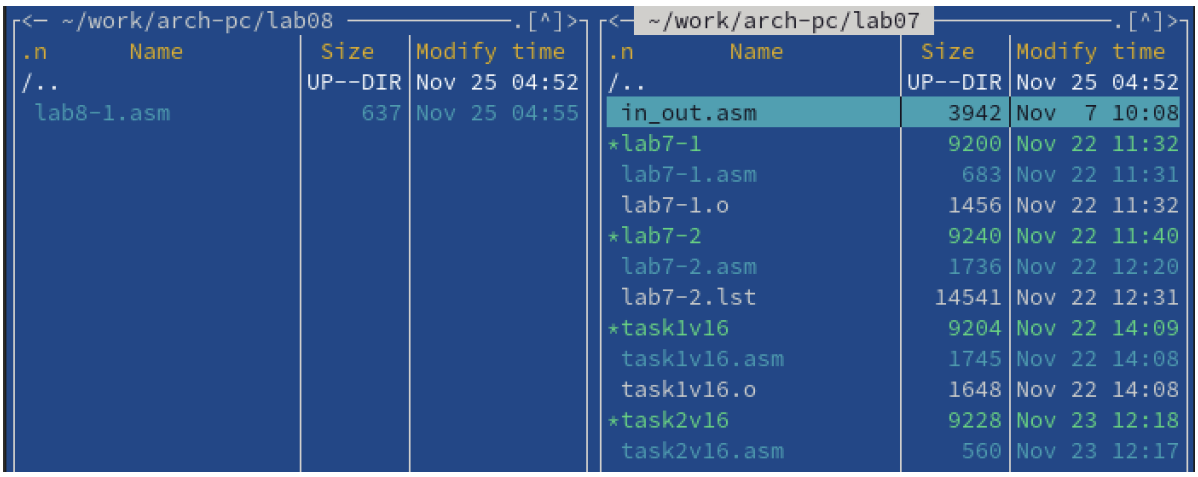

Рис. 2.3: Переносим in_out.asm

Запустим код и посмотрим результат(рис.4)

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

Рис. 2.4: Запуск кода

Видно, что программа выводит числа от N до единицы.

Теперь изменим код, чтобы в цикле отнималась единица у регистра ecx.(рис.5)

```
mov [N],eax
; ------ Организация цикла
mov ecx,[N] ; Счетчик цикла, `ecx=N`
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`
```

Рис. 2.5: Изменение кода программы

Соберем файл еще раз и при запуске тоже введем 4.(рис.6)

```
liveuser@localhost-live:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
liveuser@localhost-live:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
liveuser@localhost-live:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 4
```

Рис. 2.6: Повторный запуск программы

На выходе мы получаем бесконечный вывод, а значит количество проходов не совпадает с введенным значением.(рис.7)

```
4294949912
4294949910
4294949908
4294949906
4294949904
4294949902
4294949900
4294949898
4294949896
4294949894
4294949892
4294949890
4294949888
4294949886
4294949884
4294949882
4294949880
4294949878
4294949876
4294949874
429
```

Рис. 2.7: Бесконечный цикл на выходе

Тогда изменим программу так, чтобы она сохраняла значение регистра ecx в стек.(рис.8)

```
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'
```

Рис. 2.8: Редактируем код

Запустим код еще раз. На этот раз вывод правильный, значит код написан верно.(рис.9)

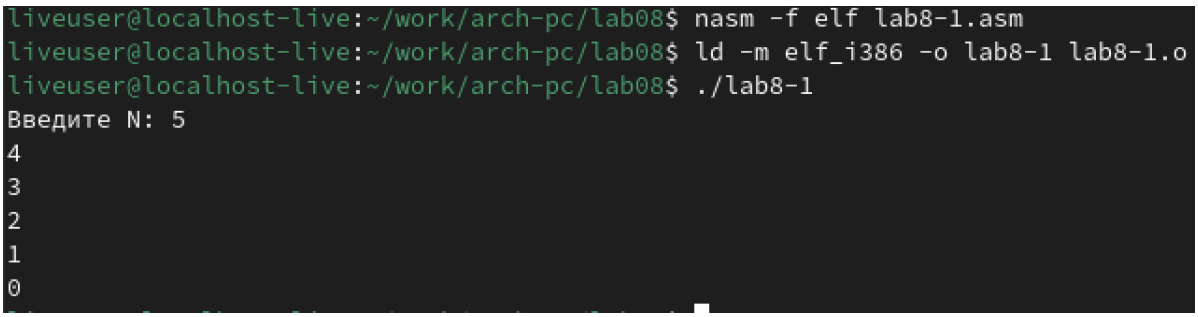

Рис. 2.9: Запуск и вывод программы

Теперь создадим второй файл.(рис.10)

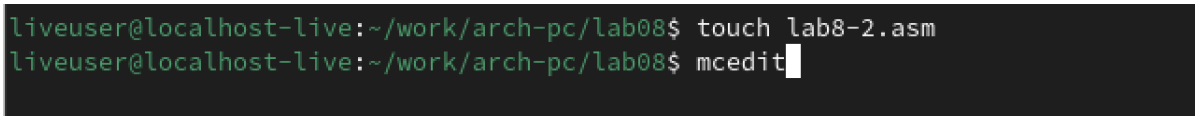

Рис. 2.10: Создание второго файла

Вставим туда код из второго листинга.(рис.11)

```
%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:
```

Рис. 2.11: Вставляем код в файл

Теперь запустим программу и дадим при запуске 3 аргумента. На выходе мы получим их же. Программа выводит их по порядку.(рис.12)

```
liveuser@localhost-live:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
liveuser@localhost-live:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
liveuser@localhost-live:~/work/arch-pc/lab08$ ./lab8-2
liveuser@localhost-live:~/work/arch-pc/lab08$ ./lab8-2 arg1 arg2 arg3
arg1
arg2
arg3
```

Рис. 2.12: Проверка работы кода

Теперь создадим 3-й файл.(рис.13)

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

Рис. 2.13: Создание третьего файла

Вставляем туда код из третьего листинга.(рис.14)

```
%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)
```

Рис. 2.14: Вставляем код в файл

Теперь снова соберем файл. При запуске введем несколько чисел, которые программа должна сложить.(рис.15)

```
liveuser@localhost-live:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
liveuser@localhost-live:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
liveuser@localhost-live:~/work/arch-pc/lab08$ ./lab8-3 3 2 5 6
Результат: 16
```

Рис. 2.15: Запуск третьего файла

Все вывелось верно. Теперь попробуем немного отредактировать код, чтобы числа перемножались.(рис.16)

```
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 ; добавляем к промежуточной сумме
; след. аргумент `esi=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
```

Рис. 2.16: Редактируем файл, чтобы числа перемножались

Теперь запустим файл еще раз. как мы видим, все числа перемножились и программа вывела верный ответ(рис.17)

```
liveuser@localhost-live:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
liveuser@localhost-live:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
liveuser@localhost-live:~/work/arch-pc/lab08$ ./lab8-3 3 2 5 6
Результат: 180
```

Рис. 2.17: Запуск отредактированного кода

#Выполнение самостоятельной работы

Для начала создадим файл, где буем писать код.(рис.18)

```
liveuser@localhost-live:~/work/arch-pc/lab08$ touch task1v16.asm
```

Рис. 2.18: Создание файла самостоятельной работы

Далее откроем файл.(рис.19)

```
task1v16.asm       [----] 12 L:[  1+ 0   1/ 30] *(12  / 372b) 0095 0x05F  [*][X]
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg2 db "Функция: f(x)=30x-11"
SECTION .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi, 0
next:
cmp ecx,0h
jz _end
pop eax
call atoi
mov ebx, 30
mul ebx
sub eax, 11
add esi,eax
loop next
```

Рис. 2.19: Открываем файл

Запишем код, который будет вычислять функцию из варианта с заданными аргументами 16 и затем сложит ответы при всех аргументах.(рис.20-21)

```
task1v16.asm       [----] 12 L:[  1+ 0   1/ 30] *(12  / 372b) 0095 0x05F  [*][X]
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg2 db "Функция: f(x)=30x-11"
SECTION .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi, 0
next:
cmp ecx,0h
jz _end
pop eax
call atoi
mov ebx, 30
mul ebx
sub eax, 11
add esi,eax
loop next
```

Рис. 2.20: Код для задания 16 ч.1

```
_end:
mov eax, msg2
call sprintLF
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
```

Рис. 2.21: Код для задания 16 ч.2

Опишем те строки кода, которые могут вызывать вопросы:

pop ecx - здесь из регистра достаем количество аргументов в стеке

esi отвечает за хранение промежуточных сумм.

cmp ecx необходимо для проверки, есть ли еще аргументы на входе.

Теперь запустим код и введем несколько разных аргументов для проверки.(рис.22)

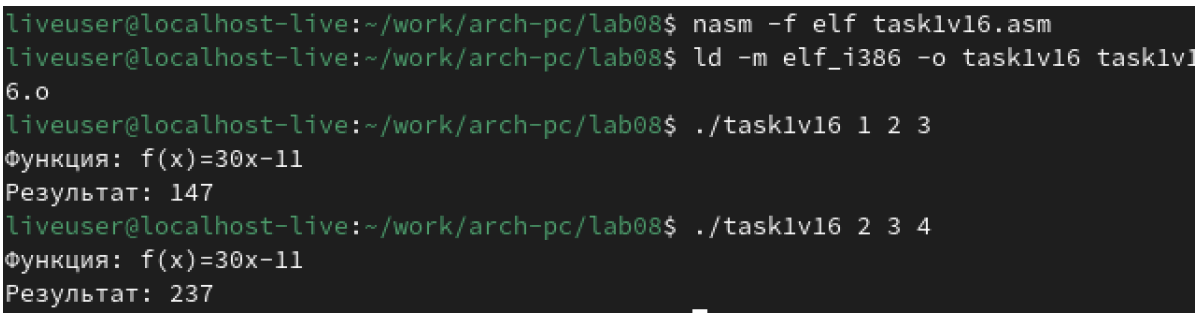

Рис. 2.22: Запуск кода для сложения значений функции

Код работает правильно, значит самостоятельная работа выполнена верно.

3 Выводы

В ходе выполнения данной лабораторной работы были получены знания о циклах на языке ассемблера и получены навыки о работе с ними.