Отчет по лабораторной работе №8
Номер студенческого билета: 1032256500

Колесников Михаил Александрович

Содержание

Список иллюстраций

Список таблиц

#Цель работы

Познакомиться с таким понятием как циклы в ассемблере. Узнать как работает стек и как извлекать из него элементы, а также записывать, а после использовать их в решении задач

1 Выполнение лабораторной работы

Создадим папку **lab08** в той директории, в которой работале ранее

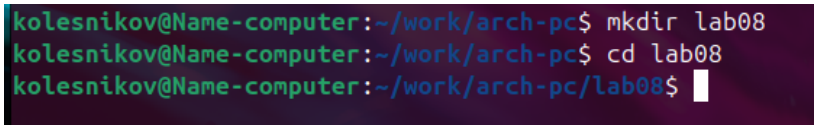

Рисунок 1.1: Создание папки и переход в нее

Создаем в этой директории файл **lab8-1.asm**

Рисунок 1.2: Создание файла

Открываем файла **lab8-1.asm** в текстовом редакторе с помощью команды **nano**

Рисунок 1.3: Открываем файл в текстовом редакторе

Записываем в него код, который нам дали в лабораторной работе

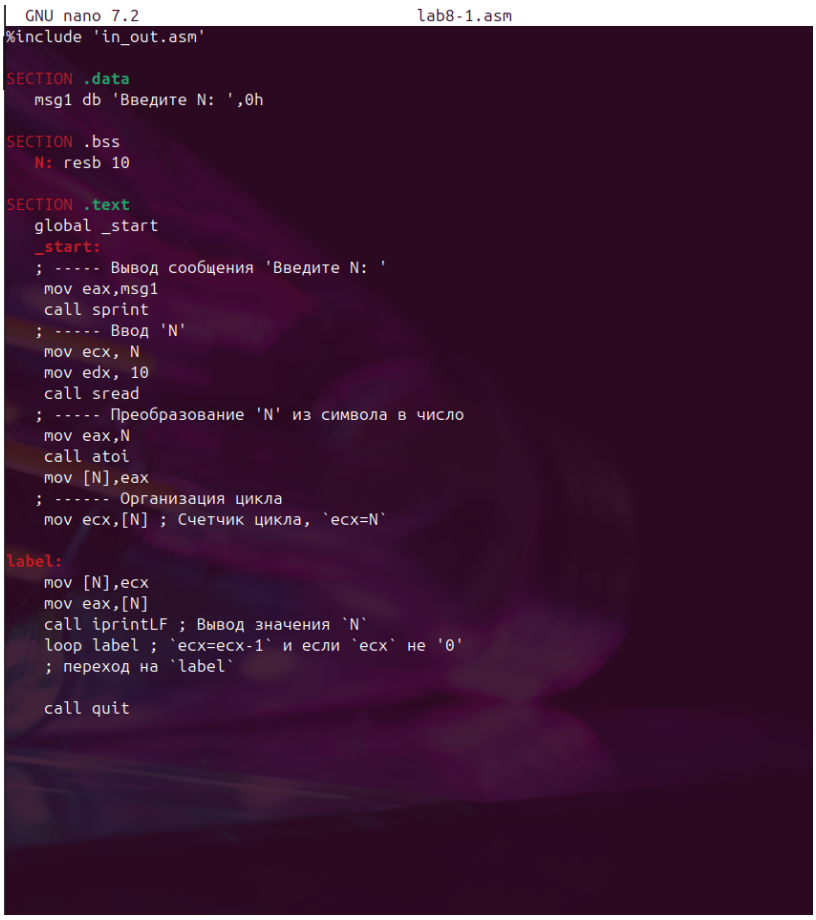

Рисунок 1.4: Запись кода из лабораторной работы

Компилируем файл и запускаем

Рисунок 1.5: Компилируем и запускаем файл

Редактируем код, заменяем метку **label** на код, который нам дали

```
 GNU nano 7.2                          lab8-1.asm *
%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:
    sub ecx,1 ; `ecx=ecx-1`
    mov [N],ecx
    mov eax,[N]
    call iprintLF
    loop label

    call quit
```

Рисунок 1.6: Редкатируем код

Компилируем и запускаем отредактированный код

```
kolesnikov@Name-computer:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
kolesnikov@Name-computer:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
kolesnikov@Name-computer:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 10
9
7
5
3
1
kolesnikov@Name-computer:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 20
19
17
15
13
11
9
7
5
3
1
```

Рисунок 1.7: Компиляция и запуск отредактированного кода

Мы можем увидеть, что первое число игнорируется, поэтому начинается счет с числа на единицу меньше

Редактируем код еще раз, заменяем метку **label**

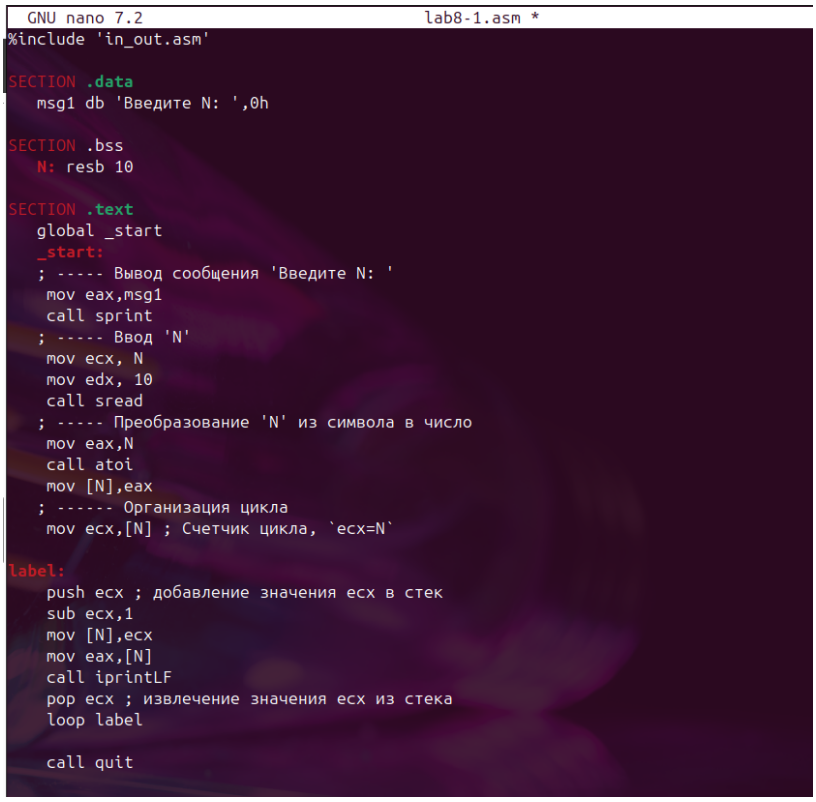

Рисунок 1.8: Еще раз редктируем код

Компилируем и запускаем отредактированный еще раз файл

Рисунок 1.9: Компилруем и запускаем отредактированный файл

Мы можем заметить,что теперь счет идет не до единицы, а до 0 и это соответствует числу чиклов

Создаем файл **lab8-2.asm**

Рисунок 1.10: Создание файла lab8-2.asm

Записываем в файл код, который нам дали в лабораторной работе

```
 GNU nano 7.2                                  lab8-2.asm *
;-------------------------------------------------------------
; Обработка аргументов командной строки
;-------------------------------------------------------------

%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
```

Рисунок 1.11: Запись кода в файл из лабораторной работы

Компилируем и запускаем файл **lab8-2.asm**

```
kolesnikov@Name-computer:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
kolesnikov@Name-computer:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
kolesnikov@Name-computer:~/work/arch-pc/lab08$ ./lab8-2
kolesnikov@Name-computer:~/work/arch-pc/lab08$ ./lab8-2
kolesnikov@Name-computer:~/work/arch-pc/lab08$ ./lab8-2 1 2 3
1
2
3
kolesnikov@Name-computer:~/work/arch-pc/lab08$ ./lab8-2 '1' '2' '3'
1
2
3
kolesnikov@Name-computer:~/work/arch-pc/lab08$ ./lab8-2 "Hello"
Hello
```

Рисунок 1.12: Компилируем и запускаем файл

Мы можем заметить,что все аргументы были аброботаны программой

Создаем файл **lab8-3.asm**

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

Рисунок 1.13: Создаем файл lab8-3.asm

Записываем в файл код, который нам дали в лабораторной работе

Рисунок 1.14: Записываем в файл lab8-3.asm код из лабораторной работы

Компиилруем и запускаем исполняемый файл **lab8-3**

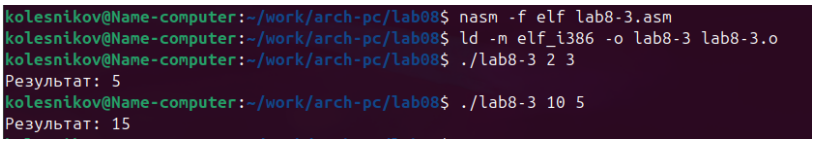

Рисунок 1.15: Компилируем и запускаем файл

Редактируем файл **lab8-3.asm** так, чтобы он выводил нам произведение аргументов, которые мы передаем

```
 GNU nano 7.2                                lab8-3.asm
%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, 1        ; Используем `esi` для хранения
                     ; промежуточных произведений(ставим 1, чтобы не получался 0)
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
   imul esi,eax      ; умножаем промежуточное произведение на аргумент
   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 iprintLF     ; печать результата
   call quit         ; завершение программы
```

Рисунок 1.16: Редкатируем файл lab8-3.asm

Компилруем и запускаем отредактированный файл

```
kolesnikov@Name-computer:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
kolesnikov@Name-computer:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
kolesnikov@Name-computer:~/work/arch-pc/lab08$ ./lab8-3
Результат: 1
kolesnikov@Name-computer:~/work/arch-pc/lab08$ ./lab8-3 2 3
Результат: 6
kolesnikov@Name-computer:~/work/arch-pc/lab08$ ./lab8-3 10 5
Результат: 50
```

Рисунок 1.17: Компилируем и запускаем отредактированный файл

Задание для самостоятельной работы

Создаем файл **task.asm**

```
kolesnikov@Name-computer:~/work/arch-pc/lab08$ touch task.asm
kolesnikov@Name-computer:~/work/arch-pc/lab08$
```

Рисунок 1.18: Создаем файл

Записываем в него код,чтобы выполнить задание для первого варианта

```
 GNU nano 7.2                                  task.asm
%include 'in_out.asm'

SECTION .data
   func_msg db "Функция: f(x)=2x+15", 10, 0
   msg db "Результат: ",0

SECTION .text
global _start

_start:

   pop ecx            ; Извлекаем количество аргументов
   pop edx            ; Извлекаем имя программы
   sub ecx,1          ; Уменьшаем на 1 (без имени программы)
   mov esi, 0         ; Начальное значение суммы

   ; Выводим сообщение о функции
   mov eax, func_msg
   call sprint

next:
   cmp ecx,0h
   jz _end
   pop eax
   call atoi
   shl eax, 1         ; 2x
   add eax, 15        ; 2x+15
   add esi, eax       ; Суммируем f(x)
   loop next

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

Рисунок 1.19: Пишем код для выполнения задания

Компилируем и запускаем исполняемый файл **task**

```
kolesnikov@Name-computer:~/work/arch-pc/lab08$  nasm -f elf task.asm
kolesnikov@Name-computer:~/work/arch-pc/lab08$ ld -m elf_i386 -o task task.o
kolesnikov@Name-computer:~/work/arch-pc/lab08$ ./task 1 2 3 4
Функция: f(x)=2x+15
Результат: 80
kolesnikov@Name-computer:~/work/arch-pc/lab08$ ./task 1
Функция: f(x)=2x+15
Результат: 17
```

Рисунок 1.20: Компилируем и запускаем файл

2 Выводы

Мы познакомились с конструкциями циклов и стека и научились использовать эти технологии для выполнения различных задач