Лабораторная работа №8

НКАбд-02-23

Выборнов Дмитрий Валерьевич

1 Цель работы

Приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки NASM.

2 Задание

1. Реализация циклов в NASM.
2. Обработка аргументов командной строки.
3. Задание для самостоятельной работы.

3 Теоретическое введение

Стек — это структура данных, организованная по принципу LIFO («Last In — First Out» или «последним пришёл — первым ушёл»). Стек является частью архитектуры процессора и реализован на аппаратном уровне. Для работы со стеком в процессоре есть специальные регистры (ss, bp, sp) и команды. Основной функцией стека является функция сохранения адресов возврата и передачи аргументов при вызове процедур. Кроме того, в нём выделяется память для локальных переменных и могут временно храниться значения регистров. Стек имеет вершину, адрес последнего добавленного элемента, который хранится в регистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. Команда push размещает значение в стеке, т.е. помещает значение в ячейку памяти, на которую указывает регистр esp, после этого значение регистра esp увеличивается на 4. Данная команда имеет один операнд — значение, которое необходимо поместить в стек. Команда pop извлекает значение из стека, т.е. извлекает значение из ячейки памяти, на которую указывает регистр esp, после этого уменьшает значение регистра esp на 4. У этой команды также один операнд, который может быть регистром или переменной в памяти. Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре ecx. Наиболее простой является инструкция loop. Она 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.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM.

Создаю каталоги и файлы для новой лабораторной работы.

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

Рис. 4.1: Первый шаг.

Ввожу в файл текст первой программы.

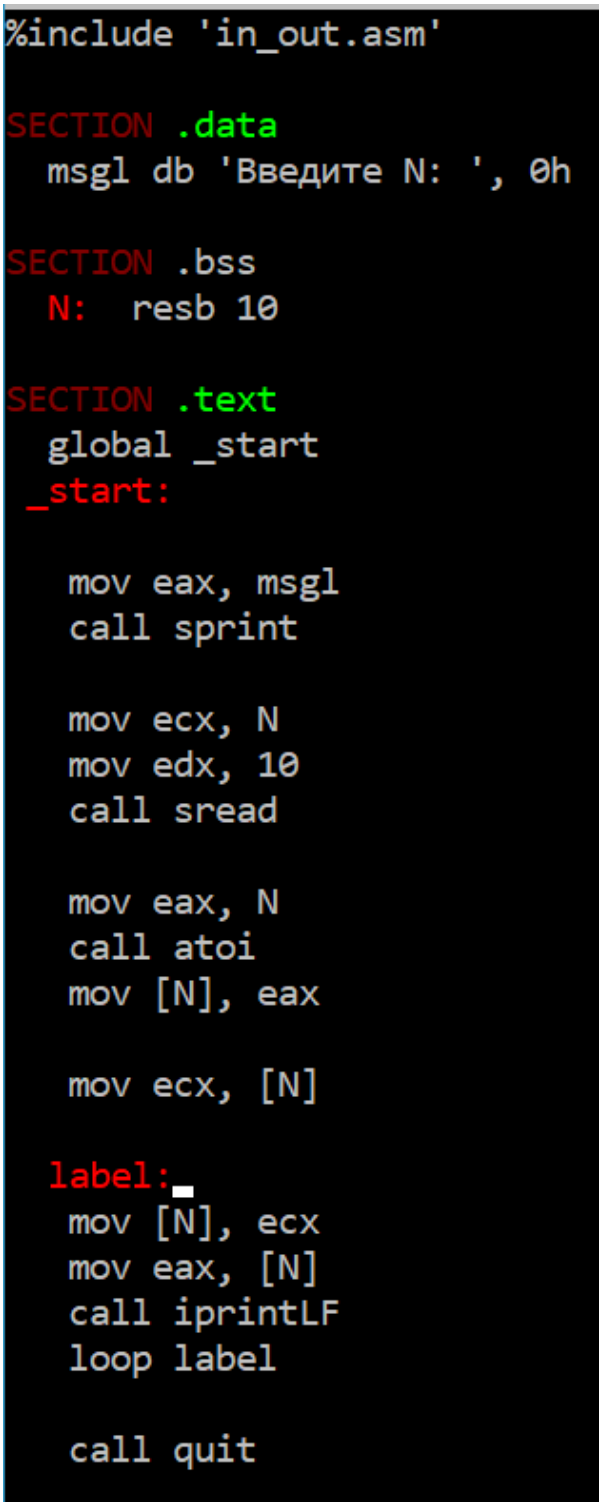

Рис. 4.2: Второй шаг.

Создаю исполняемый файл и запускаю его.

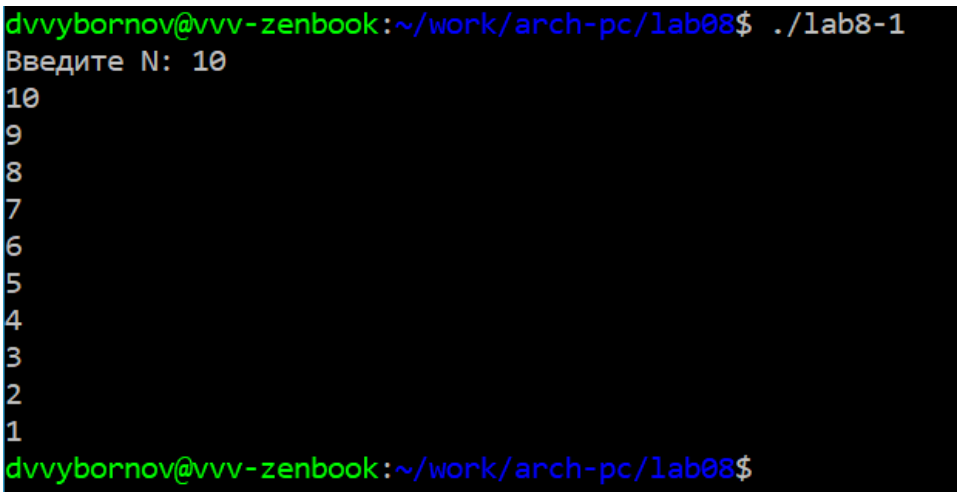

Рис. 4.3: Третий шаг.

Добавляю в программу изменение значения регистра ecx в цикле.

```
%include 'in_out.asm'

SECTION .data
  msg1 db 'Введите N: ', 0h

SECTION .bss
  N:  resb 10

SECTION .text
  global _start
 _start:

   mov eax, msg1
   call sprint

   mov ecx, N
   mov edx, 10
   call sread

   mov eax, N
   call atoi
   mov [N], eax

   mov ecx, [N]

  label:

   sub ecx, 1
   mov [N], ecx
   mov eax, [N]
   call iprintLF

   loop label

   call quit
```

Рис. 4.4: Четвёртый шаг.

Запускаю изменённую программу.

Рис. 4.5: Пятый шаг.

В данном случае число проходов цикла не соответствует значению N, так как за один проход значение регистра ecx уменьшается на 2.

Добавляю в текст программы комманды push и pop.

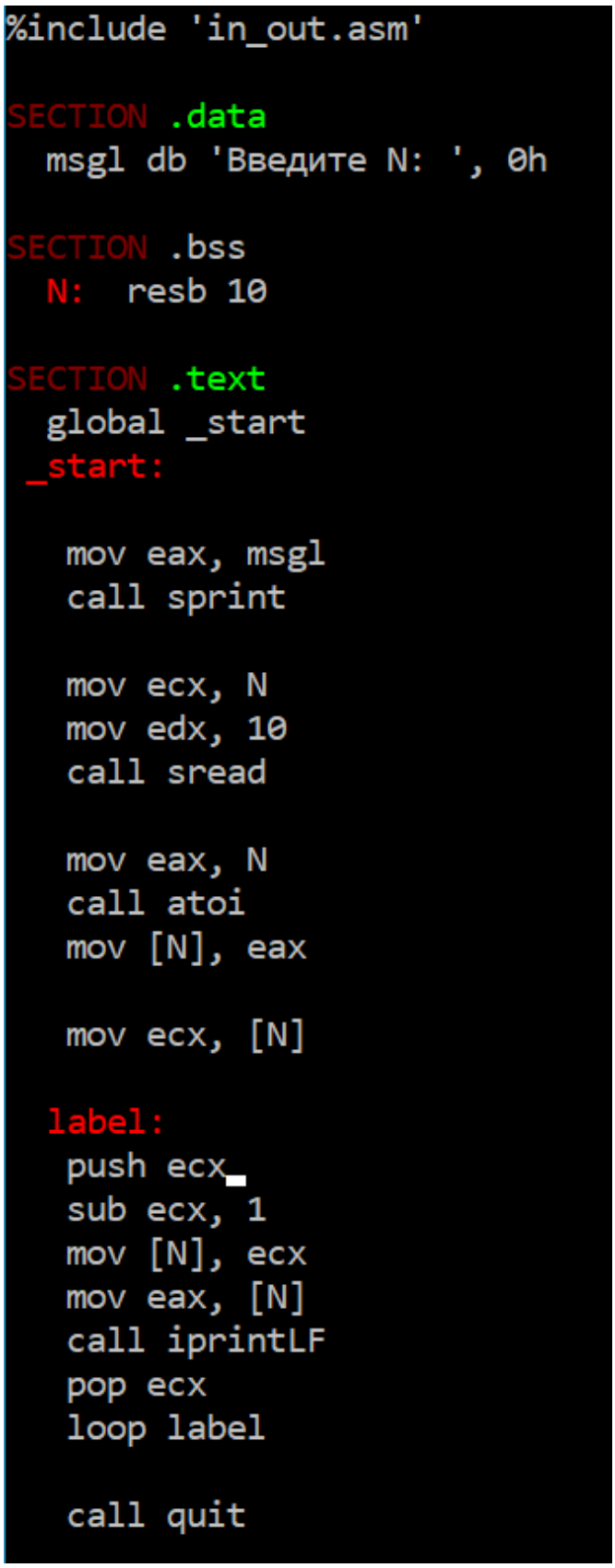

Рис. 4.6: Шестой шаг.

Зарускаю изменённую программу.

Рис. 4.7: Седьмой шаг.

Число проходов соответствует значению N, но теперь вместо “10” выводится “0”.

4.2 Реализация циклов в NASM.

Создаю новый файл и ввожу в него текст второй программы.

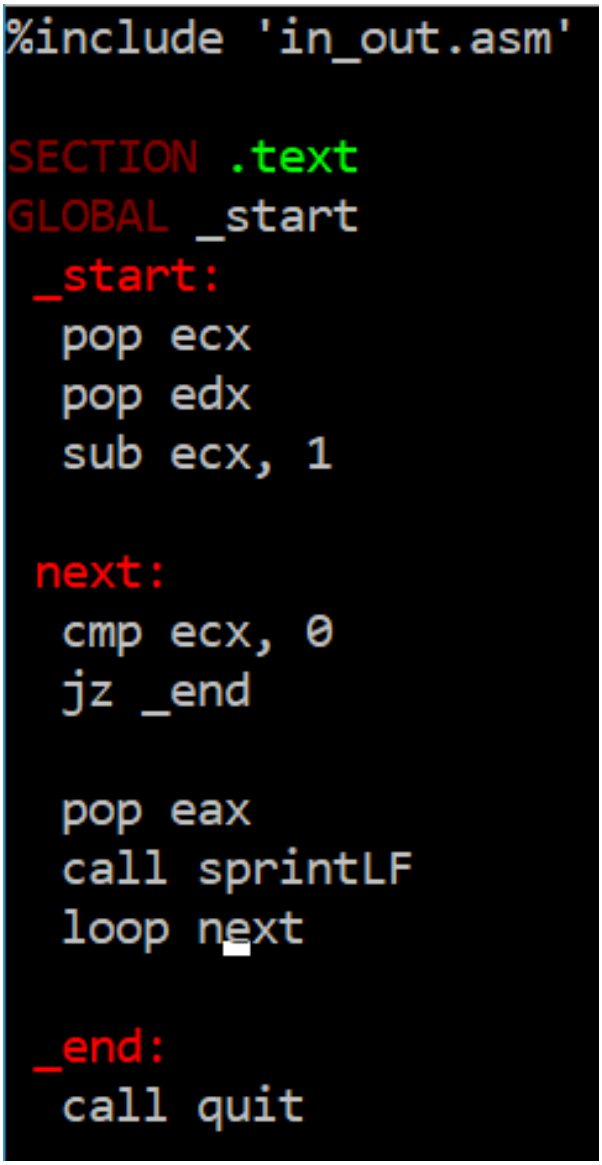

Рис. 4.8: Восьмой шаг.

Запускаю программу с указанными аргументами.

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
```

Рис. 4.9: Девятыйй шаг.

Программа обработала 4 аргумента, так как ‘аргумент’ и ‘2’ разделились.

Создаю новый файл и ввожу в него текст третьей программы.

```
%include   'in_out.asm'

SECTION .data
 msg db "Результат: ", 0
SECTION .text
GLOBAL _start
 _start:

  pop ecx
  pop edx
  sub ecx, 1
  mov esi, 1

 next:
  cmp ecx, 0h
  jz _end

  pop eax
  call atoi
  add esi, eax

  loop next

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

Рис. 4.10: Десятый шаг.

Создаю машинный файл и запускаю его.

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab08$ ./lab8-3 13 7 12 5 10
Результат: 47
dvvybornov@vvv-zenbook:~/work/arch-pc/lab08$
```

Рис. 4.11: Одиннадцатый шаг.

Изменяю программу так, чтобы она выводила произведение аргументов.

```
%include   'in_out.asm'

SECTION .data
 msg db "Результат: ", 0
SECTION .text
GLOBAL _start
 _start:

  pop ecx
  pop edx
  sub ecx, 1
  mov esi, 1

 next:
  cmp ecx, 0h
  jz _end

  pop eax
  call atoi
  mov edx, eax
  mov eax, esi
  mul edx
  mov esi, eax
  loop next

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

Рис. 4.12: Двенадцатый шаг.

Создаю исполняемый файл и запускаю его.

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab08$ ./lab8-3-2 5 4 2 10
Результат: 400
```

Рис. 4.13: Тринадцатый шаг.

4.3 Задание для самостоятельной работы.

Создаю программу, которая находит сумму значений функции 5(2 + x).

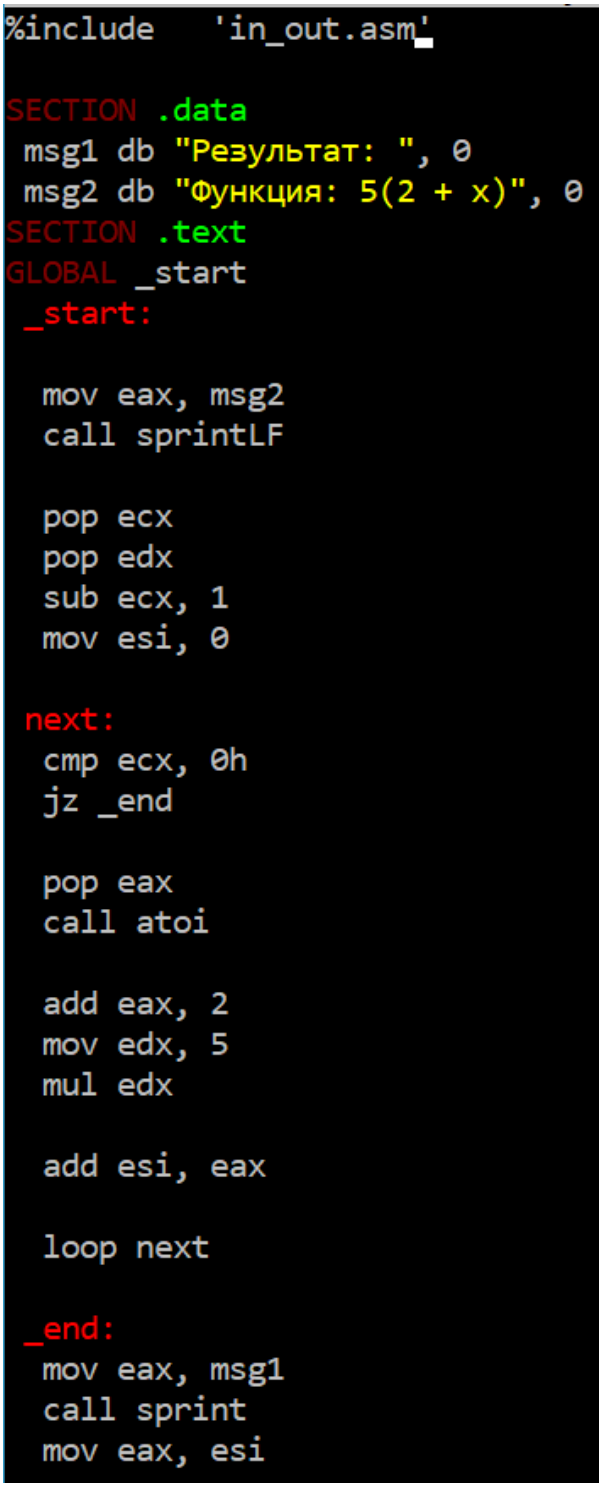

Рис. 4.14: Первый шаг самостоятельного загания.

Проверяю работу программы с несколькими наборами аргументов.

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab08$ nasm -f elf lab8-4.asm
dvvybornov@vvv-zenbook:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-4 lab8-4.o
dvvybornov@vvv-zenbook:~/work/arch-pc/lab08$ ./lab8-4 8 18 28
Функция: 5(2 + x)
Результат: 300
dvvybornov@vvv-zenbook:~/work/arch-pc/lab08$ ./lab8-4 4 10 61
Функция: 5(2 + x)
Результат: 405
dvvybornov@vvv-zenbook:~/work/arch-pc/lab08$ ./lab8-4 10 26 42 70 101
Функция: 5(2 + x)
Результат: 1295
dvvybornov@vvv-zenbook:~/work/arch-pc/lab08$
```

Рис. 4.15: Второй шаг самостоятельного задания.

5 Выводы

Завершив эту лабораторную работу, я приобрёл навыки написания программ с использованием циклов и обработкой аргументов командной строки NASM.