Отчёт по лабораторной работе №8

Дисциплина: архитектура компьютеров

Симонова Виктория Игоревна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Задание

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

3 Теоретическое введение

3.1 Организация стека

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. На рис. 8.1 показана схема организации стека в процессоре. 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. При помещении значения в стек указатель стека уменьшается, а при извлечении — увеличивается. Для стека существует две основные операции: • добавление элемента в вершину стека (push); • извлечение элемента из вершины стека (pop).

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Создаю каталог lab08 для файлов лабораторной работы и файл lab8-1.asm (рис. [4.1]).

```
[visimonova@fedora ~]$ mkdir ~/work/arch-pc/lab08
[visimonova@fedora ~]$ cd ~/work/arch-pc/lab08
[visimonova@fedora lab08]$ touch lab8-1.asm
```

Рис. 4.1: Создание каталога и файла

Ввожу текст прграммы из листинга в файл (рис. [4.2]).

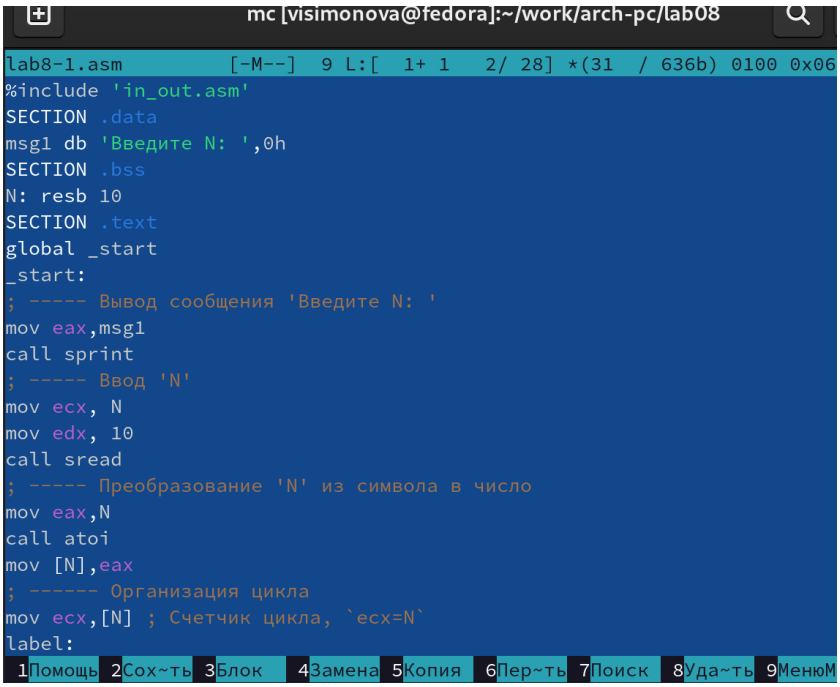

Рис. 4.2: Программа для вывода значения регистра

Создаю исполняемый файл и проверяю его работу. (рис. [4.3]).

Рис. 4.3: Запускаю исполняемый файл

Программа по порядку выводит введённое с клавиатуры значение N, которое каждый раз уменьшается на 1(инструкция loop), число проходов цикла соответсятвует значению N.

Корректирую текст программы (рис. [4.4]).

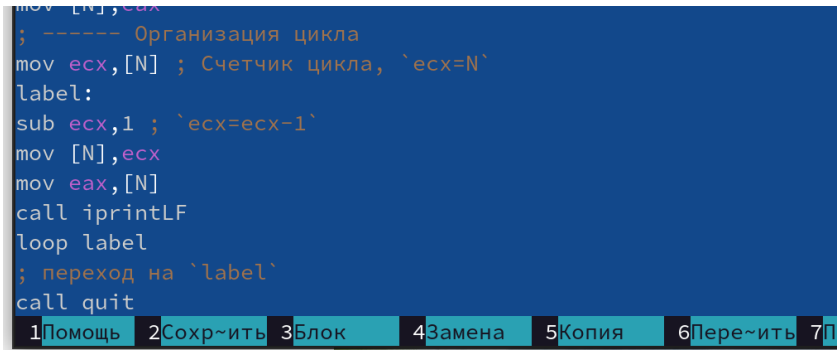

Рис. 4.4: Корректирую текст программы

Создаю исполняемый файл и проверяю его работу. (рис. [4.5]).

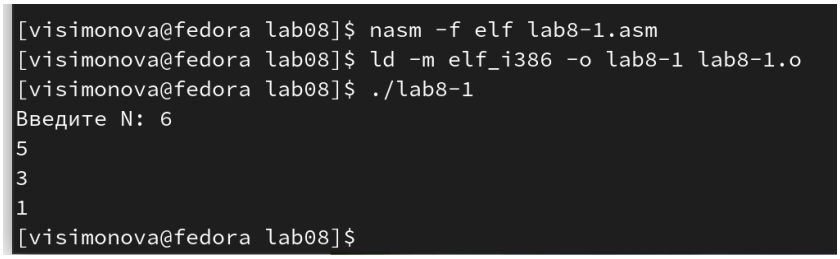

Рис. 4.5: Запускаю исполняемый файл

Число проходов цикла не соответствует введённому значению N (N принимает значения через 1, тк в теле цикла добавлено изменение знаяения регистра ecx).

Изсменяю программу, добавляя команду push(испозуем стек) (рис. [4.6]).

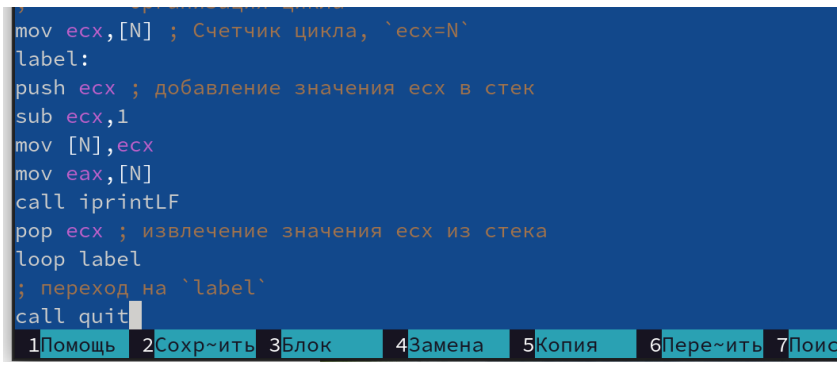

Рис. 4.6: Изменяю файл второй раз

Создаю исполняемый файл и проверяю его работу. (рис. [4.7]).

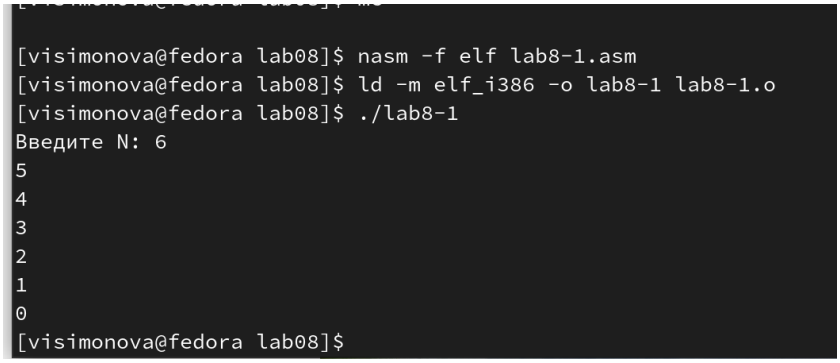

Рис. 4.7: Запускаю исполняемый файл

Число проходов цикла соответствует введённому значению N и выводит числа от N-1 до 0 включительно.

4.2 Обработка аргументов командной строки

Создаю файл lab8-2.asm (рис. [4.8]).

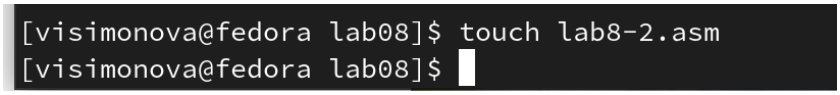

Рис. 4.8: Создаю файл

Ввожу в файл текст программы, которая выводит на экран аргументы командной строки (рис. [4.9]).

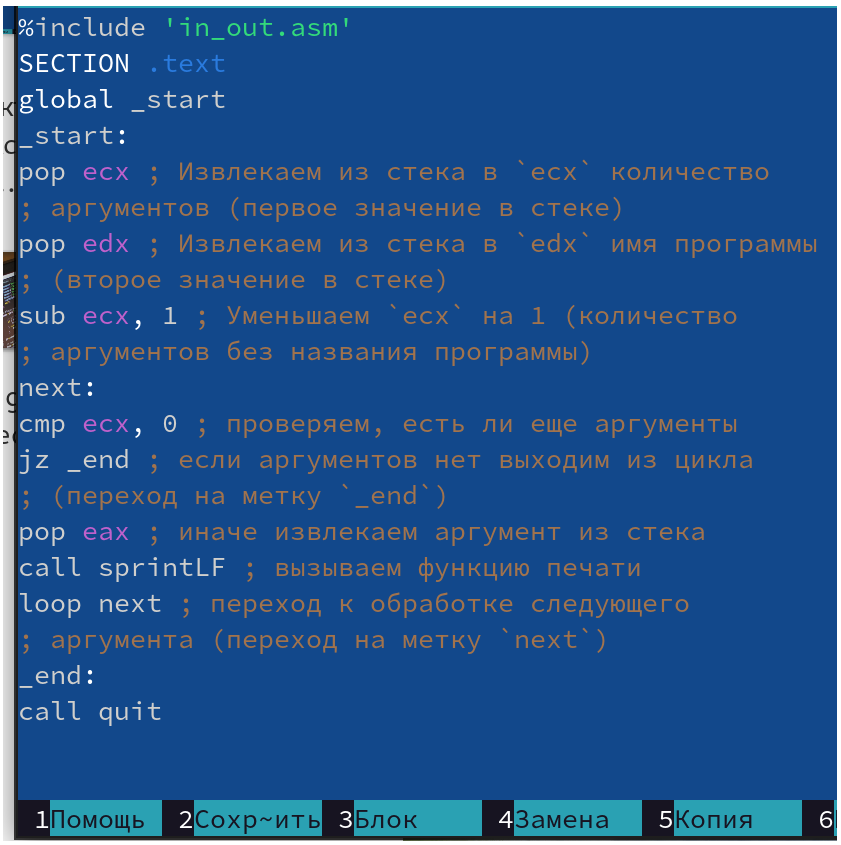

Рис. 4.9: Ввожу код

Создаю исполняемый файл и проверяю его работу. (рис. [4.10]).

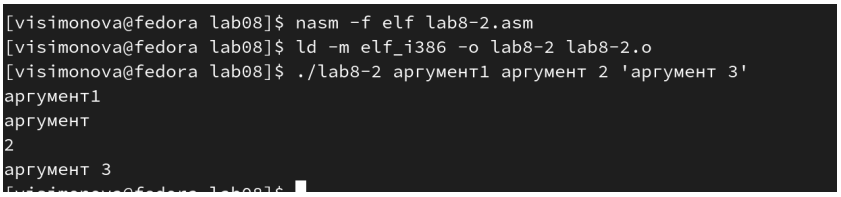

Рис. 4.10: Запускаю исполняемый файл

Программа обработала 4 аргумента, т.к. цифра “2” не была взята в кавычки вместе со словом “аргумент”, в отличие от ‘аргумент 3’, который программа восприняла как один аргумент.

Создаю файл lab8-3.asm (рис. [4.11]).

```
3
[visimonova@fedora lab08]$ touch lab8-3.asm
[visimonova@fedora lab08]$
```

Рис. 4.11: Создаю файл

Ввожу в файл текст программы, которая выводит на экран сумму чисел, которые введены как аргументы командной строки (рис. [4.12]).

```
%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
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
 1Помощь  2Сохр~ить  3Блок  4Замена  5Копия  6Пере~
```

Рис. 4.12: Ввожу код

Создаю исполняемый файл и проверяю его работу. (рис. [4.13]).

```
[visimonova@fedora lab08]$ nasm -f elf lab8-3.asm
[visimonova@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[visimonova@fedora lab08]$ ./lab8-3
Результат: 0
[visimonova@fedora lab08]$ ./lab8-3 12 13 7 10 5
Результат: 47
[visimonova@fedora lab08]$
```

Рис. 4.13: Запускаю исполняемый файл

Изменяю программу так, чтобы она вычислла произведеник аргументов,введённых из командной строки (рис. [4.14]).

```
lab8-3.asm          [----] 11 L:[  3+27  30/ 30] *(1396/1436b) 0032 0x020  [*][X]
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 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 ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
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
```

Рис. 4.14: Изменяю код

Создаю исполняемый файл и проверяю его работу. (рис. [4.15]).

```
[visimonova@fedora lab08]$ nasm -f elf lab8-3.asm
[visimonova@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[visimonova@fedora lab08]$ ./lab8-3 12 13 7 10 5
Результат: 54600
[visimonova@fedora lab08]$
```

Рис. 4.15: Запускаю исполняемый файл

4.3 Задание для самостоятельной работы

Пишу текст программы для вычисления суммы значений функции f(x)=12*x-7 в соответствии с моим номером варианта (13) для x = x1, x2, ..., xn. Значения xi передаются как аргументы (рис. [4.16]).

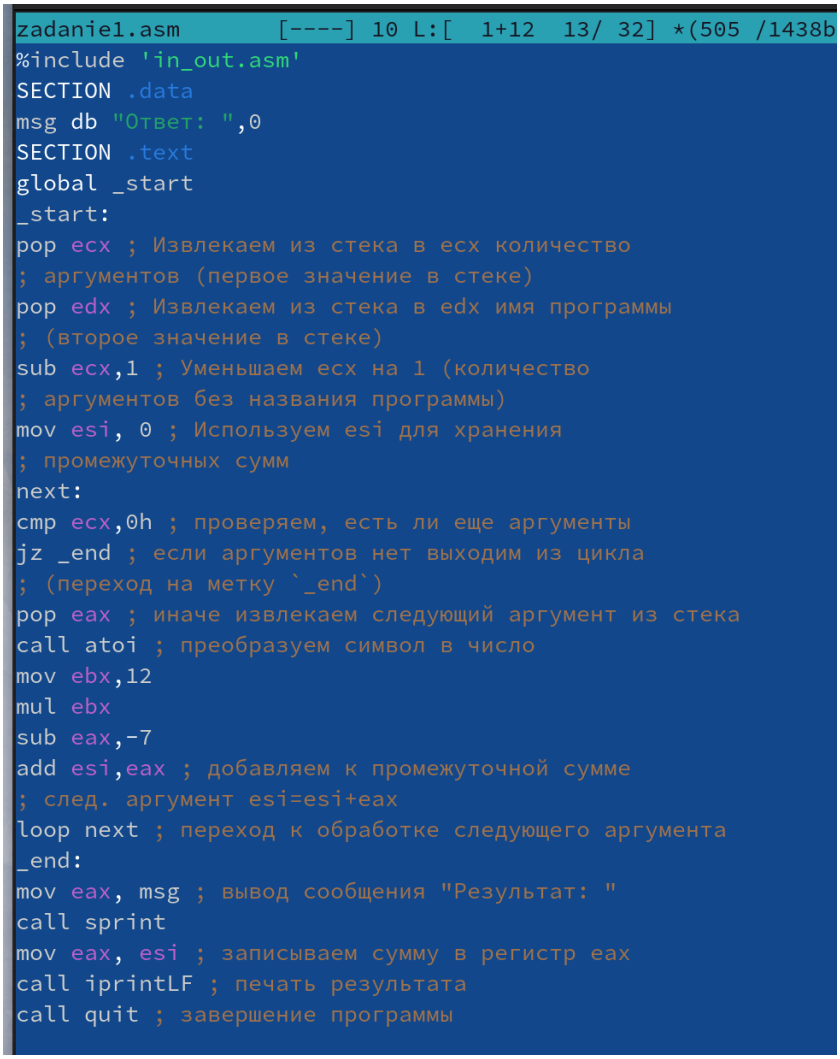

Рис. 4.16: Текст программы

Текст программы:

```
%include 'in_out.asm'
SECTION .data
```

```
msg db "Ответ: ",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
mov ebx,12
mul ebx
sub eax,-7
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент esi=esi+eax
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
```

Создаю исполняемый файл и проверяю его работу для нескольких наборов xi. (рис. [4.17]).

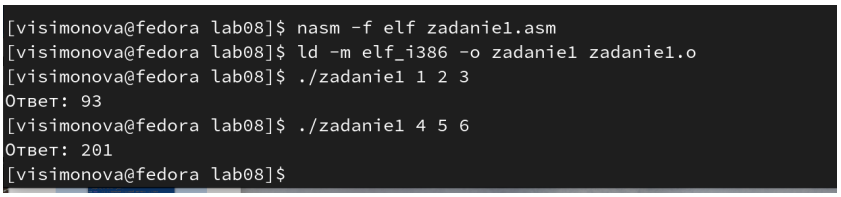

Рис. 4.17: Запускаю исполняемый файл

5 Выводы

Приобрела навыки написания программ с использованием циклов и обработкой аргументов командной строки.

Список литературы

Лабораторная работа №8