Отчёт по лабораторной работе №8

Дисциплина: Архитектура Компьютера

Курилко-Рюмин Е.М

Содержание

Список иллюстраций

1 Цель работы

Целью данной работы является приобретение практического опыта в написании программ с использованием циклов и обработкой аргументов командной строки.

2 Задание

1. Общее ознакомление с циклами и обработкой аргументов командной строки.
2. Реализация циклов в NASM.
3. Обработка аргументов командной строки.
4. Выполнение заданий 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. 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). 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. Существует ещё две команды для добавления значений в стек. Это команда pusha, которая помещает в стек содержимое всех регистров общего назначения в следующем порядке: ax, cx, dx, bx, sp, bp, si, di. А также команда pushf, которая служит для перемещения в стек содержимого регистра флагов. Обе эти команды не имеют операндов. 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. Нужно помнить, что извлечённый из стека элемент не стирается из памяти и остаётся как “мусор”, который будет перезаписан при записи нового значения в стек. Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре ecx. Наиболее простой является инструкция loop. Она позволяет организовать безусловный цикл. Иструкция loop 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.

С помощью утилиты mkdir создаю директорию lab8 для выполнения соответствующей лабораторной работы. Перехожу в созданный каталог с помощью утилиты cd. С помощью touch создаю файл lab8-1.asm. (рис.1).

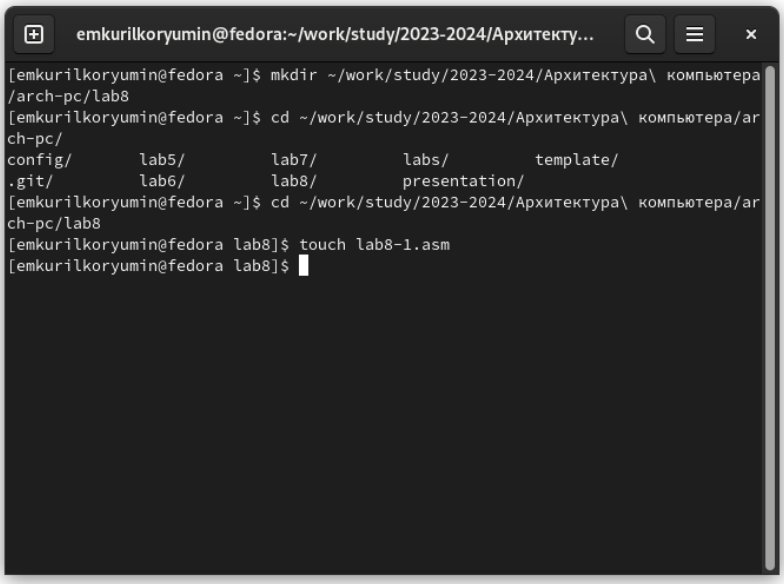

Рис. 4.1: Работа с директориями и создание файла

Открываю созданный файл lab8-1.asm, вставляю в него следующую программу: (рис.2).

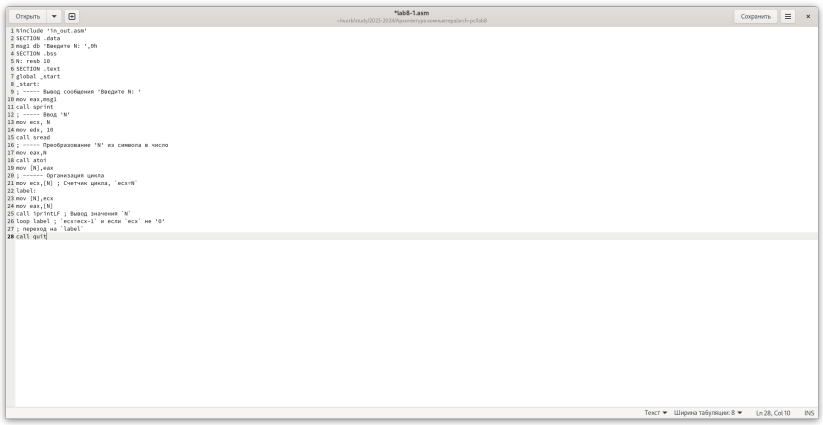

Рис. 4.2: Редактирование файла

Копирую в текущий каталог файл in_out.asm с помощью утилиты cp, ибо он будет использоваться в дальнейшем. Создаю исполняемый файл и запускаю его. Мы видим, что использование инструкции loop позволяет выводить значения регистра ecx циклично.(рис.3).

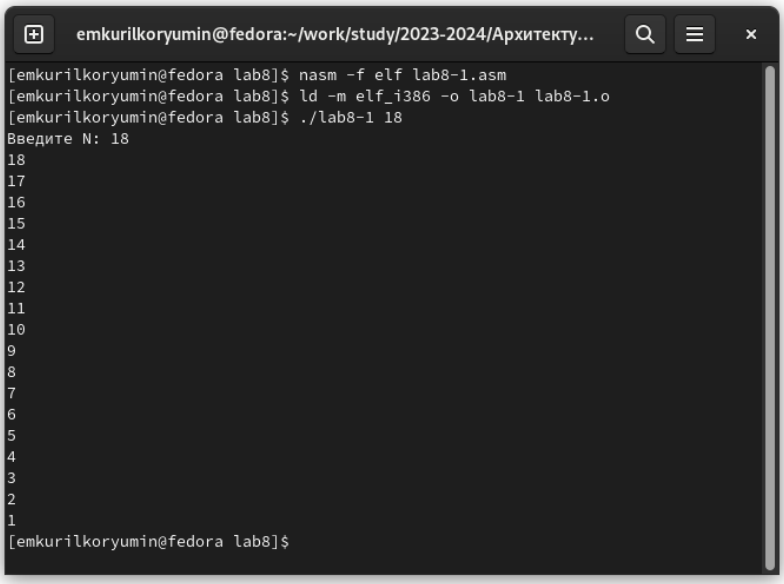

Рис. 4.3: Копирование, подготовка и исполнение файла

Изменяю значение ecx в цикле. (рис.4).

Рис. 4.4: Редактирование файла

Создаю новый исполняемый файл программы и запускаю его. Мы видим, что регистр ecx в цикле принимает совершенно разные значения. И число проходов цикла далеко не соответствует ли значению ⊠, введенному с клавиатуры. (рис.5).

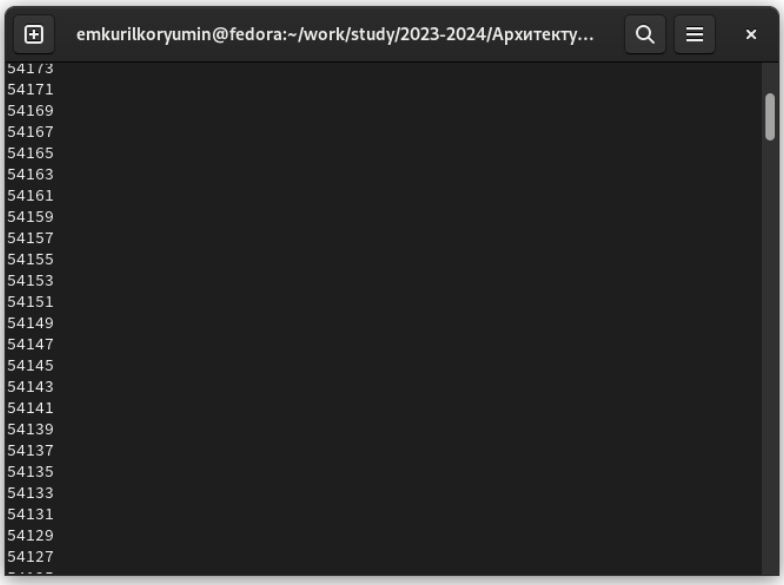

Рис. 4.5: Создание и запуск исполняемого файла

Вношу изменения в текст программы, добавив команды push, pop для сохранения значения счётчика цикла loop. (рис.6).

Рис. 4.6: Создание, редактирование файла

Выполняю компиляцию и компоновку, и запускаю исполняемый файл. В данном случае число проходов цикла соответствует значению ⊠ введенному с клавиатуры. Счёт идёт, не от 8-ми, а от 7-ми, но включается 0 (рис.7).

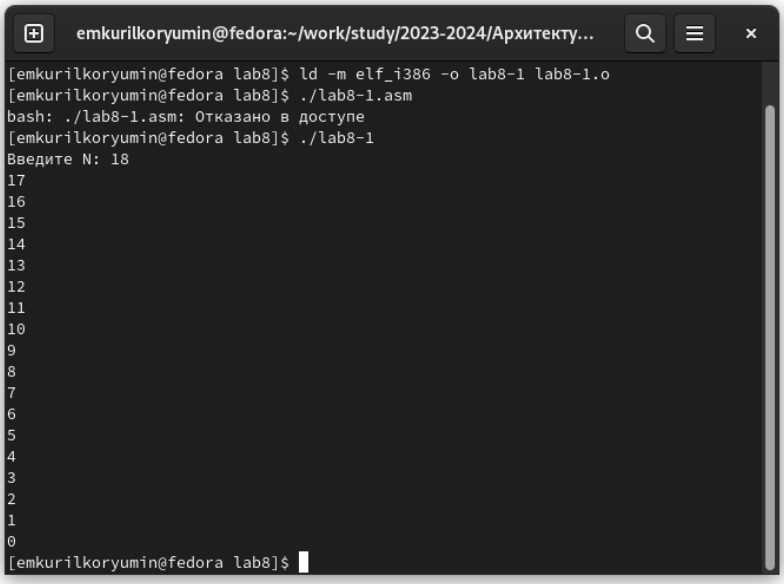

Рис. 4.7: Создание и запуск исполняемого файла

4.2) Обработка аргументов командной строки.

Создаю файл lab8-2.asm. Редактирую его, вводя предлагаемую программу.

(рис.8).

Рис. 4.8: Редактирование файла

Создаю исполняемый файл после редактирования. (рис.9).

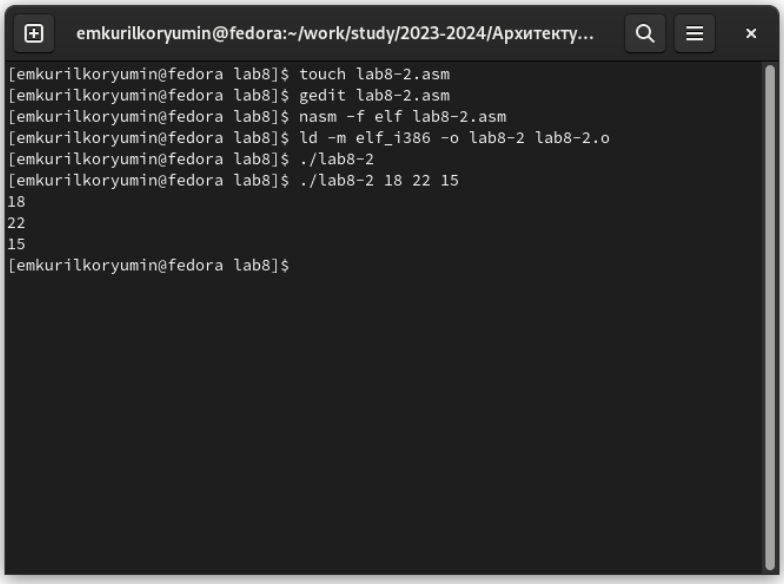

Рис. 4.9: Создание файла, открытие его в режиме правки, компиляция и обработка исполняемого файла

Запускаю исполняемый файл. Программой было обработано 3 аргумента - ровно те, которые я указал при запуске. (рис.10).

Рис. 4.10: Запуск исполняемого файла

Создаю файл lab8-3.asm. Ввожу в него следующую программу: (рис.11).

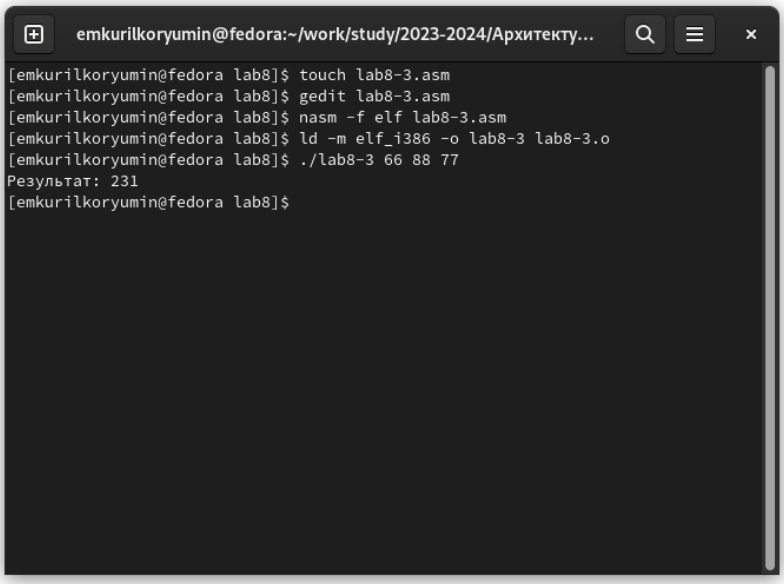

Рис. 4.11: Редактирование файла

Создаю исполняемый файл. (рис.12).

Рис. 4.12: Создание файла, открытие его в режиме правки, компиляция и обработка исполняемого файла

Указываю нужные аргументы. Выполняя устную проверку, убеждаемся в правильности работы программы. (рис.13).

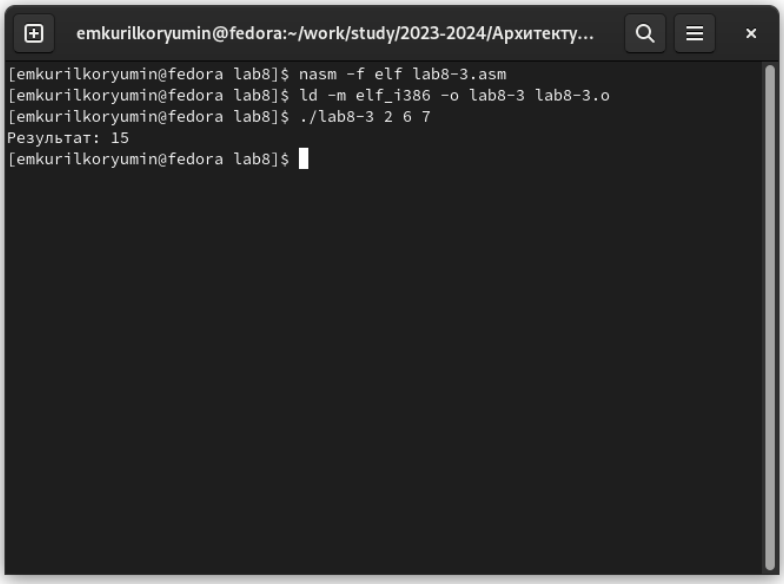

Рис. 4.13: Открытие листинга

Изменяю текст программы для вычисления произведения аргументов командной строки. (рис.14).

Рис. 4.14: Редактирование файла

Создаю и запускаю исполняемый файл. При проверке видим, что выводятся верные значения. (рис.15).

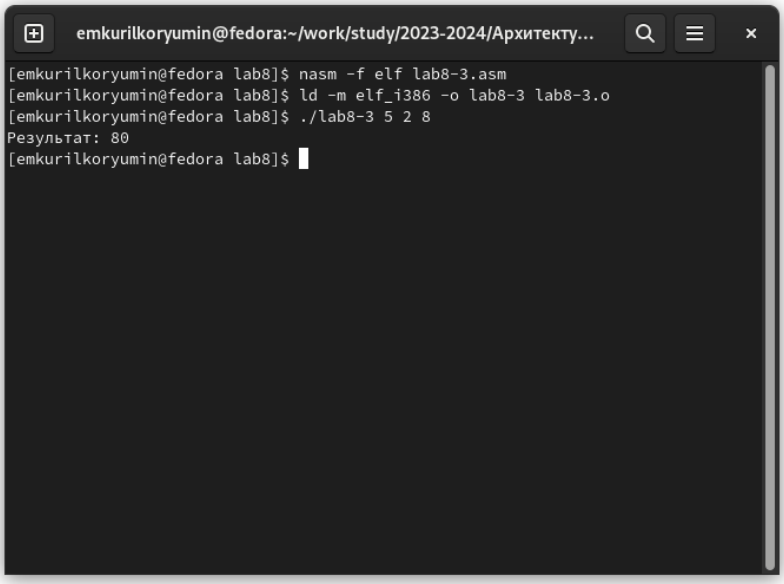

Рис. 4.15: Создание и запуск исполняемого файла

4.3) Выполнение заданий для самостоятельной работы

Создаю файл sr.asm с помощью утилиты touch. Открываю созданный файл для редактирования, ввожу в него текст программы для суммирования значений

функции, предложенной в варианте 4, полученнымпри выполнении прошлых лабораторных работ (рис.16)

Рис. 4.16: Создание и редактирование файла

Проводим привычные операции и запускаем исполняемый файл, выполняем устную проверку и убеждаемся в правильности работы программы.(рис.17)

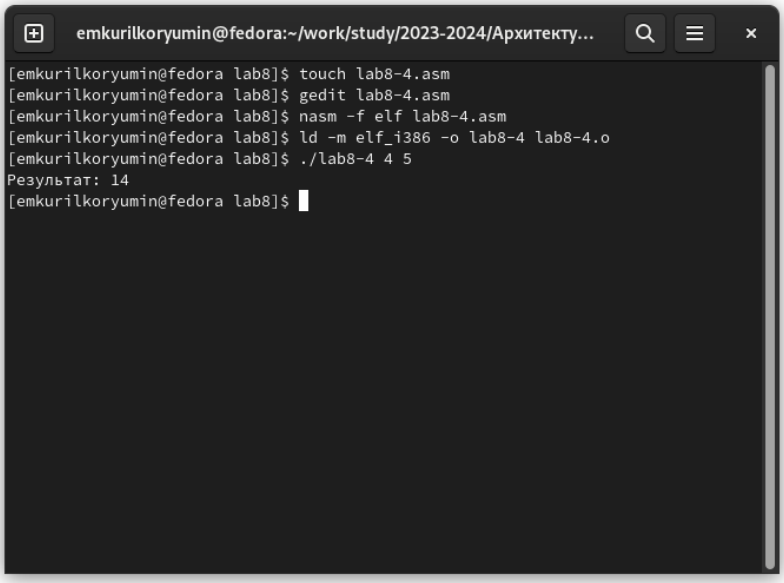

Рис. 4.17: Компиляция, обработка и запуск исполняемого файла

Листинг 4.1 - Программа для суммирования нескольких значений функции,

предложенной в варианте 4.

```
"'%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
sub eax,1
imul eax,2
add esi,eax ; добавляем к промежуточной сумме
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 “‘
```

5 Выводы

При выполнении лабораторной работы я приобрел практический опыт в написании программ с использованием циклов и обработкой аргументов командной строки.

Список литературы

Архитектура компьютера и ЭВМ