Архитектура компьютера

Отчёт по лабораторной работе №8

Арбатова Варвара Петровна

Содержание

# Цель работы

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

# Задание

Напишите программу, которая находит сумму значений функции 𝑓(𝑥) для 𝑥 = 𝑥1, 𝑥2, …, 𝑥𝑛, т.е. программа должна выводить значение 𝑓(𝑥1) + 𝑓(𝑥2) + … + 𝑓(𝑥𝑛). Значения 𝑥𝑖 передаются как аргументы. Вид функции 𝑓(𝑥) выбрать из таблицы 8.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах 𝑥 = 𝑥1, 𝑥2, …, 𝑥𝑛.

# 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. 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). 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. push -10 ; Поместить -10 в стек push ebx ; Поместить значение регистра ebx в стек push [buf] ; Поместить значение переменной buf в стек push word [ax] ; Поместить в стек слово по адресу в ax Существует ещё две команды для добавления значений в стек. Это команда pusha, которая помещает в стек содержимое всех регистров общего назначения в следующем порядке: ах, сх, dx, bх, 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. Примеры: pop eax ; Поместить значение из стека в регистр eax pop [buf] ; Поместить значение из стека в buf pop word[si] ; Поместить значение из стека в слово по адресу в si Аналогично команде записи в стек существует команда popa, которая восстанавливает из стека все регистры общего назначения, и команда popf для перемещения значений из вершины стека в регистр флагов. 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, типичная структура которого имеет следующий вид: mov ecx, 100 ; Количество проходов NextStep: … … ; тело цикла … loop NextStep ; Повторить ecx раз от метки NextStep 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.

# Выполнение лабораторной работы

1. Создаю папку, файл в этой папке, проверяю его создание

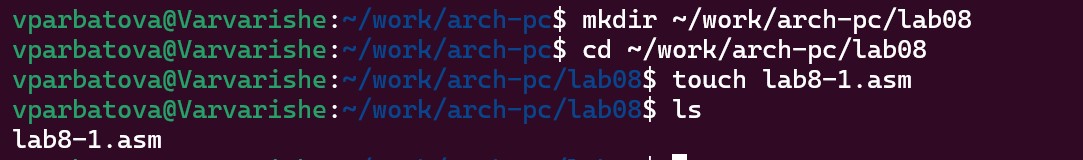


Figure 1: Создание файла

1. Ввожу в файл lab8-1.asm текст программы из листинга 8.1

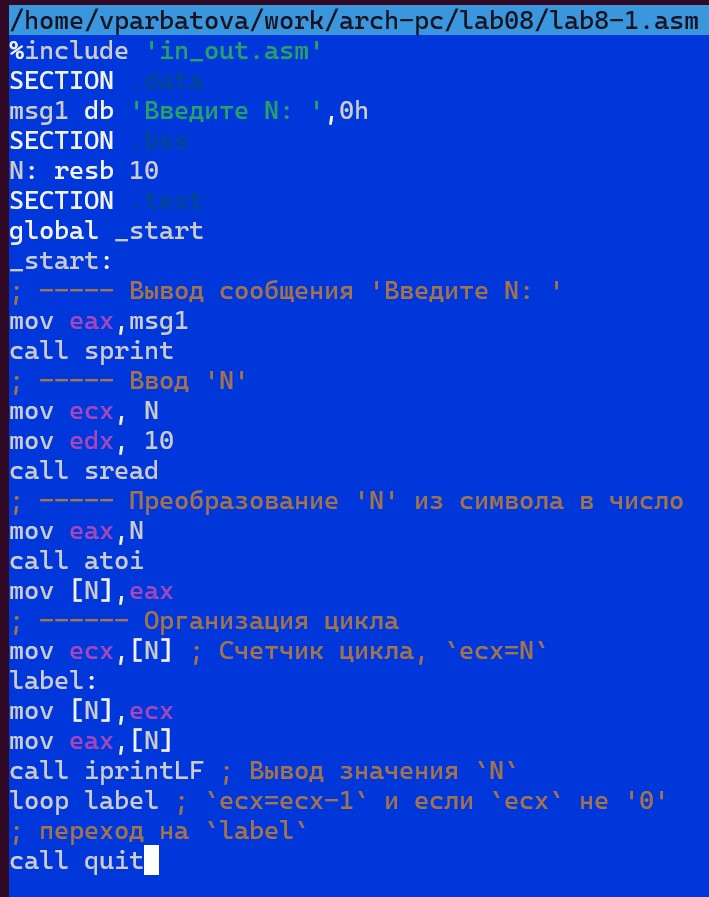


Figure 2: Текстфайла lab8-1.asm

1. Создание исполняемого файла и проверка его работы



Figure 3: Работа файла

1. Изменение текста программы, добавляю изменение значение регистра ecx в цикле

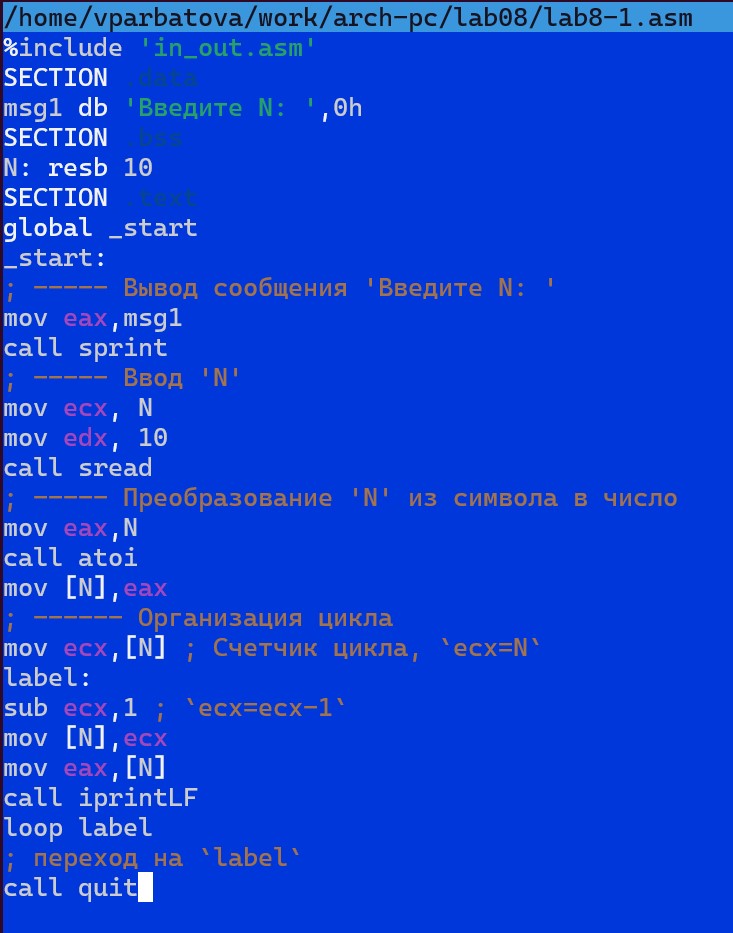


Figure 4: Измененный текст программы

1. Создаю файл и запускаю его. Значения идут с приблизительно 4350000000 и работает некорректно. Работу заскринить не получилось

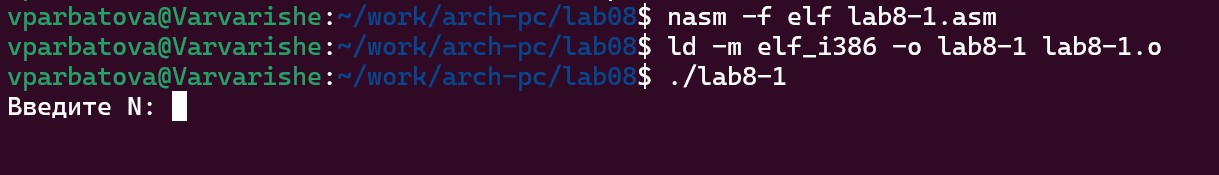


Figure 5: Созданный файл с ошибкой

1. Изменяю текст программы, добавив изменение значение регистра ecx в цикле

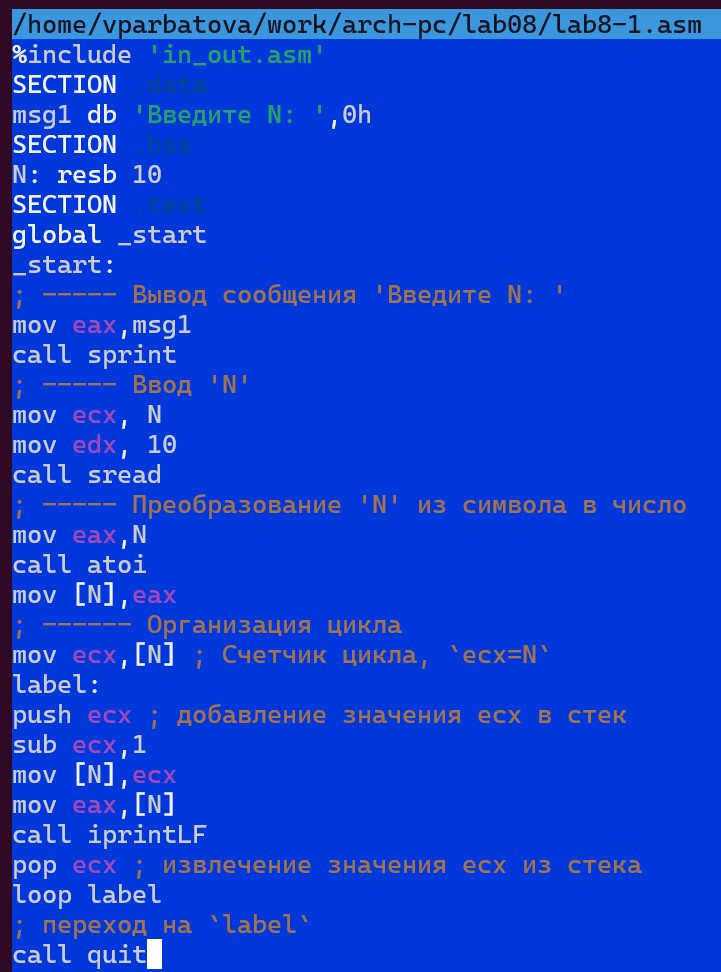


Figure 6: Измененный текст программы

1. Работа программы и создание файла

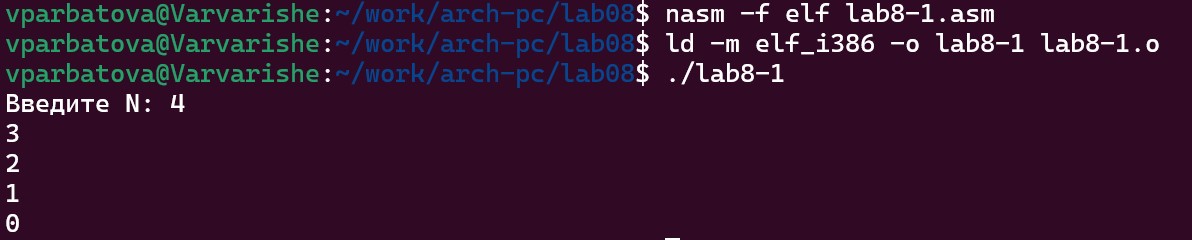


Figure 7: Работа файла

Значения идут от N-1 до 0. Их количество соответствует N, введенному с клавиатуры

1. Создаю файл lab8-2.asm



Figure 8: Название рисунка

1. Вношу изменения в текст программы добавив команды push и pop для сохранения значения счетчика цикла loop

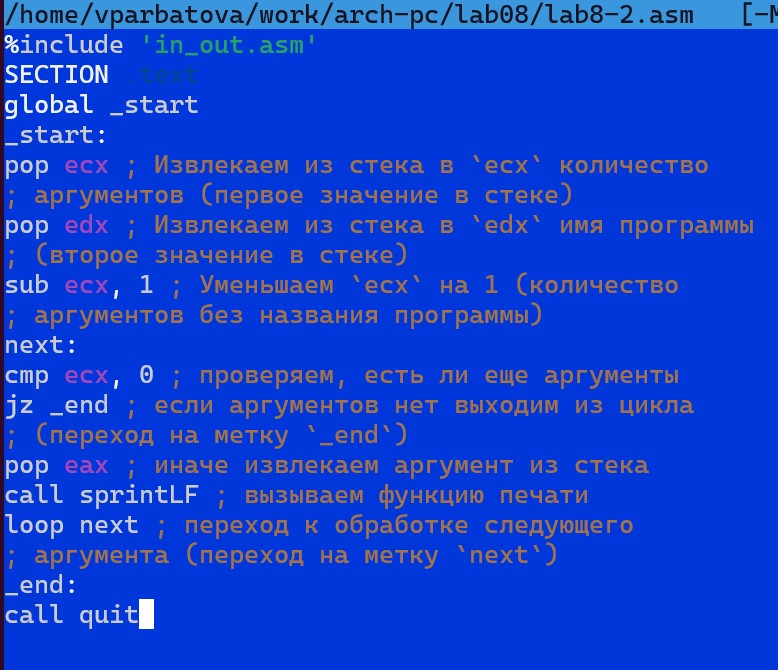


Figure 9: Новый текст файла

1. Создание файла и его работа

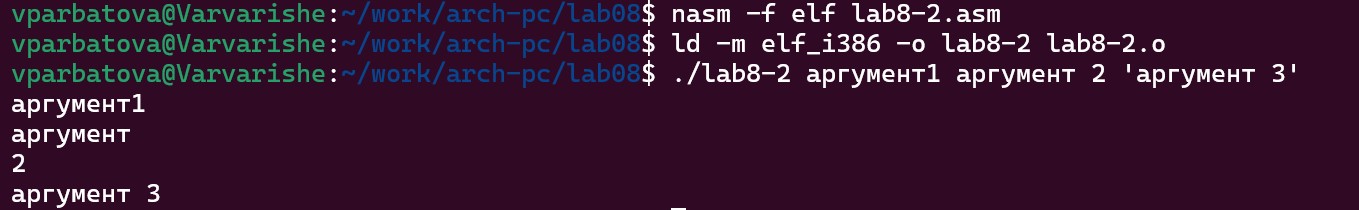


Figure 10: Работа файла

Преграммой было обработано 4 аргумента

1. Создаю файл lab8-3.asm

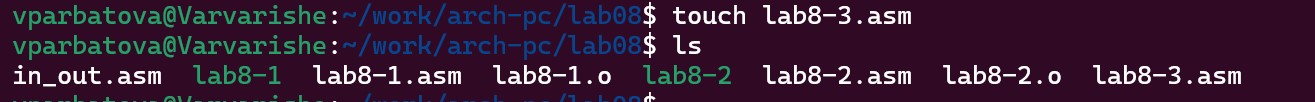


Figure 11: Создание файла

1. Ввожу в него текст программы из листинга 8.3

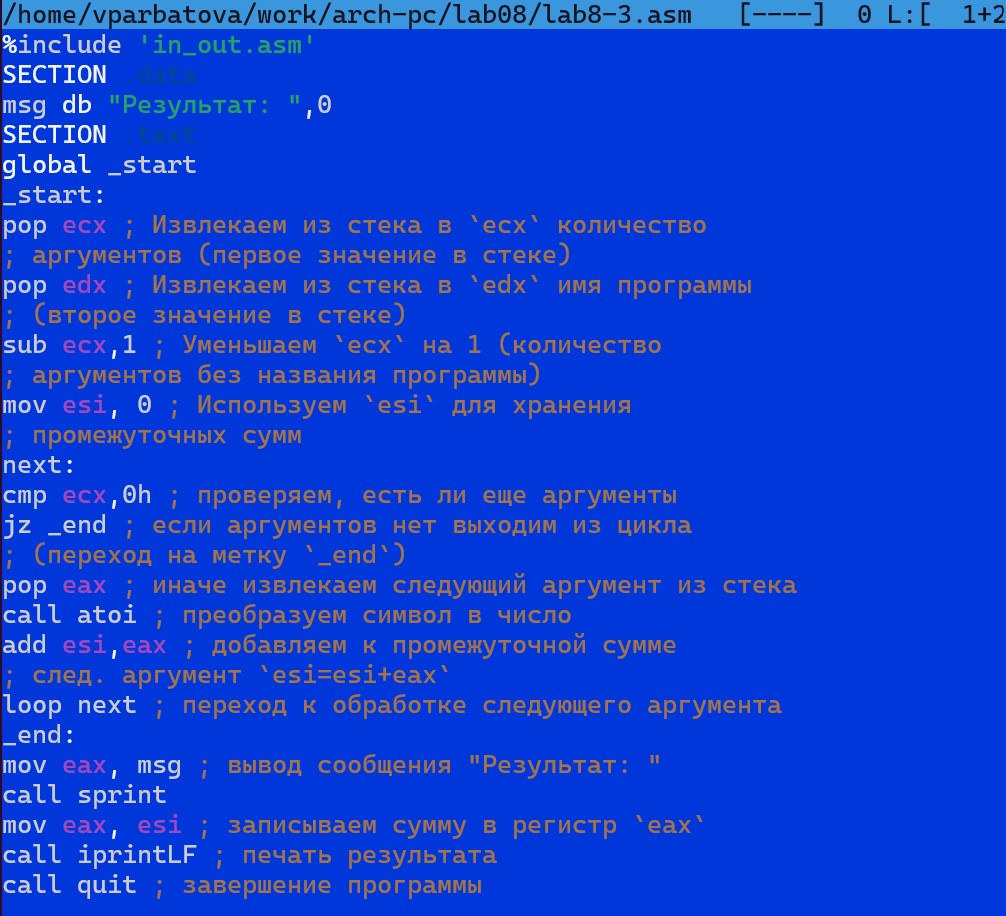


Figure 12: Текст файла

1. Создание исполняемого файла и его работа



Figure 13: Работа файла

1. Меняю текст файла, чтоб он выводил произведение

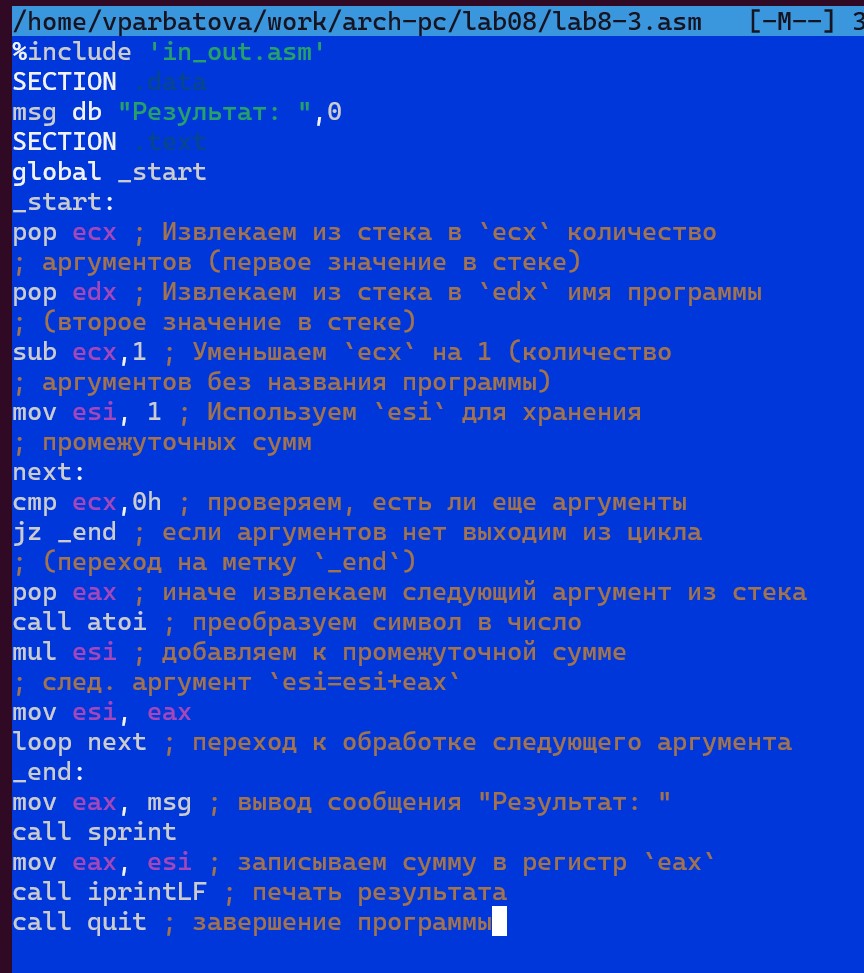


Figure 14: Измененный текст файла

%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 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 ; добавляем к промежуточной сумме ; след. аргумент esi=esi+eax mov 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

1. Создание и работа файла

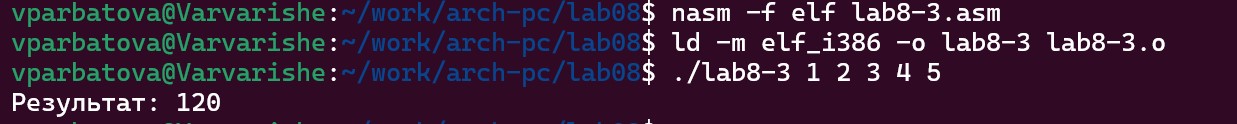


Figure 15: Работа файла

# Выполнение заданий для самостоятельной работы

1. Создаю файл lab8-4.asm



Figure 16: Создание файла

1. Создание и работа файла

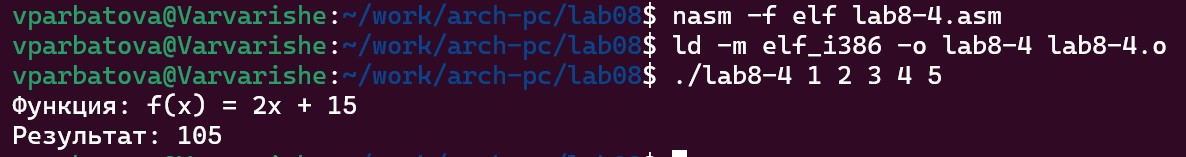


Figure 17: Работа файла

текст файла: %include ‘in\_out.asm’ SECTION .data msg db “Результат:”,0 msg1 db “Функция: f(x) = 2x + 15”,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, 2 mul ebx add eax, 15 add esi,eax ; добавляем к промежуточной сумме ; след. аргумент esi=esi+eax loop next ; переход к обработке следующего аргумента \_end: mov eax, msg1 call sprintLF mov eax, msg ; вывод сообщения “Результат:” call sprint mov eax, esi ; записываем сумму в регистр eax call iprintLF ; печать результата call quit ; завершение программы

# Выводы

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

# Список литературы