Лабораторная работа №8

Архитектура компьютера

Алехин Давид Андреевич

Содержание

# 1 Цель работы

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

# 2 Задание

# 3 Теоретическое введение

**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 ре- гистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек указа- тель стека уменьшается, а при извлечении — увеличивается. Для стека существует две основные операции: • добавление элемента в вершину стека (push); • извлечение элемента из вершины стека (pop).

**Добавление элемента в стек** 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: 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.

**Извлечение элемента из стека** 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.

**Инструкции организации циклов** Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре ecx. Наиболее простой является ин- струкция 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.

# 4 Выполнение лабораторной работы

Создаю каталог lab08 и файл lab8-1.asm .(рис. 1).

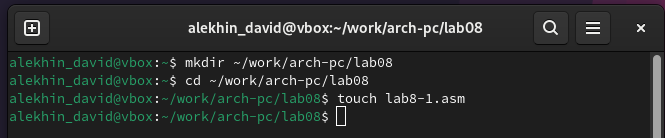


Рис. 1: lab08/lab8-1.asm

Ввожу туда текст команды 8.1. Программа вывода значений регистра ecx. (рис. 2).

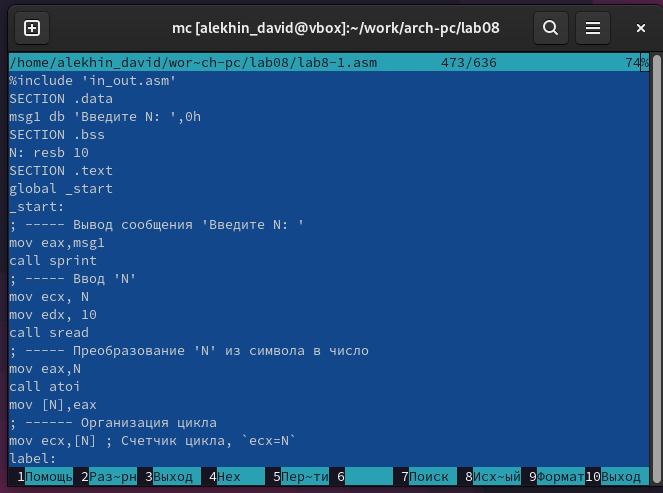


Рис. 2: 8.1

Компеллирую и запускаю команду 8.1. (рис. 3).

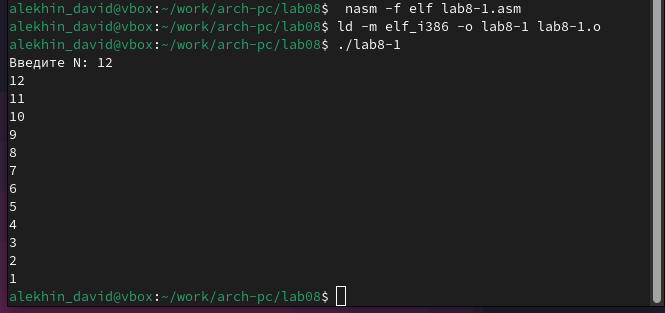


Рис. 3: ./lab8-1

Меняю текст программы, в теле label добавляю строку sub eax,1. Циклы закольцевался и стал бесконечным. (рис. 4), (рис. 5).

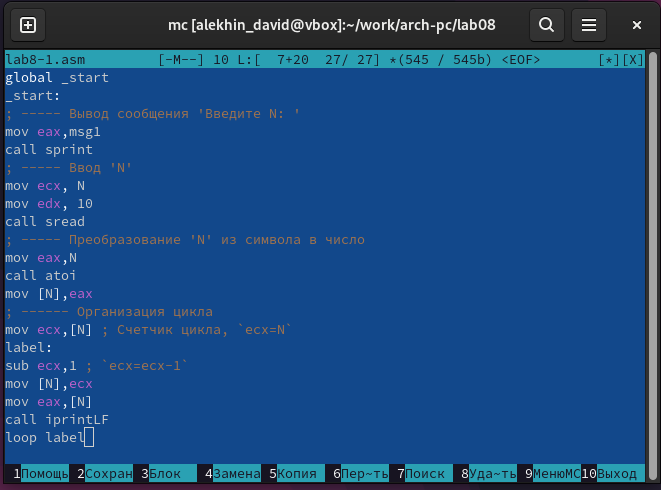


Рис. 4: 8.1v2

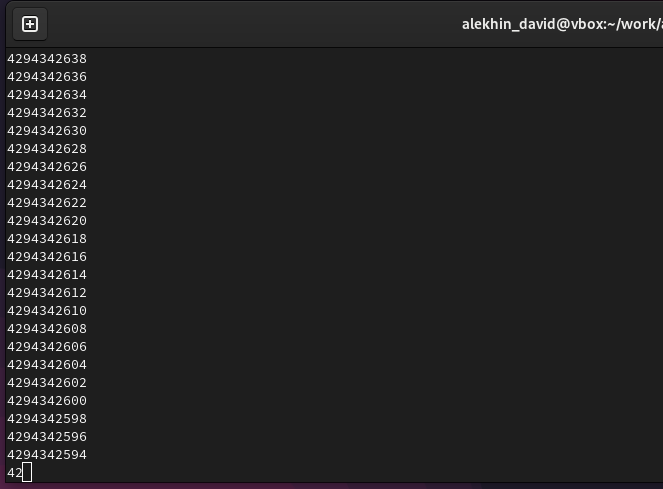


Рис. 5: ./8.1v2

Меняю текст программы так, чтобы цикл и счетчик работали корректно. (рис. 6).

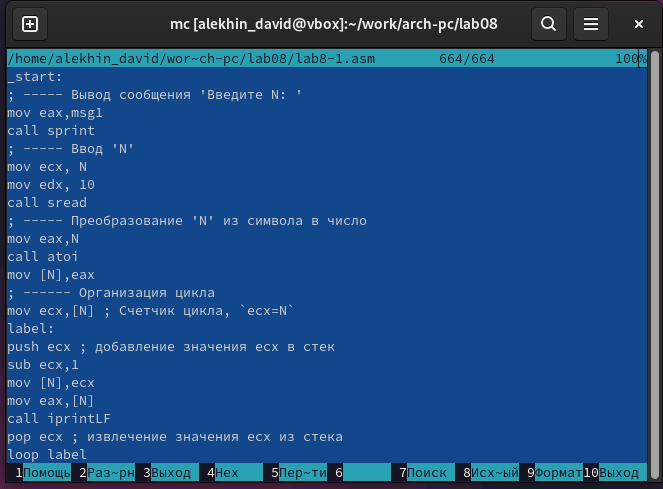


Рис. 6: 8.1v3

После изменения программы, число проходки циклов стал соответствовать числу введенному с клавиатуры. (рис. 7).

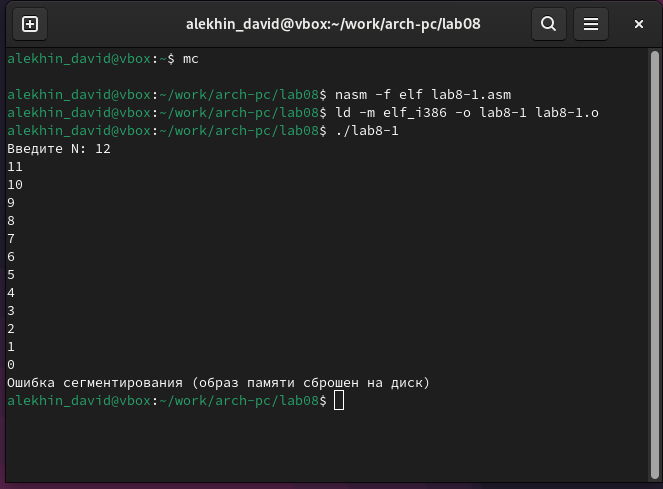


Рис. 7: ./8.1v3

Создаю lab8-2.asm и ввожу туда текст команды 8.2. Программа выводящая на экран аргументы командной строки. (рис. 8).

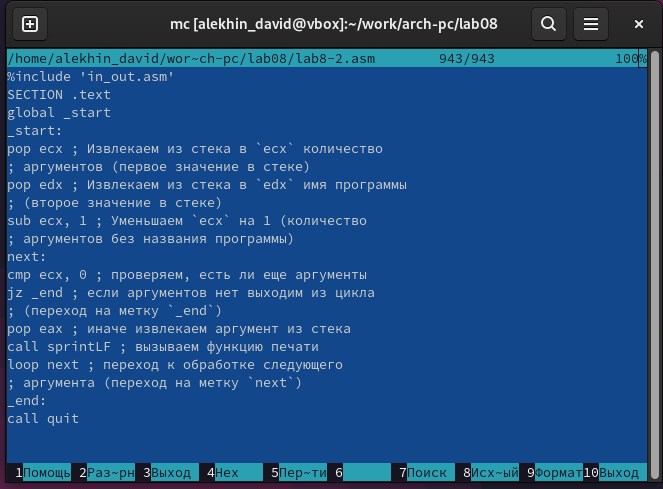


Рис. 8: 8.2

При запуске программа выводит все 3 аргумента которые ввели, но в разном виде. (рис. 9).

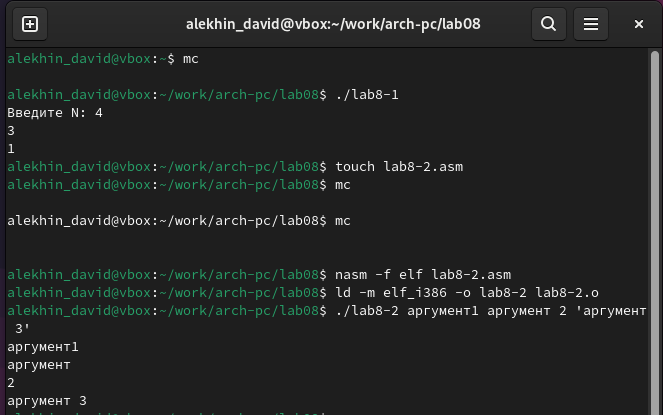


Рис. 9: ./8.2

Создаю lab8-3.asm и ввожу туда текст команды 8.3. gрограмма вычисления суммы аргументов командной строки. (рис. 10).

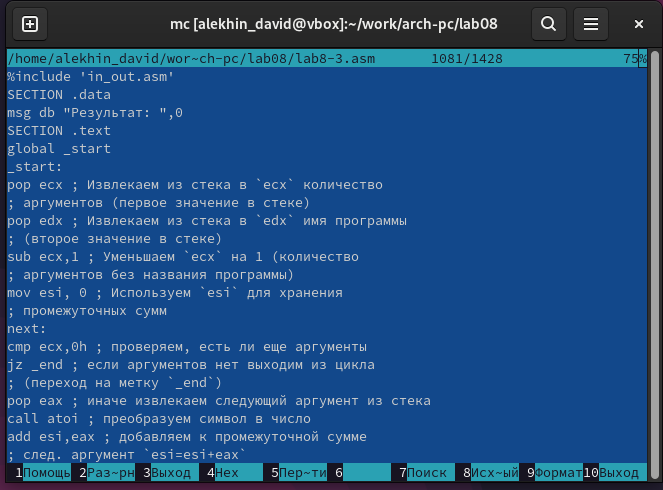


Рис. 10: 8.3

Запускаю 8.3 и получаю сумму. (рис. 11).

Рис. 11: ./8.3

Рис. 11: ./8.3

Меняю текст 8.3 так, чтобы полчить произведение аргументов. (рис. 12) (рис. 13).

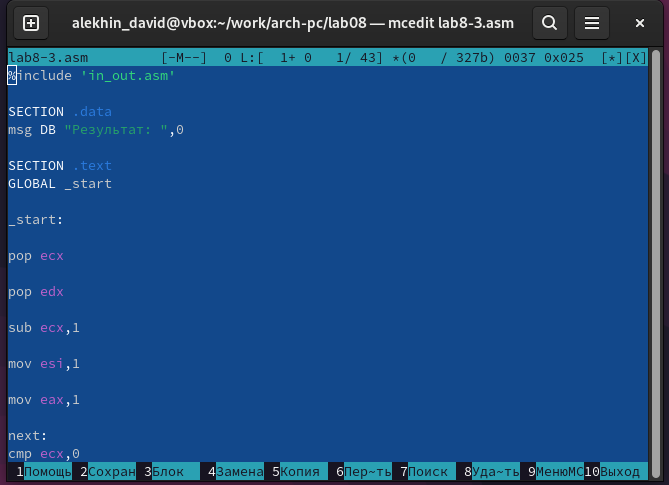


Рис. 12: 8.3v2

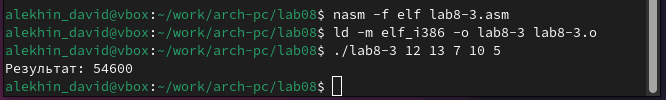


Рис. 13: ./8.3v2

Пишу команду для самостоятельной работы (1 вариант). (рис. 14).

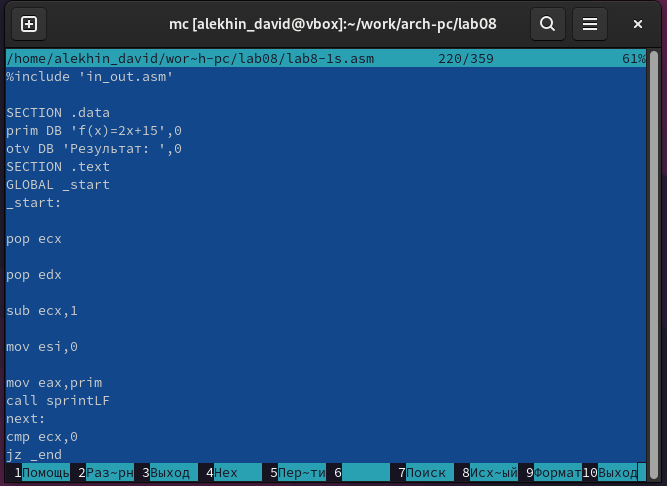


Рис. 14: lab8-1s.asm

Компеллирую и запускаю проверяя различные значения аргумента. (рис. 15).

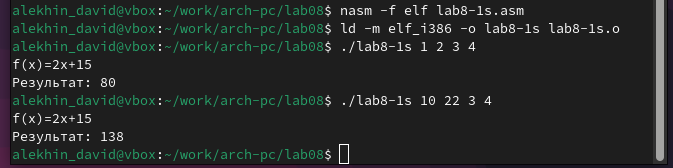


Рис. 15: ./lab8-1s.asm

# 5 Выводы

После выполнения лабораторной работы я освоил навыки написания программ с использованием циклов и обработкой аргументов командной строки.

# Список литературы

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.
2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.
3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander. org/.
4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O’Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learning- bash-Shell-Programming-Nutshell/dp/0596009658.
6. Robbins A. Bash Pocket Reference. — O’Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс,
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ- Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm\_unix.
15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер,
17. — 1120 с. — (Классика Computer Science).