Лабораторная работа №8.

Программирование цикла. Обработка аргументов командной строки.

Павленко Сергей

Содержание

# 1 Цель работы

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

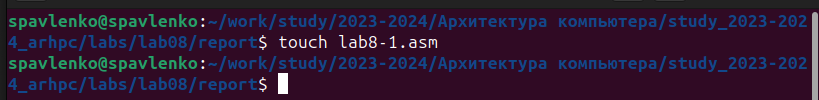
# 2 Теоретическое введение

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).

# 3 Выполнение лабораторной работы

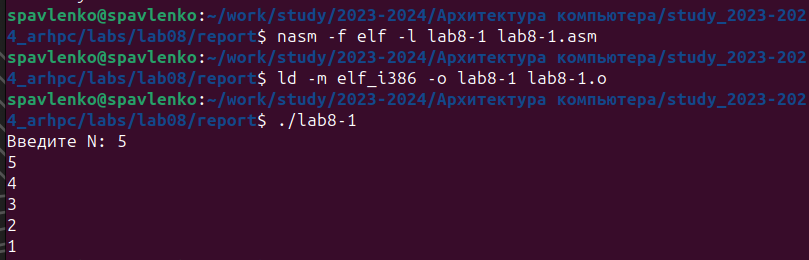
Создайте каталог для программам лабораторной работы № 8, перейдите в него и создайте файл lab8-1.asm:

cd ~/work/arch-pc/lab08 touch lab8-1.asm



1

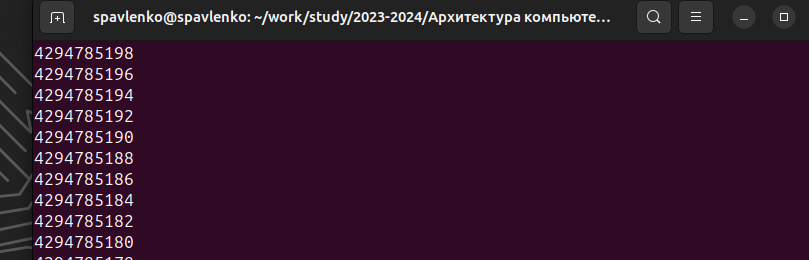
Введите в файл lab8-1.asm текст программы из листинга 8.1. Создайте исполняемый файл и проверьте его работу



2

Измените текст программы добавив изменение значение регистра ecx в цикле. Создайте исполняемый файл и проверьте его работу. Какие значения принимает регистр ecx в цикле? Соответствует ли число проходов цикла значению 𝑁 введенному с клавиатуры?

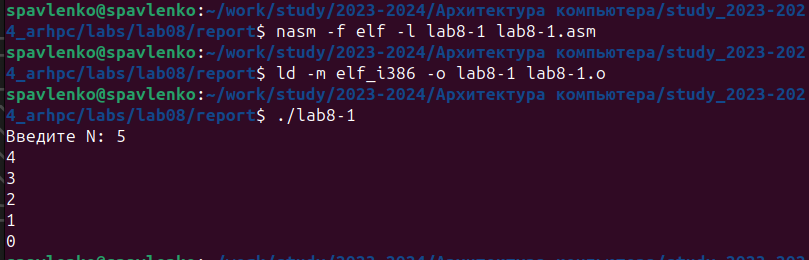
ecx принимает значение =ecx-1 Нет, не соответствует значению N



3

Для использования регистра ecx в цикле и сохранения корректности работы программы можно использовать стек. Внесите изменения в текст программы добавив команды push и pop (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла loop. Создайте исполняемый файл и проверьте его работу. Соответствует ли в данном случае число проходов цикла значению 𝑁 введенному с клавиатуры?

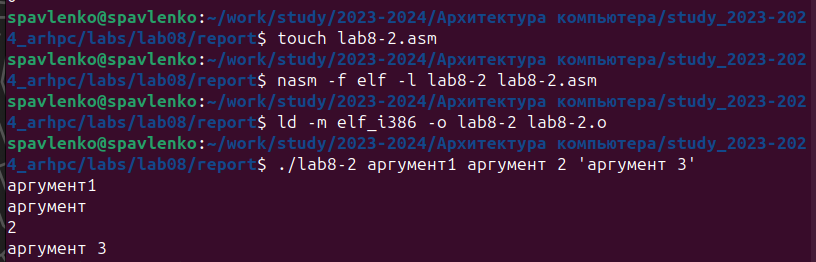
В данном случае значение N соответствует числу проходов циклов



4

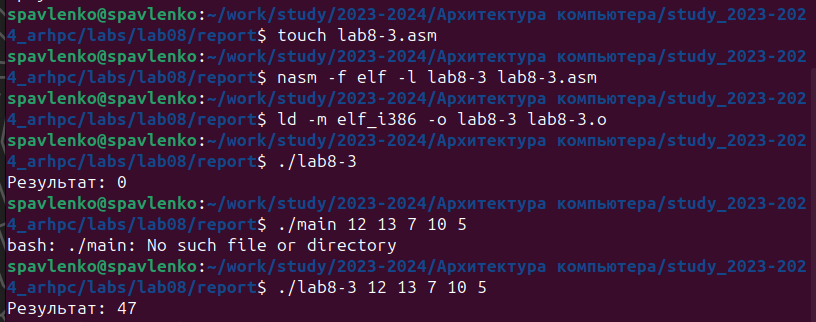
Создайте файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и введите в него текст про- граммы из листинга 8.2. Создайте исполняемый файл и запустите его, указав аргументы: user@dk4n31:~$ ./lab8-2 аргумент1 аргумент 2 ‘аргумент 3’ Сколько аргументов было обработано программой?

Было обработано 4 аргумента



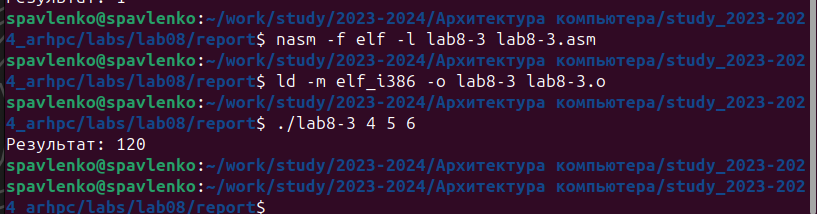
5

Создайте файл lab8-3.asm в каталоге ~/work/arch- pc/lab08 и введите в него текст программы из листинга 8.3. Создайте исполняемый файл и запустите его, указав аргументы. Пример результата работы программы: user@dk4n31:~$ ./main 12 13 7 10 5 Результат: 47 user@dk4n31:~$



6

Измените текст программы из листинга 8.3 для вычисления произведения аргументов командной строки.



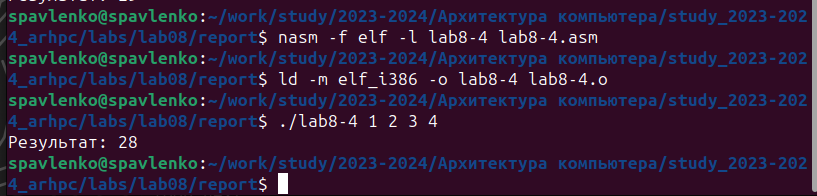
7

# 4 Выводы

Такие образом мы научились зацикливать и обрабатывать некоторые аргументы командной строки

# 5 Самостоятельная работа

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.



8

# 6 Выводы

В ходе самостоятельной работы мы на практике закрепили знания, полученые в лабораторной работе, по зацикливанию и обработке 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, 2015. — 1120 с. — (Классика Computer Science).