Лабораторная работа номер 8.

Программирование цикла. Обработка аргументов командной строки.

Сорокин Кирилл

Содержание

# 1 Цель работы

Научиться писать циклы и обрабатывать информацию из командной строки с помощью языка ассемблер.

# 2 Задание

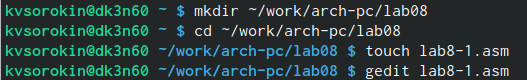
Изучить приведённый материал на практике и выполнить самостоятельную работу.

# 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.

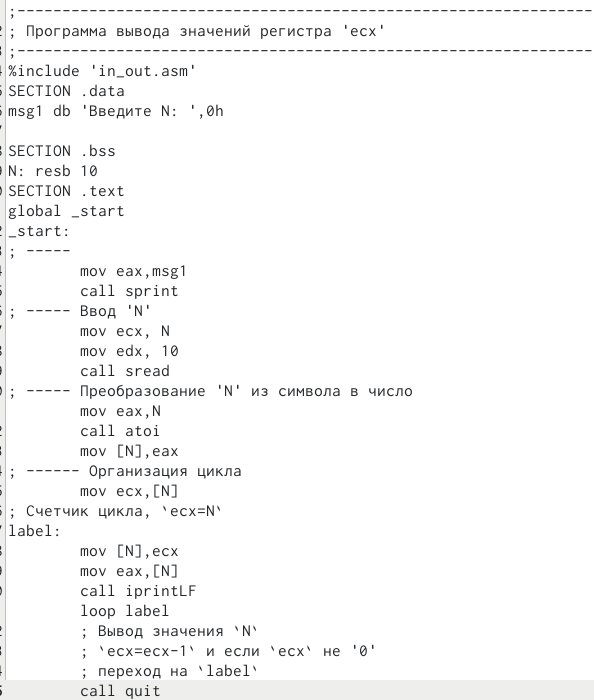
# 4 Выполнение лабораторной работы

Создадим необходимые для работы директории и файлы (рис. ??).



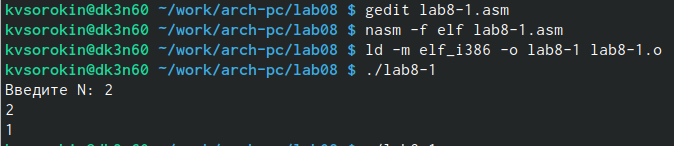
Создание файлов и директорий

Откроем файл lab8-1.asm и введём в него текст программы(рис. ??).



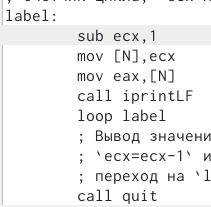
Текст первой программы

После компиляции файлов запустим программу и увидим не совсем требуемый результат число хоть и уменьшается, но не до нуля(рис. ??).



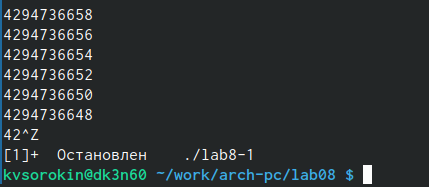
Первая попытка выполнить

Изменим содержимое label в тексте программы(рис. ??).



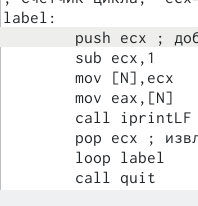
Редактирование label 1

Попробуем ещё раз выполнить программу и увидим, что программа попадает в бесконечный цикл, выводя совсем не корректные значения. Это связанно с тем, что мы вначале уменьшает ecx с помощью sub, а потом ещё раз с помощью loop, поэтому он и не принимает значения 0 и уходит в бесконечность.(рис. ??).



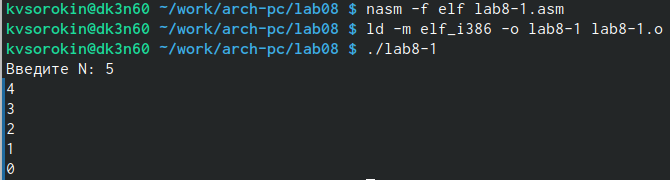
Вторая попытка выполнить

Ещё раз изменим содержимое label так, чтобы ecx мог заходить и выходить из стека.(рис. ??).



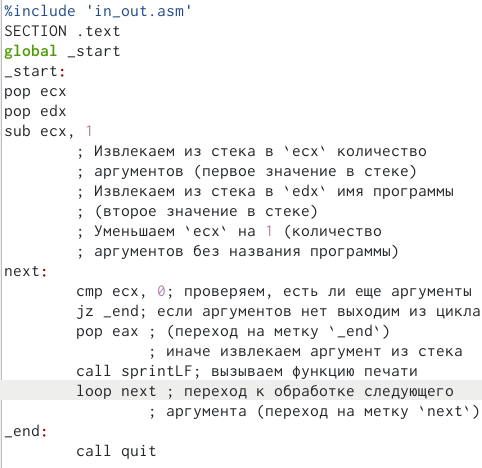
Редактирование label 2

После выполнения опять увидим, что наконец достигнут желаемый результат.(рис. ??).



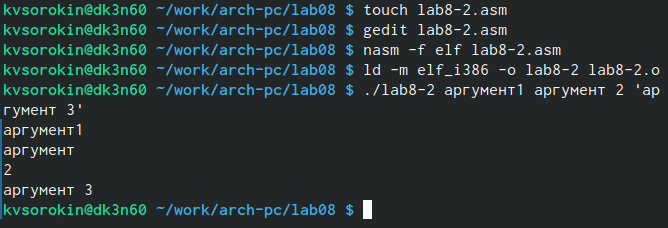
Успешный запуск цикла

Создадим файл lab8-2.asm и введём в него текст программы(рис. ??).



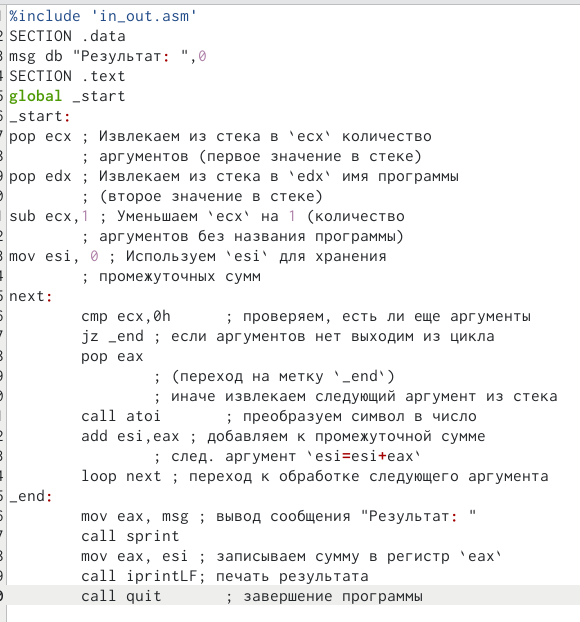
Программы lab8-2

Выполним и увидим, что программа считает за аргументы каждый данный ей элемент разделённый пробелом. (их 4) (рис. ??).



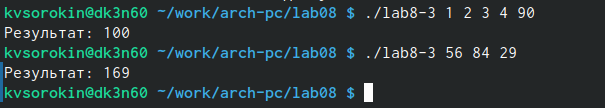
Раюота с агрументами

Создадим файл lab8-3.asm и запишем в него программу для суммы введённых чисел (рис. ??).



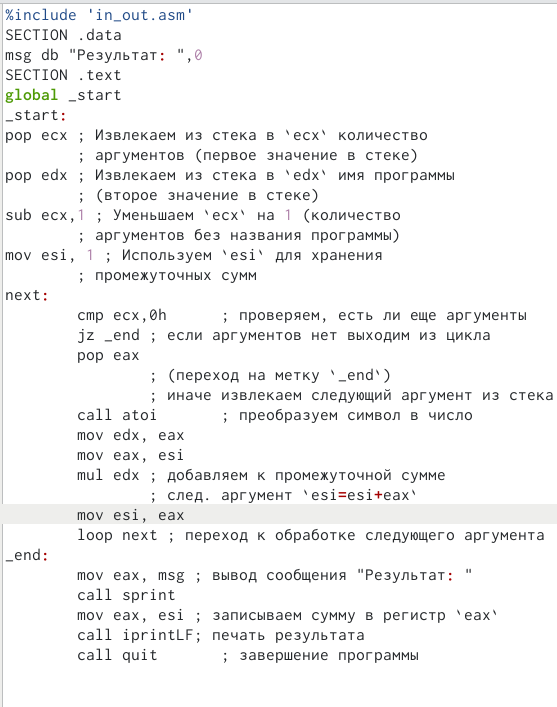
Содержимое файла lab8-3.asm

Проверим её на нескольких наборах данных. (рис. ??).



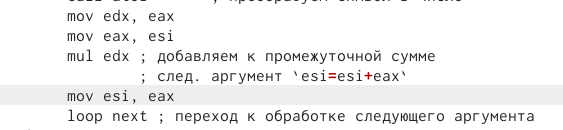
Сложение чисел

Изменим текст программы, чтобы она умножала числа(рис. ??).



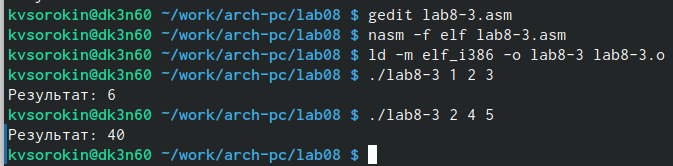
Изменённый текст lab8-3

Здесь видны основные отличия от предыдущей версии программы. Однако необходимо также изменить начально значение esi на 1, чтобы умножение выполнялось корректно. (рис. ??).



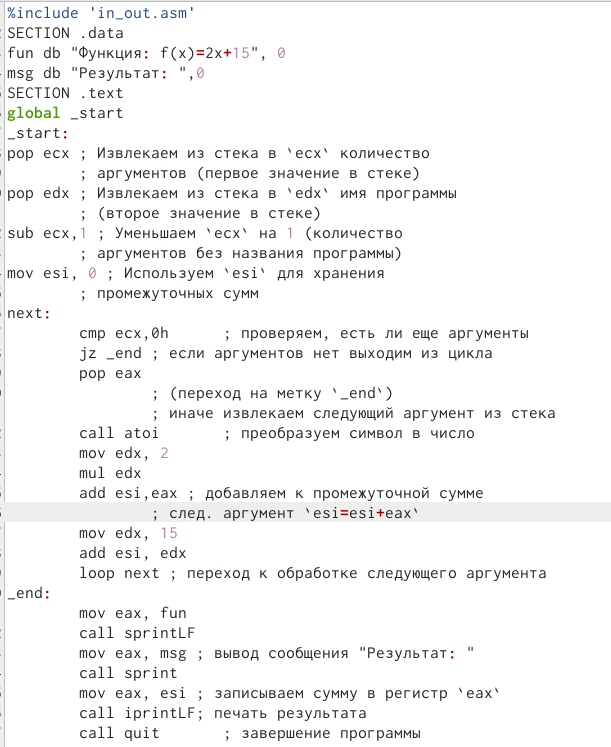
Основные изменения lab8-3

Убедимся в корректности умножения чисел (рис. ??).



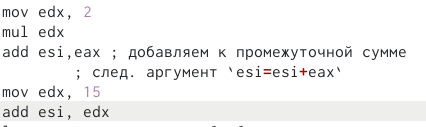
## 4.1 Самостоятельная работа

Так как наш вариант 1, напишем программу для суммы значений функции f(x)=2x+15 от введённых значений (рис. ??).



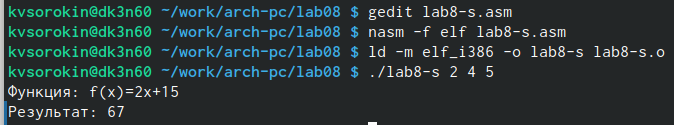
Текст программы

Основными изменениями от программы складывающей аргументы будет в необходимости домножать аргумент на 2 и прибавлять к нему 15. Однако необходимо также не забыть о том, что нужно ещё вывести функцию на экран (3 строки разбросаны по тексту программы, поэтому они не отображены на этом рисунке) (рис. ??).



Основные изменения

Скомпилировав файл, убедимся, что он работает корректно (рис. ??).



Работа самостоятельной работы

# 5 Выводы

Мы научились использовать писать циклы на языке ассемблера, а также получать информацию из 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://midnightcommander.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, 2017.
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)