Отчёт по лабораторной работе 6

Архитектура компьютера

Нилова Кристина Артуровна

Содержание

# 1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

# 2 Задания

1. Изучение арифметических действий
2. Примеры программ с вычислниями
3. Выполнение заданий для самостоятельной работы.

# 3 Теоретическое введение

Схема команды целочисленного сложения add (от англ. addition - добавление) выполняет сложение двух операндов и записывает результат по адресу первого операнда.

Команда целочисленного вычитания sub (от англ. subtraction – вычитание) работает аналогично команде add.

Существуют специальные команды: inc (от англ. increment) и dec (от англ. decrement), которые увеличивают и уменьшают на 1 свой операнд.

Умножение и деление, в отличии от сложения и вычитания, для знаковых и беззнаковых чисел производиться по-разному, поэтому существуют различные команды. Для беззнакового умножения используется команда mul (от англ. multiply – умножение), для знакового умножения используется команда imul.

Для деления, как и для умножения, существует 2 команды div (от англ. divide - деление) и idiv

# 4 Выполнение лабораторной работы

## 4.1 Символьные и численные данные в NASM

Я создала каталог для хранения программ к лабораторной работе номер шесть, после чего перешла в этот каталог и создала файл с исходным кодом программы под названием lab6-1.asm, как показано на рисунке [[1](#fig:001)].

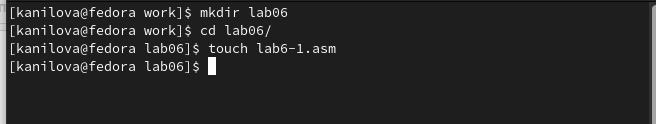


Figure 1: Создание каталога и файла

Теперь давайте взглянем на примеры программ, которые отображают символы и числовые значения на экран. Эти программы будут использовать регистр eax для вывода значений.

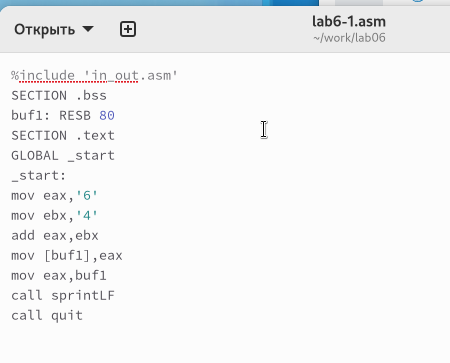


Figure 2: Заполнение файла lab6-1.asm

В одном из примеров, который вы можете увидеть на рисунке [[2](#fig:002)], программа записывает символ ‘6’ в регистр eax с помощью инструкции mov eax, ‘6’, а символ ‘4’ помещается в регистр ebx с помощью инструкции mov ebx, ‘4’. Затем я выполняю сложение значений, находящихся в регистрах ebx и eax, и результат записывается обратно в eax с помощью инструкции add eax, ebx. После этого нужно вывести результат на экран.

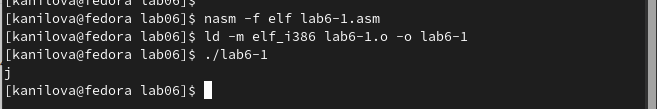


Figure 3: Компиляция текста программы lab6-1.asm

Однако, чтобы функция sprintLF работала правильно, необходимо передать в eax адрес данных. Поэтому я использую вспомогательную переменную buf1. Сначала я перемещаю значение из eax в buf1 с помощью инструкции mov [buf1], eax, а потом загружаю адрес переменной buf1 обратно в регистр eax (mov eax, buf1), прежде чем вызвать sprintLF.

В итоге, вместо того чтобы увидеть на экране число 10, мы видим символ ‘j’. Это происходит из-за особенностей кодировки символов: символ ‘6’ имеет двоичный код 00110110 (или 54 в десятичном виде), а ‘4’ — 00110100 (или 52 в десятичном виде). Сложив эти значения, мы получаем двоичный код 01101010, что в десятичной системе равно 106, и это соответствует символу ‘j’, как показано на рисунке [[3](#fig:003)].

Теперь я внесу изменения в исходный код программы, чтобы в регистрах хранились числовые значения, а не символы, как это показано на иллюстрации [[4](#fig:004)].

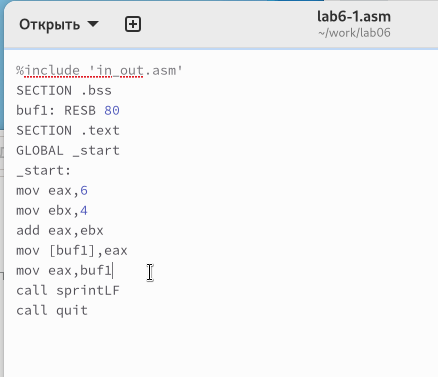


Figure 4: Заполнение файла lab6-1.asm

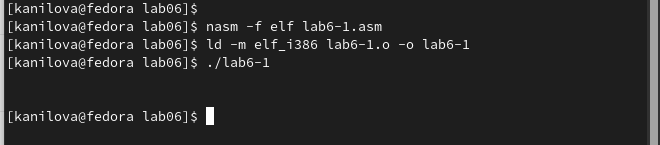


Figure 5: Компиляция текста программы lab6-1.asm

Когда я запускаю программу, она не выдаёт число 10. На экран выводится символ с ASCII-кодом 10, который означает новую строку, как видно на рисунке [[5](#fig:005)]. Этот символ невидим в терминале, но он создаёт пустую строку в моём выводе.

Я помню, что ранее в файле in\_out.asm были созданы специальные функции, чтобы преобразовывать символы ASCII в числа и обратно. Используя эти функции, я изменила код программы, что можно увидеть на иллюстрации [[6](#fig:006)].



Figure 6: Заполнение файла lab6-2.asm

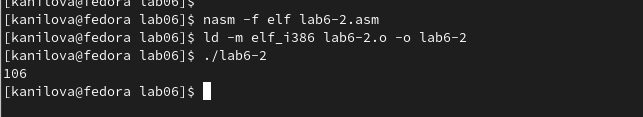


Figure 7: Компиляция текста программы lab6-2.asm

Когда я запустила модифицированную программу, на экране появилось число 106, как это показано на рисунке [[7](#fig:007)]. В этом случае, так же как и в первом примере, команда add сложила ASCII-коды цифр ‘6’ и ‘4’ (54+52=106). Но благодаря функции iprintLF, на этот раз на экран вывелось именно число, а не символ, соответствующий его ASCII-коду.

Таким же образом, как и в предыдущем примере, я заменила символы на их числовые эквиваленты, что изображено на рисунке [[8](#fig:008)].

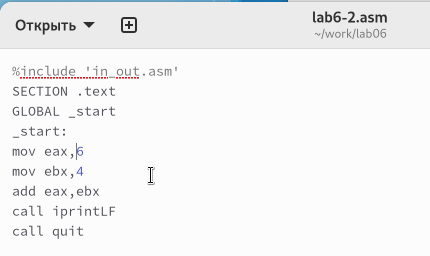


Figure 8: Заполнение файла lab6-2.asm

В этой ситуации функция iprintLF использовалась для отображения числового значения, так как операнды были числами, а не символьными кодами. Это привело к результату, который равен числу 10, как можно увидеть на рисунке [[9](#fig:009)].

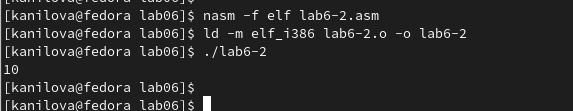


Figure 9: Компиляция текста программы lab6-2.asm

Затем я заменила функцию iprintLF на функцию iprint. После того, как я скомпилировала и запустила программу, результат изменился: теперь вывод не содержал переноса строки, что иллюстрируется на рисунке [[10](#fig:010)].

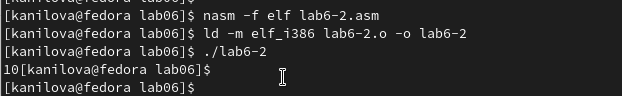


Figure 10: Компиляция текста программы lab6-2.asm

## 4.2 Выполнение арифметических операций в NASM

Возьмем, к примеру, код, который я написала для вычисления функции

. Вы можете посмотреть, как он работает, на иллюстрациях [[11](#fig:011)] и [[12](#fig:012)].

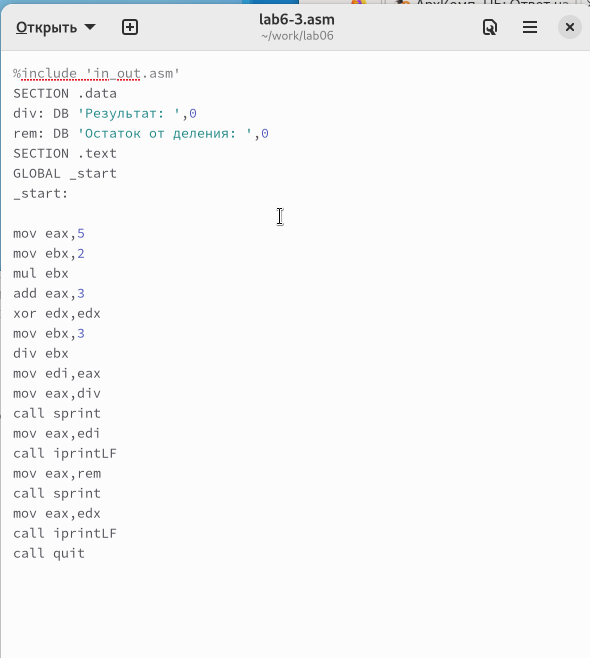


Figure 11: Заполнение файла lab6-3.asm

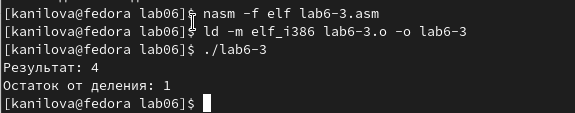


Figure 12: Компиляция текста программы lab6-3.asm

После этого я немного изменила исходный код, чтобы он мог рассчитать другую функцию:

. После того как я скомпилировала измененный код и получила исполняемый файл, я провела его тестирование. Показано на рисунках [[13](#fig:013)] и [[14](#fig:014)].

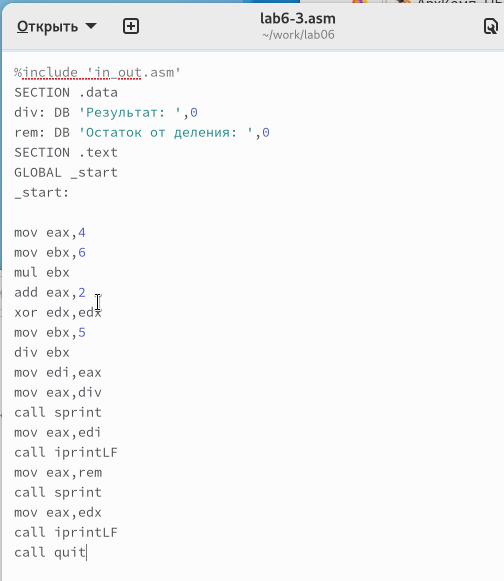


Figure 13: Заполнение файла lab6-3.asm

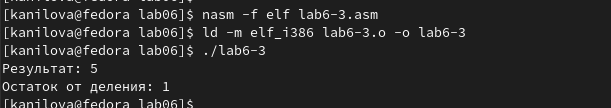


Figure 14: Компиляция текста программы lab6-3.asm

Давайте взглянем на программу, которая вычисляет результаты, исходя из номера студенческого билета. Показано на рисунках [[15](#fig:015)] и [[16](#fig:016)].

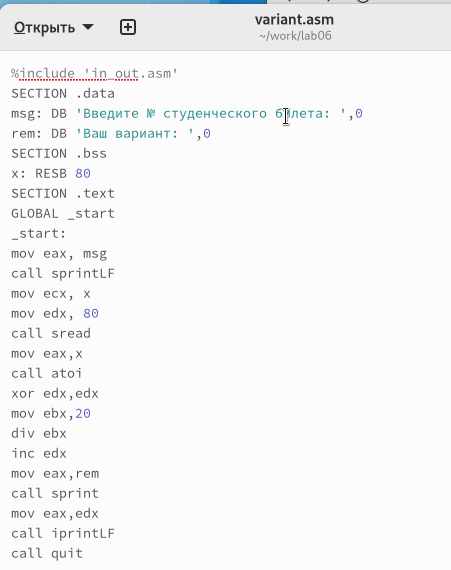


Figure 15: Заполнение файла variant.asm

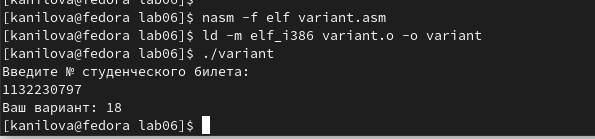


Figure 16: Компиляция текста программы variant.asm

### 4.2.1 Ответы на вопросы

1. Какие строки листинга отвечают за вывод на экран сообщения ‘Ваш вариант:’?

Строки кода, отвечающие за показ сообщения “Ваш вариант:”, включают загрузку этого сообщения в регистр с помощью команды “mov eax, rem”, а также вызов функции вывода на экран с помощью “call sprint”.

1. Для чего используются следующие инструкции?

Инструкции “mov ecx, x” и “mov edx, 80” используются для передачи значения из переменной x в регистр ecx и установки числа 80 в регистр edx соответственно. Для считывания номера студенческого билета применяется команда “call sread”.

1. Для чего используется инструкция “call atoi”?

Инструкция “call atoi” служит для конвертации строки символов в числовое значение.

1. Какие строки листинга отвечают за вычисления варианта?

Вычисление варианта производится с помощью обнуления регистра edx командой “xor edx, edx”, загрузки числа 20 в регистр ebx с помощью “mov ebx, 20”, выполнения деления на 20 через “div ebx” и увеличения результата на единицу с помощью “inc edx”.

1. В какой регистр записывается остаток от деления при выполнении инструкции “div ebx”?

При выполнении операции деления “div ebx” остаток помещается в регистр edx.

1. Для чего используется инструкция “inc edx”?

Команда “inc edx” используется для инкрементирования содержимого регистра edx на единицу, что необходимо при вычислении варианта.

1. Какие строки листинга отвечают за вывод на экран результата вычислений?

Вывод результатов вычислений на экран осуществляется путем помещения результата в регистр eax с помощью “mov eax, edx” и последующего вызова функции “call iprintLF” для вывода на экран.

## 4.3 Самостоятельное задание

Написать программу вычисления выражения y = f(x). Программа должна выводить выражение для вычисления, выводить запрос на ввод значения x, вычислять заданное выражение в зависимости от введенного x, выводить результат вычислений. Вид функции f(x) выбрать из таблицы 6.3 вариантов заданий в соответствии с номером полученным при выполнении лабораторной работы. Создайте исполняемый файл и проверьте его работу для значений x1 и x2 из 6.3.

Получили вариант 18 -

для

(рис. [[17](#fig:017)] и [[18](#fig:018)])

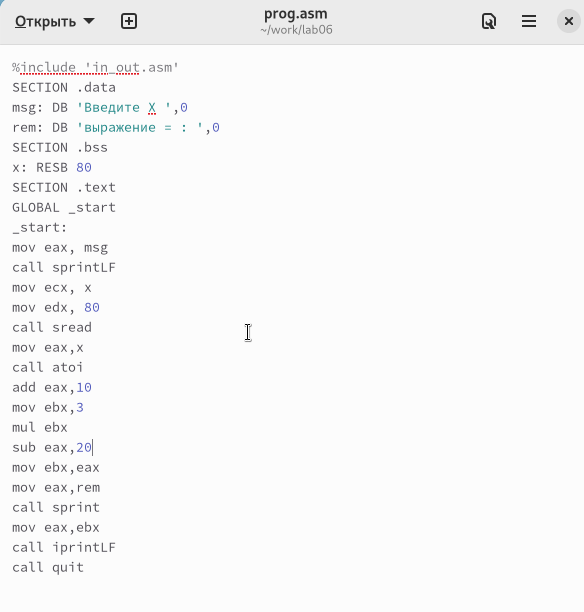


Figure 17: Заполнение файла prog.asm

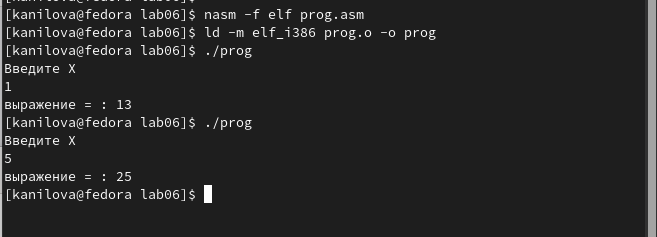


Figure 18: Компиляция текста программы prog.asm

# 5 Выводы

Изучили работу с арифметическими операциями.