Отчёт по лабораторной работе 6

Дисциплина: архитектура компьютера

Синюшко Элза Камиловна

Содержание

# 1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

# 2 Задание

1. Изучение арифметический операций в ассемблере
2. Изучение типов данных в ассемблере
3. Выполнение заданий, рассмотрение примеров
4. Выполнение заданий для самостоятельной работы

# 3 Теоретическое введение

Большинство инструкций на языке ассемблера требуют обработки операндов. Адрес операнда предоставляет место, где хранятся данные, подлежащие обработке. Это могут быть данные хранящиеся в регистре или в ячейке памяти.

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.

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 команд умножения один из сомножителей указывается в команде и должен находиться в регистре или в памяти, но не может быть непосредственным операндом. Второй сомножитель в команде явно не указывается и должен находиться в регистре EAX,AX или AL, а результат помещается в регистры EDX:EAX, DX:AX или AX, в зависимости от размера операнда.

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

В командах указывается только один операнд – делитель, который может быть регистром или ячейкой памяти, но не может быть непосредственным операндом. Местоположение делимого и результата для команд деления зависит от размера делителя. Кроме того, так как в результате деления получается два числа – частное и остаток, то эти числа помещаются в определённые регистры 6.2.

# 4 Выполнение лабораторной работы

## 4.1 Символьные и численные данные в NASM

Я создала каталог для программ лабораторной работы № 6, перешла в него и создала файл lab6-1.asm.

Рассмотрим примеры программ вывода символьных и численных значений. Программы будут выводить значения, записанные в регистр eax.

В данной программе (рис. [[1](#fig:001)]) в регистр eax записан символ 6 (mov eax,‘6’), а в регистр ebx символ 4 (mov ebx,‘4’). Затем прибавиляем значение регистра ebx к значению в регистре eax (add eax,ebx, результат сложения запишется в регистр eax). Затем вывод результата (рис. [[2](#fig:002)]).

Так как для работы функции sprintLF в регистр eax должен быть записан адрес, использовуем дополнительную переменную. Записали значение регистра eax в переменную buf1 (mov [buf1],eax), а затем записали адрес переменной buf1 в регистр eax (mov eax,buf1) и вызвали функцию sprintLF.

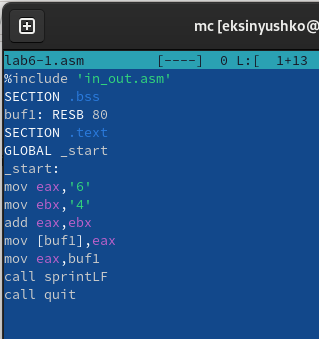


Figure 1: Изменение кода lab6-1.asm

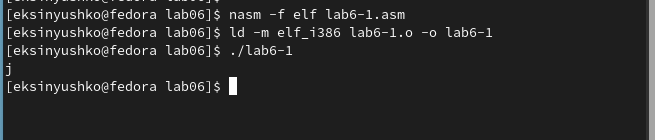


Figure 2: Компиляция текста программы lab6-1.asm

В данном случае, при выводе значения регистра eax, я ожидала увидеть число 10. Однако, результатом был символ ‘j’. Это произошло потому, что код символа 6 равен 00110110 в двоичном представлении (или 54 в десятичном представлении), а код символа 4 – 00110100 (52). Команда add eax,ebx записала в регистр eax сумму кодов – 01101010 (106), что в свою очередь является кодом символа ‘j’.

Далее я изменила текст программы и вместо символов записала в регистры числа (рис. [[3](#fig:003)])

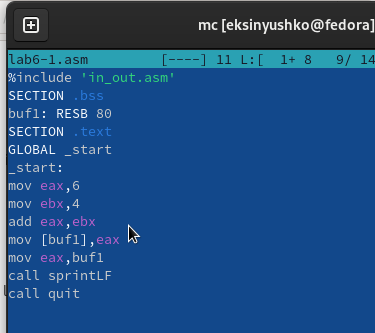


Figure 3: Изменение кода lab6-1.asm

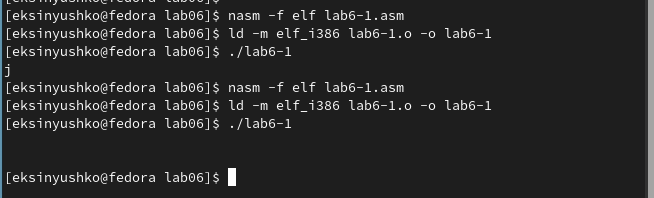


Figure 4: Компиляция текста программы lab6-1.asm

В процессе выполнения программы мы не получили ожидаемое число 10. Вместо этого был выведен символ с кодом 10 (рис. [[4](#fig:004)]). Это символ конца строки (возврат каретки), который в консоли не отображается, но добавляет пустую строку.

В файле in\_out.asm реализованы подпрограммы для работы с числами и преобразования символов ASCII. Я модифицировала текст программы с использованием этих функций (рис. [[5](#fig:005)]).

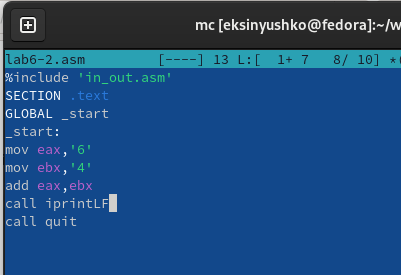


Figure 5: Изменение кода lab6-2.asm

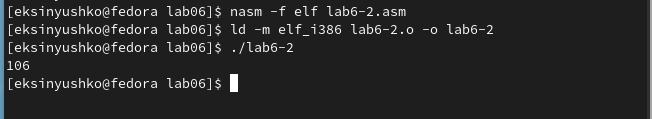


Figure 6: Компиляция текста программы lab6-2.asm

В результате выполнения обновленной программы было выведено число 106 (рис. [[6](#fig:006)]). Здесь, как и в первом случае, команда add складывает коды символов ‘6’ и ‘4’ (54 + 52 = 106). Но в отличие от предыдущей версии, функция iprintLF позволяет напечатать само число, а не символ с соответствующим кодом.

По аналогии с предыдущим примером, я заменила символы на числа (рис. [[7](#fig:007)]).

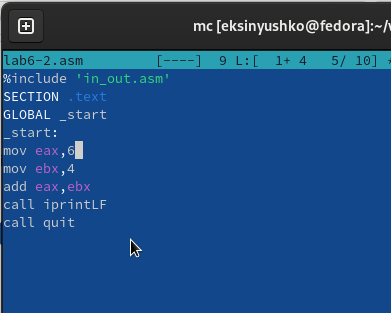


Figure 7: Изменение кода lab6-2.asm

Функция iprintLF позволяет выводить числа, и на этот раз в качестве операндов использовались именно числа, а не коды символов. В результате мы получили число 10 (рис. [[8](#fig:008)]).

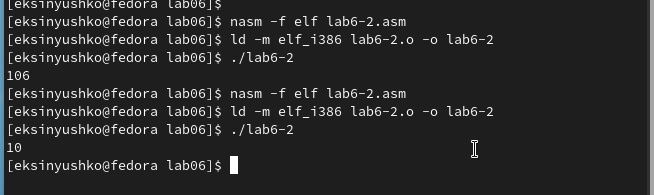


Figure 8: Компиляция текста программы lab6-2.asm

Далее я заменила функцию iprintLF на iprint, создала исполняемый файл и запустила его. Вывод теперь отличается отсутствием перехода на новую строку (рис. [[9](#fig:009)]).

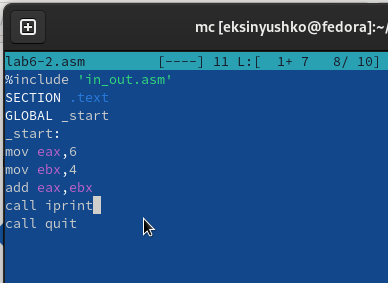


Figure 9: Изменение кода lab6-2.asm

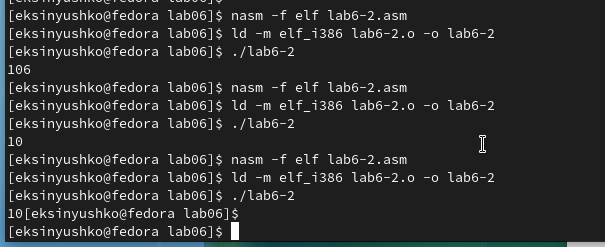


Figure 10: Компиляция текста программы lab6-2.asm

## 4.2 Выполнение арифметических операций в NASM

В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения

. (рис. [[11](#fig:011)]) (рис. [[12](#fig:012)])

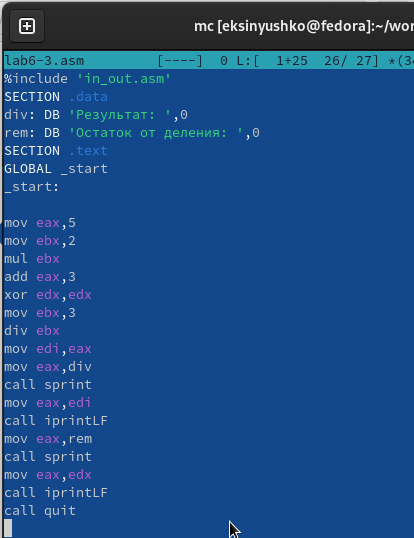


Figure 11: Изменение кода lab6-3.asm

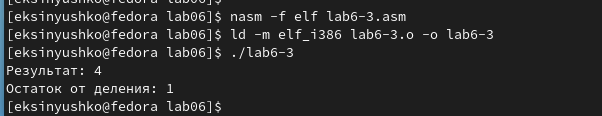


Figure 12: Компиляция текста программы lab6-3.asm

Изменила текст программы для вычисления выражения

. Создал исполняемый файл и проверил его работу. (рис. [[13](#fig:013)]) (рис. [[14](#fig:014)])

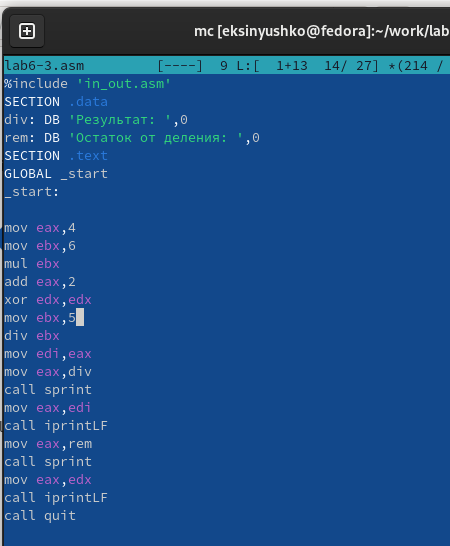


Figure 13: Изменение кода lab6-3.asm

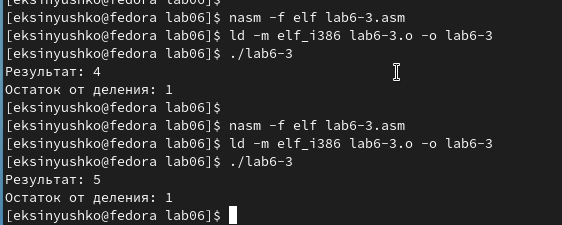


Figure 14: Компиляция текста программы lab6-3.asm

В качестве еще одного примера давайте рассмотрим программу для вычисления варианта задания на основе номера студенческого билета.

В этом случае число, над которым нужно выполнять арифметические операции, вводится с клавиатуры. Как я уже отмечала ранее, ввод с клавиатуры осуществляется в символьном виде. Для корректной работы арифметических операций в NASM эти символы необходимо преобразовать в числовой формат. С этой целью можно использовать функцию atoi из файла in\_out.asm (рис. [[15](#fig:015)]) (рис. [[16](#fig:016)]). Она конвертирует строку символов в эквивалентное decimal число.

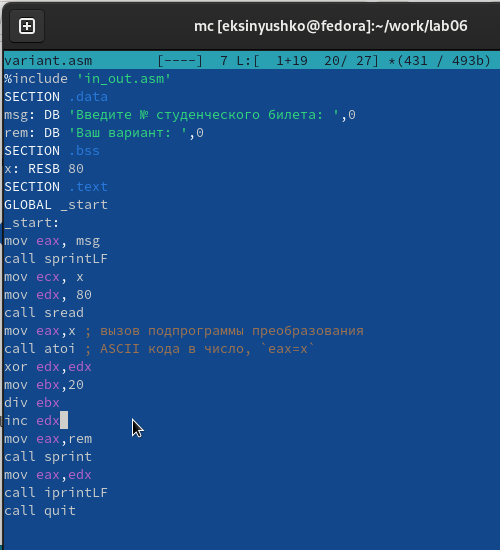


Figure 15: Изменение кода variant.asm

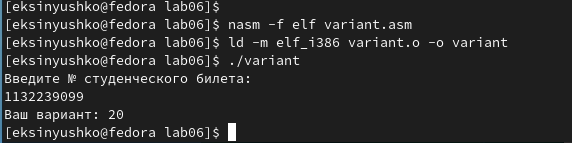


Figure 16: Компиляция текста программы variant.asm

### 4.2.1 Ответы на вопросы по программе variant.asm

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

Строки, отвечающие за вывод сообщения “Ваш вариант:”, - это mov eax, rem для помещения фразы в регистр eax и call sprint для вызова подпрограммы вывода строки.

1. Для чего используется следующие инструкции?

* mov ecx, x - сохранение регистра ecx в переменной x
* mov edx, 80 - присваивание значения 80 регистру edx
* call sread - вызов подпрограммы для считывания данных из консоли

1. Для чего используется инструкция “call atoi”?

Инструкция call atoi используется для преобразования введённых символов в числовой формат.

1. Какие строки листинга отвечают за вычисления варианта?

* xor edx, edx - обнуление регистра edx
* mov ebx, 20 - присваивание значения 20 регистру ebx
* div ebx - деление номера студента на 20
* inc edx - увеличение edx на 1

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

При выполнении div ebx остаток от деления помещается в регистр edx.

1. Для чего используется инструкция “inc edx”?

Инструкция inc edx увеличивает значение регистра edx на 1, что нужно для вычисления варианта по формуле.

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

* mov eax, edx - помещение результата в регистр eax
* call iprintLF - вызов подпрограммы вывода

## 4.3 Выполнение заданий для самостоятельной работы

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. (рис. [[17](#fig:017)]) (рис. [[18](#fig:018)]) Создайте исполняемый файл и проверьте его работу для значений x1 и x2 из 6.3.

Вариант 20 - для

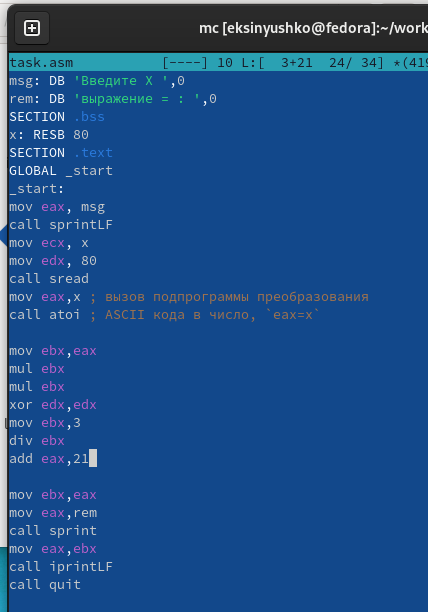


Figure 17: Изменение кода task.asm

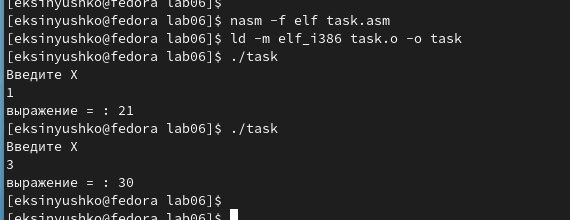


Figure 18: Компиляция текста программы task.asm

Программа считает верно.

# 5 Источники

1. Архитектура ЭВМ - Материалы курса