Отчёт по лабораторной работе №7. Арифметические операци в Nasm

дисциплина: Архитектура компьютера

Ибатулина Дарья Эдуардовна

Содержание

# 1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

# 2 Задание

1. Написание программ вывода символьных и численных значений;
2. Написание программ вычисления значений арифметичсеких выражений;
3. Выполнение заданий для самостоятельной работы.

# 3 Теоретическое введение

## 3.1 Адресация в Nasm

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. Далее рассмотрены все существующие способы задания адреса хранения операндов – способы адресации.

Существует три основных способа адресации:

* **Регистровая адресация** – операнды хранятся в регистрах и в команде используются имена этих регистров, например: mov ax,bx.
* **Непосредственная адресация** – значение операнда задается непосредственно в команде, Например: mov ax,2.
* **Адресация памяти** – операнд задает адрес в памяти. В команде указывается символическое обозначение ячейки памяти, над содержимым которой требуется выполнить операцию.

Например, определим переменную intg DD 3 – это означает, что задается область памяти размером 4 байта, адрес которой обозначен меткой intg. В таком случае, команда

mov eax,[intg]

копирует из памяти по адресу intg данные в регистр eax.

В свою очередь, команда

mov [intg],eax

запишет в память по адресу intg данные из регистра eax.

Также рассмотрим команду

mov eax,intg

В этом случае в регистр *eax* запишется адрес *intg*. Допустим, для *intg* выделена память начиная с ячейки с адресом 0x600144, тогда команда *mov eax,intg* аналогична команде *mov eax,0x600144* – т.е. эта команда запишет в регистр eax число 0x600144.

## 3.2 Арифметические операции в NASM

### 3.2.1 Целочисленное сложение add

Схема команды целочисленного сложения add (от англ. addition - добавление) выполняет сложение двух операндов и записывает результат по адресу первого операнда. Команда add работает как с числами со знаком, так и без знака и выглядит следующим образом:

add <операнд\_1>, <операнд\_2>

### 3.2.2 Целочисленное вычитание sub

Команда целочисленного вычитания sub (от англ. subtraction – вычитание) работает аналогично команде add и выглядит следующим образом:

sub <операнд\_1>, <операнд\_2>

### 3.2.3 Команды инкремента и декремента.

Довольно часто при написании программ встречается операция прибавления или вычитания единицы. Прибавление единицы называется инкрементом, а вычитание — декрементом. Для этих операций существуют специальные команды: *inc* (от англ. *increment*) и *dec* (от англ. *decrement*), которые увеличивают и уменьшают на 1 свой операнд.

Эти команды содержат один операнд и имеет следующий вид:

inc <операнд>  
dec <операнд>

### 3.2.4 Команда изменения знака операнда neg

Еще одна команда, которую можно отнести к арифметическим командам, это команда изменения знака neg:

neg <операнд>

### 3.2.5 Команды умножения mul и imul

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):

mul <операнд>

Для знакового умножения используется команда imul:

imul <операнд>

### 3.2.6 Команды деления div и idiv

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

div <делитель> ; Беззнаковое деление  
idiv <делитель> ; Знаковое деление

В командах указывается только один операнд – делитель, который может быть регистром или ячейкой памяти, но не может быть непосредственным операндом.

## 3.3 Перевод символа числа в десятичную символьную запись

Ввод информации с клавиатуры и вывод её на экран осуществляется в символьном виде. Кодирование этой информации производится согласно кодовой таблице символов ASCII. ASCII – сокращение от American Standard Code for Information Interchange (Американский стандартный код для обмена информацией). Согласно стандарту ASCII каждый символ кодируется одним байтом.

Среди инструкций NASM нет такой, которая выводит числа (не в символьном виде). Поэтому, например, чтобы вывести число, надо предварительно преобразовать его цифры в ASCII-коды этих цифр и выводить на экран эти коды, а не само число. Если же выводить число на экран непосредственно, то экран воспримет его не как число, а как последовательность ASCII-символов – каждый байт числа будет воспринят как один ASCII-символ – и выведет на экран эти символы.

Аналогичная ситуация происходит и при вводе данных с клавиатуры. Введенные данные будут представлять собой символы, что сделает невозможным получение корректного результата при выполнении над ними арифметических операций.

Для выполнения лабораторных работ в файле in\_out.asm реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Это:

* *iprint* – вывод на экран чисел в формате ASCII, перед вызовом iprint в регистр eax необходимо записать выводимое число (mov eax,).
* *iprintLF* – работает аналогично iprint, но при выводе на экран после числа добавляет к символ перевода строки.
* *atoi* – функция преобразует ascii-код символа в целое число и записает результат в регистр eax, перед вызовом atoi в регистр eax необходимо записать число (mov eax,).

# 4 Выполнение лабораторной работы

1. Создаём каталог для программ лабораторной работы № 7, переходим в него и создаём файл lab7-1.asm (рис. 1):

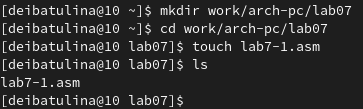


Рис. 1: Создание каталога для лабораторной работы и переход в него, создание файла

Скопируем внешний файл in\_out.asm в наш каталог для лабораторной работы, поскольку он будет использоваться в дальнейших программах (рис. 2):

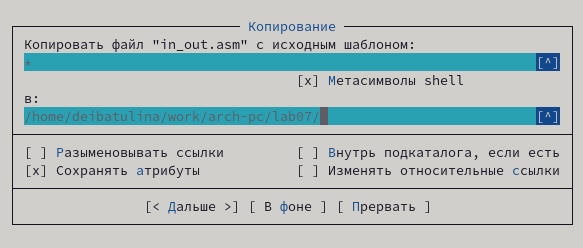


Рис. 2: Копирование внешнего файла в каталог с программами

1. Введём в файл lab7-1.asm текст программы из листинга (рис. 3), создадим исполняемый файл и запустим его (рис. 4):

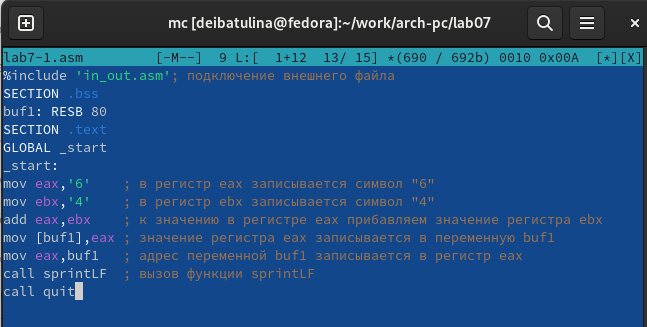


Рис. 3: Ввод текста программы в файл

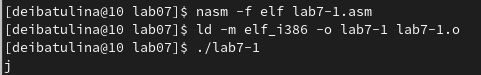


Рис. 4: Создание и запуск исполняемого файла

В данном случае при выводе значения регистра eax мы ожидаем увидеть число 10. Однако результатом будет символ j. Это происходит 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 запишет в регистр eax сумму кодов – 01101010 (106), что в свою очередь является кодом символа j.

1. Теперь изменим текст программы, заменив символы “6” и “4” на числа 6 и 4 (рис. 5). Создадим исполняемый файл и запустим его (рис. 6):

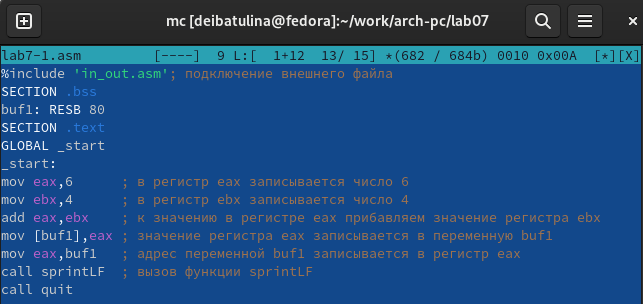


Рис. 5: Изменение текста программы в файле

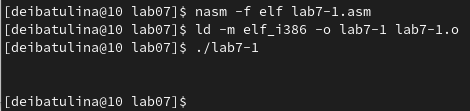


Рис. 6: Создание и запуск исполняемого файла

Как и в предыдущем случае при исполнении программы мы не получим число 10. В данном случае выводится символ с кодом 10. В соответствии с таблицей, приложенной к руководству по лабораторной работе №7, это символ перевода строки. При выводе на экран он, как видно по скриншоту, не отображается.

1. Создаём файл lab7-2.asm в каталоге ~/work/arch-pc/lab07 (рис. 7) и введите в него текст программы из листинга (рис. 8). Создаём исполняемый файл и запускаем его рис. (9):

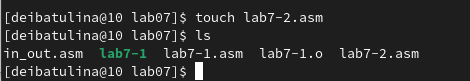


Рис. 7: Создание файла

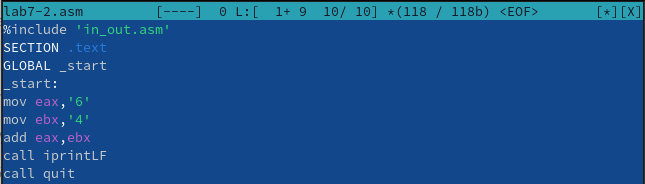


Рис. 8: Внесение изменений в файл

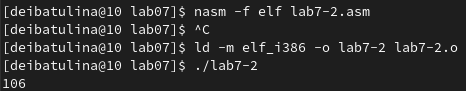


Рис. 9: Создание и запуск исполняемого файла

В результате работы программы мы получим число 106. В данном случае, как и в первом, команда *add* складывает коды символов ‘6’ и ‘4’ (54+52=106). Однако, в отличии от предыдущей программы, функция *iprintLF* позволяет вывести число, а не символ, кодом которого является это число.

1. Аналогично предыдущему примеру изменим символы на числа (рис. 10). Создадим исполняемый файл и запустим его рис. (11):

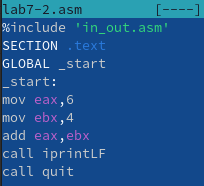


Рис. 10: Изменение текста программы

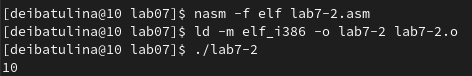


Рис. 11: Изменение текста программы

Заменим функцию iprintLF на iprint (рис. 12). Создаём исполняемый файл и запускаем его (рис. 13). Выводы функций iprintLF и iprint отличаются тем, что iprintlF делает перевод на новую строку, а iprint - нет.

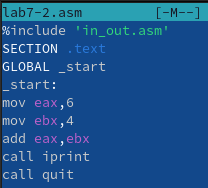


Рис. 12: Изменение текста программы

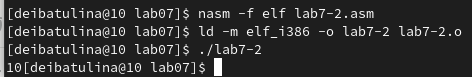


Рис. 13: Создание и запуск исполняемого файла

1. Создаём файл lab7-3.asm в каталоге ~/work/arch-pc/lab07 (рис. 14):

Рис. 14: Создание файла

Рис. 14: Создание файла

Вводим в данный файл текст программы из листинга (рис. 15). Данная программа вычисляет значение арифметического выражения (5 \* 2 + 3) / 3 (рис. 16):

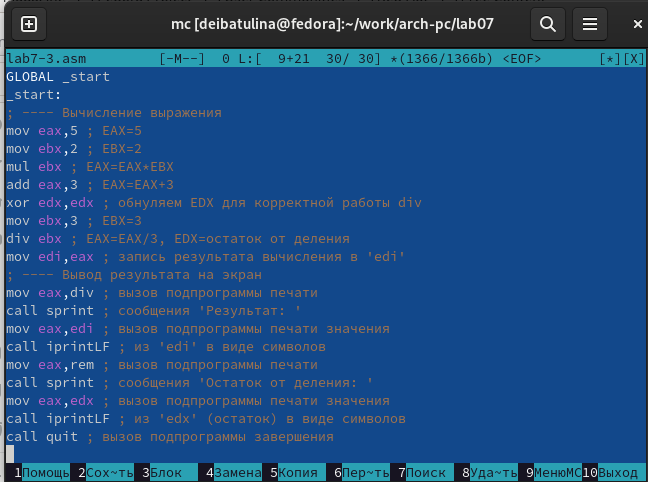


Рис. 15: Ввод текста программы для вычисления арифметического выражения

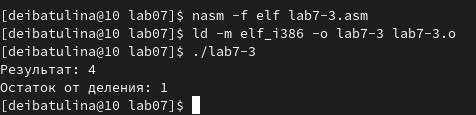


Рис. 16: Создание и запуск исполняемого файла

Вывод программы корректен.

Теперь заменим некоторые числа в этой программе (рис. 17): так, теперь она будет вычислять значение выражения (4 \* 6 + 2) / 5 (рис. 18):

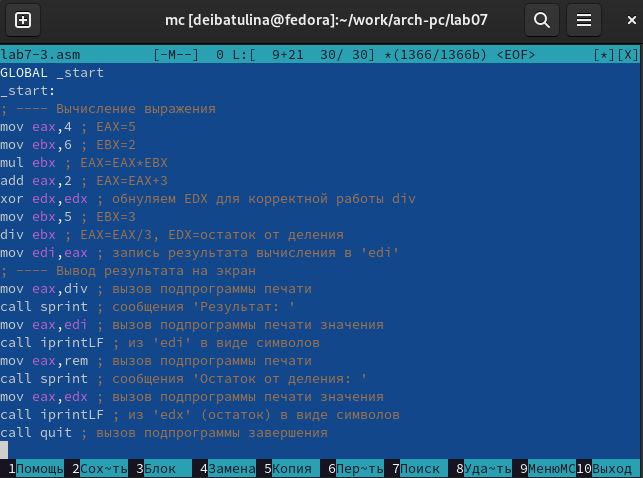


Рис. 17: Внесение изменений в текст программы

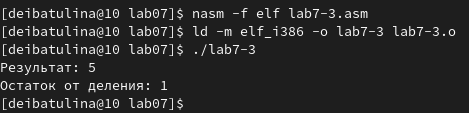


Рис. 18: Создание и запуск исполняемого файла

1. Создаём файл variant.asm в каталоге ~/work/arch-pc/lab07 (рис. 19):

Рис. 19: Создание файла

Рис. 19: Создание файла

Ввожу в файл текст программы (рис. 20):

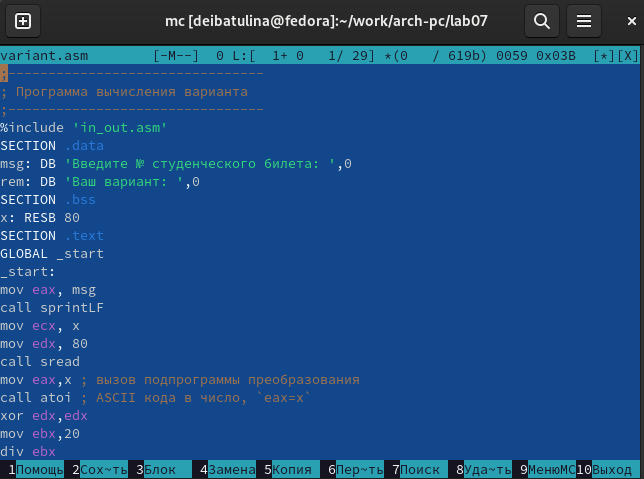


Рис. 20: Ввод текста программы

Создаём исполняемый файл и, вычислив номер варианта аналитически, приходим к выводу, что программа вычислила номер варианта корректно (рис. 21):

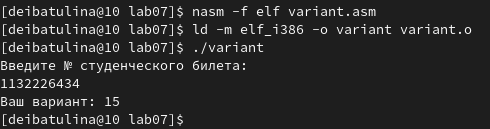


Рис. 21: Создание и запуск исполняемого файла

## 4.1 Ответы на вопросы

1. За вывод на экран сообщения “Ваш вариант” отвечают следующие строки кода программы:

mov eax,rem  
 call sprint

1. Инструкция *mov ecx,x* используется для помещения адреса вводимого значения x в регистр ecx. Инструкция *mov edx,80* предназначена для помещения длины вводимого с клавиатуры значения. Инструкция call sread относится ко внешнему файлу и позволяет прочитать введённое пользователем значение.
2. Функция *atoi* преобразует ASCII-код символа в целое число и записает результат в регистр eax, перед вызовом atoi в регистр eax необходимо записать число (*mov eax,*).
3. За вычисление номера варианта отвечают следующие строки кода программы:

xor edx,edx ; обнуляем edx для корректной работы div  
mov ebx,20 ; помещаем в регистр ebx число 20: ebx = 20  
div ebx ; eax = eax / 20  
inc edx ; edx = edx + 1

1. При выполнении инструкции *div ebx* остаток от деления записывается в регистр *edx*.
2. Инструкция *inc edx* увеличивает значение регистра *edx* на 1. Прибавление единицы называется инкрементом.
3. За вывод результата вычисления на экран отвечают следующие сроки кода программы:

mov eax,edx  
call iprintLF

## 4.2 Выполнение заданий для самостоятельной работы

1. В соответствии с номером моего варианта необходимо написать программу для вычисления значения арифметического выражения. Мой вариант №15, f(x) = (5 + x) ^ 2 - 3. Создаю файл для данной программы (рис. 22, ввожу текст программы (рис. 23, создаю и запускаю исполняемый файл, проверяю работу программы для заданных значений x: 5 и 1 (рис. 24):

Рис. 22: Создание файла

Рис. 22: Создание файла

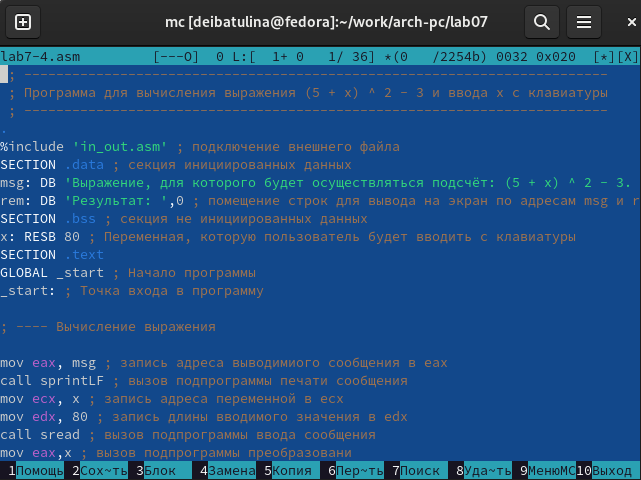


Рис. 23: Ввод кода программы в файл

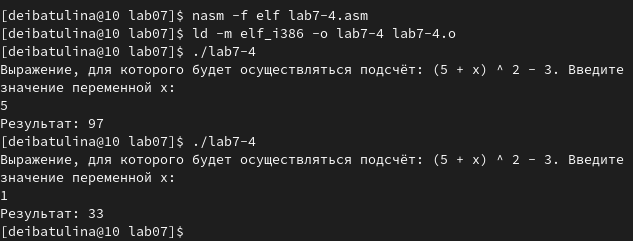


Рис. 24: Создание и запуск исполняемого файла

Приведу ниже листинг программы для вычисления значения арифметического выражения (5 + x) ^ 2 - 3:

**Листинг программы для вычисления значения арифметического выражения (5 + x) ^ 2 - 3 (Вариант 15)**

; --------------------------------------------------------------------------  
; Программа для вычисления выражения (5 + x) ^ 2 - 3 и ввода х с клавиатуры  
; --------------------------------------------------------------------------  
   
%include 'in\_out.asm' ; подключение внешнего файла  
SECTION .data ; секция инициированных данных  
msg: DB 'Выражение, для которого будет осуществляться подсчёт: (5 + x) ^ 2 - 3.   
 Введите значение переменной х: ',0  
rem: DB 'Результат: ',0 ; помещение строк для вывода на экран по адресам msg и rem  
SECTION .bss ; секция не инициированных данных  
x: RESB 80 ; Переменная, которую пользователь будет вводить с клавиатуры  
SECTION .text  
GLOBAL \_start ; Начало программы  
\_start: ; Точка входа в программу  
  
; ---- Вычисление выражения  
  
mov eax, msg ; запись адреса выводимиого сообщения в eax  
call sprintLF ; вызов подпрограммы печати сообщения  
mov ecx, x ; запись адреса переменной в ecx  
mov edx, 80 ; запись длины вводимого значения в edx  
call sread ; вызов подпрограммы ввода сообщения  
mov eax,x ; вызов подпрограммы преобразовани  
call atoi ; ASCII кода в число, `eax = x`  
add eax,5; eax = eax + 5 = x + 5  
mul eax ; возведение выражения (х + 5) в квадрат  
add eax,-3 ; eax = eax - 3 = (x + 5) ^ 2 - 3  
mov edi,eax ; запись результата вычисления в 'edi'  
  
; ---- Вывод результата на экран  
  
mov eax,rem ; вызов подпрограммы печати  
call sprint ; сообщения 'Результат: '  
mov eax,edi ; вызов подпрограммы печати значения  
call iprintLF ; из 'edi' в виде символов  
call quit ; вызов подпрограммы завершения

Выполнив подсчёты аналитически, я пришла к выводу, что программа работает корректно.

# 5 Выводы

При выполнении лабораторной работы я освоила арифметические инструкции языка Ассемблер.

# Список литературы

* Руководство по выполнению лабораторной работы №7.