Отчёт по лабораторной работе №7

Дисциплина: Архитектура компьютера

Канева Екатерина Павловна

Содержание

# 1 Цель работы

Целью лабораторной работы является освоение арифметических инструкций языка ассемблера NASM.

# 2 Задание

1. 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 вариантов заданий в соответствии с номером полученным при выполнении лабораторной работы.
2. Создать исполняемый файл и проверить его работу для значений и .

# 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. 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.

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>

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

sub <операнд\_1>, <операнд\_2>

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>

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

neg <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):

mul <операнд>

Для знакового умножения используется команда imul:

imul <операнд>

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

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

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,<int>).
* 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 записывает результат в регистр eax, перед вызовом atoi в регистр eax необходимо записать число (mov eax,<int>).

# 4 Выполнение лабораторной работы

## 4.1 Лабораторная работа

Создадим каталог для программам лабораторной работы №7, перейдём в него и создадим файл lab7-1.asm (рис. 1):

mkdir work/study/2022-2023/"Архитектура компьютера"/arh-pc/lab07  
cd work/study/2022-2023/"Архитектура компьютера"/arh-pc/lab07  
touch lab7-1.asm

Рис. 1: Создание каталога и файла.

Рис. 1: Создание каталога и файла.

В файл lab7-1.asm введём текст программы, указанный в лабораторной работе (рис. 2):

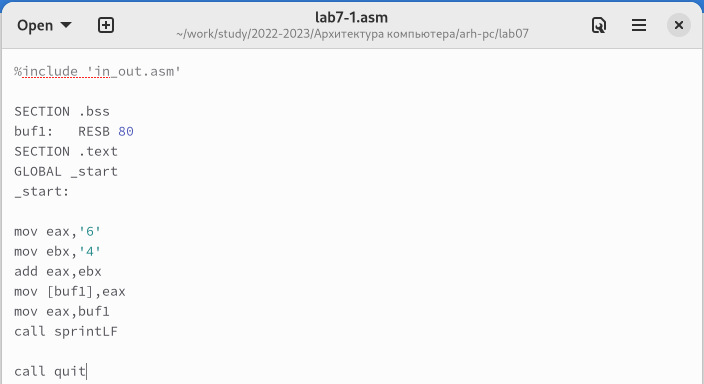


Рис. 2: Ввод текста программы.

Создадим исполняемый файл и запустим его (рис. 3):

nasm -f elf lab7-1.asm  
ld -m elf\_i386 -o lab7-1 lab7-1.o  
./lab7-1

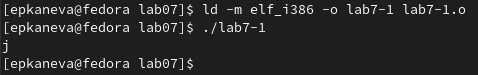


Рис. 3: Запуск программы.

Видим, что программа вывела символ j.

Далее изменим текст программы и вместо символов запишем в регистры числа. Исправим текст программы (рис. 4):



Рис. 4: Изменение текста программы.

Создадим исполняемый файл и запустим его (рис. 5):

nasm -f elf lab7-1.asm  
ld -m elf\_i386 -o lab7-1 lab7-1.o  
./lab7-1

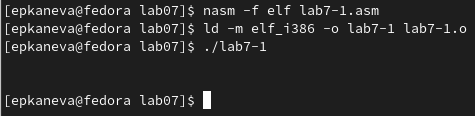


Рис. 5: Запуск программы.

Программа выводит пустой символ. Это соответствует десятому символу в таблице ASCII.

Создадим файл lab7-2.asm (рис. 6) и введём текст программы (рис. 7):

touch lab7-2.asm

Рис. 6: Создание файла.

Рис. 6: Создание файла.



Рис. 7: Ввод текста программы.

Создадим исполняемый файл и запустим его (рис. 8):

nasm -f elf lab7-2.asm  
ld -m elf\_i386 -o lab7-2 lab7-2.o  
./lab7-2

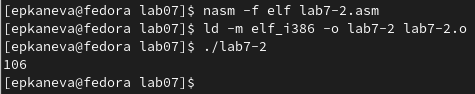


Рис. 8: Запуск программы.

На экран вывелось число 106.

Теперь изменим символы на числа, как ранее (рис. 9) и запустим программу (рис. 10):

nasm -f elf lab7-2.asm  
ld -m elf\_i386 -o lab7-2 lab7-2.o  
./lab7-2



Рис. 9: Изменение текста программы.

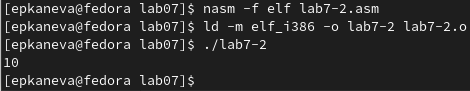


Рис. 10: Запуск программы.

Заменим функцию iprintLF на iprint (рис. 11) и запустим программу (рис. 12):

nasm -f elf lab7-2.asm  
ld -m elf\_i386 -o lab7-2 lab7-2.o  
./lab7-2



Рис. 11: Изменение текста программы.

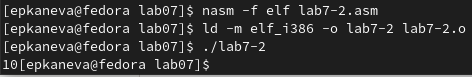


Рис. 12: Запуск программы.

Программа вывела число 10 и в той же строке приглашение для следующей команды. Вывод функций отличается тем, что функция iprintLF выполняет переход на новую строку после вывода, а iprint нет.

Создадим файл lab7-3.asm (рис. 13), в него введём текст программы из лабораторной работы (рис. 14):

touch lab7-3.asm

Рис. 13: Создание файла.

Рис. 13: Создание файла.

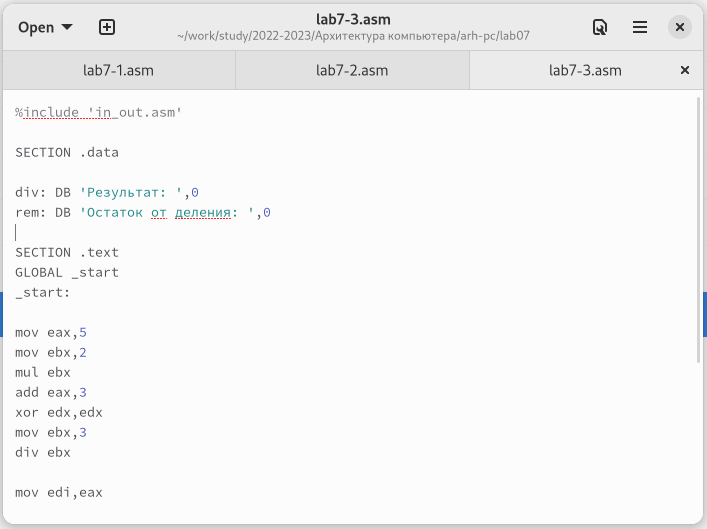


Рис. 14: Ввод текста программы.

Создадим исполняемый файл и запустим его (рис. 15):

nasm -f elf lab7-3.asm  
ld -m elf\_i386 -o lab7-3 lab7-3.o  
./lab7-3

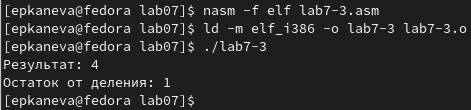


Рис. 15: Запуск программы.

Программа работает корректно.

Далее создадим файл variant.asm (рис. 16):

touch variant.asm

Рис. 16: Создание файла.

Рис. 16: Создание файла.

Введём текст программы вычисления номера варианта по номеру студенческого билета (рис. 17):

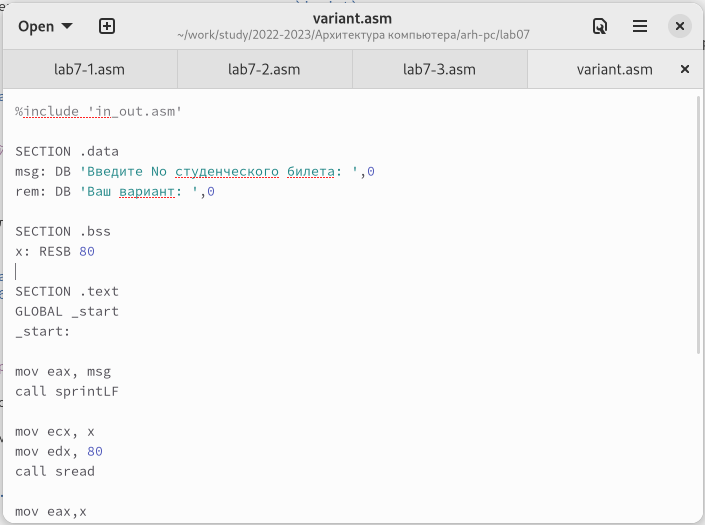


Рис. 17: Ввод текста программы.

Создадим исполняемый файл и запустим его (рис. 18):

nasm -f elf variant.asm  
ld -m elf\_i386 -o variant variant.o  
./variant

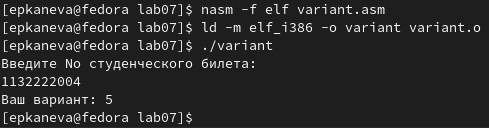


Рис. 18: Запуск программы.

Программа вывела номер 5. Программа вычисляет вариант, прибавляя к остатку от деления номера варианта на 20 1 – результат действительно должен получиться таким.

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

Строки, отвечающите за вывод этого сообщения на экран:

mov eax,rem  
call sprint

1. Для чего используется следующие инструкции?

mov ecx, x  
mov edx, 80   
call sread

Эти строки отвечают за чтение вводимого номера студенческого билета.

1. Для чего используется инструкция “call atoi”?

Инструкция call atoi вызывает функцию atoi, которая преобразует ASCII-код символа в целое число и записывает результат в регистр eax.

1. Какие строки отвечают за вычисление варианта?

Строки, отвечающие за вычисление варианта:

xor edx,edx  
mov ebx,20  
div ebx  
inc edx

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

В регистр edx.

1. Для чего используется инструкция “inc edx”?

Для увеличения значения, полученного при взятии остатка, на 1.

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

Строки, отвечающие за вывод результата со словами “Ваш вариант:”:

mov eax,rem  
call sprint  
mov eax,edx  
call iprintLF

## 4.2 Самостоятельная работа

Для написания программы создадим файл function.asm (рис. 19):

touch function.asm

Рис. 19: Создание файла.

Рис. 19: Создание файла.

Введём текст программы (рис. 20):

%include 'in\_out.asm'  
  
SECTION .data  
stm: DB 'y = (9x - 8) / 8', 0  
msg: DB 'Введите значение x: ', 0  
res: DB 'Результат вычислений: ', 0  
  
SECTION .bss  
x: RESB 80  
  
SECTION .text  
GLOBAL \_start  
\_start:  
  
mov eax, stm  
call sprintLF  
mov eax, msg  
call sprintLF  
  
mov ecx, x  
mov edx, 80  
call sread  
  
mov eax, x  
call atoi  
  
mov ebx, 12  
mul ebx  
add eax, 3  
xor edx, edx  
mov ebx, 5  
idiv ebx  
mov edi, eax  
  
mov eax, res  
call sprint  
mov eax, edi  
call iprintLF  
  
call quit

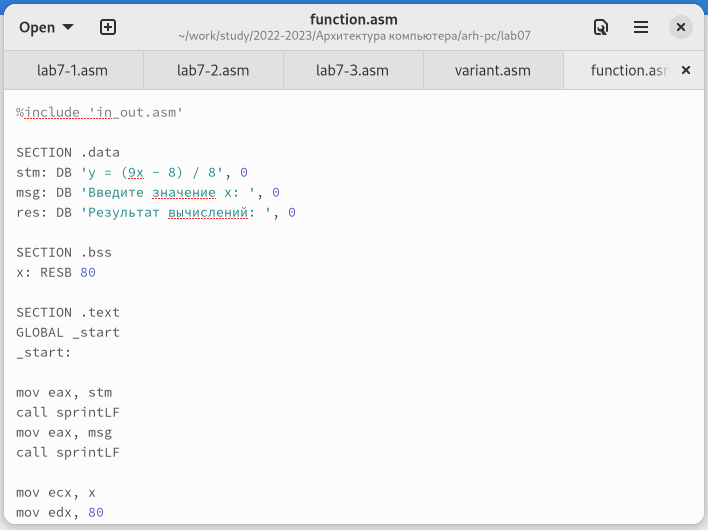


Рис. 20: Ввод текста программы.

Создадим исполняемый файл и запустим его (рис. 21 и 22):

nasm -f elf function.asm  
ld -m elf\_i386 -o function function.o  
./function

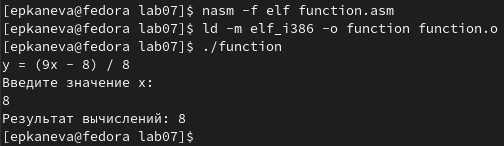


Рис. 21: Запуск программы со значением 8.

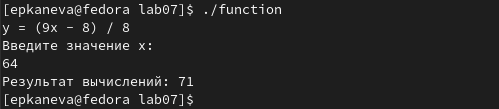


Рис. 22: Запуск программы со значением 64.

Проверим вычисления, выполненные программой:

(9 \* 8 - 8) / 8 = 8 (9 \* 64 - 8) / 8 = 71

Программа работает корректно.

# 5 Выводы

Освоили арифметические инструкции языка ассемблера NASM.