Отчёт по лабораторной работе №6

Дисциплина: архитектура компьютера

Черкашина Ангелина Максимовна

Содержание

# 1 Цель работы

Цель данной лабораторной работы - освоение арифметических инструкций языка ассемблера NASM.

# 2 Задание

1. Символьные и численные данные в NASM
2. Выполнение арифметических операций в NASM
3. 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. - 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. 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 решения этой проблемы необходимо проводить преобразование ASCII символов в числа и обратно.

# 4 Выполнение лабораторной работы

## 4.1 Символьные и численные данные в NASM

С помощью команды mkdir создаю каталог для программ данной лабораторной работы, перехожу в созданный каталог с помощьб cd (рис. 1).

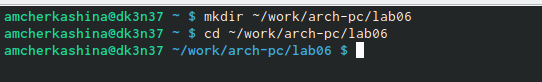


Рис. 1: Создание директории

С помощью команды touch создаю файл lab6-1.asm (рис. 2).

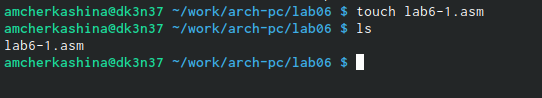


Рис. 2: Создание файла

С помощью команды cp копирую в текущий каталог файл in\_out.asm, т.к. он будет использоваться в других программах (рис. 3).

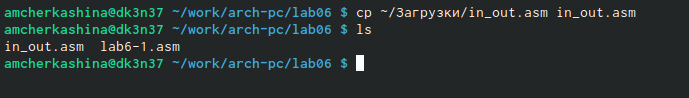


Рис. 3: Создание копии файла

Открываю созданный файл lab6-1.asm, вставляю в него программу вывода значения регистра eax (рис. 4).



Рис. 4: Редактирование файла

Создаю исполняемый файл программы и запускаю его (рис. 5). Вывод программы: 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 по кодовой таблице ASCII.

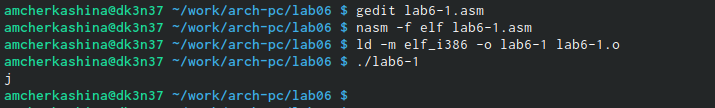


Рис. 5: Запуск исполняемого файла

Изменяю текст программы и вместо символов записываю в регистры числа, т.е. исправляю символы “6” и “4” на числа 6 и 4 (рис. 6).



Рис. 6: Изменение текста программы

Создаю новый исполняемый файл программы и запускаю его. Теперь вывелся символ с кодом 10, согласно таблице ASCII код 10 соответствует символу перевода строки. Поэтому этот символ не отображается при выводе на экран (рис. 7).

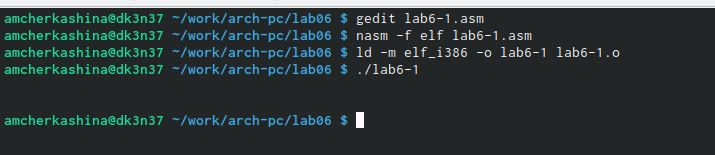


Рис. 7: Запуск нового исполняемого файла

Создаю в каталоге ~/work/arch-pc/lab06 новый файл lab6-2.asm с помощью команды touch (рис. 8).

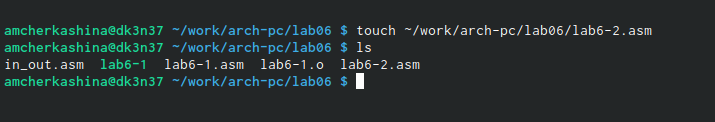


Рис. 8: Создание файла

Ввожу в созданный файл текст другой программы для вывода значения регистра eax (рис. 9).



Рис. 9: Редактирование файла

Создаю и запускаю исполняемый файл lab6-2 (рис. 10). В результате работы программы получается число 106. В данном случае, как и в первом, команда add складывает коды символов ‘6’ и ‘4’ (54+52=106). Однако, в отличии от программы файла lab6-1, функция iprintLF позволяет вывести число, а не символ, кодом которого является это число.

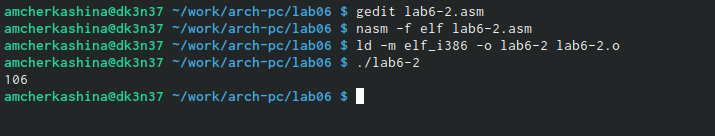


Рис. 10: Запуск исполняемого файла

Аналогично предыдущей программе корректирую текст программы файла lab6-2 и меняю символы на числа (исправляю символы “6” и “4” на числа 6 и 4) (рис. 11).



Рис. 11: Изменение текста программы

Создаю и запускаю новый исполняемый файл. Теперь программа складывает не соответствующие символам коды в системе ASCII, а сами числа, поэтому вывод 10 (рис. 12).

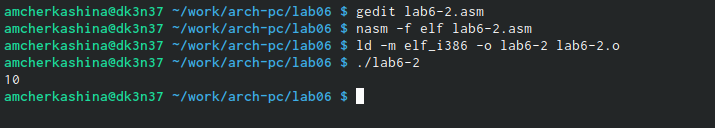


Рис. 12: Запуск нового исполняемого файла

Заменяю в тексте программы функцию iprintLF на iprint (рис. 13).



Рис. 13: Изменение текста программы

Создаю и запускаю новый исполняемый файл (рис. 14). Функция iprint не добавляет к выводу символ переноса строки, в отличие от iprintLF.

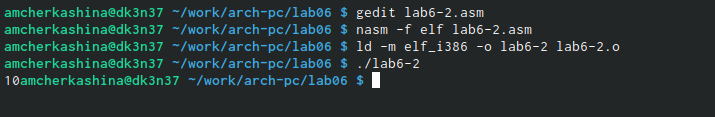


Рис. 14: Запуск нового исполняемого файла

## 4.2 Выполнение арифметических операций в NASM

С помощью команды touch создаю файл lab6-3.asm в каталоге ~/work/arch-pc/lab06 (рис. 15).

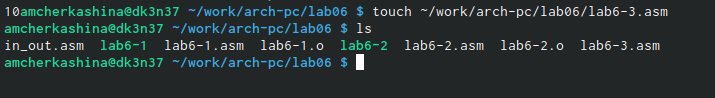


Рис. 15: Создание файла

Ввожу в созданный файл текст программы вычисления значения выражения f(x) = (5 \* 2 + 3)/3 (рис. 16).

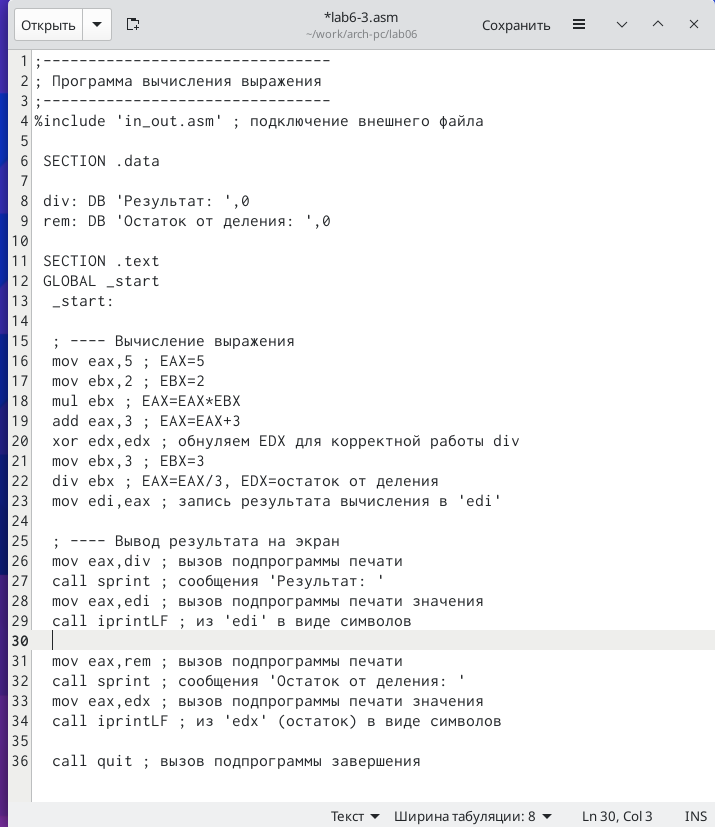


Рис. 16: Редактирование файла

Создаю исполняемый файл и запускаю его (рис. 17).

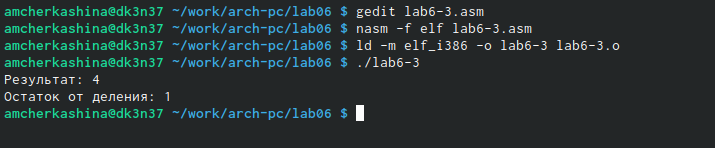


Рис. 17: Запуск исполняемого файла

Изменяю текст программы так, чтобы она вычисляла значение выражения f(x) = (4 \* 6 + 2)/5 (рис. 18).

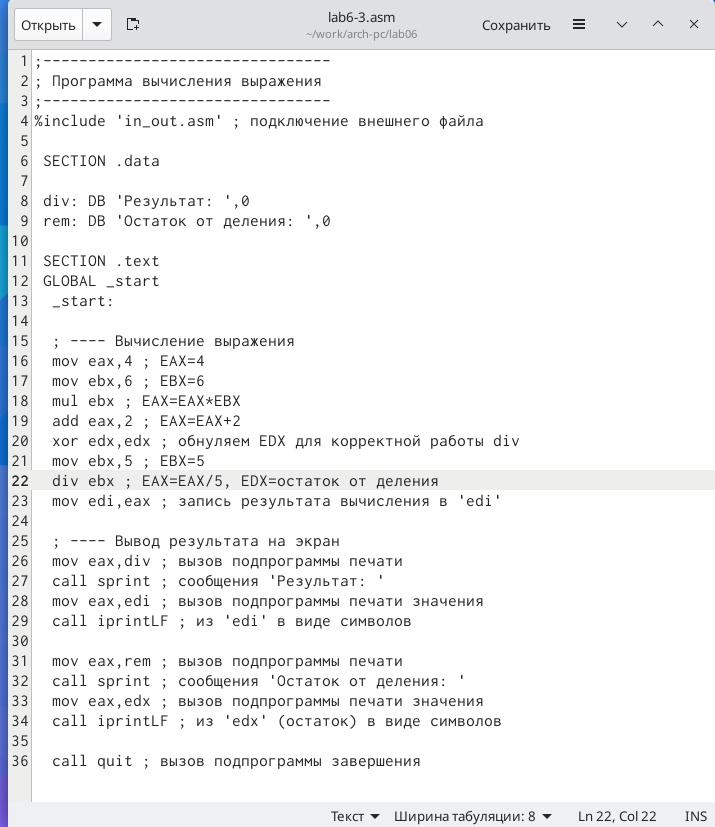


Рис. 18: Изменение программы

Создаю новый исполняемый файл и запускаю его. Проверяю его работу, посчитав значение выражения самостоятельно. Убеждаюсь, что программа сработала верно (рис. 19).

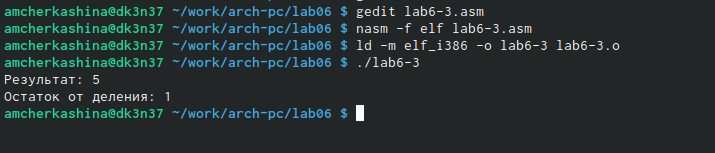


Рис. 19: Запуск нового исполняемого файла

Создаю файл variant.asm в каталоге ~/work/arch-pc/lab06 с помощью команды touch (рис. 20).

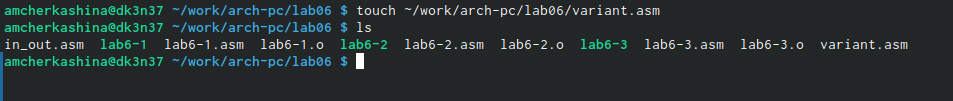


Рис. 20: Создание файла

Ввожу в созданный файл текст программы вычисления варианта задания по номеру студенческого билета (рис. 21).

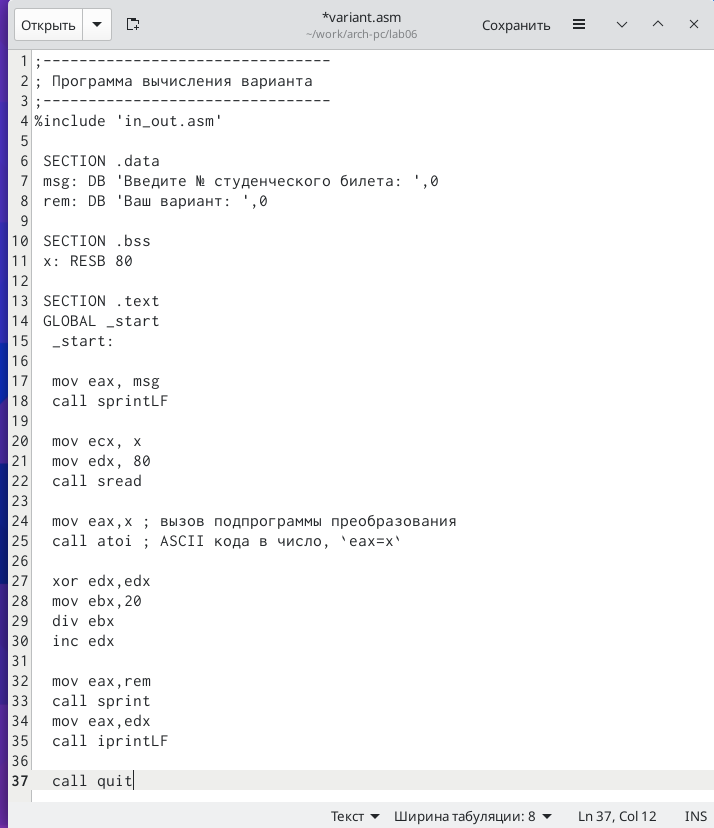


Рис. 21: Редактирование файла

Создаю и запускаю исполняемый файл. Ввожу номер своего студенческого билета с клавиатуры, программа выводит мой вариант - 11. Проверяю результат работы программы, вычислив номер своего варианта аналитически. Убеждаюсь, что программа сработала верно (рис. 22).

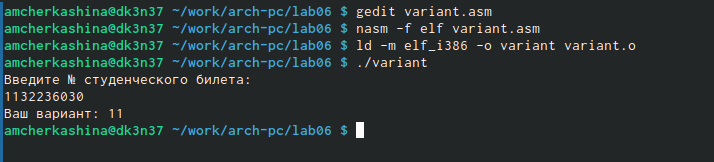


Рис. 22: Запуск исполняемого файла

### 4.2.1 Ответы на вопросы по программе

1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’?

За вывод сообщения “Ваш вариант” отвечают следующие строки кода:

mov eax,rem  
call sprint

1. Для чего используется следующие инструкции?

mov ecx, x  
mov edx, 80  
call sread

Инструкция mov ecx, x используется, чтобы положить адрес вводимой строки x в регистр ecx; mov edx, 80 - запись в регистр edx длины вводимой строки; call sread - вызов подпрограммы из внешнего файла, обеспечивающей ввод сообщения с клавиатуры.

1. Для чего используется инструкция “call atoi”?

Инструкция call atoi используется для вызова подпрограммы из внешнего файла, которая преобразует ASCII-код символа в целое число и записывает результат в регистр eax.

1. Какие строки листинга 6.4 отвечают за вычисления варианта?

За вычисления варианта отвечают следующие строки кода:

xor edx,edx ; обнуление edx для корректной работы div  
mov ebx,20 ; ebx = 20  
div ebx ; eax = eax/20, edx - остаток от деления  
inc edx ; edx = edx + 1

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

При выполнении инструкции div ebx остаток от деления записывается в регистр edx.

1. Для чего используется инструкция “inc edx”?

Инструкция inc edx увеличивает значение регистра edx на 1.

1. Какие строки листинга 6.4 отвечают за вывод на экран результата вычислений?

За вывод на экран результата вычислений отвечают следующие строки:

mov eax,edx  
call iprintLF

## 4.3 Выполнение заданий для самостоятельной работы

Создаю файл lab6-4.asm с помощью команды touch (рис. 23).

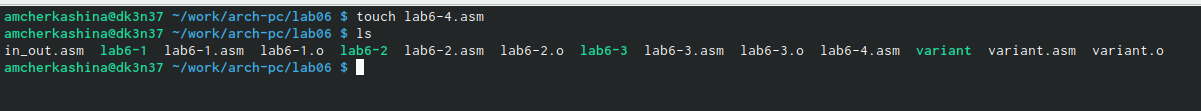


Рис. 23: Создание файла

Открываю созданный файл для редактирования, ввожу в него текст программы для вычисления значения выражения 10\*(x + 1)-10 (вариант 11) (рис. 24).

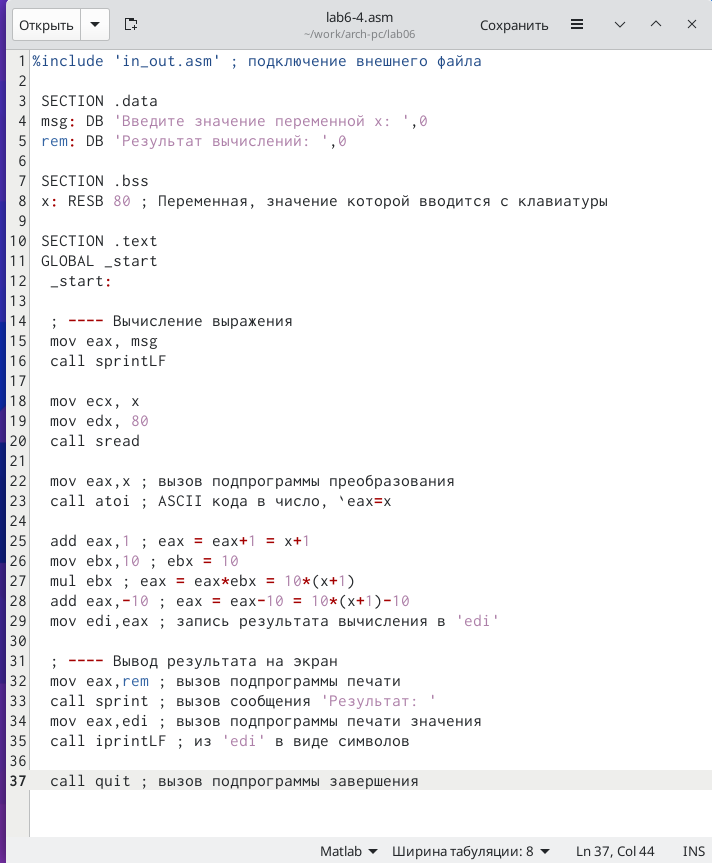


Рис. 24: Написание программы

Создаю и запускаю исполняемый файл. Ввожу значение x1 = 1 с клавиатуры. В результате выполнения программы получаю число 10 (рис. 25). Проверяю результат работы программы, подсчитав значение выражения самостоятельно. Убеждаюсь, что программа сработала верно.

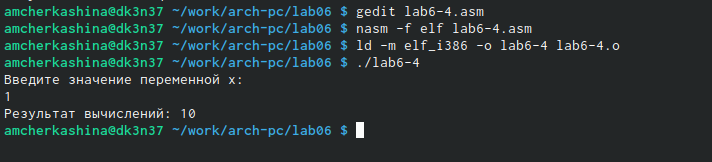


Рис. 25: Запуск исполняемого файла

Снова запускаю исполняемый файл для проверки работы программы с другим значением на вводе (x2 = 7). Программа выводит результат 70 (рис. 26). Проверяю результат работы программы, подсчитав значение выражения самостоятельно. Убеждаюсь, что программа сработала верно.

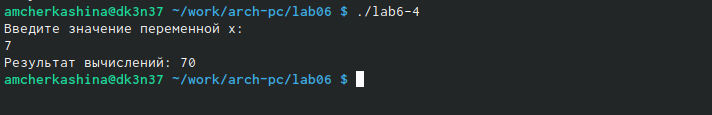


Рис. 26: Запуск исполняемого файла

Листинг 4.1. Программа для вычисления значения выражения 10\*(x + 1)-10:

%include 'in\_out.asm' ; подключение внешнего файла  
  
 SECTION .data  
 msg: DB 'Введите значение переменной x: ',0  
 rem: DB 'Результат вычислений: ',0  
   
 SECTION .bss  
 x: RESB 80 ; Переменная, значение которой вводится с клавиатуры  
   
 SECTION .text  
 GLOBAL \_start  
 \_start:  
   
 ; ---- Вычисление выражения  
 mov eax, msg  
 call sprintLF  
  
 mov ecx, x  
 mov edx, 80  
 call sread  
  
 mov eax,x ; вызов подпрограммы преобразования  
 call atoi ; ASCII кода в число, `eax=x  
  
 add eax,1 ; eax = eax+1 = x+1  
 mov ebx,10 ; ebx = 10  
 mul ebx ; eax = eax\*ebx = 10\*(x+1)  
 add eax,-10 ; eax = eax-10 = 10\*(x+1)-10  
 mov edi,eax ; запись результата вычисления в 'edi'  
  
 ; ---- Вывод результата на экран  
 mov eax,rem ; вызов подпрограммы печати  
 call sprint ; вызов сообщения 'Результат: '  
 mov eax,edi ; вызов подпрограммы печати значения  
 call iprintLF ; из 'edi' в виде символов  
  
 call quit ; вызов подпрограммы завершения

# 5 Выводы

При выполнении данной лабораторной работы я освоила арифметические инструкции языка ассемблера NASM.

# Список литературы

1. Архитектура ЭВМ
2. Таблица ASCII