Арифметические операции в NASM

Лабораторная работа №6

Губайдуллина Софья Романовна

Содержание

# 1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

# 2 Задание

1. Символьные и численные данные в NASM
2. Выполнение арифметических операций в NASM
3. Ответы на вопросы по лабораторной работе
4. Задание 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.

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 , .

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

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 и dec, которые увеличивают и уменьшают на 1 свой операнд.

Еще одна команда, которую можно отнести к арифметическим командам это команда изменения знака neg: neg Команда neg рассматривает свой операнд как число со знаком и меняет знак операнда на противоположный. Операндом может быть регистр или ячейка памяти любого размера.

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 Для знакового умножения используется команда imul: imul

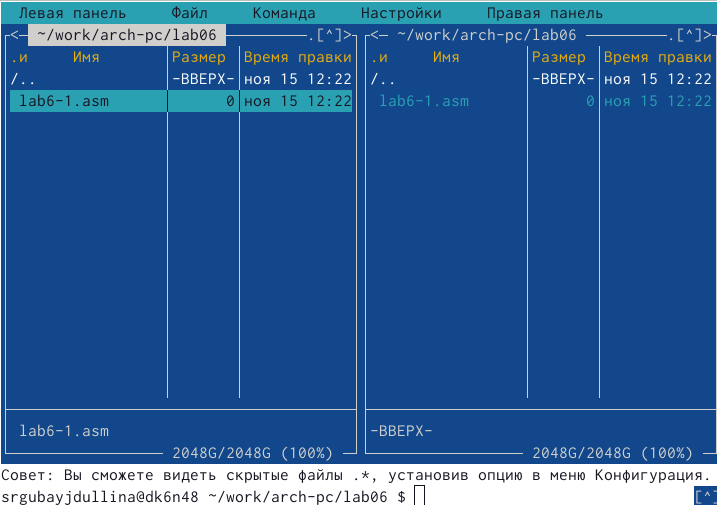
Для деления, как и для умножения, существует 2 команды div (от англ. divide - деление) и idiv: div ; Беззнаковое деление idiv ; Знаковое деление

Ввод информации с клавиатуры и вывод её на экран осуществляется в символьном виде. Кодирование этой информации производится согласно кодовой таблице символов ASCII.Согласно стандарту ASCII каждый символ кодируется одним байтом. Первая часть таблицы (символы с кодами 0-127) является универсальной (см. Приложение.), а вторая (коды 128-255) предназначена для специальных символов и букв национальных алфавитов и на компьютерах разных типов может меняться.

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. Начинаю выполнение своей лабораторной работы с создания нового каталога, с которым мне в дальнейшем придется работать, после чего перехожу в него и создаю нужный мне файл lab6-1.asm (рис. ??).

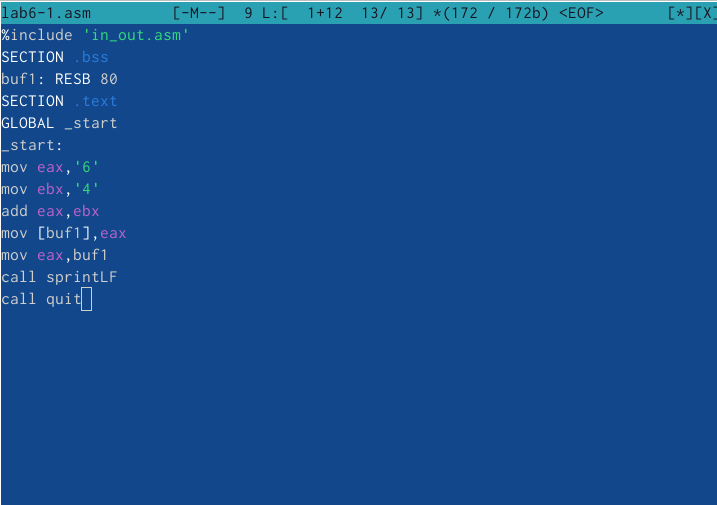


Создания новых каталога и файла lab6-1.asm

В lab6-1.asm при помощи F4 ввожу следующий листинг:

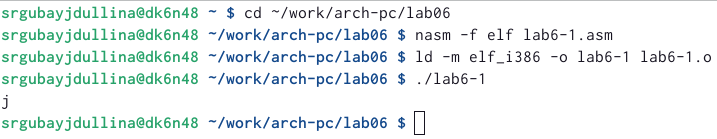
%include ‘in\_out.asm’ SECTION .bss buf1: RESB 80 SECTION .text GLOBAL \_start \_start: mov eax,‘6’ mov ebx,‘4’ add eax,ebx mov [buf1],eax mov eax,buf1 call sprintLF call quit

В данной программе в регистр eax записывается символ 6 (mov eax,‘6’), в регистр ebx символ 4 (mov ebx,‘4’), после чего к значению в регистре eax прибавляется значение регистра ebx (add eax,ebx, результат сложения запишется в регистр eax). (рис. ??)



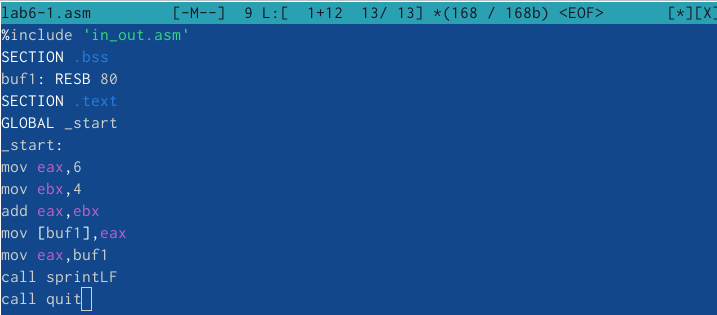
Запись листинга в файл lab6-1.asm

Создаю через терминал исполняемый файл файла lab6-1.asm и запускаю его, получая необходимый результат (рис. ??).



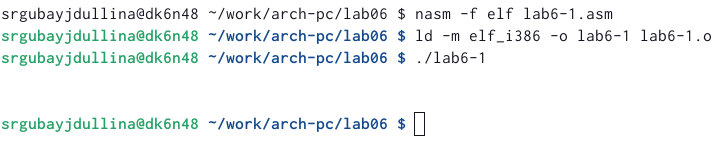
Создание и запуск исполняемого файла lab6-1.asm

Далее по заданию изменяю текст программы и вместо символов записываю в регистры числа следующим образом: Меняю mov eax,‘6’ mov ebx,‘4’ на строки mov eax,6 mov ebx,4 (рис. ??)



Замена строчек в программе

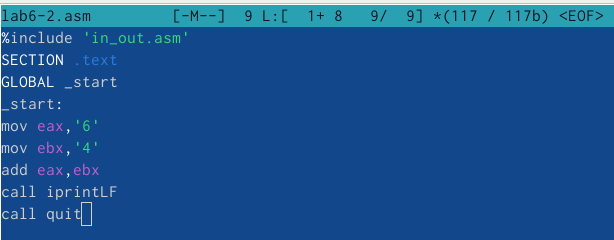
Создаю новый исполняемый файл и запускаю его, получая ожидаемый результат (рис. ??). Согласно таблице ASCII 10 соответствует следующим символам: 0A,◙,LF. При выводе на экран символ не отображается.



Создание исполняемого файла и проверка работы программы

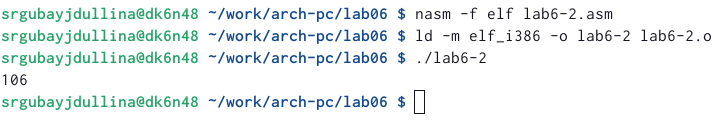
Далее создаю файл lab6-2.asm в каталоге ~/work/arch-pc/lab06 и ввожу в него текст программы из листинга 6.2 (рис. ??):

%include ‘in\_out.asm’ SECTION .text GLOBAL \_start \_start: mov eax,‘6’ mov ebx,‘4’ add eax,ebx call iprintLF call qui



Создание и заполнение файла lab6-2.asm

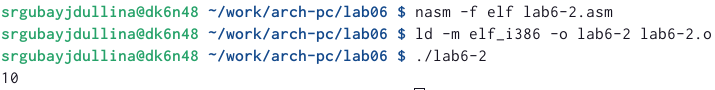
Проделываю все то же самое, а именно: созданию исполняемый файл для lab6-2.asm и запускаю его (рис. ??). В результате работы программы получаю число 106.



Создание исполняемого файла и запуск lab6-2.asm

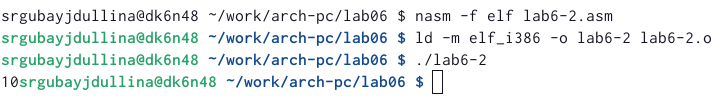
Аналогично предыдущему примеру заменяю символы на числа, а именно: mov eax,‘6’ mov ebx,‘4’ на строки mov eax,6 mov ebx,4

Создаю исполняемый файл и запускаю (рис. ??)



Запуск измененного файла lab6-2.asm

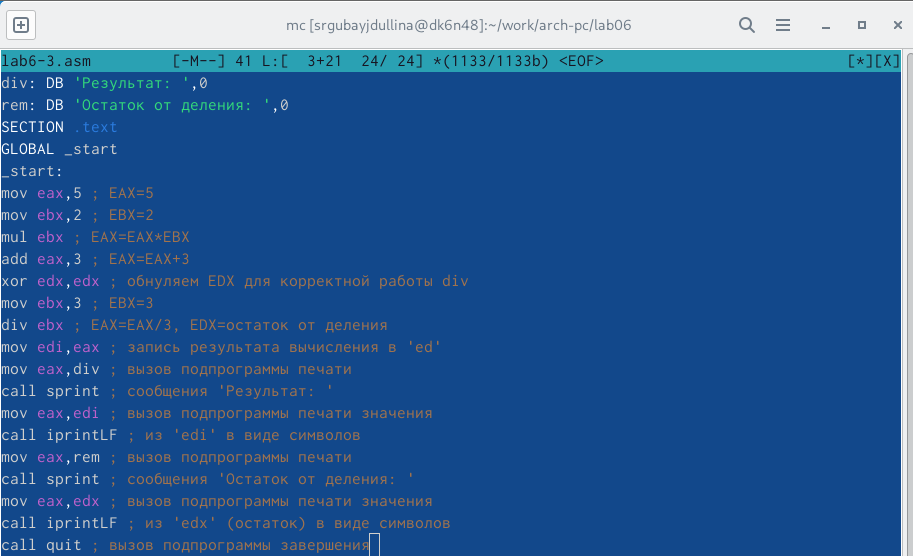
На этот раз я снова меняю текст программы: теперь заменим функцию iprintLF на iprint. iprint это вывод на экран чисел в формате ASCII, тогда как iprintLF при выводе на экран после числа добавляет к символ перевода строки. Снова создаю исполняемый файл для нового lab6-2.asm, после чего запускаю (рис. ??)



Запуск lab6-2.asm с измененным iprintLF на iprint

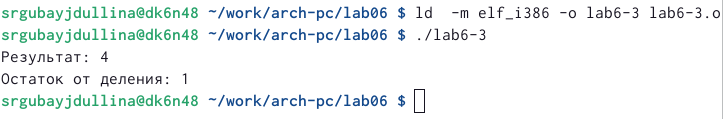
1. Для выполнения второй части лабораторной работы мне необходимо создать новый файл lab6-3.asm, после чего в новый файл при помощи F4 ввожу следующий листинг - программу для вычисления выражения f(𝑥) = (5 ∗ 2 + 3)/3 (рис. ??):

%include ‘in\_out.asm’ ; подключение внешнего файла SECTION .data div: DB ‘Результат:’,0 rem: DB ‘Остаток от деления:’,0 SECTION .text GLOBAL \_start \_start: ; —- Вычисление выражения mov eax,5 ; EAX=5 mov ebx,2 ; EBX=2 mul ebx ; EAX=EAX\*EBX add eax,3 ; EAX=EAX+3 xor edx,edx ; обнуляем EDX для корректной работы div mov ebx,3 ; EBX=3 div ebx ; EAX=EAX/3, EDX=остаток от деления mov edi,eax ; запись результата вычисления в ‘edi’ mov eax,div ; вызов подпрограммы печати call sprint ; сообщения ‘Результат:’ mov eax,edi ; вызов подпрограммы печати значения call iprintLF ; из ‘edi’ в виде символов mov eax,rem ; вызов подпрограммы печати call sprint ; сообщения ‘Остаток от деления:’ mov eax,edx ; вызов подпрограммы печати значения call iprintLF ; из ‘edx’ (остаток) в виде символов call quit ; вызов подпрограммы завершения



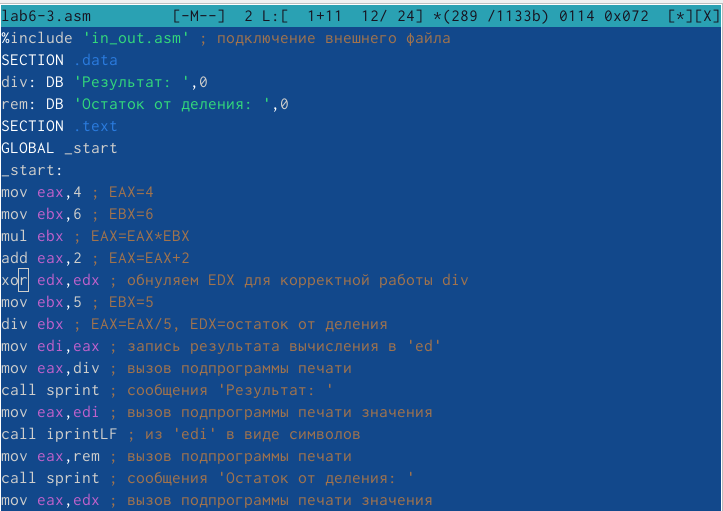
Ввод программы из листинга в lab6-3.asm

Как и ранее создаю исполняемый файл и запускаю его (рис. ??)



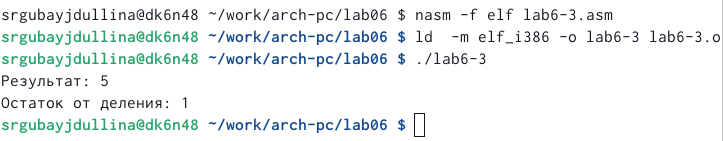
Создание исполнительного файла и запуск lab6-3.asm

Теперь мне необходимо скорректировать программу из листинга в lab6-3.asm для вычисления выражения 𝑓(𝑥) = (4 ∗ 6 + 2)/5 (рис. ??)



Программа для вычисления 𝑓(𝑥) = (4 ∗ 6 + 2)/5.

Создаю исполняемый файл, запускаю (рис. ??)



Создание исполняемого файла и запуск для вычисления выражения

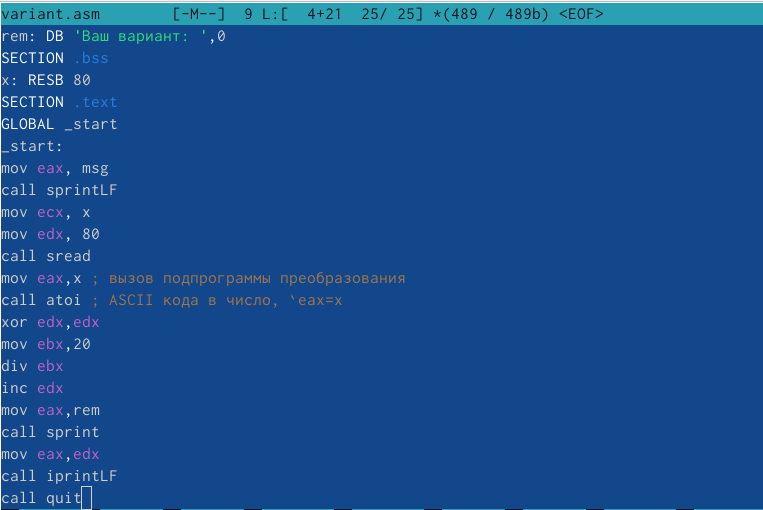
Для дальнейших не необходим новый файл - создаю variant.asm в каталоге ~/work/arch-pc/lab06 (рис. ??)

Создание файла variant.asm

Создание файла variant.asm

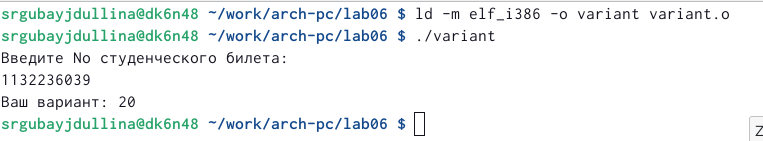
С F4 ввожу в новый файл программа для вычисления варианта задания по номеру студенческого билета из следующего листинга (рис. ??):

%include ‘in\_out.asm’ SECTION .data msg: DB ‘Введите № студенческого билета:’,0 rem: DB ‘Ваш вариант:’,0 SECTION .bss x: RESB 80 SECTION .text GLOBAL \_start \_start: 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’ xor edx,edx mov ebx,20 div ebx inc edx mov eax,rem call sprint mov eax,edx call iprintLF call quit



Ввод программы для вычисления варианта

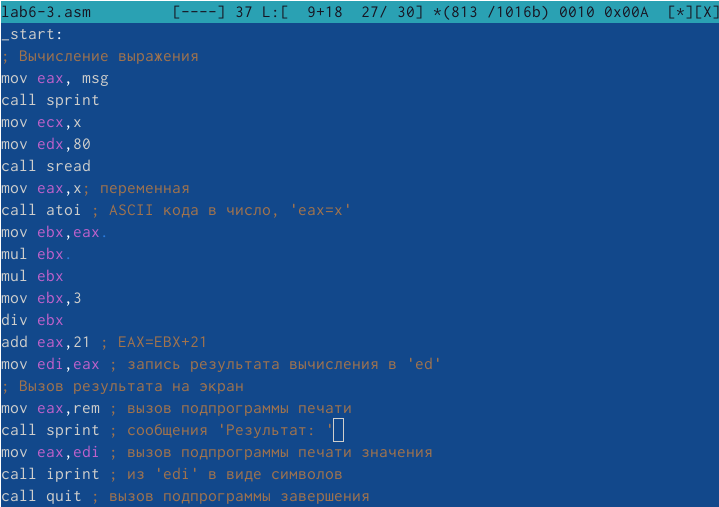
Создаю исполняемый файл для variant.asm и запускаю. Мой вариант - 20 (рис. ??)



Запуск файла variant.asm

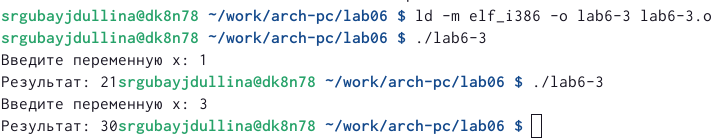
1. Ответы на вопросы:
   1. mov eax, rem call sprintLF/call sprint - строки, отвечающие за вывод сообщения ‘Ваш вариант:’
   2. Первая строчка (mov ecx,x) кладет в ecx адрес строки x. mov edx, 80 записывает в edx длину вводимой строки call sread - вызов подпрограммы из файла извне.
   3. Функция call atoi вызывает подпрограммы из внешнего файла. Программа преобразовывает ASCII код символ в число.
   4. xor edx,edx mov ebx,20 div ebx inc edx - вычисление варианта
   5. Остаток от деления div ebx запишется соответственно в регистр edx.
   6. Инструкция inc edx используется для увеличения значения регистра (здесь edx) на 1.
   7. Для того, чтобы вывести на экран результат вычисления мы используем: mov eax,edx call iprintLF
2. Для начала выполнения самостоятельной работы мне потребуется файл в lab06 с нужным листингом для его дальнейшего редактирования, чтобы решить поставленную задачу. Решаю работать в файле lab6-3.asm с последующим его редактированием. Открываю при помощи F4.

Необходимо править программу, чтобы решать пример из варианта 20 - x^3 ⋅ 1/3 + 21 (рис. ??)



Программа для решения примера из варианта 20

Создаю исполняемый файл и запускаю, подставляя данные для х значения: сначала 1, затем 3 (рис. ??)



Проверка программы для x^3\*(1/3)+21

Листинг для вычисления x^3\*(1/3)+21:

%include ‘in\_out.asm’ ; подключение внешнего файла SECTION .data msg: DB ‘Введите значения переменной х:’,0 rem: DB ‘Результат:’, 0 x: RESB 80; переменная SECTION .text GLOBAL \_start \_start: ; —- Вычисление выражения mov eax, msg ; call sprint; mov ecx,x mo edx,80 call sread mov eax,x ; переменная call atoi mov ebx, eax mul ebx mul ebx mov ebx,3 div ebx add eax,21 ; EAX=EBX+21 mul ebx ; EAX=EAX\*EBX add eax,3 ; EAX=EAX+3 mov edi,eax ; запись результата вычисления в ‘edi’ mov eax,rem ; вызов подпрограммы печати call sprint ; сообщения ‘Результат:’ mov eax,edi ; вызов подпрограммы печати значения call iprint ; из ‘edi’ в виде символов call quit ; вызов подпрограммы завершения.

# 5 Выводы

В ходе выполнения лабораторной работы я освоила арифметические инструкции языка ассемблера NASM.

# Список литературы