Отчёт по лабораторной работе №6

2023

Просина Ксения Максимовна

Содержание

# 1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

# 2 Теоретическое введение

## 2.1 Адресация в NASM

Большинство инструкций на языке ассемблера требуют обработки операндов. Адрес опе- ранда 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 ко- манде, Например: mov ax,2. • Адресация памяти – операнд задает адрес в памяти. В команде указывается символи- ческое 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

## 2.2 Арифметические операции в NASM

### 2.2.1 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 ,   
Допустимые сочетания операндов для команды add аналогичны сочетаниям операндов для команды mov. Так, например, команда add eax,ebx прибавит значение из регистра eax к значению из регистра ebx и запишет результат в регистр eax. Примеры: add ax,5 ; AX = AX + 5  
add dx,cx ; DX = DX + CX  
add dx,cl ; Ошибка: разный размер операндов.

### 2.2.2 Целочисленное вычитание sub.

Команда целочисленного вычитания sub (от англ. subtraction – вычитание) работает анало- гично команде add и выглядит следующим образом: sub ,   
Так, например, команда sub ebx,5 уменьшает значение регистра ebx на 5 и записывает результат в регистр ebx.

### 2.2.3 Команды инкремента и декремента.

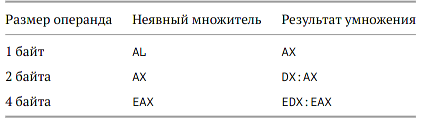
Довольно часто при написании программ встречается операция прибавления или вычита- ния единицы. Прибавление единицы называется инкрементом, а вычитание — декрементом. Для этих операций существуют специальные команды: inc (от англ. increment) и dec (от англ. decrement), которые увеличивают и уменьшают на 1 свой операнд. Эти команды содержат один операнд и имеет следующий вид: inc   
dec   
Операндом может быть регистр или ячейка памяти любого размера. Команды инкремента и декремента выгодны тем, что они занимают меньше места, чем соответствующие команды сложения и вычитания. Так, например, команда inc ebx увеличивает значение регистра ebx на 1, а команда inc ax уменьшает значение регистра ax на 1.

### 2.2.4 Команда изменения знака операнда neg.

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. mov ax,1 ; AX = 1  
neg ax ; AX = -1

### 2.2.5 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   
Для знакового умножения используется команда imul: imul   
Для команд умножения один из сомножителей указывается в команде и должен нахо- диться 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.

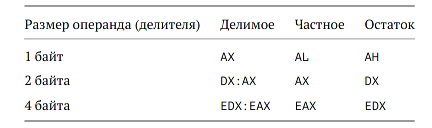


Регистры используемые командами умножения в Nasm

Пример использования инструкции mul: a dw 270  
mov ax, 100 ; AX = 100  
mul a ; AX = AX*a,*  
*mul bl ; AX = AL*BL  
mul ax ; DX:AX = AX\*AX

### 2.2.6 Команды деления div и idiv.

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



Регистры используемые командами деления в Nasm

Например, после выполнения инструкций mov ax,31  
mov dl,15  
div dl  
результат 2 (31/15) будет записан в регистр al, а остаток 1 (остаток от деления 31/15) — в регистр ah. Если делитель — это слово (16-бит), то делимое должно записываться в регистрах dx:ax. Так в результате выполнения инструкций mov ax,2 ; загрузить в регистровую  
mov dx,1 ; пару dx:ax значение 10002h  
mov bx,10h  
div bx  
в регистр ax запишется частное 1000h (результат деления 10002h на 10h), а в регистр dx — 2 (остаток от деления).

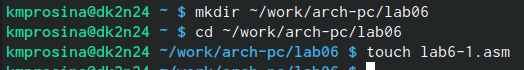
## 2.3 Перевод символа числа в десятичную символьную запись

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. Расширенная таблица ASCII состоит из двух частей. Первая (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. Среди инструкций NASM нет такой, которая выводит числа (не в символьном виде). По- этому, 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 дан- ные 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. 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,).

# 3 Выполнение лабораторной работы

## 3.1 Символьные и численные данные в NASM

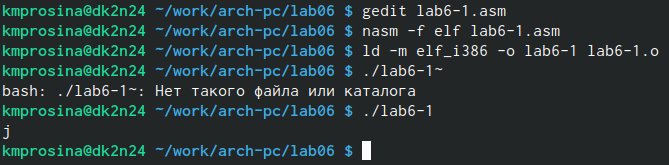
1. Создайте каталог для программам лабораторной работы № 6, перейдите в него и создайте файл lab6-1.asm: mkdir ~/work/arch-pc/lab06  
   cd ~/work/arch-pc/lab06  
   touch lab6-1.asm



Создание каталога и файла

1. Рассмотрим примеры программ вывода символьных и численных значений. Програм- мы будут выводить значения записанные в регистр eax. Введите в файл lab6-1.asm текст программы из листинга 6.1. В данной программе в ре- гистр eax записывается символ 6 (mov eax,‘6’), в регистр ebx символ 4 (mov ebx,‘4’). Далее к значению в регистре eax прибавляем значение регистра ebx (add eax,ebx, ре- зультат сложения запишется в регистр eax). Далее выводим результат. Так как для работы функции sprintLF в регистр eax должен быть записан адрес, необходимо использовать до- полнительную переменную. Для этого запишем значение регистра eax в переменную buf1 (mov [buf1],eax), а затем запишем адрес переменной buf1 в регистр eax (mov eax,buf1) и вызовем функцию sprintLF. Создайте исполняемый файл и запустите его. nasm -f elf lab6-1.asm  
   ld -m elf\_i386 -o lab6-1 lab6-1.o  
   ./lab6-1

ВАЖНО! Для корректной работы программы подключаемый файл in\_out.asm должен лежать в том же каталоге, что и файл с текстом программы. Перед созданием исполня- емого файла создайте копию файла in\_out.asm в каталоге ~/work/arch-pc/lab06.



Создание исполнямого файла и проверка работы

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 (см. таблицу ASCII в приложении). 3. Далее изменим текст программы и вместо символов, запишем в регистры числа. Ис- правьте текст программы (Листинг 6.1) следующим образом: замените строки mov eax,‘6’  
mov ebx,‘4’  
на строки mov eax,6  
mov ebx,4  
Создайте исполняемый файл и запустите его. Как и в предыдущем случае при исполнении программы мы не получим число 10. В данном случае выводится символ с кодом 10. Пользуясь таблицей ASCII определите какому символу соответствует код 10. Отображается ли этот символ при выводе на экран?

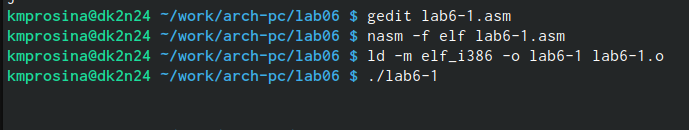


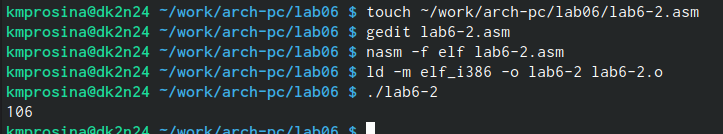
Рис 5

Код 10 соответствует символу LF, который не может отображаться в консоли

1. Как отмечалось выше, для работы с числами в файле in\_out.asm реализованы подпро- граммы для преобразования ASCII символов в числа и обратно. Преобразуем текст программы из Листинга 6.1 с использованием этих функций.

Создайте файл lab6-2.asm в каталоге ~/work/arch-pc/lab06 и введите в него текст про- граммы из листинга 6.2. touch ~/work/arch-pc/lab06/lab6-2.asm

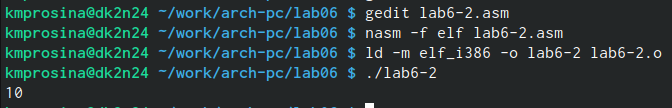
Создайте исполняемый файл и запустите его. nasm -f elf lab6-2.asm  
ld -m elf\_i386 -o lab6-2 lab6-2.o  
./lab6-2



Создание файла, исполняемого файла и проверка

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 программы из листинга 6.1, функция iprintLF позволяет вывести число, а не символ, кодом которого является это число.

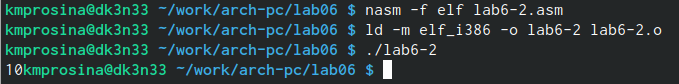
1. Аналогично предыдущему примеру изменим символы на числа. Замените строки mov eax,‘6’  
   mov ebx,‘4’  
   на строки mov eax,6  
   mov ebx,4  
   Создайте исполняемый файл и запустите его. Какой результат будет получен при исполне- нии программы?



Создание исполняемого файла и проверка

После изменений код сложил 6 и 4, в итоге получилось 10

Замените функцию iprintLF на iprint. Создайте исполняемый файл и запустите его. Чем отличается вывод функций iprintLF и iprint?



Создание исполняемого файла и проверка

iprintLF выводит ответ и переводит следующую информацию на новой строчке. Поскольку мы заменили его на iprint, новая строчка консоли не вывелась на следующую и последовала сразу после выведенного числа

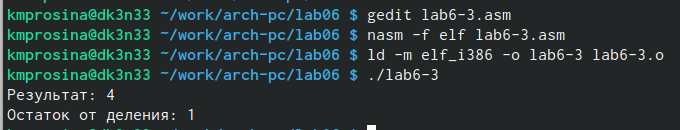
## 3.2 Выполнение арифметических операций в NASM

1. В качестве примера выполнения арифметических операций в NASM приведем про- грамму вычисления арифметического выражения 𝑓(𝑥) = (5 ∗ 2 + 3)/3. Создайте файл lab6-3.asm в каталоге ~/work/arch-pc/lab06: touch ~/work/arch-pc/lab06/lab6-3.asm

Создание файла

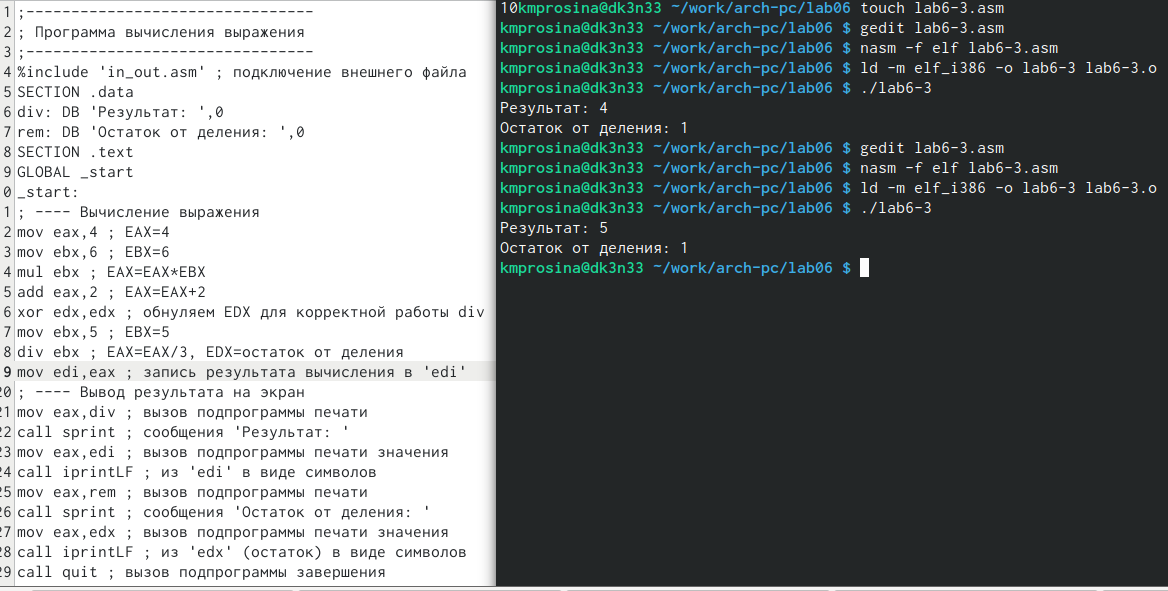
Создание файла

Внимательно изучите текст программы из листинга 6.3 и введите в lab6-3.asm. Создайте исполняемый файл и запустите его. Результат работы программы должен быть следующим: user@dk4n31:~$ ./lab6-3 Результат: 4 Остаток от деления: 1 user@dk4n31:~$



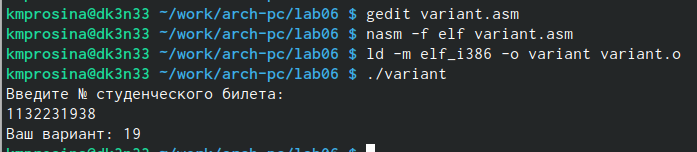
Создание исполняемого файла и проверка

Измените текст программы для вычисления выражения 𝑓(𝑥) = (4 ∗ 6 + 2)/5. Создайте исполняемый файл и проверьте его работу.



Изменение кода, создание исполняемого файла и проверка

1. В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета, работающую по следующему алгоритму: • вывести запрос на введение № студенческого билета • вычислить номер варианта по формуле: (𝑆𝑛 mod 20) + 1, где 𝑆𝑛 – номер студен- ческого билета (В данном случае 𝑎 mod 𝑏 – это остаток от деления 𝑎 на 𝑏). • вывести на экран номер варианта. В данном случае число, над которым необходимо проводить арифметические операции, вводится с клавиатуры. Как отмечалось выше ввод с клавиатуры осуществляется в символь- ном виде и для корректной работы арифметических операций в NASM символы необхо- димо преобразовать в числа. Для этого может быть использована функция atoi из файла in\_out.asm. Создайте файл variant.asm в каталоге ~/work/arch-pc/lab06: touch ~/work/arch-pc/lab06/variant.asm Внимательно изучите текст программы из листинга 6.4 и введите в файл variant.asm. Создайте исполняемый файл и запустите его. Проверьте результат работы программы вычислив номер варианта аналитически.



Cоздание исполняемого файла и проверка

Включите в отчет по выполнению лабораторной работы ответы на следующие вопросы: 1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’? rem: ‘DB ’Ваш вариант:’ ,0  
и mov eax,rem  
2. Для чего используется следующие инструкции? mov ecx, x  
mov edx, 80  
call sread

1. Для чего используется инструкция “call atoi”?

Для корректной работы арифметических операций в NASM символы необходимо преобразовать в числа.

1. Какие строки листинга 6.4 отвечают за вычисления варианта?

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

1. В какой регистр записывается остаток от деления при выполнении инструкции “div ebx”? edx
2. Для чего используется инструкция “inc edx”?

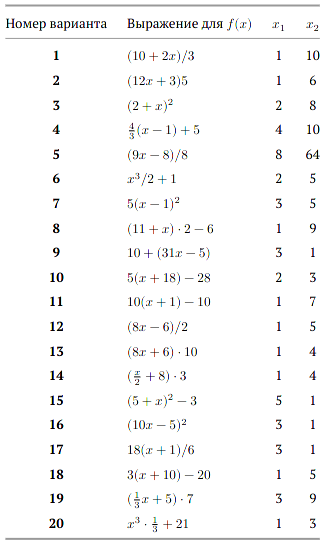
Для добавления к числу единицы

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

call sprint  
mov eax,edx  
call iprintLF

## 3.3 Задание для самостоятельной работы

1. Написать программу вычисления выражения 𝑦 = 𝑓(𝑥). Программа должна выводить выражение для вычисления, выводить запрос на ввод значения 𝑥, вычислять задан- ное 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. Создайте исполняемый файл и проверьте его работу для значений 𝑥1 и 𝑥2.



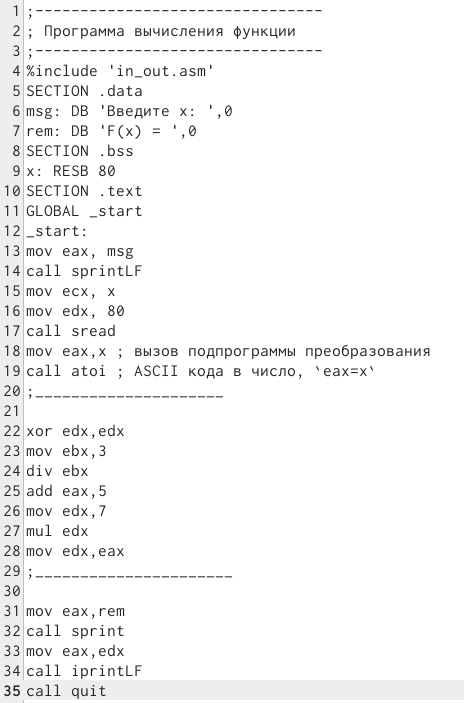
Выражения для 𝑓(𝑥) для задания №1

При выполнении задания преобразовывать (упрощать) выражения для 𝑓(𝑥) нельзя. При выполнении деления в качестве результата можно использовать только целую часть от деления и не учитывать остаток (т.е. 5 ∶ 2 = 2).

Мне попался номер 19, это (x/3 + 5) \* 7

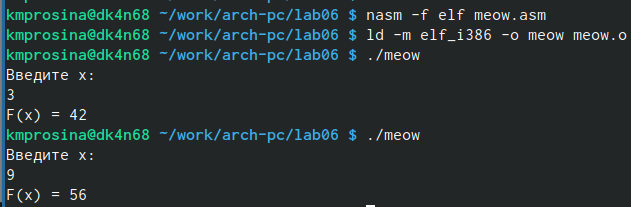
Получившийся код:

;——————————– ; Программа вычисления функции ;——————————– %include ‘in\_out.asm’  
SECTION .data  
msg: DB ‘Введите x:’,0  
rem: DB ‘F(x) =’,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,3  
div ebx  
add eax,5  
mov edx,7  
mul edx  
mov edx,eax  
;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
mov eax,rem  
call sprint  
mov eax,edx  
call iprintLF  
call quit



Код для вычисления функции

mov ebx,3  присвоил ebx 3 div ebx  разделил x на 3 add eax,5  прибавил к x 5 mov edx,7  присвоил 7 mul edx  умножил на 7 mov edx,eax  присвоил ebx значение x



Компоновка и проверка

Посчитав уравнение самостоятельно, я сверилась с ответами, которые оказались верными

# 4 Выводы

Были освоены арифметические функции ассемблера NASM и применены знания на практике

# Список литературы

::: {#refs} ::: https://esystem.rudn.ru/pluginfile.php/2089662/mod\_resource/content/0/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%966.%20%D0%90%D1%80%D0%B8%D1%84%D0%BC%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5%20%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8%20%D0%B2%20NASM..pdf