Лабораторная работа №7

Арифметические операции в NASM

Вершинина Ангелина Алексеевна

Содержание

# 1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM

# 2 Задание

Написание программ с использование арифметических действий

# 3 Теоретическое введение

Большинство инструкций на языке ассемблера требуют обработки операндов. Адрес операнда предоставляет место, где хранятся данные, подлежащие обра- ботке. 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 Большинство инструкций на языке ассемблера требуют обработки операндов. Адрес операнда предоставляет место, где хранятся данные, подлежащие обра- ботке. 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:

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

# 4 Выполнение лабораторной работы

## 4.1 Символьные и численные данные в NASM

Создам каталог для программам лабораторной работы №7, перейду в него и создам файл lab7-1.asm (рис. 1)

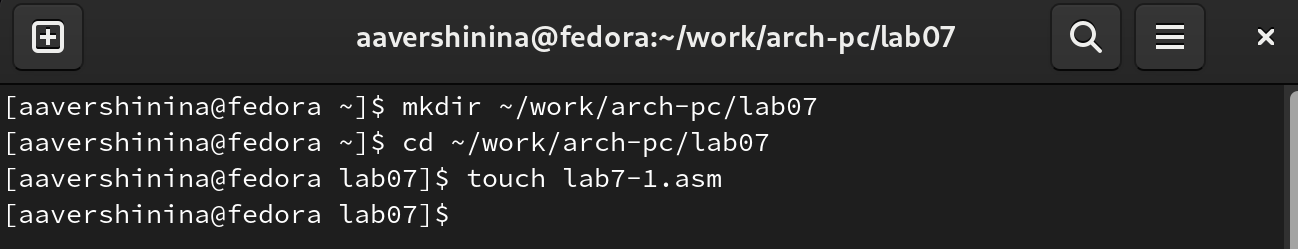


Рис. 1: Создание каталога и файла

**Пример программы 1**

Программы буду выводить значения записанные в регистр eax.

Введу в файл lab7-1.asm текст программы из листинга 7.1. В данной программе в регистр eax записывается символ 6, в регистр ebx символ 4. Далее к значению в регистре eax прибавляю значение регистра ebx и результат сложения запишу в регистр ea. Далее вывожу результат. Так как для работы функции sprintLF в регистр eax должен быть записан адрес, необходимо использовать дополнительную переменную. Для этого запишу значение регистра eax в переменную buf1, а затем запишу адрес переменной buf1 в регистр eax и вызову функцию sprintLF.

Далее создам исполняемый файл и запущу его. (рис. 2) Результатом является 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.

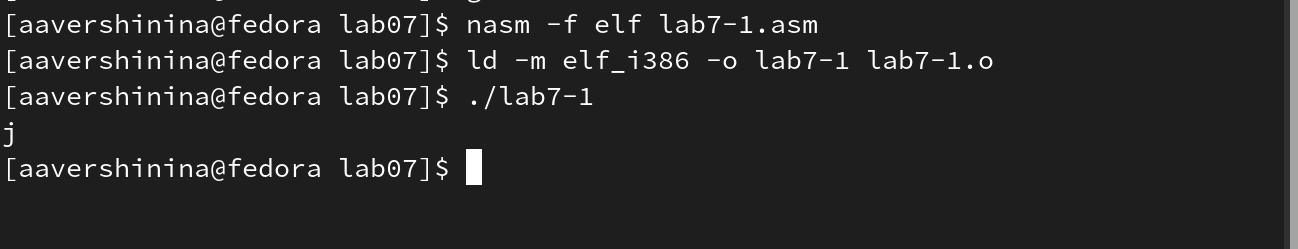


Рис. 2: Создание и запуск исполняемого файла программы 1

**Пример программы 2**

Далее изменю текст программы и вместо символов, запишу в регистры числа. Исправлю текст программы 1 следующим образом: (рис. 3)

заменю строки

mov eax,'6'  
mov ebx,'4'

на строки

mov eax,6  
mov ebx,4

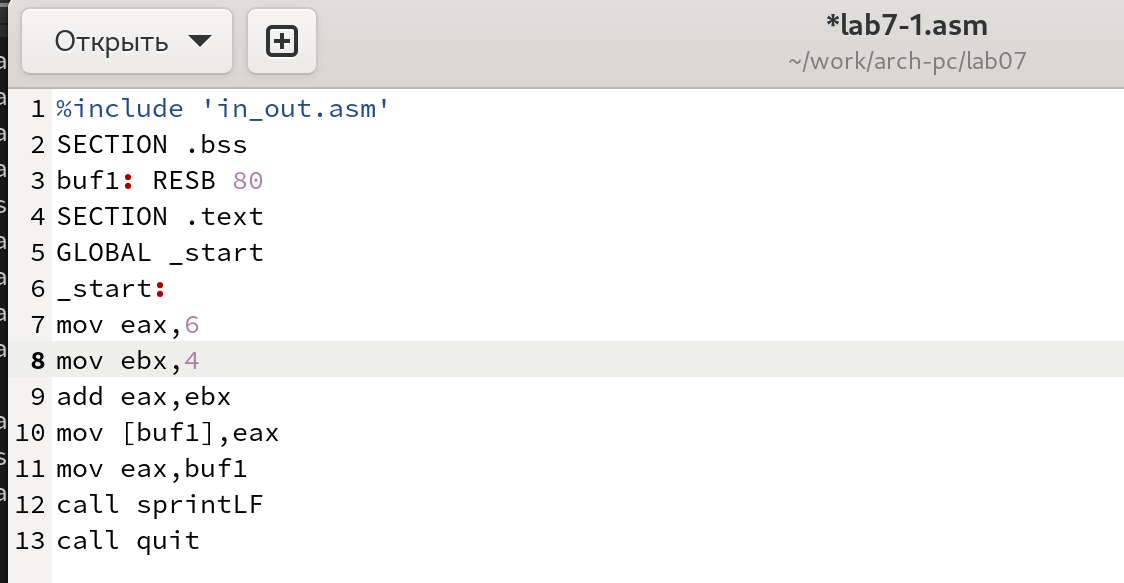


Рис. 3: Замена в тексте программы символов на числа

Создам исполняемый файл и запущу его. (рис. 4) Как и в предыдущем случае при исполнении программы мы не получили число 10. В данном случае выводится символ с кодом 10. Пользуясь таблицей ASCII определю какому символу соответствует код 10. Это символ перевода строки и он не отображается при выводе на экран.

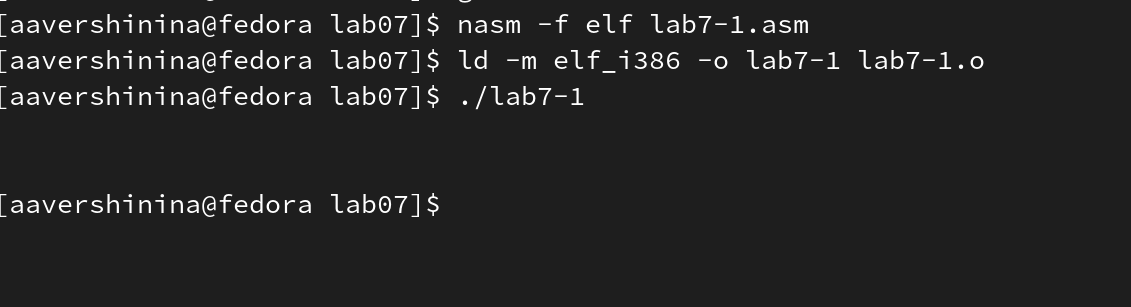


Рис. 4: Создание и запуск исполняемого файла программы 2

**Пример программы 3**

Создам файл lab7-2.asm в каталоге ~/work/arch-pc/lab07 (рис. 5) и введу в него текст программы из листинга 7.2. Создам исполняемый файл и запущу его.(рис. 6) В результате работы программы получилось 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 7.1, функция iprintLF позволяет вывести число, а не символ, кодом которого является это число.

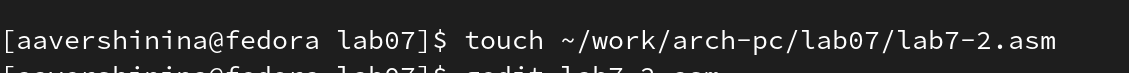


Рис. 5: Создание файла программы 3

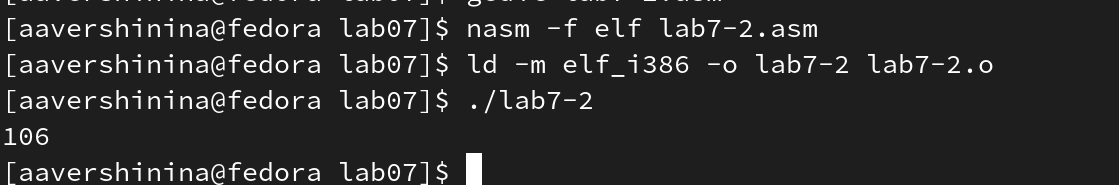


Рис. 6: Создание и запуск исполняемого файла программы 3

Аналогично предыдущему примеру изменим символы на числа.

Заменю строки

mov eax,'6'  
mov ebx,'4'

на строки (рис. 7)

mov eax,6  
mov ebx,4

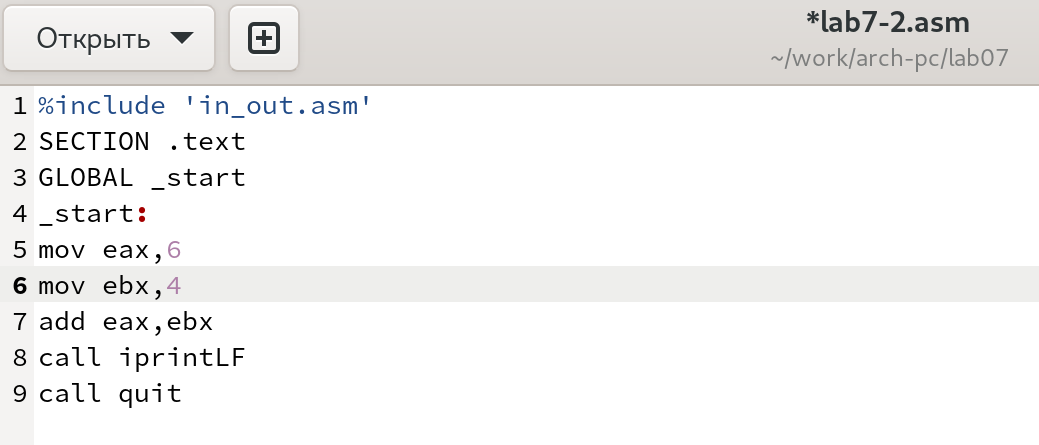


Рис. 7: Замены строк

Создам исполняемый файл и запущу его. (рис. 8) результат выполнения программы - 10.

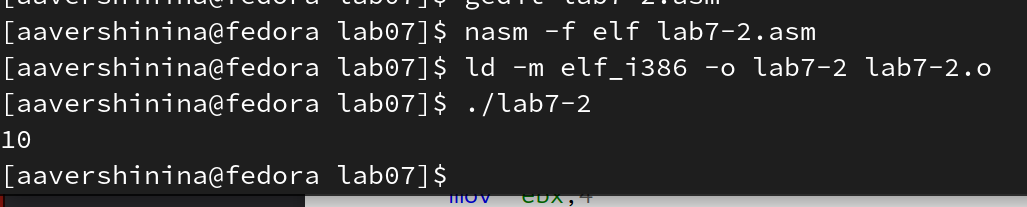


Рис. 8: Создание и запуск исполняемого файла программы

Заменю функцию iprintLF на iprint. (рис. 9) Создам исполняемый файл и запущу его.(рис. 10) Отличие вывода iprintLF и iprint: в первом случае после вывода происходит перенос строки, а втором нет переноса на новую строку

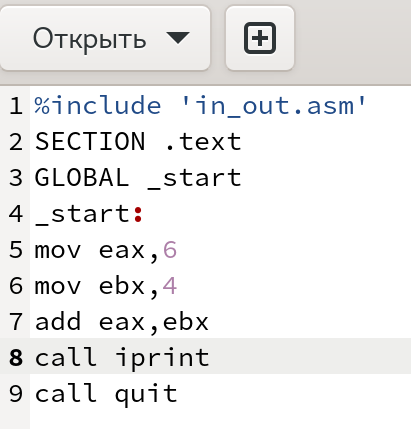


Рис. 9: Замена функции

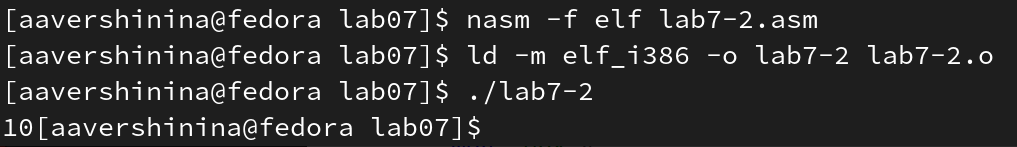


Рис. 10: Создание и запуск исполняемого файла программы

## 4.2 Выполнение арифметических операций в NASM

В качестве примера выполнения арифметических операций в NASM приведe программу вычисления арифметического выражения 𝑓(𝑥) = (5 ∗ 2 + 3)/3

Создам файл lab7-3.asm в каталоге ~/work/arch-pc/lab07 (рис. 11)

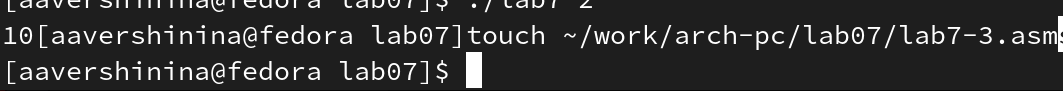


Рис. 11: Создание файла программы

Внимательно изучу текст программы из листинга 7.3 и введу в lab7-3.asm. Создам исполняемый файл и запущу его (рис. 12)

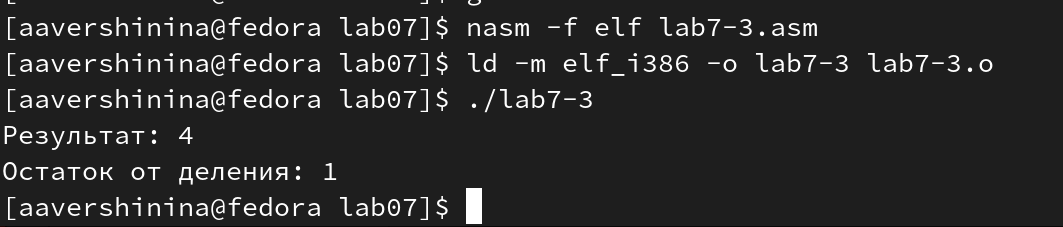


Рис. 12: Создание и запуск исполняемого файла программы

Изменю текст программы для вычисления выражения 𝑓(𝑥) = (4 ∗ 6 + 2)/5.(рис. 13) Создам исполняемый файл и проверю его работу.(рис. 14)

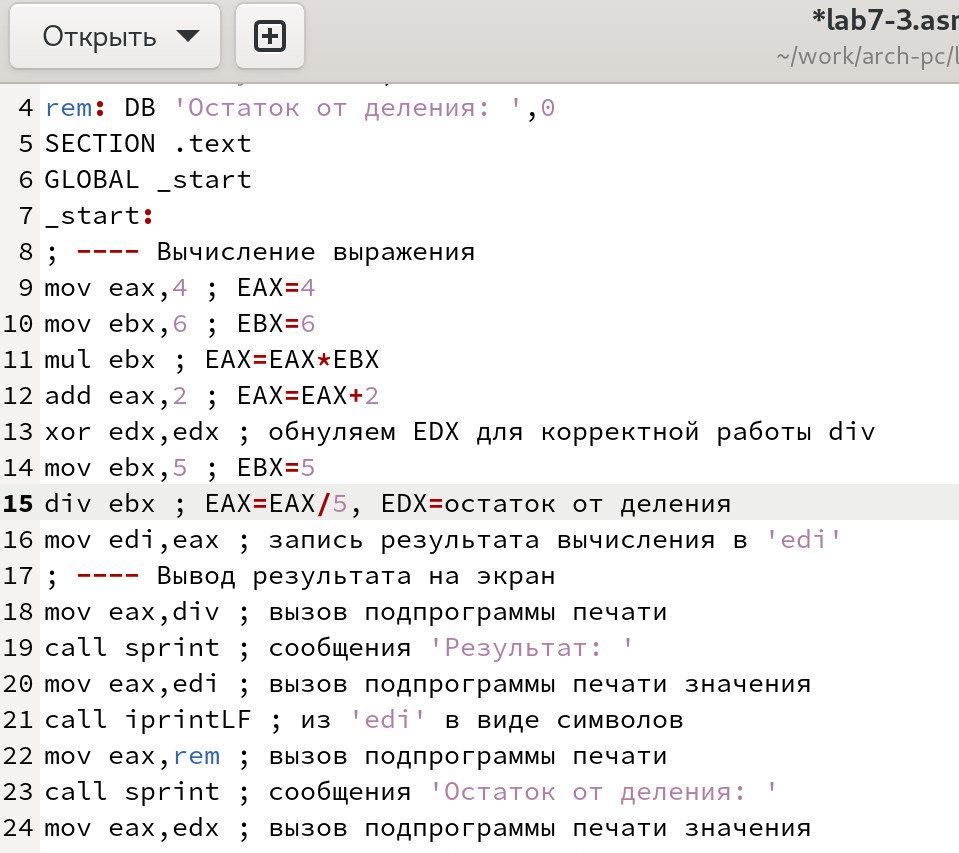


Рис. 13: Изменение текста программы

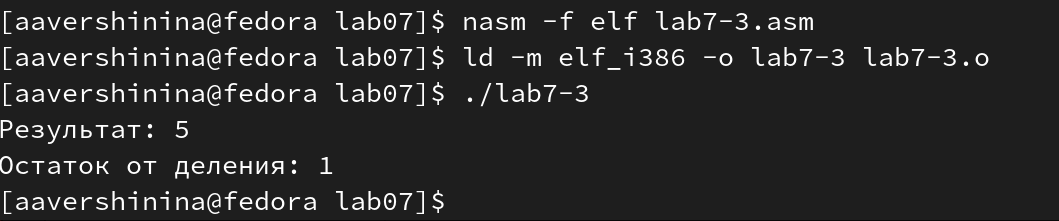


Рис. 14: Создание и запуск исполняемого файла программы

В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета, работающую по следующему алгоритму:

1. вывести запрос на введение No студенческого билета
2. вычислить номер варианта по формуле: (𝑆𝑛 mod 20) + 1, где 𝑆𝑛 – номер студенческого билета (В данном случае 𝑎 mod 𝑏 – это остаток от деления 𝑎 на 𝑏).
3. 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 символьном виде и для корректной работы арифметических операций в NASM символы необходимо преобразовать в числа. Для этого может быть использована функция atoi из файла in\_out.asm.

Создам файл variant.asm в каталоге ~/work/arch-pc/lab07. Внимательно изучу текст программы из листинга 7.4 и введу в файл variant.asm. Создам исполняемый файл и запущу его. (рис. 15)

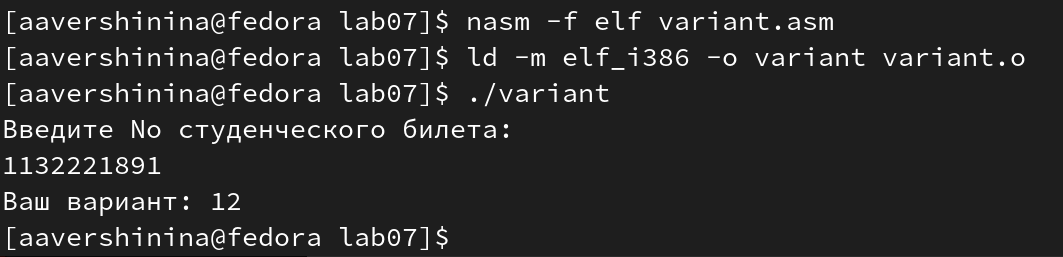


Рис. 15: Работа программы

**Ответы на вопросы**

1. За вывод сообщения “Ваш вариант” отвечают строки кода

mov eax,rem  
call sprint

1. 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
2. call atoi используется для вызова подпрограммы из внешнего файла, которая преобразует ASCII-код символа в целое число и записывает результат в регистр eax
3. За вычисления варианта отвечают строки

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. При выполнении инструкции div ebx остаток от деления записывается в регистр edx
2. Инструкция inc edx увеличивает значение регистра edx на 1
3. За вывод на экран результатов вычислений отвечают строки

mov eax,edx  
call iprintLF

## 4.3 Задание для самостоятельной работы

Необходимо 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, выво- дить результат вычислений. Вид функции 𝑓(𝑥)= (8х-6)/2

Создам файл для выполнения задания (рис. 16)

Рис. 16: Создание файла

Рис. 16: Создание файла

Создам исполняемый файл и проверю его работу для значений 𝑥1 = 1 и 𝑥2 = 5. (рис. 17 и 18)

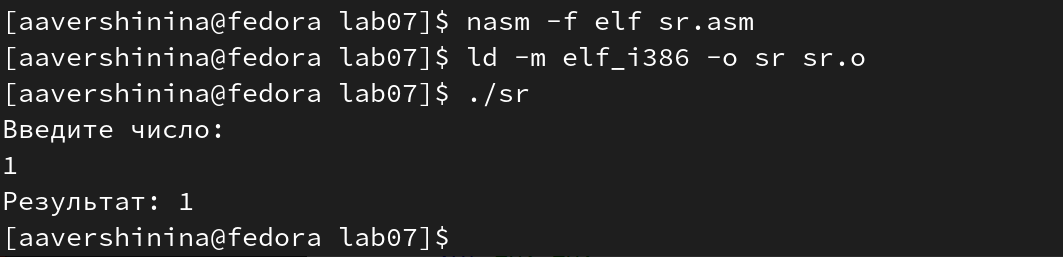


Рис. 17: Проверка работы программы на х1

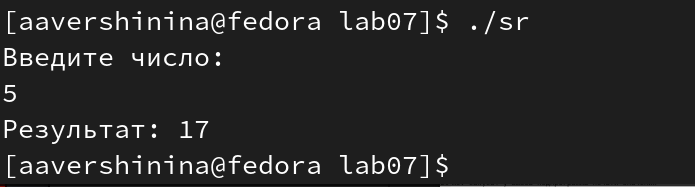


Рис. 18: Проверка работы программы на х2

**Листинг программы**

%include 'in\_out.asm'  
SECTION .data  
fun: DB 'F(x)=(8x-6)/2',0  
msg: DB 'Введите число: ',0  
rem: DB 'Результат: ',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 ; eax=x  
call atoi ; ASCII кода в число, `eax=x`  
mov ebx, 8 ; ebx = 8  
mul ebx ; eax = eax\*ebx  
sub eax, 6 ; eax - 6  
mov ebx, 2 ; ebx = 2  
div ebx ; eax = eax/2  
mov edi, eax ; запись результата в edi  
mov eax, rem  
call sprint  
mov eax, edi  
call iprintLF  
call quit

# 5 Выводы

В результате проделанной работы я освоила арифметических инструкций языка ассемблера NASM.

# Список литературы