

Отчет по лабораторной работе №6

Дисциплина: Архитектура компьютера

Мустафина Аделя Юрисовна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
3.1	6.2.1. Адресация в NASM	7
3.2	6.2.2. Арифметические операции в NASM	8
3.2.1	6.2.2.1. Целочисленное сложение add.	8
3.2.2	6.2.2.2. Целочисленное вычитание sub.	8
3.2.3	6.2.2.3. Команды инкремента и декремента.	8
3.2.4	6.2.2.4. Команда изменения знака операнда neg.	9
3.2.5	6.2.2.5. Команды умножения mul и imul.	9
3.2.6	6.2.2.6. Команды деления div и idiv.	9
4	Выполнение лабораторной работы	11
4.1	6.3.1. Символьные и численные данные в NASM	11
4.2	6.3.2. Выполнение арифметических операций в NASM	16
4.3	6.4. Задание для самостоятельной работы	22
4.4	Листинг для задания для самостоятельной работы	23
5	Выводы	26
6	Список литературы	27

Список иллюстраций

4.1	Каталог	11
4.2	Открытый редактор	11
4.3	Листинг	12
4.4	Исполняемый файл	12
4.5	Запуск файла	13
4.6	Изменения в коде	13
4.7	Исполняемый файл	13
4.8	Запуск программы	13
4.9	Создание файла	14
4.10	Текст программы	14
4.11	Исполняемый файл	14
4.12	Запуск файла	15
4.13	Запуск файла	15
4.14	Запуск файла	15
4.15	Изменение файла	15
4.16	Запуск файла	16
4.17	Ввод программы	17
4.18	Исполняемый файл	18
4.19	Новый файл	19
4.20	Файл	19
4.21	Файл с вариантом	20
4.22	Вариант	20
4.23	Программа	23
4.24	Компиляция файла	23
4.25	Запуск файла	23

Список таблиц

1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

2 Задание

1. Порядок выполнения лабораторной работы.
2. Выполнение заданий для самостоятельной работы.

3 Теоретическое введение

3.1 6.2.1. Адресация в NASM

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. 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`.

3.2 6.2.2. Арифметические операции в NASM

3.2.1 6.2.2.1. 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.

3.2.2 6.2.2.2. Целочисленное вычитание sub.

Команда целочисленного вычитания sub (от англ. subtraction – вычитание) работает аналогично 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.

3.2.3 6.2.2.3. Команды инкремента и декремента.

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 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 dec

ах уменьшает значение регистра ах на 1.

3.2.4 6.2.2.4. 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`

3.2.5 6.2.2.5. Команды умножения mul и imul.

Умножение и деление, в отличие 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 находиться 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 6.1.

3.2.6 6.2.2.6. 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 6.2. ## 6.2.3. 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 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. 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 записывает результат в регистр `eax`, перед вызовом `atoi` в регистр `eax` необходимо записать число (`mov eax, ...`).

4 Выполнение лабораторной работы

4.1 6.3.1. Символьные и численные данные в NASM

Создаю каталог для программ лабораторной работы №6 (рис. 4.1).

```
aymustafina@vbox:~$ mkdir ~/work/arch-pc/lab06  
aymustafina@vbox:~$ cd ~/work/arch-pc/lab06  
aymustafina@vbox:~/work/arch-pc/lab06$ touch lab6-1.asm  
aymustafina@vbox:~/work/arch-pc/lab06$
```

Рис. 4.1: Каталог

Открываю созданный файл (рис. 4.2).



Рис. 4.2: Открытый редактор

Ввожу текст из листинга 6.1 (рис. 4.3).



```
GNU nano 7.2 /home/aymustafina/work/arch-pc/lab06/lab6-1.asm Изменён
#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
```

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выводить ^_ К строке

Рис. 4.3: Листинг

Листинг 6.1. Программа вывода значения регистра eax

```
%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
```

Создаю исполняемый файл (рис. 4.4).

```
aymustafina@vbox:~/work/arch-pc/lab06$ nasm -f elf lab6-1.asm
aymustafina@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-1 lab6-1.o
aymustafina@vbox:~/work/arch-pc/lab06$ ./lab6-1
```

Рис. 4.4: Исполняемый файл

Запускаю его и вижу вывод буквы j, хотя мы ожидали увидеть число 10, это произошло из-за того, что команда `add eax,ebx` записала в регистр `eax` сумму кодов – 01101010, так через таблицу ASCII мы получаем j (рис. 4.5).

```
aymustafina@vbox:~/work/arch-pc/lab06$ ./lab6-1
j
aymustafina@vbox:~/work/arch-pc/lab06$ █
```

Рис. 4.5: Запуск файла

Теперь изменим наш код программы, убрав кавычки возле чисел (рис. 4.6).

```
GNU nano 7.2 /home/aymustafina/work/arch-pc/lab06/lab6-11.asm Изменён
%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
```

Рис. 4.6: Изменения в коде

Создам исполняемый файл (рис. 4.7).

```
aymustafina@vbox:~$ cd ~/work/arch-pc/lab06
aymustafina@vbox:~/work/arch-pc/lab06$ nasm -f elf lab6-11.asm
aymustafina@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-11 lab6-11.o
aymustafina@vbox:~/work/arch-pc/lab06$ ./lab6-11
```

Рис. 4.7: Исполняемый файл

При запуске программы получаем пустую строку, так мы получили символ с кодом 10. Пользуясь таблицей ASCII, видим, что код числа 10 это “LF”, а они обозначают начало новой строки, теперь понятно, почему нам вывело пустую строку (рис. 4.8).

```
aymustafina@vbox:~/work/arch-pc/lab06$ ./lab6-11

aymustafina@vbox:~/work/arch-pc/lab06$ ./lab6-11
```

Рис. 4.8: Запуск программы

Создаю новый файл с названием lab6-2.asm в каталоге ~/work/arch-pc/lab06 (рис. 4.9).

```
aymustafina@vbox:~/work/arch-pc/lab06$ touch lab6-2.asm
aymustafina@vbox:~/work/arch-pc/lab06$ mc
aymustafina@vbox:~/work/arch-pc/lab06$ █
```

Рис. 4.9: Создание файла

Ввожу в него текст программы из листинга 6.2. (рис. 4.10).

```
GNU nano 7.2 /home/aymustafina/work/arch-pc/lab06/lab6-2.asm
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,'6'
mov ebx,'4'
add eax,ebx
call iprintLF
call quit
```

Рис. 4.10: Текст программы

Листинг 6.2. Программа вывода значения регистра eax

```
%include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,'6'
mov ebx,'4'
add eax,ebx
call iprintLF
call quit
```

Создаю исполняемый файл (рис. 4.11).

```
aymustafina@vbox:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
aymustafina@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
```

Рис. 4.11: Исполняемый файл

Запускаю его функция `iprintLF` позволяет вывести число, а не символ, кодом которого является это число, поэтому мы получили число 106 (рис. 4.12).

```
aymustafina@vbox:~/work/arch-pc/lab06$ ./lab6-2
106
aymustafina@vbox:~/work/arch-pc/lab06$ █
```

Рис. 4.12: Запуск файла

Снова изменим пару строк в тесте программы, убрав кавычки (рис. 4.13).

```
GNU nano 7.2 /home/aymustafina/work/arch-pc/lab06/lab6-2.asm
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprintLF
call quit
```

Рис. 4.13: Запуск файла

После создания исполняемого файла запускаю его и получаю 10 (рис. 4.14).

```
aymustafina@vbox:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
aymustafina@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
aymustafina@vbox:~/work/arch-pc/lab06$ ./lab6-2
10
aymustafina@vbox:~/work/arch-pc/lab06$ █
```

Рис. 4.14: Запуск файла

Теперь снова меняю строки в коде, заменив функцию `iprintLF` на `iprint` (рис. 4.15).

```
GNU nano 7.2 /home/aymustafina/work/arch-pc/lab06/lab6-2.asm
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprint
call quit
```

Рис. 4.15: Изменение файла

В выводе получаю число 10 без перехода на новую строку и все слиплось, так произошло из-за того, что мы убрали символ LF, который и делал пропуск строки. (рис. 4.16).

```
aymustafina@vbox:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
aymustafina@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
aymustafina@vbox:~/work/arch-pc/lab06$ ./lab6-2
10aymustafina@vbox:~/work/arch-pc/lab06$ █
```

Рис. 4.16: Запуск файла

4.2 6.3.2. Выполнение арифметических операций в NASM

В качестве примера выполнения арифметических операций в NASM приведу программу вычисления арифметического выражения $f(x) = (5 * 2 + 3)/3$. Создаю файл lab6-3.asm в каталоге ~/work/arch-pc/lab06: touch ~/work/arch-pc/lab06/lab6-3.asm И ввожу текст программы из листинга 6.3 (рис. 4.17).


```

GNU nano 7.2 /home/aymustafina/work/arch-pc/lab06/lab6-3.asm
;-----
; Программа вычисления выражения
;-----
#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'
; ---- Вывод результата на экран
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

```

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
 ^X Выход ^R ЧитФайл ^_ Замена ^U Вставить ^J Выводить ^_ К строке

Рис. 4.17: Ввод программы

Листинг 6.3. Программа вычисления выражения $f(x) = (5 * 2 + 3)/3$.

```

;-----
; Программа вычисления выражения
;-----
#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'
; ---- Вывод результата на экран
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,edx ; 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

```

Создаю исполняемый файл и запускаю его (рис. 4.18).

```

10aymustafina@vbox:~/work/arch-pc/lab06 touch lab6-3.asm
aymustafina@vbox:~/work/arch-pc/lab06$ mc

aymustafina@vbox:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
aymustafina@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
aymustafina@vbox:~/work/arch-pc/lab06$ ./lab6-3
Результат: 4
Остаток от деления: 1
aymustafina@vbox:~/work/arch-pc/lab06$

```

Рис. 4.18: Исполняемый файл

Изменяю текст программы для вычисления выражения $f(x) = (4 * 6 + 2)/5$ (рис. 4.19).

```

aymustafina@vbox:~/work/arch-pc/lab06
GNU nano 7.2 /home/aymustafina/work/arch-pc/lab06/lab6-3.asm
;-----
; Программа вычисления выражения
;-----
#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,4 ; EAX=4
mov ebx,6 ; EBX=6
mul ebx ; EAX=EAX*EBX
add eax,2 ; EAX=EAX+2
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,5 ; EBX=5
div ebx ; EAX=EAX/5, EDX=остаток от деления
mov edi,eax ; запись результата вычисления в 'edi'
; ---- Вывод результата на экран
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

```

[^]G Справка [^]O Записать [^]W Поиск [^]K Вырезать [^]T Выполнить [^]C Позиция
[^]X Выход [^]R ЧитФайл [^]\ Замена [^]U Вставить [^]J Выводить [^]/ К строку

Рис. 4.19: Новый файл

Создаю исполняемый файл и проверяю его работу (рис. 4.20).

```

aymustafina@vbox:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
aymustafina@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
aymustafina@vbox:~/work/arch-pc/lab06$ ./lab6-3
Результат: 5
Остаток от деления: 1
aymustafina@vbox:~/work/arch-pc/lab06$ █

```

Рис. 4.20: Файл

Создаю файл variant.asm в каталоге ~/work/arch-pc/lab06: touch ~/work/arch-pc/lab06/variant.asm Проверила результат работы программы, вычислив номер варианта аналитически и получаю номер варианта 20 (рис. 4.21).

```

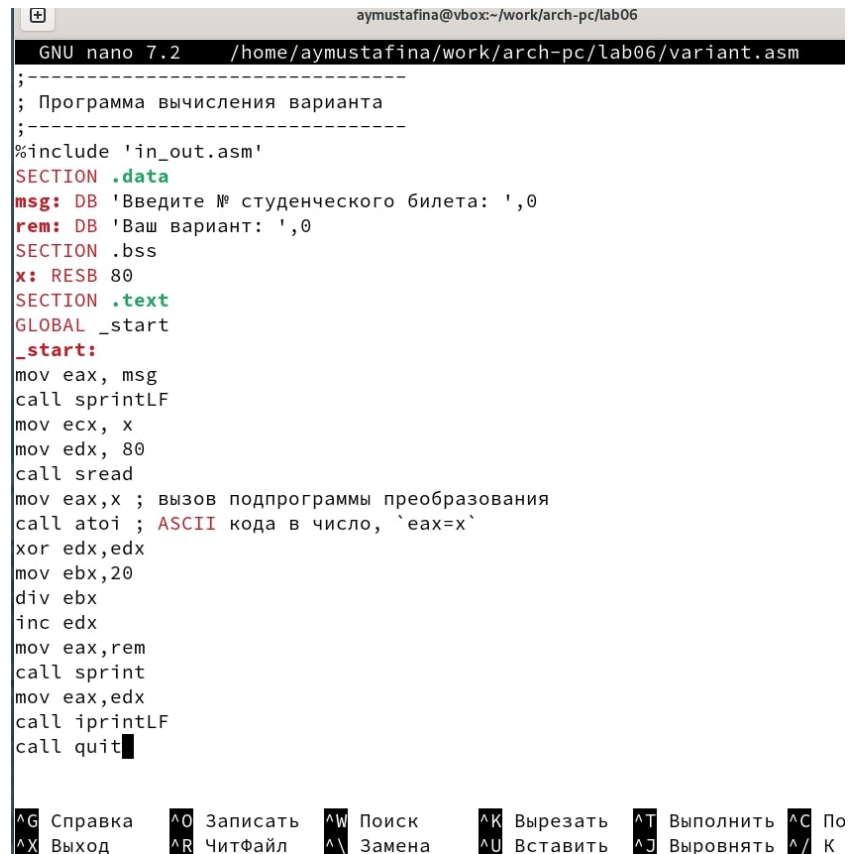
aymustafina@vbox:~/work/arch-pc/lab06$ touch variant.asm
aymustafina@vbox:~/work/arch-pc/lab06$ mc

aymustafina@vbox:~/work/arch-pc/lab06$ nasm -f elf variant.asm
aymustafina@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o variant variant.o
aymustafina@vbox:~/work/arch-pc/lab06$ ./variant
Введите № студенческого билета:
1132246719
Ваш вариант: 20

```

Рис. 4.21: Файл с вариантом

Текст файла из листинга 6.4 (рис. 4.22).



```

GNU nano 7.2 /home/aymustafina/work/arch-pc/lab06/variant.asm
;-----
; Программа вычисления варианта
;-----
%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 sprintf
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 sprintf
mov eax, edx
call iprintLF
call quit

```

[^]G Справка [^]O Записать [^]W Поиск [^]K Вырезать [^]T Выполнить [^]C По
[^]X Выход [^]R ЧитФайл [^]\ Замена [^]U Вставить [^]J Выводить [^]/ К

Рис. 4.22: Вариант

Листинг 6.4 Программа вычисления вычисления варианта задания по номеру студенческого билета

```

;-----
; Программа вычисления варианта
;-----

```

```

#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 sprintf
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 sprintf
mov eax,edx
call iprintLF
call quit

```

Ответы на вопросы из лекции. 1. За вывод на экран сообщения “Ваш вариант:” отвечают строки:

```

mov eax,rem          ; вызов подпрограммы печати сообщения "Ваш вариант:"
call sprintf

```

2. Данные инструкции используются для чтения 80 байтов данных из ввода и сохранения их по адресу, указанном в `x`. В данной программе в регистр `ecx` записывается символ `x` (`mov ecx, x`), в регистр `edx` значение 80 (`mov edx, 80`), это значение используется для указания количества байтов.

```
mov ecx, x
mov edx, 80
call sread
```

3. Эта инструкция используется для преобразования ASCII кода в число, `eax=x`.

```
call atoi
```

4. За вычисление варианта отвечают строки:

```
xor edx,edx    ; Обнуляем EDX для корректной работы div
mov ebx,20     ; Задали для ebx значение 20
div ebx        ; AX=EAX/20, EDX=остаток от деления
inc edx        ; edx + 1
```

5. Остаток от деления при выполнении инструкции “`div ebx`” записывается в регистр `EDX`.

6. Инструкция “`inc edx`” используется для увеличения значения `edx` на 1 (`edx + 1`).

7. За вывод результата вычислений отвечают строки:

```
mov eax,edx ; вызов подпрограммы печати значения
call iprintLF ; из 'edx' (остаток) в виде символов
```

4.3 6.4. Задание для самостоятельной работы

Вводим текст программы для вычисления выражения $y = f(x)$. В предыдущем задании мы получили № нашего варианта - 20. Значит пишем программу вычисления выражения $y = x^3 * 1/3 + 21$ (рис. 4.23).

```

GNU nano 7.2 /home/aymustafina/work/arch-pc/lab06/lab6-4.asm
;-----
; Программа вычисления выражения
;-----
#include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg: DB 'Введите значение для переменной x: ',0
res: DB 'Результат: ',0
SECTION .bss
x: RESB 80; переменная
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения

mov eax, msg
call sprintf

mov ecx, x
mov edx, 80
call sread
mov eax, x ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, 'eax=x'

mov ebx, eax
mul ebx, ebx ; x^3
mul ebx, ebx ; перемножили x для получения куба
xor ebx, ebx ; обнуляем для корректного деления
mov ebx, 3

div ebx, ebx ; x^3 / 3 делим
xor ebx, ebx
add eax, 21 ; x^3 / 3 + 21 прибавляем

```

Рис. 4.23: Программа

Компилирую созданный файл (рис. 4.24).

```

aymustafina@vbox:~/work/arch-pc/lab06$ nasm -f elf lab6-4.asm
aymustafina@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-4 lab6-4.o

```

Рис. 4.24: Компиляция файла

Запускаю его (рис. 4.25).

```

aymustafina@vbox:~/work/arch-pc/lab06$ ./lab6-4
Введите значение для переменной x:
1
Результат: 21aymustafina@vbox:~/work/arch-pc/lab06$ ./lab6-4
Введите значение для переменной x:
3
Результат: 30aymustafina@vbox:~/work/arch-pc/lab06$

```

Рис. 4.25: Запуск файла

4.4 Листинг для задания для самостоятельной работы

```

;-----
; Программа вычисления выражения
;-----
#include 'in_out.asm' ; подключение внешнего файла

```

```

SECTION .data
msg: DB 'Введите значение для переменной x: ',0
rem: DB 'Результат: ',0
SECTION .bss
x: RESB 80; переменная
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения

mov eax, msg
call printf

mov ecx, x
mov edx, 80
call sread
mov eax,x ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, `eax=x

mov ebx, eax
mul eax      ;x^3
mul ebx      ; перемножили x для получения куба
xor ebx, ebx      ;Обнуляем для корректного деления
mov ebx, 3      ;

div ebx      ;x^3 / 3 делим
xor ebx, ebx
add eax, 21    ;x^3 / 3 + 21 прибавляем

```



```
mov edi,eax      ;  
; ---- Вывод результата на экран
```

```
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 мы научились работать с арифметическими инструкциями языка ассемблера NASM.

6 Список литературы

1. Лабораторная работа №6