

Архитектура компьютера

Отчёт по лабораторной работе №6

Ибрахим Мохсейн Алькамаль

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	11
5	Выполнение заданий для самостоятельной работы	19
6	Выводы	22
	Список литературы	23

Список иллюстраций

4.1	Создание папки и файла	11
4.2	Содержание файла lab6-1.asm	11
4.3	Копирование файла	12
4.4	Результат работы файла	12
4.5	Измененный код	12
4.6	Работа файла	13
4.7	Создание файла lab6-2.asm	13
4.8	Содержание файла	13
4.9	Название рисунка	14
4.10	Новое содержание файла	14
4.11	Работа созданного файла	14
4.12	Новый текст файла	15
4.13	Работа исполняемого файла	15
4.14	Создание файла	15
4.15	Работа исполняемого файла	16
4.16	Текст программы	16
4.17	Работа файла	16
4.18	Создание файла и проверка его наличия	17
4.19	Содержание файла	17
4.20	Работа файла	17
5.1	Создание файла	19
5.2	Текст файла	20
5.3	Работа файла	20

Список таблиц

1 Цель работы

Освоить арифметические инструкции языка ассемблера NASM.

2 Задание

1. Создайте каталог для программ лабораторной работы № 6, перейдите в него и создайте файл lab6-1.asm
2. Рассмотрим примеры программ вывода символьных и численных значений. Программы будут выводить значения записанные в регистр eax
3. Далее изменим текст программы и вместо символов, запишем в регистры числа.
4. Как отмечалось выше, для работы с числами в файле in_out.asm реализованы подпрограммы для преобразования ASCII символов в числа и обратно.
5. Аналогично предыдущему примеру изменим символы на числа.
6. В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения.

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. 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`.

6.2.2. Арифметические операции в NASM

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 выгля-

дит 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` ; $DX = DX + CX `add dx,cl` ; Ошибка: разный размер операндов.$

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`.

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 ax` уменьшает значение регистра `ax` на 1.

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` ; $AX = -1$

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. Вторым сомножителем 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. Таблица 6.1. Регистры используемые командами умножения

Размер операнда	Неявный множитель	Результат умножения
1 байт	AL	AX
2 байта	AX	DX:AX
4 байта	EAX	EDX:EAX

Пример использования инструкции mul:
`a dw 270 mov ax, 100 ; AX = 100 mul a ; AX = AXa, mul bl ; AX = ALBL mul ax ; DX:AX = AX*AX`

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. Таблица 6.2. Регистры используемые командами деления

Размер операнда (делителя)	Делимое	Частное	Остаток
1 байт	AX	AL	AH
2 байта	DX:AX	AX	DX
4 байта	EDX:EAX	EAX	EDX

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).

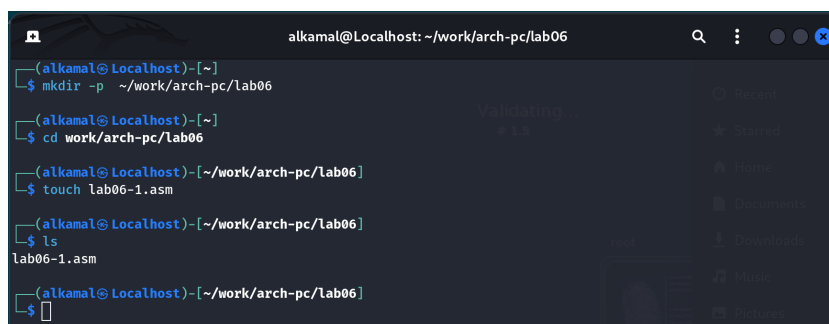
6.2.3. Перевод символа числа в десятичную символьную запись Ввод информации с клавиатуры и вывод её на экран осуществляется в символьном виде. Кодирование этой информации производится согласно кодовой таблице символов ASCII. ASCII – сокращение от American Standard Code for Information Interchange (Американский 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). 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. 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

- 1) Создаю каталог для программ лабораторной работы № 6, перехожу в него и создаю файл lab6-1.asm



```
alkamal@Localhost: ~/work/arch-pc/lab06
(alkamal@Localhost)-[~]
$ mkdir -p ~/work/arch-pc/lab06
(alkamal@Localhost)-[~]
$ cd work/arch-pc/lab06
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ touch lab06-1.asm
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ls
lab06-1.asm
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$
```

Рис. 4.1: Создание папки и файла

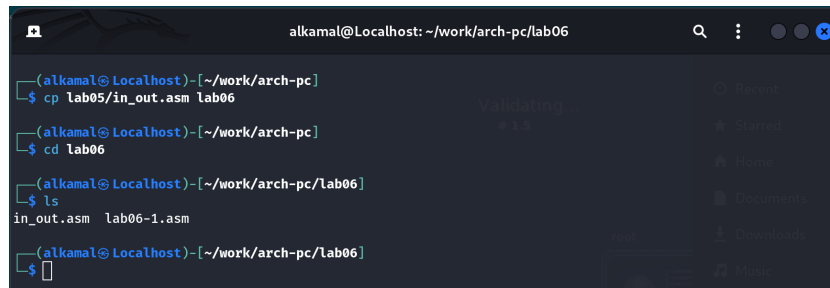
- 2.1) Ввожу в файл lab6-1.asm текст программы из листинга 6.1



```
GNU nano 7.2 /home/alkamal/work/arch-pc/lab06/lab06-1.asm
#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 sprintf
call quit
```

Рис. 4.2: Содержание файла lab6-1.asm

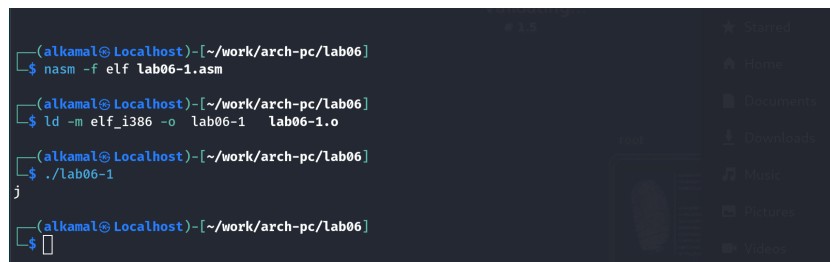
- 2.2) Копирую из папки lab5 в папку lab6 файл in_out.asm, проверяю его наличие



```
alkamal@Localhost: ~/work/arch-pc/lab06
(alkamal@Localhost)-[~/work/arch-pc]
$ cp lab05/in_out.asm lab06
(alkamal@Localhost)-[~/work/arch-pc]
$ cd lab06
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ls
in_out.asm  lab06-1.asm
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$
```

Рис. 4.3: Копирование файла

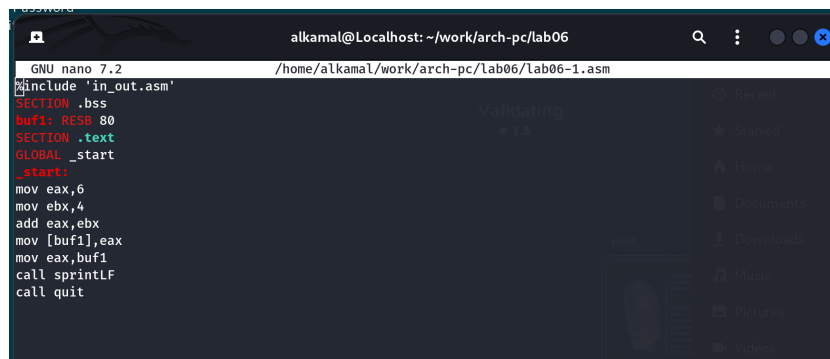
2.3) Создаю исполняемый файл и запускаю его. В результате получается j



```
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ nasm -f elf lab06-1.asm
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ld -m elf_i386 -o lab06-1 lab06-1.o
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ./lab06-1
j
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$
```

Рис. 4.4: Результат работы файла

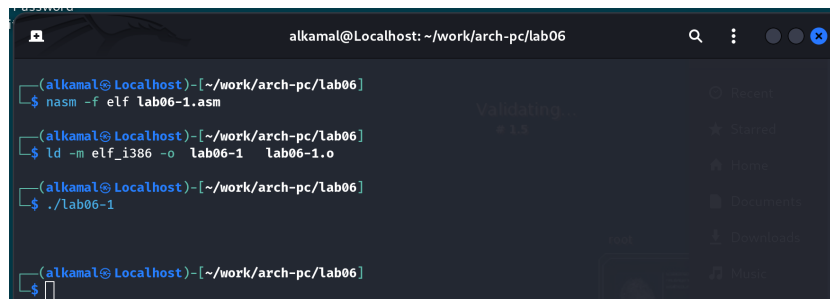
3.1) Меняю “6” и “4” на 6 и 4 в файле lab6-1.asm



```
GNU nano 7.2 /home/alkamal/work/arch-pc/lab06/lab06-1.asm
#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 sprintf
call quit
```

Рис. 4.5: Измененный код

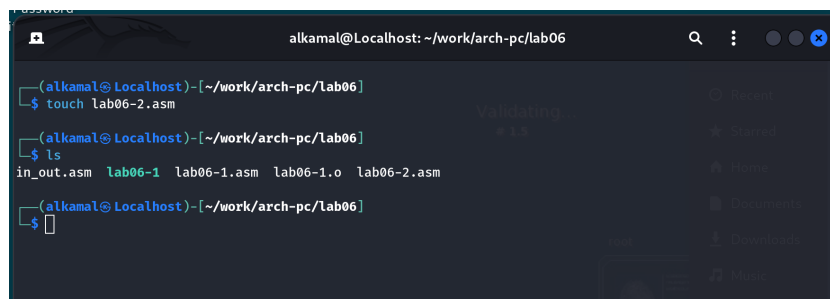
3.2) Создаю исполняемый файл и запускаю его. В результате получается символ, который не отображается в консоли (спец.LF Возвр. каретки)



```
alkamal@Localhost: ~/work/arch-pc/lab06
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ nasm -f elf lab06-1.asm
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ld -m elf_i386 -o lab06-1 lab06-1.o
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ./lab06-1
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$
```

Рис. 4.6: Работа файла

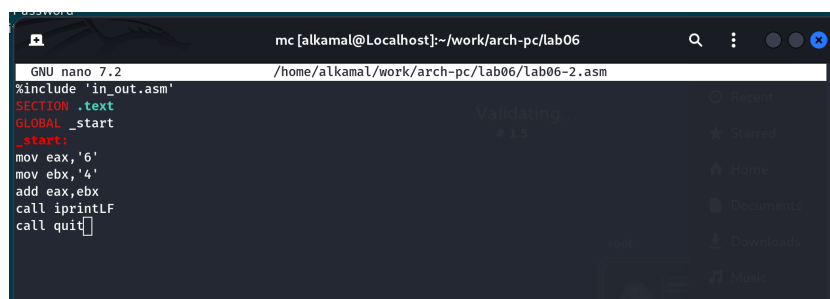
4.1) Создаю файл lab6-2.asm и проверяю его наличие



```
alkamal@Localhost: ~/work/arch-pc/lab06
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ touch lab06-2.asm
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ls
in_out.asm lab06-1 lab06-1.asm lab06-1.o lab06-2.asm
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$
```

Рис. 4.7: Создание файла lab6-2.asm

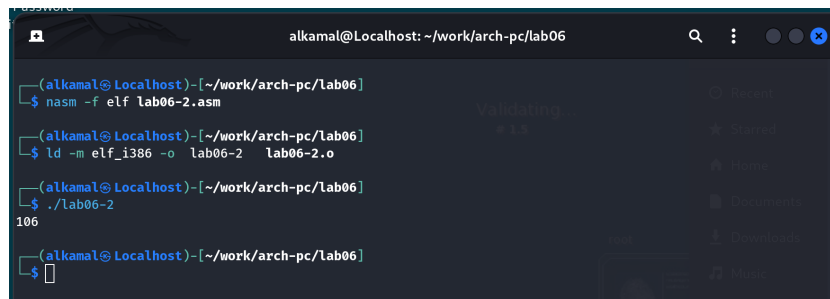
4.2) Ввожу в файл текст из листинга 6.2



```
mc [alkamal@Localhost]:~/work/arch-pc/lab06
GNU nano 7.2 /home/alkamal/work/arch-pc/lab06/lab06-2.asm
%include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,'6'
mov ebx,'4'
add eax,ebx
call iprintLF
call quit
```

Рис. 4.8: Содержание файла

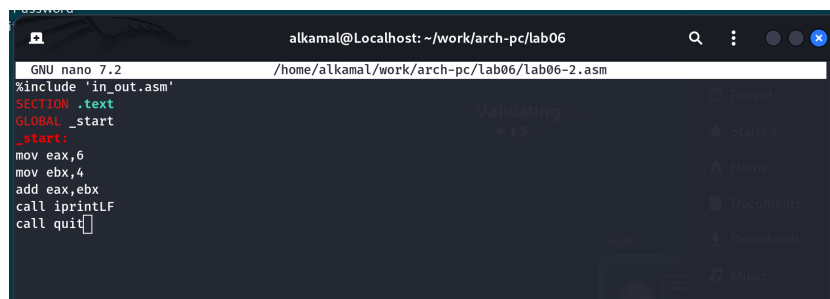
4.3) Создаю файл и запускаю его



```
alkamal@Localhost: ~/work/arch-pc/lab06
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ nasm -f elf lab06-2.asm
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ld -m elf_i386 -o lab06-2 lab06-2.o
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ./lab06-2
106
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$
```

Рис. 4.9: Название рисунка

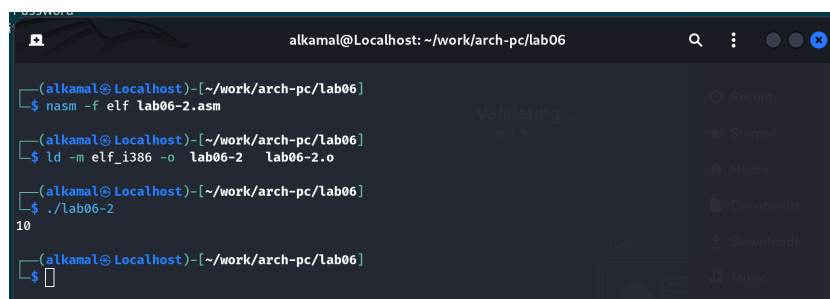
5.1) Заменяю “6” и “4” на 6 и 4 в файле lab6-2.asm



```
GNU nano 7.2 /home/alkamal/work/arch-pc/lab06/lab06-2.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: Новое содержание файла

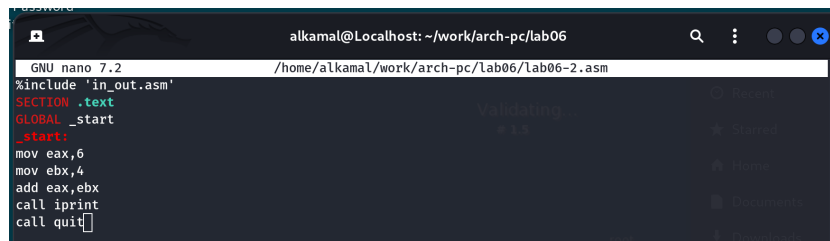
5.2) Создаю исполняемый файл и запускаю его. В результате получается 10



```
alkamal@Localhost: ~/work/arch-pc/lab06
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ nasm -f elf lab06-2.asm
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ld -m elf_i386 -o lab06-2 lab06-2.o
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ./lab06-2
10
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$
```

Рис. 4.11: Работа созданного файла

5.3) Заменяю функцию iprintLF на iprint



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

Рис. 4.12: Новый текст файла

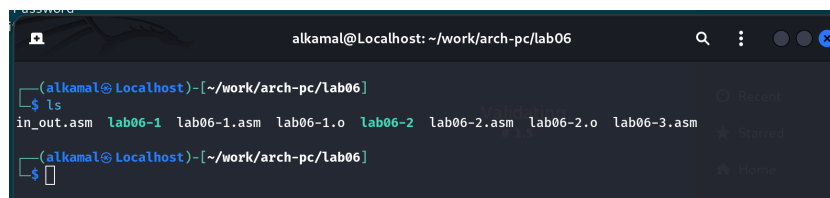
5.4) Создание и запуск исполняемого файла. Результат такой же, но без переноса строки. Но оно не появилось из-за терминала Kali Linux



```
alkamal@Localhost: ~/work/arch-pc/lab06
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ nasm -f elf lab06-2.asm
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ld -m elf_i386 -o lab06-2 lab06-2.o
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ./lab06-2
10
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$
```

Рис. 4.13: Работа исполняемого файла

6.1) Создаю файл lab6-3.asm и проверяю, создан ли он



```
alkamal@Localhost: ~/work/arch-pc/lab06
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ls
in_out.asm lab06-1 lab06-1.asm lab06-1.o lab06-2 lab06-2.asm lab06-2.o lab06-3.asm
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$
```

Рис. 4.14: Создание файла

6.2) Ввожу в файл текст из листинга 6.3, создаю исполняемый файл и запускаю его

```

alkamal@Localhost: ~/work/arch-pc/lab06
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ nasm -f elf lab06-3.asm
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ld -m elf_i386 -o lab06-3 lab06-3.o
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ./lab06-3
Результат: 4
Остаток от деления: 1
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$

```

Рис. 4.15: Работа исполняемого файла

6.3) Изменяю текст программы так, чтобы она выводила результат выражения $(4*6 + 2)/5$

```

GNU nano 7.2 /home/alkamal/work/arch-pc/lab06/lab06-3.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

```

Рис. 4.16: Текст программы

6.4) Создание и работа исполняемого файла

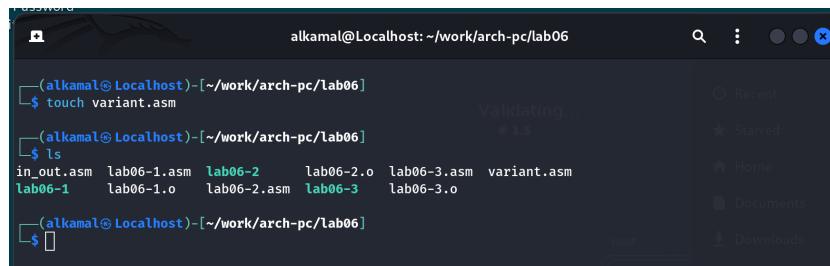
```

alkamal@Localhost: ~/work/arch-pc/lab06
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ nasm -f elf lab06-3.asm
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ld -m elf_i386 -o lab06-3 lab06-3.o
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ./lab06-3
Результат: 5
Остаток от деления: 1
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$

```

Рис. 4.17: Работа файла

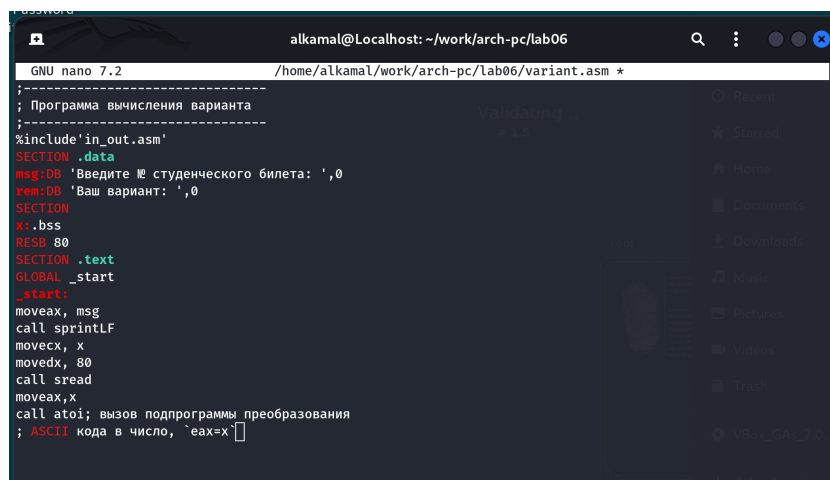
7.1) Создание файла variant.asm



```
alkamal@Localhost: ~/work/arch-pc/lab06
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ touch variant.asm
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ls
in_out.asm  lab06-1.asm  lab06-2    lab06-2.o  lab06-3.asm  variant.asm
lab06-1     lab06-1.o   lab06-2.asm  lab06-3    lab06-3.o
```

Рис. 4.18: Создание файла и проверка его наличия

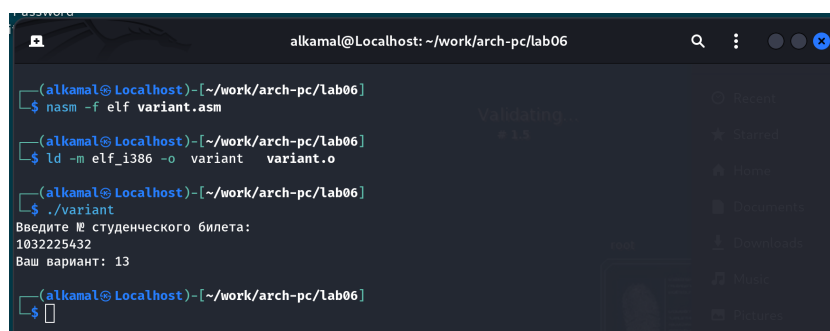
7.2) Программа файла



```
GNU nano 7.2 /home/alkamal/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
x:.bss
RESB 80
SECTION .text
GLOBAL _start
_start:
moveax, msg
call sprintf
movecx, x
movedx, 80
call sread
moveax, x
call atoi; вызов подпрограммы преобразования
; ASCII кода в число, 'eax=x'
```

Рис. 4.19: Содержание файла

7.3) Работа файла



```
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ nasm -f elf variant.asm
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ld -m elf_i386 -o variant variant.o
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$ ./variant
Введите № студенческого билета:
1032225432
Ваш вариант: 13
(alkamal@Localhost)-[~/work/arch-pc/lab06]
$
```

Рис. 4.20: Работа файла

7.4) Ответы на вопросы

1. `mov eax,rem call sprint`
2. `mov ecx, x` - с помощью этой команды мы кладем адрес вводимой строки `x` в регистр `ecx` `mov edx, 80` - с помощью этой команды записываем в регистр `edx` длину вводимой строки (80) `call sread` - вызываем подпрограмму из стороннего файла, чтобы обеспечить ввод сообщения с клавиатуры
3. `call atoi` - подключение сторонней программы, которая преобразует `ascii`-код символа в целое число и запишет результат в регистр `eax`, перед вызовом `atoi` в регистр `eax` необходимо записать число (`mov eax,`)
4. `xor edx,edx mov ebx,20 div ebx inc edx`
5. В регистр `edx`
6. Для увеличения значения регистра `edx` на 1
7. `mov eax,edx call iprintLF`

5 Выполнение заданий для самостоятельной работы

1) Создаю файл lab6-4.asm для написания программы по вариантам



```
alkamal@Localhost: ~/work/arch-pc/lab06
(alkamal@ Localhost) - [~/work/arch-pc/lab06]
$ touch lab06-4.asm
(alkamal@ Localhost) - [~/work/arch-pc/lab06]
$ ls
in_out.asm  lab06-1.asm  lab06-2      lab06-2.o  lab06-3.asm  lab06-4.asm  variant.asm
lab06-1     lab06-1.o   lab06-2.asm  lab06-3    lab06-3.o    variant      variant.o
```

Рис. 5.1: Создание файла

2) Редактирование файла

```

alkamal@Localhost: ~/work/arch-pc/lab06
GNU nano 7.2 /home/alkamal/work/arch-pc/lab06/lab06-4.asm
#include 'in_out.asm' ; подключение внешнего файла
SECTION .data
div: DB 'Введите значение x: ',0
rem: DB 'Результат: ',0
SECTION .bss
x: RESB 80 ;
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения
mov eax,div ;
call sprint ;
mov ecx, x ;
mov edx, 80 ;
call sread ;
mov eax, x ;
call atoi ;
mov ebx, 8 ;
mul ebx ; eax = x*8
add eax, 6 ; eax = x*2 + 6
mov ebx, 10 ;
mul ebx ; eax = (x*8+10)*10
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 iprintLF ; из 'edx' (остаток) в виде символов
call quit ; вызов подпрограммы завершения

```

Рис. 5.2: Текст файла

3) Создание и запуск исполняемого файла. Он выдает верный результат

```

(alkamal@ Localhost)-[~/work/arch-pc/lab06]
$ nasm -f elf lab06-4.asm

(alkamal@ Localhost)-[~/work/arch-pc/lab06]
$ ld -m elf_i386 -o lab06-4 lab06-4.o

(alkamal@ Localhost)-[~/work/arch-pc/lab06]
$ ./lab06-4
Введите значение x: 1
Результат: 140

(alkamal@ Localhost)-[~/work/arch-pc/lab06]
$ ./lab06-4
Введите значение x: 4
Результат: 380

(alkamal@ Localhost)-[~/work/arch-pc/lab06]
$

```

Рис. 5.3: Работа файла

```

#include 'in_out.asm' ; подключение внешнего файла
SECTION .data
div: DB 'Введите значение x: ',0
rem: DB 'Результат: ',0
SECTION .bss
x: RESB 80 ;
SECTION .text

```

```

GLOBAL _start

_start:
; ---- Вычисление выражения
mov eax,div ;
call sprint ;
mov ecx, x ;
mov edx, 80 ;
call sread ;
mov eax, x ;
call atoi ;
mov ebx, 8 ;
mul ebx ; eax = x*8
add eax, 6 ; eax = x*2 + 6
mov ebx, 10 ;
mul ebx ; eax = (x*8+10)*10
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 iprintLF ; из 'edx' (остаток) в виде символов
call quit ; вызов подпрограммы завершения

```

6 Выводы

У меня получилось освоить арифметические инструкции языка ассемблера NASM.

Список литературы