

Отчёт по лабораторной работе №6

Дисциплина: архитектура компьютера

Черкашина Ангелина Максимовна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Символьные и численные данные в NASM	9
4.2	Выполнение арифметических операций в NASM	14
4.2.1	Ответы на вопросы по программе	18
4.3	Выполнение заданий для самостоятельной работы	20
5	Выводы	24
	Список литературы	25

Список иллюстраций

4.1	Создание директории	9
4.2	Создание файла	9
4.3	Создание копии файла	9
4.4	Редактирование файла	10
4.5	Запуск исполняемого файла	10
4.6	Изменение текста программы	11
4.7	Запуск нового исполняемого файла	11
4.8	Создание файла	11
4.9	Редактирование файла	12
4.10	Запуск исполняемого файла	12
4.11	Изменение текста программы	13
4.12	Запуск нового исполняемого файла	13
4.13	Изменение текста программы	14
4.14	Запуск нового исполняемого файла	14
4.15	Создание файла	14
4.16	Редактирование файла	15
4.17	Запуск исполняемого файла	15
4.18	Изменение программы	16
4.19	Запуск нового исполняемого файла	16
4.20	Создание файла	17
4.21	Редактирование файла	17
4.22	Запуск исполняемого файла	18
4.23	Создание файла	20
4.24	Написание программы	20
4.25	Запуск исполняемого файла	21
4.26	Запуск исполняемого файла	21

Список таблиц

1 Цель работы

Цель данной лабораторной работы - освоение арифметических инструкций языка ассемблера NASM.

2 Задание

1. Символьные и численные данные в NASM
2. Выполнение арифметических операций в NASM
3. Выполнение заданий 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. - 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. 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. 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.

4 Выполнение лабораторной работы

4.1 Символьные и численные данные в NASM

С помощью команды `mkdir` создаю каталог для программ данной лабораторной работы, перехожу в созданный каталог с помощью `cd` (рис. 4.1).

```
amcherkashina@dk3n37 ~ $ mkdir ~/work/arch-pc/lab06
amcherkashina@dk3n37 ~ $ cd ~/work/arch-pc/lab06
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $
```

Рис. 4.1: Создание директории

С помощью команды `touch` создаю файл `lab6-1.asm` (рис. 4.2).

```
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ touch lab6-1.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ls
lab6-1.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $
```

Рис. 4.2: Создание файла

С помощью команды `cp` копирую в текущий каталог файл `in_out.asm`, т.к. он будет использоваться в других программах (рис. 4.3).

```
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ cp ~/Загрузки/in_out.asm in_out.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ls
in_out.asm  lab6-1.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $
```

Рис. 4.3: Создание копии файла

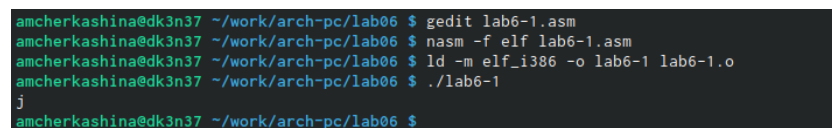
Открываю созданный файл lab6-1.asm, вставляю в него программу вывода значения регистра eax (рис. 4.4).



```
1 %include 'in_out.asm'
2
3 SECTION .bss
4 buf1: RESB 80
5
6 SECTION .text
7 GLOBAL _start
8 _start:
9
10 mov eax, '6'
11 mov ebx, '4'
12 add eax, ebx
13 mov [buf1], eax
14 mov eax, buf1
15 call sprintf
16
17 call quit
```

Рис. 4.4: Редактирование файла

Создаю исполняемый файл программы и запускаю его (рис. 4.5). Вывод программы: 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 по кодовой таблице ASCII.



```
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ gedit lab6-1.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ./lab6-1
j
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $
```

Рис. 4.5: Запуск исполняемого файла

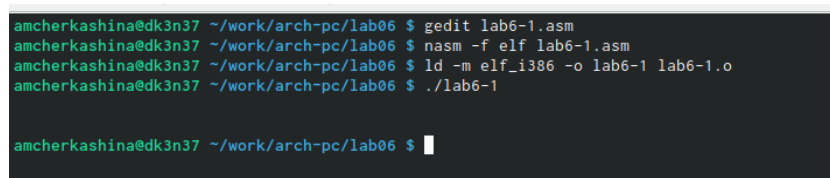
Изменяю текст программы и вместо символов записываю в регистры числа, т.е. исправляю символы “6” и “4” на числа 6 и 4 (рис. 4.6).



```
1 %include 'in_out.asm'
2
3 SECTION .bss
4 buf1: RESB 80
5
6 SECTION .text
7 GLOBAL _start
8 _start:
9
10 mov eax,6
11 mov ebx,4
12 add eax,ebx
13 mov [buf1],eax
14 mov eax,buf1
15 call sprintLF
16
17 call quit
```

Рис. 4.6: Изменение текста программы

Создаю новый исполняемый файл программы и запускаю его. Теперь вывелся символ с кодом 10, согласно таблице ASCII код 10 соответствует символу перевода строки. Поэтому этот символ не отображается при выводе на экран (рис. 4.7).

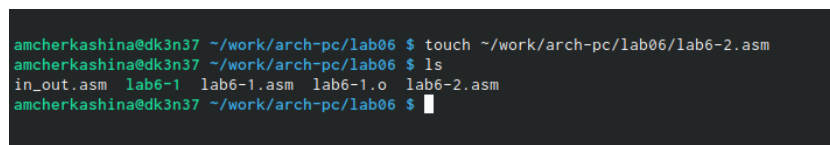


```
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ gedit lab6-1.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ./lab6-1

amcherkashina@dk3n37 ~/work/arch-pc/lab06 $
```

Рис. 4.7: Запуск нового исполняемого файла

Создаю в каталоге ~/work/arch-pc/lab06 новый файл lab6-2.asm с помощью команды touch (рис. 4.8).



```
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/lab6-2.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ls
in_out.asm lab6-1 lab6-1.asm lab6-1.o lab6-2.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $
```

Рис. 4.8: Создание файла

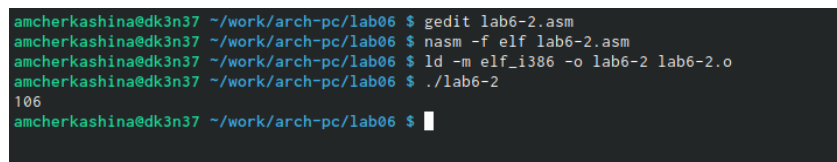
Ввожу в созданный файл текст другой программы для вывода значения регистра eax (рис. 4.9).



```
1 %include 'in_out.asm'
2
3 SECTION .text
4 GLOBAL _start
5 _start:
6
7 mov eax, '6'
8 mov ebx, '4'
9 add eax, ebx
10 call iprintLF
11
12 call quit
```

Рис. 4.9: Редактирование файла

Создаю и запускаю исполняемый файл lab6-2 (рис. 4.10). В результате работы программы получается число 106. В данном случае, как и в первом, команда add складывает коды символов '6' и '4' ($54+52=106$). Однако, в отличие от программы файла lab6-1, функция iprintLF позволяет вывести число, а не символ, кодом которого является это число.



```
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ gedit lab6-2.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ./lab6-2
106
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $
```

Рис. 4.10: Запуск исполняемого файла

Аналогично предыдущей программе корректирую текст программы файла lab6-2 и меняю символы на числа (исправляю символы “6” и “4” на числа 6 и 4) (рис. 4.11).

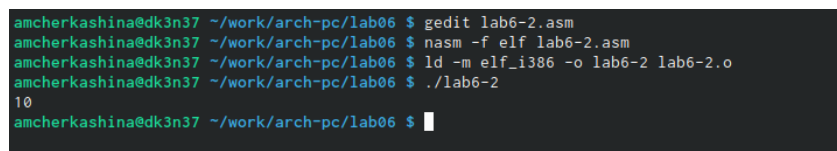


```
1 %include 'in_out.asm'
2
3 SECTION .text
4 GLOBAL _start
5 _start:
6
7 mov eax,6
8 mov ebx,4
9 add eax,ebx
10 call iprintLF
11
12 call quit
```

Matlab Ширина табуляции: 8 Ln 8, Col 11 INS

Рис. 4.11: Изменение текста программы

Создаю и запускаю новый исполняемый файл. Теперь программа складывает не соответствующие символы коды в системе ASCII, а сами числа, поэтому вывод 10 (рис. 4.12).



```
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ gedit lab6-2.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ./lab6-2
10
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $
```

Рис. 4.12: Запуск нового исполняемого файла

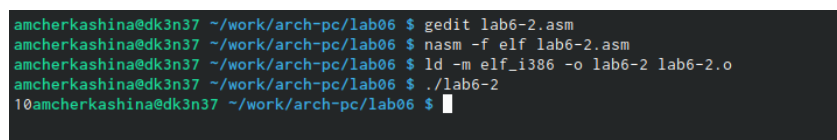
Заменяю в тексте программы функцию iprintLF на iprint (рис. 4.13).



```
1 %include 'in_out.asm'
2
3 SECTION .text
4 GLOBAL _start
5 _start:
6
7 mov eax,6
8 mov ebx,4
9 add eax,ebx
10 call iprint
11
12 call quit
```

Рис. 4.13: Изменение текста программы

Создаю и запускаю новый исполняемый файл (рис. 4.14). Функция `iprint` не добавляет к выводу символ переноса строки, в отличие от `iprintLF`.

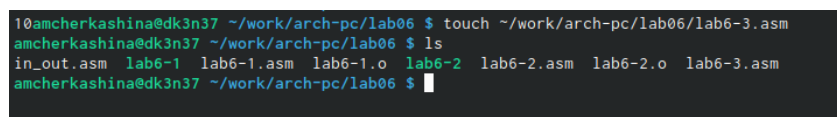


```
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ gedit lab6-2.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ./lab6-2
10amcherkashina@dk3n37 ~/work/arch-pc/lab06 $
```

Рис. 4.14: Запуск нового исполняемого файла

4.2 Выполнение арифметических операций в NASM

С помощью команды `touch` создаю файл `lab6-3.asm` в каталоге `~/work/arch-pc/lab06` (рис. 4.15).

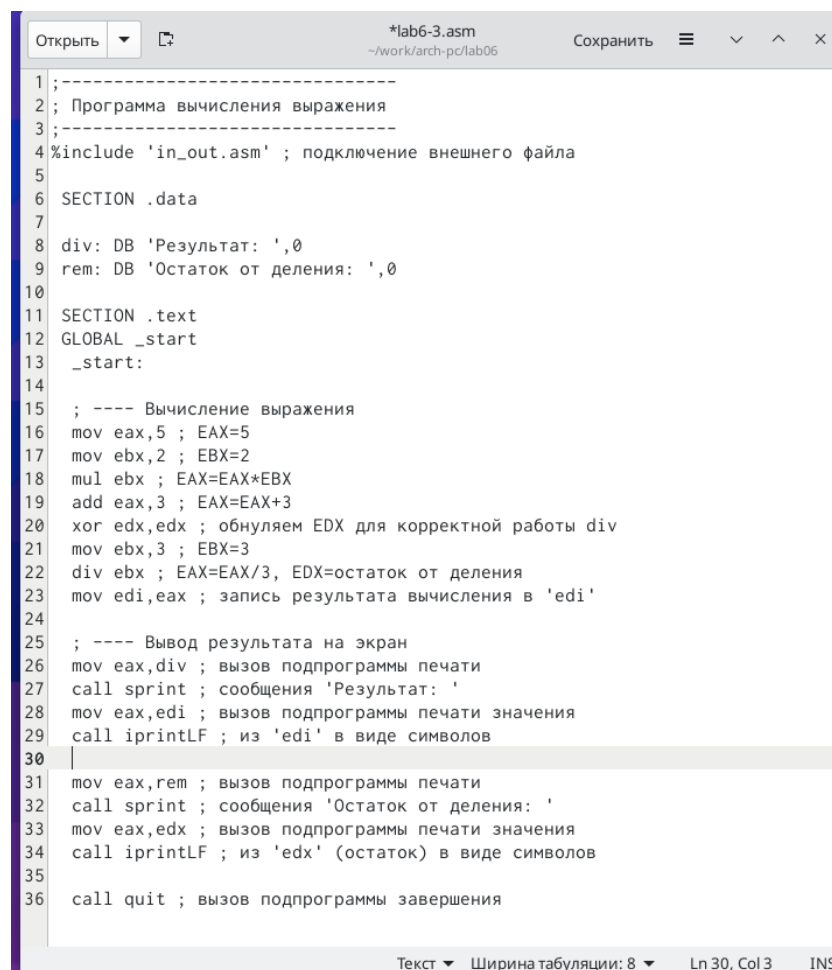


```
10amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/lab6-3.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ls
in_out.asm lab6-1 lab6-1.asm lab6-1.o lab6-2 lab6-2.asm lab6-2.o lab6-3.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $
```

Рис. 4.15: Создание файла

Ввожу в созданный файл текст программы вычисления значения выражения

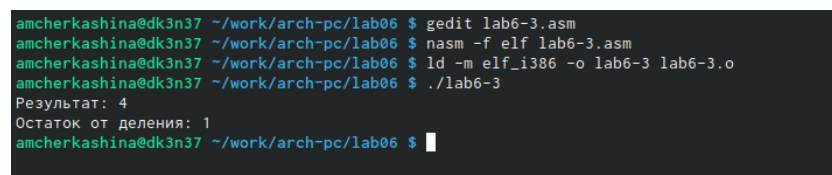
$f(x) = (5 * 2 + 3)/3$ (рис. 4.16).



```
1 ;-----
2 ; Программа вычисления выражения
3 ;-----
4 %include 'in_out.asm' ; подключение внешнего файла
5
6 SECTION .data
7
8 div: DB 'Результат: ',0
9 rem: DB 'Остаток от деления: ',0
10
11 SECTION .text
12 GLOBAL _start
13 _start:
14
15 ; ---- Вычисление выражения
16 mov eax,5 ; EAX=5
17 mov ebx,2 ; EBX=2
18 mul ebx ; EAX=EAX*EBX
19 add eax,3 ; EAX=EAX+3
20 xor edx,edx ; обнуляем EDX для корректной работы div
21 mov ebx,3 ; EBX=3
22 div ebx ; EAX=EAX/3, EDX=остаток от деления
23 mov edi,eax ; запись результата вычисления в 'edi'
24
25 ; ---- Вывод результата на экран
26 mov eax,div ; вызов подпрограммы печати
27 call sprint ; сообщения 'Результат: '
28 mov eax,edi ; вызов подпрограммы печати значения
29 call iprintLF ; из 'edi' в виде символов
30
31 mov eax,rem ; вызов подпрограммы печати
32 call sprint ; сообщения 'Остаток от деления: '
33 mov eax,edx ; вызов подпрограммы печати значения
34 call iprintLF ; из 'edx' (остаток) в виде символов
35
36 call quit ; вызов подпрограммы завершения
```

Рис. 4.16: Редактирование файла

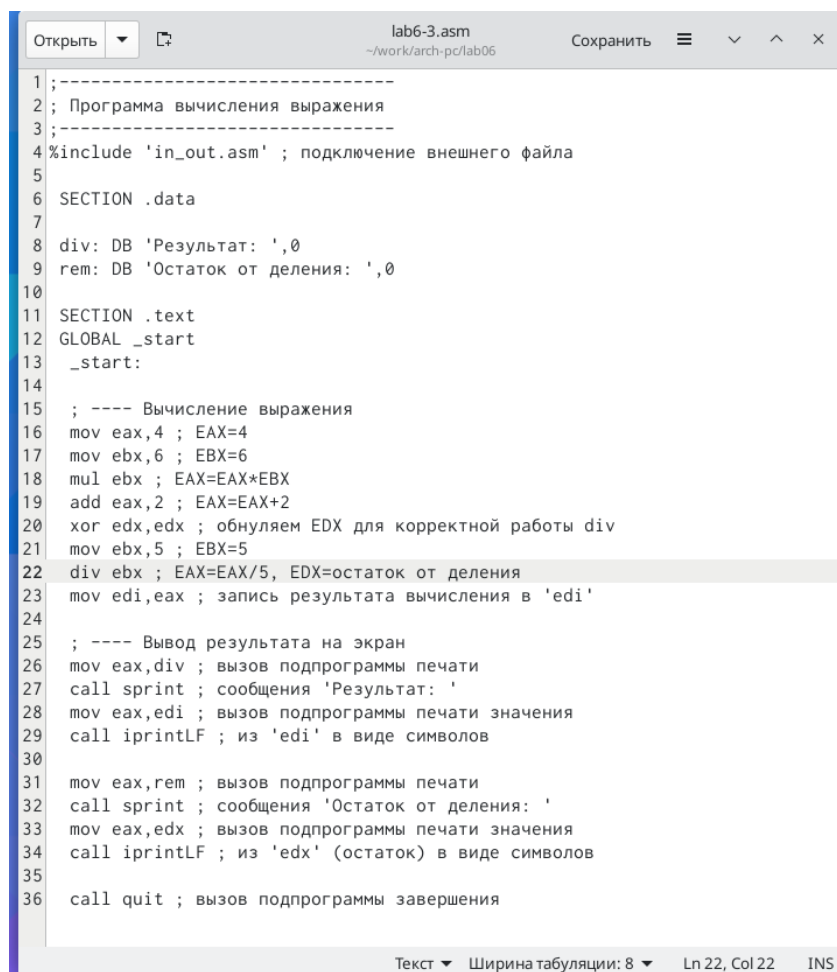
Создаю исполняемый файл и запускаю его (рис. 4.17).



```
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ gedit lab6-3.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 4
Остаток от деления: 1
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $
```

Рис. 4.17: Запуск исполняемого файла

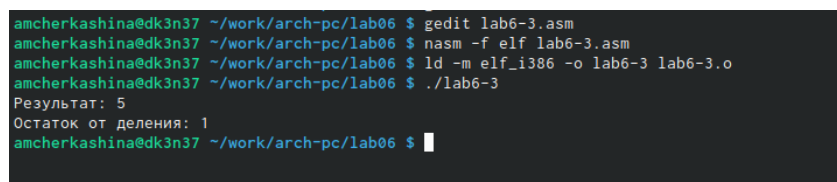
Изменяю текст программы так, чтобы она вычисляла значение выражения $f(x) = (4 * 6 + 2)/5$ (рис. 4.18).



```
1 ;-----
2 ; Программа вычисления выражения
3 ;-----
4 %include 'in_out.asm' ; подключение внешнего файла
5
6 SECTION .data
7
8 div: DB 'Результат: ',0
9 rem: DB 'Остаток от деления: ',0
10
11 SECTION .text
12 GLOBAL _start
13 _start:
14
15 ; ---- Вычисление выражения
16 mov eax,4 ; EAX=4
17 mov ebx,6 ; EBX=6
18 mul ebx ; EAX=EAX*EBX
19 add eax,2 ; EAX=EAX+2
20 xor edx,edx ; обнуляем EDX для корректной работы div
21 mov ebx,5 ; EBX=5
22 div ebx ; EAX=EAX/5, EDX=остаток от деления
23 mov edi,eax ; запись результата вычисления в 'edi'
24
25 ; ---- Вывод результата на экран
26 mov eax,div ; вызов подпрограммы печати
27 call sprint ; сообщения 'Результат: '
28 mov eax,edi ; вызов подпрограммы печати значения
29 call iprintLF ; из 'edi' в виде символов
30
31 mov eax,rem ; вызов подпрограммы печати
32 call sprint ; сообщения 'Остаток от деления: '
33 mov eax,edx ; вызов подпрограммы печати значения
34 call iprintLF ; из 'edx' (остаток) в виде символов
35
36 call quit ; вызов подпрограммы завершения
```

Рис. 4.18: Изменение программы

Создаю новый исполняемый файл и запускаю его. Проверяю его работу, посчитав значение выражения самостоятельно. Убеждаюсь, что программа сработала верно (рис. 4.19).



```
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ gedit lab6-3.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 5
Остаток от деления: 1
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $
```

Рис. 4.19: Запуск нового исполняемого файла

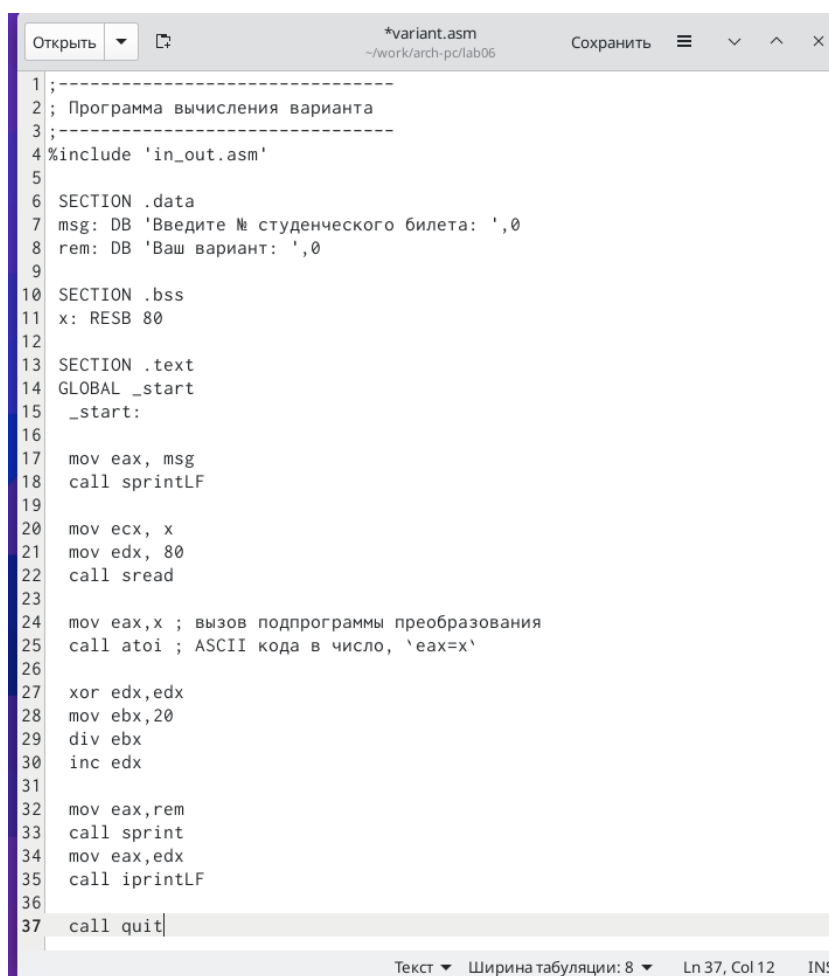
Создаю файл variant.asm в каталоге ~/work/arch-pc/lab06 с помощью команды

touch (рис. 4.20).

```
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/variant.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ls
in_out.asm lab6-1 lab6-1.asm lab6-1.o lab6-2 lab6-2.asm lab6-2.o lab6-3 lab6-3.asm lab6-3.o variant.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $
```

Рис. 4.20: Создание файла

Ввожу в созданный файл текст программы вычисления варианта задания по номеру студенческого билета (рис. 4.21).



```
1 ;-----
2 ; Программа вычисления варианта
3 ;-----
4 %include 'in_out.asm'
5
6 SECTION .data
7 msg: DB 'Введите № студенческого билета: ',0
8 rem: DB 'Ваш вариант: ',0
9
10 SECTION .bss
11 x: RESB 80
12
13 SECTION .text
14 GLOBAL _start
15 _start:
16
17 mov eax, msg
18 call sprintf
19
20 mov ecx, x
21 mov edx, 80
22 call sread
23
24 mov eax, x ; вызов подпрограммы преобразования
25 call atoi ; ASCII кода в число, 'eax=x'
26
27 xor edx, edx
28 mov ebx, 20
29 div ebx
30 inc edx
31
32 mov eax, rem
33 call sprintf
34 mov eax, edx
35 call iprintLF
36
37 call quit
```

Рис. 4.21: Редактирование файла

Создаю и запускаю исполняемый файл. Ввожу номер своего студенческого билета с клавиатуры, программа выводит мой вариант - 11. Проверяю результат

работы программы, вычислив номер своего варианта аналитически. Убеждаюсь, что программа работала верно (рис. 4.22).

```
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ gedit variant.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ nasm -f elf variant.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o variant variant.o
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ./variant
Введите № студенческого билета:
1132236030
Ваш вариант: 11
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $
```

Рис. 4.22: Запуск исполняемого файла

4.2.1 Ответы на вопросы по программе

1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’?

За вывод сообщения “Ваш вариант” отвечают следующие строки кода:

```
mov eax, rem
call sprint
```

2. Для чего используются следующие инструкции?

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

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.

3. Для чего используется инструкция “`call atoi`”?

Инструкция `call atoi` используется для вызова подпрограммы из внешнего файла, которая преобразует ASCII-код символа в целое число и записывает результат в регистр `eax`.

4. Какие строки листинга 6.4 отвечают за вычисления варианта?

За вычисления варианта отвечают следующие строки кода:

```
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
```

5. В какой регистр записывается остаток от деления при выполнении инструкции “`div ebx`”?

При выполнении инструкции `div ebx` остаток от деления записывается в регистр `edx`.

6. Для чего используется инструкция “`inc edx`”?

Инструкция `inc edx` увеличивает значение регистра `edx` на 1.

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

За вывод на экран результата вычислений отвечают следующие строки:

```
mov eax,edx
call iprintLF
```

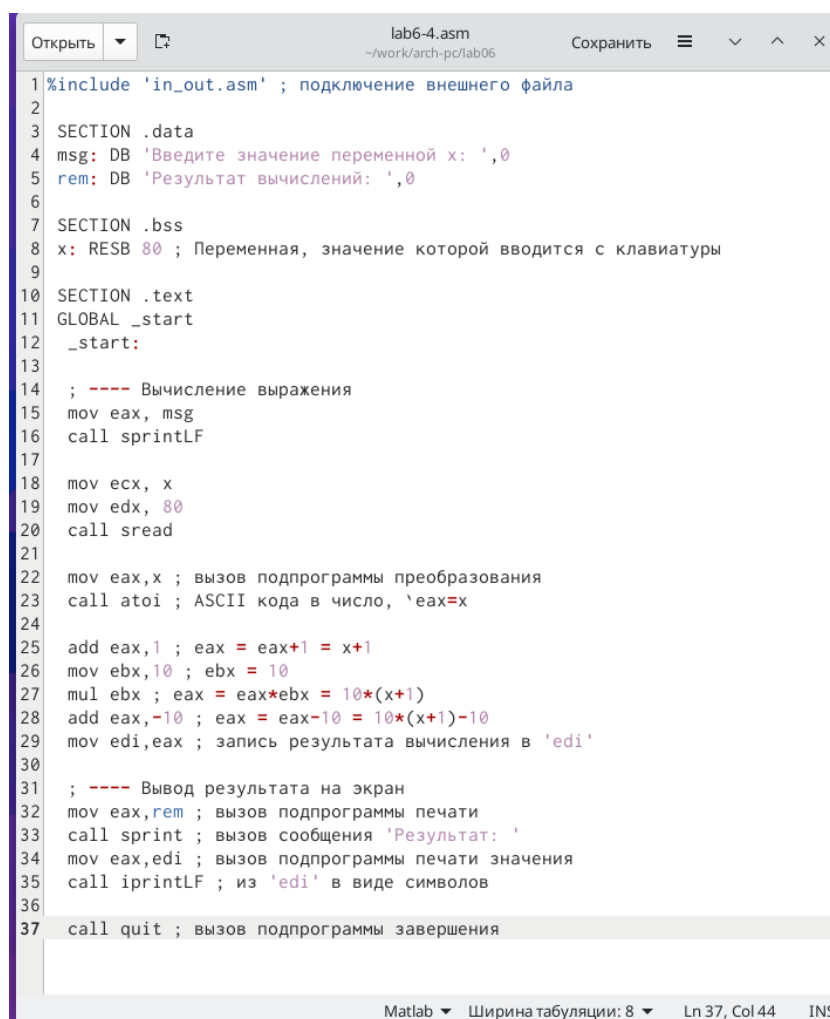
4.3 Выполнение заданий для самостоятельной работы

Создаю файл lab6-4.asm с помощью команды touch (рис. 4.23).

```
ancherkashina@dk3n37 ~/work/arch-pc/lab06 $ touch lab6-4.asm
ancherkashina@dk3n37 ~/work/arch-pc/lab06 $ ls
in_out.asm lab6-1 lab6-1.o lab6-2 lab6-2.o lab6-3 lab6-3.o lab6-4.asm variant variant.o
ancherkashina@dk3n37 ~/work/arch-pc/lab06 $
```

Рис. 4.23: Создание файла

Открываю созданный файл для редактирования, ввожу в него текст программы для вычисления значения выражения $10 \cdot (x + 1) - 10$ (вариант 11) (рис. 4.24).



```
1 %include 'in_out.asm' ; подключение внешнего файла
2
3 SECTION .data
4 msg: DB 'Введите значение переменной x: ',0
5 rem: DB 'Результат вычислений: ',0
6
7 SECTION .bss
8 x: RESB 80 ; Переменная, значение которой вводится с клавиатуры
9
10 SECTION .text
11 GLOBAL _start
12 _start:
13
14 ; ---- Вычисление выражения
15 mov eax, msg
16 call sprintf
17
18 mov ecx, x
19 mov edx, 80
20 call sread
21
22 mov eax, x ; вызов подпрограммы преобразования
23 call atoi ; ASCII кода в число, 'eax=x'
24
25 add eax, 1 ; eax = eax + 1 = x + 1
26 mov ebx, 10 ; ebx = 10
27 mul ebx ; eax = eax * ebx = 10 * (x + 1)
28 add eax, -10 ; eax = eax - 10 = 10 * (x + 1) - 10
29 mov edi, eax ; запись результата вычисления в 'edi'
30
31 ; ---- Вывод результата на экран
32 mov eax, rem ; вызов подпрограммы печати
33 call sprint ; вызов сообщения 'Результат: '
34 mov eax, edi ; вызов подпрограммы печати значения
35 call iprintLF ; из 'edi' в виде символов
36
37 call quit ; вызов подпрограммы завершения
```

Рис. 4.24: Написание программы

Создаю и запускаю исполняемый файл. Ввожу значение $x_1 = 1$ с клавиатуры. В результате выполнения программы получаю число 10 (рис. 4.25). Проверяю результат работы программы, подсчитав значение выражения самостоятельно. Убеждаюсь, что программа сработала верно.

```
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ gedit lab6-4.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ nasm -f elf lab6-4.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-4 lab6-4.o
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ./lab6-4
Введите значение переменной x:
1
Результат вычислений: 10
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $
```

Рис. 4.25: Запуск исполняемого файла

Снова запускаю исполняемый файл для проверки работы программы с другим значением на вводе ($x_2 = 7$). Программа выводит результат 70 (рис. 4.26). Проверяю результат работы программы, подсчитав значение выражения самостоятельно. Убеждаюсь, что программа сработала верно.

```
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $ ./lab6-4
Введите значение переменной x:
7
Результат вычислений: 70
amcherkashina@dk3n37 ~/work/arch-pc/lab06 $
```

Рис. 4.26: Запуск исполняемого файла

Листинг 4.1. Программа для вычисления значения выражения $10 \cdot (x + 1) - 10$:

```
%include 'in_out.asm' ; подключение внешнего файла
```

```
SECTION .data
```

```
msg: DB 'Введите значение переменной x: ',0
```

```
rem: DB 'Результат вычислений: ',0
```

```
SECTION .bss
```

x: **RESB** 80 ; Переменная, значение которой вводится с клавиатуры

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

add **eax**, 1 ; $eax = eax + 1 = x + 1$

mov **ebx**, 10 ; $ebx = 10$

mul **ebx** ; $eax = eax * ebx = 10 * (x + 1)$

add **eax**, -10 ; $eax = eax - 10 = 10 * (x + 1) - 10$

mov **edi**, **eax** ; запись результата вычисления в 'edi'

; ---- Вывод результата на экран

mov **eax**, rem ; вызов подпрограммы печати

call sprintf ; вызов сообщения 'Результат: '

mov **eax**, **edi** ; вызов подпрограммы печати значения

call iprintLF ; из 'edi' в виде символов

call quit ; вызов подпрограммы завершения

5 Выводы

При выполнении данной лабораторной работы я освоила арифметические инструкции языка ассемблера NASM.

Список литературы

1. Архитектура ЭВМ
2. Таблица ASCII